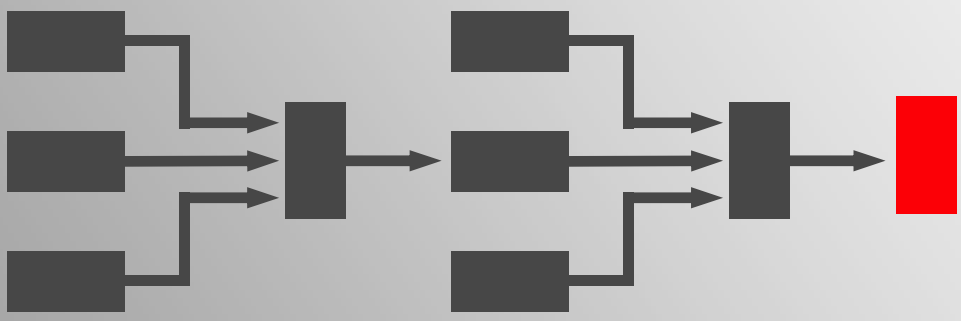
Intelligente Diagnose in der industriellen Anwendung



Intelligent Diagnosis in Industrial Applications

Lothar Hotz, Peter Struss, Thomas Guckenbiehl (Hrsg.)

Lothar Hotz, Peter Struss, Thomas Guckenbiehl (Hrsg.)

INDIA

Intelligente Diagnose in der Anwendung

Intelligent Diagnosis for Industrial Applications

SEITE WIRD VOM VERLAG DURCH ZIPSEITE ERSETZT!

Herausgeber:

Lothar Hotz
Universität Hamburg
Fachbereich Informatik
Vogt-Kölln-Strasse 30
D-22527 Hamburg
E-Mail: hotz@informatik.uni-hamburg.de

Prof. Dr. Dr. Peter Struss
Technische Universität München
Institut für Informatik
Orleansstrasse 34
D-81667 München
E-Mail: struss@informatik.uni-muenchen.de

Thomas Guckenbiehl
Fraunhofer-Institut für Informations- und Datenverarbeitung IITB
Fraunhoferstrasse 1
D-76131 Karlsruhe
E-Mail: guc@iitb.fhg.de

Das diesem Buch zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie unter den Förderkennzeichen 01 IN 509 [A2, B5, D0, F6, G9, H1, I4, K4(E3)] gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

INDIA - Intelligente Diagnose in der Anwendung
Intelligent Diagnosis for Industrial Applications
Lothar Hotz, Peter Struss, Thomas Guckenbiehl (Hrsg.)
ISBN ....

Vorwort

Modellbasierte Diagnosetechniken werden seit fast 20 Jahren theoretisch und praktisch entwickelt. Gegenwärtig dringen diese Techniken verstärkt in die Anwendung im industriellen Umfeld. Diese Entwicklung sollte in dem vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) geförderten Verbundprojekt INDIA – intelligente Diagnose in der industriellen Anwendung – weiter vorangetrieben werden. Das vorliegende Buch beschreibt die wesentlichen Ergebnisse dieses Projekts. Die Vorgehensweise im INDIA Projekt war durch drei Zielsetzungen bestimmt:

- Transfer der vorhandenen Techniken direkt in Anwendungen
- Orientierung von Grundlagenforschung an konkreten Anwendungen
- Erarbeitung von übertragbaren Verfahren für den Umgang mit verschiedenartigen Anwendungen.

Entsprechend setzte sich das Konsortium aus Forschungsinstituten, Systemhäusern und Endanwendern zusammen, die in drei Säulen jeweils ein Problemfeld bearbeiteten: mechatronische Fahrzeugsysteme, Flurförderzeuge (z.B. Gabelstapler) und Färbeanlagen.

Modellbasierte Systeme, wie sie in der Künstlichen Intelligenz entwickelt wurden, erlauben die Repräsentation von Wissen über einen Bereich, wie z.B. eine Klasse technischer Systeme als Gegenstand von Diagnoseproblemen, und die Anwendung automatischer Schlußfolgerungsverfahren für die computergestützte Problemlösung. Sie gehen somit über die rein mathematische Modellierung hinaus und unterstützen die Darstellung und Nutzung von Wissen auf einer konzeptuellen Ebene. Die Grundlage aller Teilprojekte in INDIA sind modellbasierte Techniken mit zwei Merkmalen. Zum einen werden Modelle eines technischen Systems automatisch aus Teilmodellen aus einer Bibliothek zusammengesetzt. Zum anderen sind die in der Auswertung betrachteten Zeitachsen und Werteräume häufig nicht numerisch, sondern qualitativ abstrahiert.

Die Anwendung gebietsunabhängiger Diagnose-Algorithmen reduziert die Erstellung von gerätespezifischen Diagnosesystemen auf die Modellerstellung. Die Verwendung kompositionaler Modelle reduziert die Modellerstellung auf die Beschreibung der Gerätestruktur. Modellbasierte Diagnosesysteme sind daher die Antwort auf den wachsenden Aufwand für Diagnose und Fehleranalyse aufgrund von zunehmender Komplexität und Variantenvielfalt der Systeme.

Die modellbasierten Techniken sind in der Lage, Abweichungen zwischen dem beobachteten Verhalten und dem aus dem Modell hergeleiteten Verhalten auf Teilmodelle zurückzuführen. Das Modell kann damit nicht nur, wie in klassischen mathematischen Verfahren, zur Fehler*detektion* verwendet werden, sondern erlaubt eine *Lokalisierung* des Fehlers in Komponenten oder Teilfunktionen, bei Einsatz von Fehlermodellen sogar eine Fehler*identifikation*.

Das Buch wendet sich an forschungsinteressierte Industrievertreter, anwendungsinteressierte KI-Wissenschaftler und Studenten, sowie an Ingenieure aus dem akademischen und industriellen Bereich.

Industriellen Anwendern soll das Buch Gelegenheit geben, sich an konkreten Anwendungsbeispielen statt durch KI-Textbücher oder reine Forschungspapiere ein Bild vom Reifegrad

und dem zu erwartenden Nutzen modellbasierter Systeme zu machen. In verschiedenen Beiträgen wird versucht, die sich aus praktischen Randbedingungen und den gegenwärtigen Arbeitsprozessen ergebenden Anforderungen darzustellen und zu analysieren. Auf dieser Grundlage wird dann der Gewinn durch den Einsatz modellbasierter Techniken erkennbar, ohne zu detailliert auf die theoretischen und technischen Grundlagen einzugehen. Natürlich sind die bearbeiteten Anwendungsprobleme nur exemplarisch und bei weitem nicht erschöpfend, aber hoffentlich ausreichend, um Assoziationen zu weiteren möglichen Anwendungen zu wecken.

Forscher und Studenten der KI finden hier ein Forschungsgebiet dargestellt, das sich relativ erfolgreich auf den Weg zu echten Anwendungsaufgaben begeben hat. Wir hoffen, daß das Buch hier vor allem zwei Erkenntnisse illustriert: zum einen, daß die bloße Entwicklung einer „prinzipiellen" Problemlösung (etwa eines Diagnose-Algorithmus) oder gar nur einer Theorie dazu nur der erste Schritt sein kann und dieser häufig völlig wertlos ist, wenn die Anforderungen der praktischen Seite der Aufgabe unberücksichtigt bleiben (etwa die eingeschränkte Meßbarkeit von Größen oder die Kosten bestimmter Arbeitsschritte). Zum anderen, daß die frühe und essentielle Einbeziehung solcher Aspekte nicht eine Abkehr von den „wesentlichen" Forschungsaufgaben bedeutet, sondern geradezu eine Fokussierung auf sie. Dies sollen die in diesen Band aufgenommenen forschungsbezogenen Beiträge belegen, die z.T. bereits international publiziert wurden.

Ingenieuren an Hochschulen und in der industriellen Praxis bereitet die andersartige Sicht auf Problemstellungen aus ihrer „Hausmacht" oft erst einmal Probleme, denn sie bringt auch andere Methoden und Sprachen mit sich. Die Darstellung konkreter Anwendungen sollte illustrieren, daß die KI das Repertoire der Techniken im Ingenieurbereich nicht einfach weiter verallgemeinert, sondern qualitativ ergänzt: erstens wird ermöglicht, technische Systeme und Abläufe nicht nur rein mathematisch, sondern auch konzeptuell auf dem Rechner zu beschreiben und damit ingenieurmäßiges Wissen automatischen Schlußfolgerungsprozessen zugrunde zu legen. Zweitens bietet sie Techniken an, um die in den geläufigen quantitativen mathematischen Modellen enthaltene Charakterisierung von Systemverhalten auf einer qualitativen Abstraktionsebene auszudrücken, die für Diagnoseaufgaben häufig angemessener und oft unabdingbar ist. *Nur die wesentlichen und möglichen Unterscheidungen* sollte ein Modell treffen, um die nötige Effizienz (etwa für den Realzeiteinsatz) und Robustheit (z.B. angesichts von Meßungenauigkeiten) zu erzielen. Die geschilderten Anwendungen zeigen, daß diese KI-Techniken wirksam auf die Befriedigung praktischer industrieller Anforderungen zielen und daher perspektivisch die Standardwerkzeuge der Ingenieure ergänzen sollten.

Das Buch gibt zunächst einen knappen Überblick über die drei durch jeweils einen Anwendungsbereich definierten Projektsäulen. Jede Säule ist dann durch einen gesonderten Teil vertreten:

- Teil I beschreibt die Diagnose von mechatronischen Fahrzeugsystemen. Diese Beiträge wurden von den Projektpartnern Technische Universität München Institut für Informatik und Robert Bosch GmbH Stuttgart verfaßt. Als Schwerpunkte wurden hier automatische Generierung von Prüfplänen für die Anleitung zur Fehlersuche und Reparatur und Verhaltensvorhersagen für die FMEA bearbeitet.
- Teil II beschreibt die Diagnose von Flurförderzeugen. Diese Beiträge wurden von den Projektpartnern Labor für Künstliche Intelligenz der Universität Hamburg, ServiceXpert

GmbH Hamburg und Still GmbH Hamburg verfaßt. Schwerpunkt war hier die automatische Generierung von Fehlerbäume aus Modellen technischer Systeme.

- Teil III beschreibt die Diagnose von Versorgungseinrichtungen für Färbereien. Diese Beiträge wurden von den Projektpartnern Fraunhofer-Institut für Informations- und Datenverarbeitung Karlsruhe, R.O.S.E. Informatik GmbH Heidenheim und Then Maschinen- und Apparatebau GmbH Schwäbisch Hall verfaßt. Schwerpunkte waren hier konsistenzbasierte Diagnoseverfahren und ihre effektive industrielle Einführung.

INDIA verbindet Ansätze aus der Künstlichen Intelligenz und den klassischen Ingenieurwissenschaften. Diese (teilweise durchaus spannungsreiche) Verbindung, insbesondere aber die intensive Auseinandersetzung mit den Anforderungen der industriellen Praxis trieben das Projekt voran. In den folgenden Geleitworten werden die Inhalte des Buches aus den genannten Perspektiven eingeordnet und reflektiert: durch Herrn Thomas Forchert von Daimler-Chrysler aus Sicht der Industrie, durch Prof. Wolfgang Wahlster vom Deutschen Forschungszentrum für KI und der Universität Saarbrücken aus Sicht der KI-Forschung und durch Prof. Jan Lunze von der Technischen Universität Hamburg-Harburg aus Sicht der klassischen Ingenieurwissenschaften.

Das Projekt INDIA wurde vom Bundesministerium für Bildung und Forschung gefördert. Ohne diese Förderung hätten die hier vorgestellten Arbeiten nicht durchgeführt werden können. Hierfür und für die projektbegleitende Unterstützung durch den Projektträger Informationstechnik der Deutschen Forschungsanstalt für Luft- und Raumfahrt Berlin, danken wir. Weiterhin möchten wir uns bei allen Personen bedanken, welche zum Erfolg des Projekts beigetragen haben. Zusätzlich zu den Autoren dieses Buches sind dies insbesondere: Ben Auch, Sven Bertel, Anton Beschta, Claudia Böttcher, Axel Brinkop, Andrea Brkic, Frank Buhr, Oskar Dressler, Georg Fromme, Oliver Gülden, Christoph Hansen, Ulrike Höfer, Robert Illner, Thomas Jäpel, Michael Kappert, Sabine Kockskämper, Heinz Marburger, Hans-Werner Früchtenicht, Ralf Möller, Omar Nuri, Michael Okon, Volker Palm, Ulf Sauerland, Christopher Schmid, Rainer Schmitz, Wolfgang Schneider, Werner Seibold, Gerhard Sutschet, Robby Technow, Guido Vehring, Uwe Zimmer und Matthias Zimmermann-Sturm.

Lothar Hotz
Peter Struss
Thomas Guckenbiehl
Hamburg, München, Karlsruhe im Dezember 2000

Geleitwort aus Sicht der Industrie

Dipl.-Ing. Thomas Forchert, DaimlerChrysler AG, Forschung und Technologie

Die „intelligente Diagnose in der industriellen Anwendung“ ist seit Einführung des Mikroprozessors Gegenstand der Forschung und Entwicklung in der Automobilindustrie. Die fortschreitende Entwicklung von mikroprozessorgesteuerten Elektroniksystemen ermöglichte eine programmierte Störungserkennung und Kommunikation dieser Störungsinformation an den Werkstattechniker oder Bediener des technischen Systems. In der frühen Phase (1985 – 1995) der intelligenten Diagnose wurden insbesondere wissensbasierte Techniken betrachtet, in der aktuellen Phase sind modellbasierte Techniken im Fokus.

Das vorliegende Buch mit einer Zusammenfassung der Ergebnisse des BMBF geförderten Projekts INDIA trifft somit einen aktuellen Forschungsfokus der Industrie. Die Vision einer technischen Begleitung von Fahrzeugen und Anlagen durch Callcenter, mit Ferndiagnosefunktionen, wird in der Zukunft insbesondere durch den Einsatz modellbasierter Techniken in die Realität umzusetzen sein. Während die wissensbasierten Techniken weitgehend auf den ergänzenden Einsatz in Werkstätten oder Bedienzentralen „Offboard“ beschränkt sind, bestehen berechtigte Hoffnungen, die modellbasierten Techniken im technischen System selbst „Onboard“ zum Einsatz zu bringen.

Die Perspektive dieser Techniken ist als exzellent einzuschätzen. Es liegt nun an den Toolherstellern, den Systemlieferanten, sowie den Fahrzeug-/Anlagenherstellern, den industriellen Einsatz zügig herbeizuführen.

Geleitwort aus Sicht der KI-Forschung

Prof. Dr. Wolfgang Wahlster, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken

Die modellbasierte Diagnose und das qualitative Schließen gehören zu den faszinierendsten Gebieten der Künstlichen Intelligenz, weil sie tiefliegende und prinzipielle Problemstellungen mit hohem Anwendungspotential verbinden. Die Theorie und Methodik der Künstlichen Intelligenz wurde durch die fundamentalen Beiträge aus dem Bereich der modellbasierten Systeme u.a. für die Teilgebiete Wissensrepräsentation, Default Reasoning, nicht-monotones Schließen, Abduktion, zeitliches, räumliches und fallbasiertes Schließen, Constraint-Systeme, Truth Maintanance Systeme sowie probabilistische und entscheidungstheoretische Ansätze für den Umgang mit unsicherer Information wesentlich geprägt. Die modellbasierte Diagnose hat heute die traditionellen regelbasierten Diagnoseexpertensysteme weitestgehend abgelöst. Sehr positiv ist auch zu werten, daß für weite Teile dieses KI-Fachgebietes eine mathematisch-logisch fundierte Theorie vorliegt.

In der deutschen Forschung zur Künstlichen Intelligenz gehören die modellbasierte Diagnose und das qualitative Schließen, die im vorliegenden Sammelband behandelt werden, seit mehr als fünfzehn Jahren zu den international konkurrenzfähigen Forschungsgebieten. Neben der Sprachverarbeitung, den Deduktionssystemen, der Kognitiven Robotik und der Interpretation von Bildfolgen spielen die deutschen Forschungsgruppen zur modellbasierten Diagnose weltweit in der ersten Liga. Besonders die Gruppe um Peter Struss an der Technischen Universität München, der Gruppe um Bernd Neumann an der Universität Hamburg und die Gruppe um Hans-Werner Früchtenicht am Fraunhofer-Institut für Informations- und Datenverarbeitung in Karlsruhe gelten als Pioniere auf diesem wichtigen Gebiet der Informatik. Alle drei genannten Forschungsgruppen haben im vorliegenden Band tiefliegende Forschungsresultate publiziert, die den internationalen Forschungsstand vorantreiben. Auf den bisherigen elf internationalen Workshops über „Principles of Diagnosis“ und den vierzehn Workshops über „Qualitative Reasoning“ spielten deutsche Beiträge immer eine wichtige Rolle. Auch auf den renommiertesten internationalen KI-Tagungen wie der zweijährigen Weltkonferenz für Künstliche Intelligenz der IJCAI, der europäischen KI-Konferenz ECAI und der amerikanischen AAAI wurden stets Beiträge aus diesem Bereich der deutschen KI-Forschung akzeptiert.

Unvergessen bleibt für viele KI-Forscher sicherlich die überzeugende Praxis-Demonstration der modellbasierten Online-Diagnose auf der letzten IJCAI-99 in Stockholm. In einem Volvo 850 konnte ein pneumatisches Leck in dessen elektronischer Einspritzung induziert werden. Das defekte Versuchsfahrzeug, das im thematisch verwandten VMBD-Projekt der EU instrumentiert wurde, war vor dem Ausstellungsgebäude geparkt. Man konnte die Russentwicklung durch den Defekt am Auspuff beobachten und dann am Bildschirm des Diagnosesystems die Arbeitsweise des in INDIA entwickelten Systems verfolgen und die Diagnoseresultate nachvollziehen.

Die internationalen Erfolge auf diesem Gebiet sind nicht zuletzt auf die langjährige Förderung durch die Deutsche Forschungsgemeinschaft (DFG) und das Bundesministerium für Bildung und Forschung (BMBF) zurückzuführen. Zunächst wurde die Grundlagenforschung auf dem Gebiet der wissensbasierten Fehlerdiagnose, etwa in unserem DFG-Sonderforschungsbereich 314 „Künstliche Intelligenz – Wissensbasierte Systeme“ von 1985 – 1995, initiiert. Dann folgte eine Serie von drei aufeinanderfolgenden BMBF-Verbundprojekten TEX-B,

BEHAVIOUR und INDIA, welche immer stärker in Richtung des industriellen Einsatzes dieser Technologie strebten.

Am Ende dieser rund fünfzehnjährigen Entwicklung stehen erfreulicherweise auch Softwareprodukte, die von Aus- und Neugründungen der beteiligten Forscher kommerziell vertrieben werden. Spin-Off Unternehmen wie die von Peter Struss gegründete OCC'M GmbH, die das Modellierungs- und Diagnosesystem RAZ'R vertreibt, oder die aus dem MAZ in Hamburg ausgegründete Firma ServiceXpert GmbH, die das Diagnosesystem DiaMon vertreibt, sowie die modellbasierte Analysesoftware RODON der R.O.S.E Informatik GmbH sind eine ausgezeichnete Bilanz der langjährigen BMBF-Förderung.

Trotz der langen Forschungstradition handelt es sich um ein hochaktuelles Gebiet. Die digitale Modellierung technischer und biologischer Systeme in einem über das Internet austauschbaren Repräsentationsformat, das sowohl für den menschlichen Leser als auch für maschinelle Software-Agenten verständlich ist, kann heute als eine der wichtigsten Herausforderungen an das Wissensmanagement betrachtet werden. Produkt-Ontologien, die in XML-Erweiterungen wie DAML und OIL spezifiziert werden, sind die Grundlage für agentenbasierten elekronischen Handel und globales Wissensmanagement in weltweit agierenden Unternehmen.

Das Anwendungspotential des qualitativen Schließens ist mit den Erfolgen im Bereich der technischen Systeme bei weitem nicht erschöpft: So ist es derzeit in der Bioinformatik eines der zentralen Probleme, die metabolischen Pfade im Körper zu analysieren, wozu eine Kombination von qualitativen und quantitativen Modellierungsverfahren erforderlich ist. Es wird modelliert, wie die einzelnen Stoffe und Proteine in einem Organismus eingesetzt werden können und wie sie zusammenspielen müssen, um eine gewünschte Wirkung im Organismus auszulösen. Auch in der hochwertigen Computeranimation wird immer stärker die Berücksichtigung physikalischer Effekte (z.B. Reibungskräfte) gefordert, was bei Echtzeitanimation aber ohne qualitative physische Modelle nicht möglich sein wird. Schließlich kommt auch anspruchsvolle Bildungssoftware der nächsten Generation kaum ohne eine modellbasierte Diagnose von Fehlkonzepten beim Lernenden aus, wenn intelligente Tutor- und Coaching-Systeme für virtuelle Lernumgebungen zu entwickeln sind.

Obwohl die modellbasierte Diagnose nicht mein eigenes Forschungsgebiet innerhalb der Künstlichen Intelligenz ist, habe ich in meinen Einführungsvorlesungen seit mehr als zehn Jahren immer ein Kapitel zu diesen für die KI grundlegenden Methoden aufgenommen. Das vorliegende Buch bietet für Dozenten und Studenten der KI eine ausgezeichnete Vertiefungsmöglichkeit für diesen anspruchsvollen Stoff, da es anhand zahlreicher, gut motivierter Beispiele modernste Methoden der intelligenten Diagnose auf hohem fachlichen Niveau darstellt und eine Vielzahl von Querbezügen zu fundamentalen Fragen der KI enthält.

Geleitwort aus Sicht der klassischen Ingenieurwissenschaften

Prof. Dr.-Ing. Jan Lunze, Technische Universität Hamburg-Harburg

Es ist ein alter Traum der Menschheit, „intelligente Maschinen“ zu entwickeln, die sich wie Menschen Wissen aneignen, die zu lösenden Probleme selbständig erkennen und flexibel auf sich verändernde Situationen reagieren. Mehrere Fachgebiete arbeiten auf unterschiedliche Weise an der Realisierung dieser Vision. Die Technische Kybernetik versucht, die Steuerungsprinzipien der Natur in die Technik zu übertragen und dabei neue Lösungswege zu erkennen. Die Automatisierungstechnik will die Autonomie technischer Systeme dadurch erhöhen, daß sich die Systeme selbständig den durch die Umwelt vorgegebenen, wechselnden Bedingungen anpassen. Schließlich versucht die Künstliche Intelligenz, kognitive Fähigkeiten des Menschen nachzubilden und damit intelligentes Verhalten künstlich zu erzeugen.

In der Künstlichen Intelligenz hat man dabei die Ziele am weitesten gesteckt und durch die Verwendung symbolischer (anstelle rein numerischer) Informationen bei der Repräsentation und bei der Verarbeitung von Wissen den prinzipiell größten Spielraum geschaffen. Mit Bezug auf ingenieurtechnische Anwendungen konzentriert sich die Forschung auf typische Aufgabenklassen wie Analyse, Diagnose oder Entwurf, wobei die dabei betrachteten Aufgaben und deren Lösungswege der Wissensverarbeitung zugänglich gemacht werden. Da hierbei die größtmögliche Allgemeinheit angestrebt wird, erscheinen die in den klassischen Ingenieurdisziplinen wie Elektrotechnik oder Maschinenbau untersuchten Probleme als Spezialfälle einer allgemeinen Theorie, ohne daß jedoch diese Theorie zu einem Standardhandwerkzeug des Ingenieurs geworden wäre.

Die in den letzten fünfzehn Jahren durchlaufenen Entwicklungsstufen werden an drei vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie geförderten Projekten deutlich. In TEX-B, einem Vorgängerprojekt zu dem in diesem Buch beschriebenen Projekt INDIA, wurden Methoden untersucht, mit dem die qualitativen Schlußweisen des Ingenieurs nachgebildet werden sollten. Sie folgten dem Vorbild, daß der Ingenieur oft schon nach einem kurzen Blick auf einen Schaltplan die Funktionsweise einer elektrischen Schaltung erkennen kann, weil er das Verhalten aus dem die Struktur beschreibenden Schaltplan abliest. Im Nachfolgeprojekt BEHAVIOR wurden diese Methoden in Werkzeuge für die rechnergestützte Arbeitsweise umgesetzt und an einfachen Beispielen wie einem Ballasttank erprobt. Das in diesem Band vorgestellte Projekt INDIA beschäftigte sich nun mit der industriellen Anwendung von Wissensverarbeitungsmethoden, wobei innerhalb einer dreistufigen Projektstruktur durch Forschungsinstitute Modellierung- und Diagnosemethoden erarbeitet, von Softwareentwicklern die für die Umsetzung notwendigen Systeme entwickelt und durch die industriellen Anwender die konkreten Beispiele einschließlich der dabei auftretenden praktischen Randbedingungen vorgegeben wurden. Im Mittelpunkt standen Diagnoseaufgaben, die an mechatronischen Fahrzeugsystemen, Gabelstablern und Färbereiversorgungssystemen erprobt wurden.

Der vorliegende Band gibt ein vielschichtiges Bild dieser Untersuchungen. Einerseits beschreibt er im Detail die praktischen Randbedingungen, die neue Methoden erfüllen müssen, um in der ingenieurtechnischen Praxis eingesetzt zu werden. Es genügt nicht, daß die betrachtete Aufgabe gegenüber den bisher lösbaren Problemen deutlich erweitert ist. Für eine praktische Anwendung ist es mindestens genauso wichtig, daß alle einem erfahrenen Fachmann verfügbaren Informationen nutzbar und die entwickelten Lösungen in den Arbeitspro-

zeß des entsprechenden Anwendungsgebietes integrierbar sind. Darüber berichten die Beiträge dieses Bandes, die die konkreten Diagnoseprobleme aus Anwendersicht beschreiben.

Aus methodischer Sicht konzentrieren sich die Berichte auf die Anwendung des Prinzips der konsistenzbasierten Diagnose unter den in den drei genannten Gebieten dominierenden Randbedingungen. Die Form der Modelle ist verglichen mit den für die Diagnose in der Vergangenheit vorgeschlagenen qualitativen Modellformen relativ einfach. Der Schwerpunkt wurde hier jedoch nicht auf neue Modellformen, sondern auf den Modellbildungsweg gelegt, also auf den Weg von den in der Praxis vorhandenen Informationen zu den in der Diagnose einsetzbaren Modellen. Daß man diesen Weg mit der symbolischen Informationsverarbeitung wesentlich unterstützen kann, demonstriert dieser Band nachdrücklich.

INDIA ist zweifellos ein wichtiger Schritt in Richtung Anwendung, auch wenn die in der ingenieurtechnischen Praxis oft dominierenden Forderungen nach Robustheit der Lösung gegenüber Meßunsicherheiten bzw. nach Einhaltung von Echtzeitbedingungen und Sicherheitsaspekten nur am Rande behandelt werden. Der Schwerpunkt liegt auf der Methodik, die hier in anwendungsorientierter Weise weiterentwickelt und dadurch einer praktischen Anwendung zugänglich gemacht wurde. Dabei mußte sich die Forschung der Künstlichen Intelligenz, die traditionell nach höchstmöglicher Allgemeinheit strebt, an den praktischen Bedingungen der Praxis orientieren, was zu konkreten, rechnergestützt realisierten Lösungen geführt und damit die Verbreitung der Künstlichen Intelligenz vorangetrieben hat. Ein noch ausstehender, aber mindestens ebenso wichtiger Schritt ist die Darstellung dieser Methoden in einer dem Ingenieur geläufigen Form, bei der die Begriffswelt und die Herangehensweise des Ingenieurs mit den Methoden der Künstlichen Intelligenz kombiniert und dabei neben dem Wissen über den Aufbau und die Funktion des zu diagnostizierenden Objektes auch die häufig in quantitativen Modellen enthaltenen Informationen über die dynamischen Eigenschaften der Prozesse für die Diagnose genutzt werden.

Inhaltsverzeichnis

Anhang

INDIA — Intelligente Diagnose in der industriellen Anwendung

Roman Cunis – ServiceXpert GmbH Hamburg

Peter Struss – TU München, Institut für Informatik

1 Überblick

Gegenstand des Projekts INDIA waren **modellbasierte Diagnosesysteme zur automatischen Diagnose und Überwachung technischer Systeme und Prozesse**. In diesem Bereich gab es bisher im wesentlichen nur einige Forschungsprototypen, die zeigten, daß bestimmte in der Forschung entwickelte Techniken „im Prinzip“ — aber oft unter Abstraktion von wesentlichen realen Zielen, und Bedingungen — funktionieren.

Die Ziele von INDIA waren

- der Transfer der Technologie in reale industrielle Anwendungen,
- die Fokussierung der Forschung auf die dabei auftretenden Probleme und
- die Entwicklung übertragbarer Verfahren.

Letztere sollten entstehen, indem verschiedene, sehr heterogene Anwendungen in drei sogenannten „Säulen“ bearbeitet werden und deren Anforderungen und Lösungsansätze im projektinternen Austausch abgeglichen werden.

Die drei Säulen bestanden jeweils aus

- einem Anwender für das konkrete Problem,
- einer Forschungsinstitution für die theoretischen Grundlagen und prinzipiellen Verfahren und
- einem Systemhaus für den Brückenschlag zwischen Theorie und Praxis, sprich für die Umsetzung der Verfahren in industriell einsetzbare Systeme.

Die drei Säulen umfaßten jeweils die folgenden Anwendungen mit ihren Partnern:

- Flurförderzeuge:
  - STILL GmbH, Hamburg
  - ServiceXpert GmbH, Hamburg
  - LKI Labor für Künstliche Intelligenz, Universität Hamburg
- Anlagenbau:
  - THEN Maschinen- und Apparatebau, Schwäbisch Hall
  - R.O.S.E. Informatik GmbH, Heidenheim
  - Fraunhofer-Institut für Informations- und Datenverarbeitung IITB, Karlsruhe
- Mechatronische Kfz-Systeme:
  - Robert Bosch GmbH, Stuttgart
  - TU München, Model-Based Systems & Qualitative Reasoning Group

Die folgenden Probleme und Aufgaben standen bei der Projektarbeit im Vordergrund:

- Die **Reduktion der Kosten zur Entwicklung und Pflege** von Diagnosesystemen ist insbesondere dann unabdingbar, wenn technische Systeme in vielen verschiedenen Varianten produziert werden (**„Variantenproblem“**). Dies tritt bei Serienprodukten (z.B. Kfz-Subsysteme) genauso auf wie bei komplexen Systemen, die als Unikate hergestellt werden (z.B. in der Anlagentechnik): In beiden Fällen gibt es viele ähnliche Komponenten, die hinsichtlich der Diagnose gleich oder vergleichbar behandelt werden sollen, aber immer wieder in unterschiedlichen Varianten und Kombinationen vorkommen.
- **Die Integration neuer Diagnosetechnologien in den Arbeitsablauf** ist neben der Entwicklung der eigentlichen Technologien ein wesentlicher Aspekt, da sich mit der Einführung computergestützter Diagnose herkömmliche Diagnose- und Serviceabläufe deutlich ändern können. Hier ist frühzeitig ein Konzept für eine Integration zu vor- und nachgeschalteten Prozessen zu schaffen, damit bestehende Arbeitsabläufe so wenig wie möglich gestört werden.

- werden: Dies betrifft insbesondere die Wiederverwendung (Re-Use) von Wissen über das zu diagnostizierende technische System und Softwarekomponenten für deren Diagnose in verschiedenen Aspekten der Fehlersuche, wie z.B. Werkstattdiagnose, Fehler-Möglichkeits- und Einfluß-Analysen (FMEA) und Testgenerierung.

Die computergestützte Diagnose stellt also Prüfsoftware, Fehlersuchanleitungen, FME-Analysen, Prozeßstatus-Anzeigen und Fehlermeldungen zu einem technischen System bereit, mit der auf einheitliche oder zumindest weitgehend ähnliche Weise Diagnose in verschiedenen Einsatzszenarien durchgeführt werden kann:

- bei der Bearbeitung von Kundenproblemen in Call-Centern,
- für die Ferndiagnose von einer Servicezentrale aus zu entfernt liegenden Einsatzorten,
- für die Kundenunterstützung via Internet,
- auf Servicekoffern, d.h. Notebooks mit integrierter Ansteuerung zum Gerät, für den Einsatz vor Ort,
- für die Bedienerunterstützung vor Ort.

Grundlage aller im Projekt untersuchter Lösungsansätze ist die sogenannte modellbasierte Diagnose: Ausgang des Verfahrens ist eine Modellbibliothek und eine Strukturbeschreibung des Systems. Die Modellbibliothek enthält Struktur- und/oder Verhaltensbeschreibungen der Komponenten eines Systems. Aus diesen beiden wird automatisch ein Struktur- und Verhaltensmodell des konkreten Systems generiert, das dann (weitgehend) automatisch analysiert werden kann, um die Daten für die Elemente der computergestützten Diagnose und Fehleranalyse zu generieren (vgl. Abbildung 1).

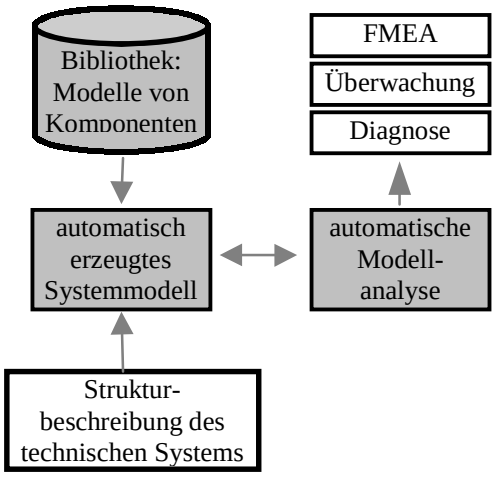


Abbildung 1: Prinzip der modellbasierten Diagnose

2 Anwendung „Mechatronische Kfz-Systeme”

BOSCH ist ein führender Entwickler und Zulieferer von mechatronischen Fahrzeugsubsystemen für den Automobilbau. Beispiele für mechatronische Subsysteme sind Motorsteuerungen für Diesel- und Benzinmotoren, Brems- und ABS-Systeme oder Klimaanlagen. Dieses Arbeitsgebiet ist gekennzeichnet durch eine hohe Variantenvielfalt: Die Subsysteme variieren von Version zu Version und zusätzlich je nach Einbauumgebung in Abhängigkeit vom Fahrzeughersteller, Fahrzeugtypen, Jahrgängen usw. Während des Lebenszyklus eines Subsystems fallen eine Reihe ähnlicher Aufgaben an, zu deren Bearbeitung gemeinsames Wissen benötigt wird: Fehler-Möglichkeits- und Einfluß-Analyse (FMEA), Erzeugung von Diagnoseanleitungen, Werkstattdiagnose und *Onboard*-Diagnose sind Beispiele hierfür. Alle diese Aufgaben erfordern eine Verhaltensanalyse des betrachteten Subsystems. Für die FMEA der Motorsteuerung eines Turboladers ist beispielsweise zu klären, wie sich ein verklemmtes Ventil in der Turboregelung auswirkt. Für das Schreiben eines Prüfplans für die Werkstattdiagnose muß geklärt werden, welche mögliche Ursachen für beobachtete schwarze Abgase in Frage kommen. Wegen des oftmals sicherheitskritischen Charakters der Anwendungen (z.B. *Onboard*-Diagnose des Bremssystems) ist oft zusätzlich Vollständigkeit der Verhaltensanalyse gefordert. In INDIA wurde hier folgender Ansatz verfolgt:

- Gemeinsame Modelle als Grundlage zur Bearbeitung verwandter Aufgaben
- Wiederverwendung von Software, z.B. zur Diagnose und zur Verhaltensanalyse

Im Rahmen von INDIA wurde von den Partnern BOSCH und der TU München die Realisierung dreier Zielsysteme bearbeitet (vgl. Abbildung 2):

- Ein System zur modellbasierten Verhaltensanalyse zur **Unterstützung der FMEA-Erstellung**. Basis der Verhaltensanalyse ist ein kompositionales Systemmodell. Dieses ist abgeleitet aus einer Beschreibung der Systemstruktur und einer wiederverwendbaren Modellbibliothek, die das Verhalten und mögliche Fehlverhalten von Komponenten (Bauteilen) beschreibt.
- Ein modellbasiertes **Autorensystem** zur Entwicklung von Fehlersuchanleitungen für Werkstätten. Basis ist wie für die FMEA ein kompositionales Systemmodell und (dieselbe) Software zur Verhaltensanalyse. Darüber hinaus wird dasselbe Modell zusammen mit zusätzlichem Kostenwissen für die Generierung von Tests (Meßvorschlägen, Prüfanweisungen) genutzt.
- Ein interaktives modellbasiertes **Diagnosesystem als Kern** für die geführte Diagnose und Reparatur in der Werkstatt. Zusätzlich zu den für FMEA und Autorensystem benötigten Methoden zur Verhaltensanalyse und Testgenerierung kommt hier ein Diagnosealgorithmus zum Einsatz, der dem Dialogcharakter der geführten Werkstattdiagnose Rechnung trägt.
- Ein **Online-Diagnosesystem**, das Diagnose basierend auf zeitlichen Sequenzen von Beobachtungen durchführt, die im Steuergerät verfügbar sind. Diese Beobachtungen werden durch eine Komponente für qualitative Werteabstraktion aus den Steuergerätesignalen erzeugt und einem modellbasierten Runtime-System übergeben. Dieses System ist als Prototyp für die Onboard-Diagnose und die Diagnose auf einem Werkstatt-Tester zu sehen.

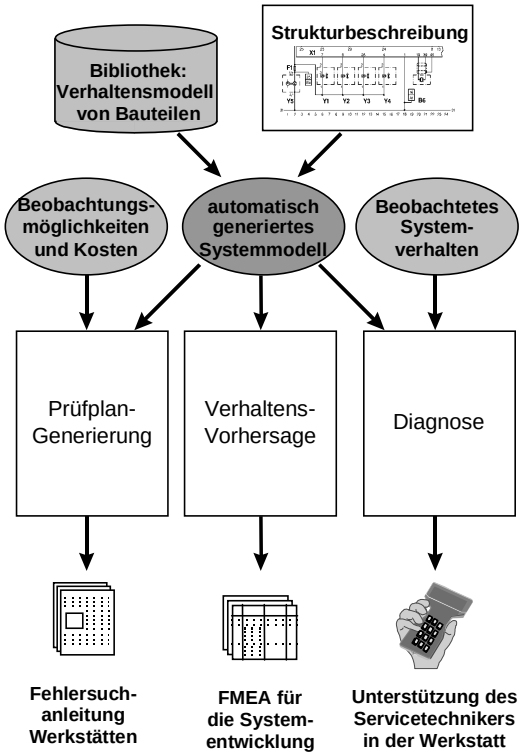


Abbildung 2: Modellbasierte Verfahren in der Kfz-Technik

Folgende Punkte wurden realisiert und sind am Rechner demonstrierbar:

- Modellbibliotheken für elektrische, hydraulische, pneumatische Komponenten und Motor,
- Diagnose aufgrund von Kundenbeanstandungen (Werkstattdiagnose),
- Verhaltensvorhersage (für FMEA),
- Wissensrepräsentation für Fehlersuchanleitungen,
- Modellbasierter Testgenerator,

- Modellbasiertes Online-Diagnosesystem, das auf echten Steuergerätesignalen arbeitet.

Softwarebasis für die Realisierung war ein kommerzielles Tool (RAZ'R der Firma OCC'M Software), das Softwarekomponenten zur Erstellung kompositionaler Modelle, zur Verhaltensanalyse und zur Diagnose liefert.

Künftige Aufgaben umfassen

- die Entwicklung abstrakter Subsystem-Modelle,
- die Integration der Methoden zur Verhaltensanalyse in ein praxistaugliches FMEA-Werkzeug,
- die Umsetzung automatisch generierter Prüfpläne in textuelle Fehlersuchanleitungen,
- weiterer Aufbau der Modellbibliotheken für die Anwendungsfelder,
- Konzeption von Schnittstellen zum Austausch von Modellen.

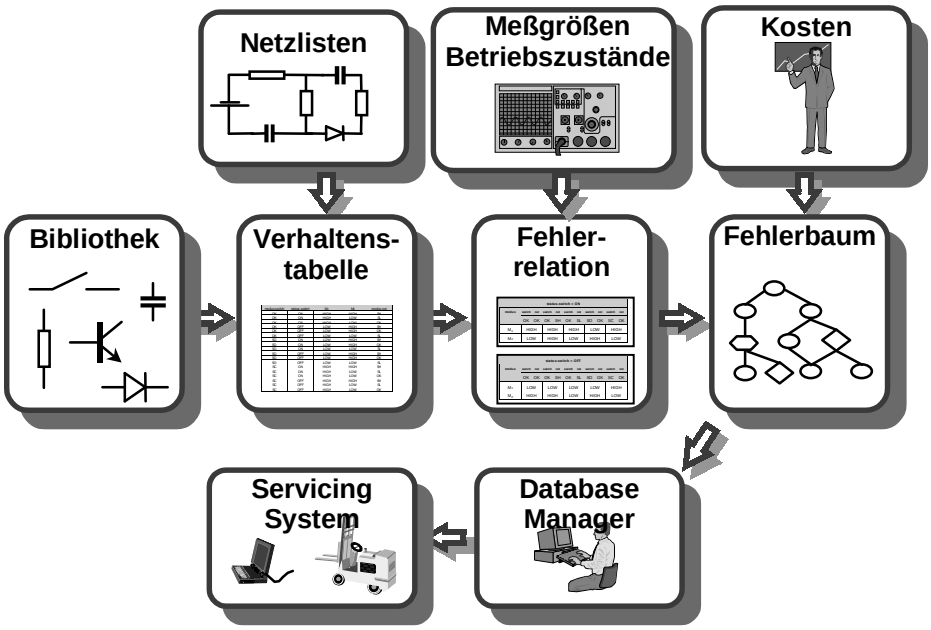


Abbildung 3: Verfahrenskonzept in der Anwendung „Flurförderzeuge“

3 Anwendung „Flurförderzeuge“

Die Firma STILL GmbH in Hamburg stellt Flurförderzeuge, allen voran Gabelstapler her. Aus Integrationssicht ist hier der Einsatz der modellbasierten Diagnosestrategie besonders interessant. Denn bei Projektbeginn hatte STILL bereits ein computergestütztes Diagnosesystem im Einsatz, das auf der Basis von Fehlerbäumen einen Servicetechniker durch die Diagnose führt und ggf. Reparaturhinweise und –anleitungen gibt. Diese Fehlerbäume werden manuell mit einer grafischen Erfassungsumgebung erstellt. Diese manuelle Generierung ist allerdings sehr zeitaufwendig: ein Servicemitarbeiter bei STILL benötigt für die Erstellung eines Fehlerbaums für **einen** Staplertyp ca. 2–3 Personenmonate. Das System war zu der Zeit bereits für drei Staplertypen im Einsatz, weitere Typen wurden und werden nach und nach ergänzt.

Mit der modellbasierten Diagnose sollten in dieser Anwendung die Fehlerbäume für das Diagnosesystem automatisch generiert und hinsichtlich flexibler Kriterien (z.B. Meßkosten) optimiert werden. Ein CAD-ähnliches Werkzeug sollte für die Erstellung von Komponenten- und Strukturmodellen bereitgestellt werden. Aus diesen sollten Fehlerauswirkungen automatisch ermittelt werden. Die daraus generierten Fehlerbäume sollten in das vorhandene Diagnosesystem übernommen werden.

Der erwartete Nutzen für STILL bestand darin,

- daß die Generierungskosten erheblich reduziert werden können,
- daß die erzeugten Fehlerbäume weniger Fehler enthalten, da sie ja aus einer vollständigen Modellbeschreibung erzeugt wurden und,

- daß die Diagnose mit einem kostenoptimierten Fehlerbaum im Einsatz zu besseren und vor allem schnelleren Ergebnissen führt.

Besonders wichtig dabei war, daß das vorhandene Diagnosesystem und der zugehörige Arbeitsablauf unverändert bleiben. Das Gesamtkonzept im Rahmen des INDIA-Projekts umfaßt die folgenden Schritte (vgl. Abbildung 3):

1. Aus einer Bibliothek von Komponenten, die hinsichtlich ihres qualitativen Ein-/Ausgangsverhaltens modelliert sind und einer Netzliste, die diese Komponenten instantiiert und in Relation setzt, entsteht ein Systemmodell (beispielhaft bisher an elektronischen Schaltungen durchgeführt).
2. Aus diesem Systemmodell wird durch vollständige Betrachtung aller möglichen Ein- und Ausgangsbelegungen eine Verhaltenstabelle erzeugt.
3. Aus dieser Verhaltenstabelle werden unter Hinzunahme von Informationen über konkrete Meßgrößen und deren Grenzwerte sowie über Betriebszustände eine Fehlerrelation erzeugt, die alle möglichen Fehlerfälle wiederspiegelt.
4. Unter Hinzunahme von Kosten oder anderen Optimierungsinformationen werden aus den Fehlerrelationen Fehlerbäume erzeugt.
5. Diese Fehlerbäume werden in das Erfassungswerkzeug des Diagnosesystems importiert und können dort manuell erweitert und mit bereits (auch manuell) erzeugten Fehlerbäumen kombiniert werden.
6. Die so entstandenen Fehlerbäume kommen dann im Feld auf einem Servicekoffer zum Einsatz.

Im Projekt konnten die folgenden Ziele realisiert werden:

- Es existiert ein Prototyp für die gesamte Strecke.
- Es wurde ein grafischer Modelleditor mit integrierter Analysekomponente (MAD) entwickelt.
- Die generierten Fehlerstrukturen können in das existierende Diagnosesystem (DiaMon) importiert werden.
- Für bessere Integration des Gesamtsystems in das organisatorische Umfeld wurde die Integration von Standardsystemen wie Kundendienst-Informations- und Steuerungssystemen (KISS) und elektronischem Daten- und Dokumentenmanagement (EDM) untersucht und Konzepte dazu entwickelt.

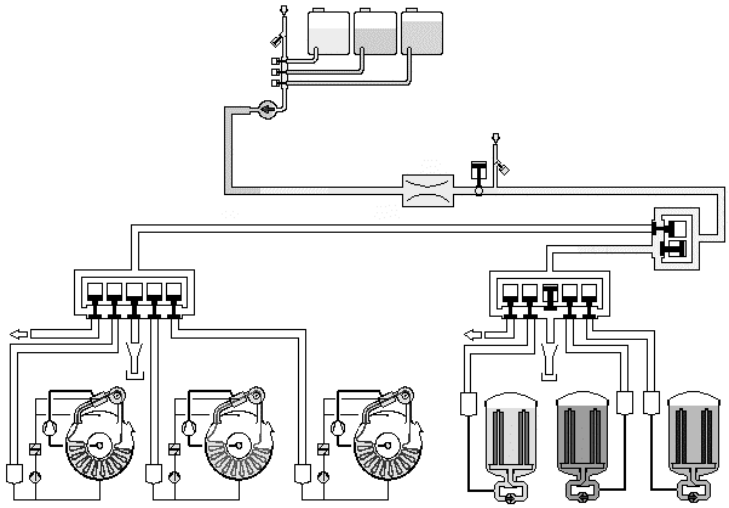


Abbildung 4: Schematische Darstellung eines Chemikaliendistributors

.

4 Anwendung „Anlagenbau“

Die Firma THEN Maschinen- und Anlagenbau in Schwäbisch Hall ist Systemlieferant für Färbereianlagen. Für das Projekt wurden sowohl Färbemaschinen als auch Anlagen zur Versorgung der Färbemaschinen mit Chemikalien (Chemikaliendistributoren) betrachtet (vgl. Abbildung 4).

Dabei handelt es sich jeweils um komplexe technische Anlagen mit elektrischen, hydraulischen und pneumatischen Baugruppen. Gefordert waren in beiden Fällen PC-basierte Systeme zur Überwachung und zur Unterstützung des Anlagenpersonals bei der Diagnose und Beseitigung von Störungen. Der Zugriff auf die Projektierungsdaten und die Daten des aktuellen Auftrags sollte möglich sein (vgl. Abbildung 5).

Durch den Einsatz solcher Systeme erwartete THEN folgende Vorteile:

- Vermeidung von Produktionsausfällen durch rechtzeitiges Erkennen von Störungen
- Reduzierung der Inbetriebnahmezeiten und Wartungsaufwendungen durch automatisierte vollständige Überwachung der Gesamtanlage

Gleichberechtigt neben der Realisierung solcher Diagnosesysteme stand die Suche nach Methoden, mit denen sich der Aufwand zur Entwicklung von Diagnosesystemen reduzieren läßt. Dies ist im Anlagenbau von essentieller Bedeutung, da die meisten Anlagen aufgrund unterschiedlicher Kundenanforderungen Unikate sind, die ein spezielles Diagnosesystem benötigen. THEN war dabei insbesondere daran interessiert, das für die Diagnose gesammelte Anlagenwissen schon in der Anlagenentwurfsphase für FMEA-Untersuchungen zu verwenden.

In Hinsicht auf die genannten Ziele wurden im Projekt die folgenden Ergebnisse erarbeitet:

- Für beide Anlagenklassen wurden prototypische Diagnosesysteme entwickelt und *offline* anhand von Beispieldaten getestet. Die dazu benötigten Betriebsdaten wurden soweit möglich direkt an der Anlage erfaßt, andernfalls durch Simulation gewonnen.
- Für die typischen Komponenten von Rohrleitungssystemen (Ventile, Rohre, Pumpen usw.) wurde eine Bibliothek von Verhaltensmodellen entwickelt. Da solche Systeme im Anlagenbau weit verbreitet sind, können diese Modelle für eine große Klasse von Anwendungen wiederverwendet werden.
- Gerade das Verhalten ablaufgesteuerter Maschinen und Anlagen wird von Fachexperten immer häufiger durch Statecharts modelliert. Mit Hilfe eines neuen Ansatzes können solche Modelle im Hinblick auf die Diagnose ergänzt und in ein Diagnosesystem importiert werden. Dies erleichtert die Wissensakquisition durch Fachexperten wesentlich und senkt dadurch die Entwicklungskosten.
- Gerade in der Verfahrenstechnik wird das Verhalten einer Anlage häufig als Abfolge verschiedener Phasen beschrieben. Solche Beschreibungen können mit Hilfe eines neuen, auf Zustandsgraphen basierenden Verfahrens leichter für die modellbasierte Diagnose genutzt werden.
- Es wurde ein Ansatz zur modellbasierten Erzeugung von Fehlerbäumen entwickelt. Das Erstellen eines Fehlerbaums für eine neue Variante wird dadurch erheblich einfacher.
- Die vorhandenen Fehlermodelle können von einem neuen Modul zur FMEA wiederverwendet werden.

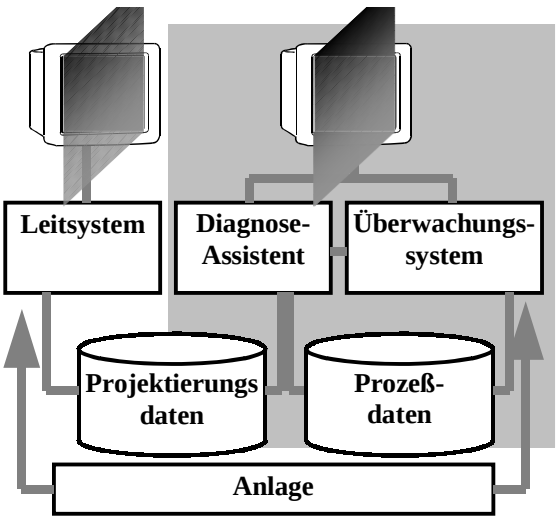


Abbildung 5: Schema eines PC-basierten Überwachungssystems

Entscheidend für die Reduktion der Kosten zur Entwicklung von Diagnosesystemen ist bei all diesen Ansätzen das modellbasierte Vorgehen. Zum einen werden die Modelle einer Anlage für verschiedene Zwecke wiederverwendet

(z. B. FMEA und Diagnose), zum anderen besitzen verschiedene Varianten desselben Anlagentyps sehr viele gemeinsame Komponenten, deren Modelle somit ebenfalls wiederverwendet werden können.

5 Technische Grundlagen der Lösungen

Die Grundlage aller Teilprojekte in INDIA sind modellbasierte Techniken. Alle Problemlösungen basieren auf kompositionalen Modellen. Ein solches Modell kann automatisch oder halbautomatisch aus einer gebietsspezifischen Bibliothek von Komponentenmodellen und einer systemspezifischen Strukturbeschreibung, z.B. CAD-Daten, gewonnen werden. Da die Komponentenmodelle das Verhalten und oft auch das mögliche Fehlverhalten beschreiben, ermöglicht das resultierende Systemmodell Vorhersagen über das Systemverhalten im Normal- und ggf. auch im Fehlerfall.

Ein Vorzug der kompositionalen Modellierung ist die Wiederverwendbarkeit der Komponentenmodelle und das damit verbundenen Potential der Reduktion von Aufwand und Kosten für die Modellierung. Ein weiterer Vorteil ist die leichte (automatische) Anpaßbarkeit von Problemlösungen an Varianten. Dies ist sehr oft allein durch eine lokale Strukturänderung am Systemmodell implementierbar. Weder die Komponentenmodelle der Modellbibliothek noch die Software des Problemlösers sind von solchen Varianten betroffen.

Eine zweite Grundlage aller Teilprojekte in INDIA ist die Verwendung generischer Problemlöser. Ziel ist es, aus einem Systemmodell, zusätzlichem aufgabenspezifischen Wissen und generischer Problemlösungssoftware einen maßgeschneiderten Problemlöser, z.B. für Diagnoseaufgaben, automatisch zu generieren. Die so erzielbare Wiederverwendung von Softwarekomponenten erhöht die Zuverlässigkeit der resultierenden Software und senkt deren Entwicklungskosten.

In den meisten INDIA-Teilprojekten wurden Verfahren der sog. konsistenzbasierten Diagnose angewendet. Eine Diagnose für ein Teilsystem bestehend aus Komponenten C_1, C_2 ... C_n ist hier definiert als eine Zuordnung von je einem Verhaltensmodus (z.B. ok, verstopft, undicht) zu jeder Komponente des Teilsystems, so daß das aus diesen Zuweisungen und dem Modell ableitbare Gesamtverhalten des Systems zusammen mit allen vorliegenden Messungen und Beobachtungen OBS (wie *observations*) logisch konsistent ist. Formal: $Verh(C_i)$ ist eine Diagnose genau dann, wenn die logische Theorie $MODEL \cup \{Verh(C_i)\} \cup OBS$ konsistent ist. Der Diagnoseprozeß wird getrieben durch sog. Konflikte. Ein Konflikt ist ein logischer Widerspruch zwischen den Vorhersagen des Modells unter bestimmten Verhaltensannahmen für die Komponenten und den tatsächlichen Beobachtungen, den Elementen der Menge OBS. Ein solcher Konflikt liefert Information darüber, welche dieser Annahmen revidiert werden müssen, um den Widerspruch zu den Beobachtungen aufzuheben.

Vorteile konsistenzbasierter Diagnose sind

- **Generisches Diagnoseverfahren**: die entsprechende Software ist für viele Diagnoseaufgaben und unterschiedliche Anwendungsbereiche verwendbar.
- **Unabhängigkeit von Fehlermodellen**: der konsistenzbasierte Ansatz erzwingt nicht, neben dem Normalverhalten auch das mögliche Fehlverhalten von Komponenten zu modellieren. Bereits die Abweichung des beobachteten Verhaltens vom modellierten Normalverhalten reicht hier, um Fehler zu hypothetisieren. Dabei garantiert der Algorithmus, daß der tatsächlich vorliegende Fehler in der Menge der erzeugten Hypothesen enthalten ist (Korrektheit des Modells vorausgesetzt).
- **Mehrfachfehler-Diagnosen**: das Standardverfahren zur konsistenzbasierten Diagnose ermöglicht die Diagnose unabhängig auftretender Mehrfachfehler. Das auftretende kombinatorische Problem wird dabei wirkungsvoll durch Ausnutzung des lokalen Charakters von Komponentenverhalten gelöst: Lokalität bewirkt hier, daß Vorhersagen über die Auswirkungen des Verhaltens einer Komponente in kombinatorisch vielen Kontexten, d.h. unter vielen verschiedenen Annahmen über das Verhalten der übrigen Komponenten simultan gültig sind.

Eine weitere Grundlage für viele Arbeiten in INDIA ist die Verwendung qualitativer Modelle. Im Bereich wissenschaftlichen und ingenieurmäßigen Denkens ist ein konzeptuelles, qualitatives Verständnis von Systemzusammenhängen grundlegende Voraussetzung für das Problemlösen. Qualitative Modellierung zielt darauf, ein natürliches oder technisches System auf einer konzeptuellen Ebene zu beschreiben. Damit ist gemeint, daß nicht nur ein mathematisches Modell repräsentiert wird, sondern auch eine begriffliche Ebene der physikalischen Phänomene (z.B. in Form von Komponenten und Prozessen), auf der entsprechende Schlußfolgerungsprozesse ablaufen können. Dies ist gerade bei Diagnosefragestellungen von erheblicher Bedeutung, da abweichendes Verhalten zwar in der Modifikation des mathematischen Modells seinen Ausdruck findet, aber natürlich auf der physikalischen Ebene seine Ursachen und Bedeutung hat. Um diese konzeptuelle Sicht und das Schließen mit partieller Information mit effizienten Verfahren, arbeitet qualitative Modellierung mit einem mathematischen Verhaltensmodell auf geeigneter Abstraktionsebene, die möglichst nur die zur Problemlösung nötigen Begriffe und Unterscheidungen einführt. Ein Beispiel für einen solchen Abstraktionsschritt ist die Vorzeichenabstraktion: Eine reellwertige Variable x wird dabei nur durch ihr Vorzeichen repräsentiert:

[x] := sign (x). Eine andere nützliche Abstraktion ist das Vorzeichen der zeitlichen Ableitung einer Variable: δx:= [dx/dt]. Dies drückt aus, ob eine Variable steigt, fällt oder momentan konstant ist. Die Unterscheidungen dieser drei qualitativen Fälle reicht für modellbasiertes Problemlösen, etwa die Unterscheidung von Verhaltensalternativen (Diagnosen), oft völlig aus. Eine dritte nützliche Abstraktion ist das Vorzeichen der Abweichung eines aktuellen Wertes x_{act} von seinem erwarteten Referenzwert x_{ref}: $[\Delta x] := x_{act} - x_{ref}$. Umgangssprachlich drückt diese Abstraktion aus, ob ein Wert normal, zu hoch, oder zu niedrig ist — auch dieser Begriff der Abweichung ist ein Konzept, das Ingenieure zur qualitativen Analyse technischer Systeme gerne verwenden. Qualitatives Schließen darf keinesfalls gleichgesetzt werden mit unscharfem oder heuristischem Schließen. Mit qualitativen Abweichungen läßt sich vielmehr für die Modellanalyse exakt und kalkülmäßig rechnen: zum Beispiel gilt für die qualitativen Abweichungen

$$\Delta (x * y) = x_{act} * \Delta y + y_{act} * \Delta x - \Delta x * \Delta y$$

Der Referenzwert x_{ref} bzw. y_{ref} kann dabei bemerkenswerterweise unspezifiziert bleiben, und auf einer qualitativen Ebene genügt die Kenntnis der Vorzeichen der aktuellen Werte, um die Folgen oder Ursachen von qualitativ charakterisierten Abweichungen gewisser Systemgrößen abzuleiten. Die Verwendung qualitativer Modelle erhöht die Mächtigkeit modellbasierter Methoden: Die damit erzielten Ergebnisse gelten für ganze Klassen von Systemen, unabhängig von den numerischen Ausprägungen ihrer Parameter. Qualitative Verhaltensmodelle decken unabhängig von spezifischen numerischen Ausprägungen ganze Klassen von Komponentenfehlern ab, z.B. eine verzögerte Ventilöffnung oder eine verstopfte Rohrleitung. Dies kann z.B. wirkungsvoll für die FMEA verwendet werden. Qualitative Modellierung führt zu kompakten Modellbibliotheken und einer natürlichen, nachvollziehbaren Repräsentation des Komponentenverhaltens. Alle Variablen haben endliche, oft sehr kleine Wertebereiche. Die Zahl der möglichen Systemzustände ist damit endlich. Dies ermöglicht effiziente und vollständige Analysen der Modelle.

Ein Anwendungsbeispiel hierfür ist die modellbasierte Testgenerierung. Die Aufgabe ist hier, Tests automatisch zu generieren, mit denen entschieden werden kann, welche von zwei Fehlerhypothesen zutrifft. Dazu wird das qualitative Modell unter beiden Annahmen analysiert. Wegen der endlichen Wertebereiche der Variablen liefert diese Analyse zwei endliche Mengen von Zuständen Z_1 und Z_2. Die Schnittmenge $Z_1 \cap Z_2$ enthält genau die Testmuster, für die das Verhalten des Systems unter beiden Hypothesen ununterscheidbar ist. Das Komplement $Z_1 \cup Z_2 - Z_1 \cap Z_2$ enthält dagegen genau die Testmuster, für die sich das System unterschiedlich verhalten kann. Unterschiede in den Zustandsmengen können so zum Ableiten geeigneter Tests genutzt werden.

Eine weitere Leitlinie in INDIA ist die Wiederverwendung und der Austausch von Modellen. Ein Systemmodell kann während des gesamten Lebenszyklus eines Systems verwendet werden, um unterschiedliche Aufgaben zu bearbeiten. Diagnose, FMEA und Testgenerierung sind Beispiele hierfür. Modelle helfen so, die verschiedenen Arbeitsprozesse im Produktlebenszyklus zu integrieren und effizienter und redundanzfreier zu gestalten.

6 Lehren und Perspektiven

Fachlich haben die Arbeiten in INDIA bestätigt, daß die grundlegenden Formalismen und Algorithmen auf dem Gebiet der modellbasierten Diagnose ausreichen, um typische Aufgaben in der industriellen Diagnose erfolgreich zu bearbeiten. Die Herausforderung besteht in INDIA vor allem darin, die Brücke von der Prinzipienlösung zur praktischen Lösung zu schlagen. Hierbei rücken pragmatische Randbedingungen industrieller Diagnoseprozesse in den Vordergrund, wie zum Beispiel

- (Nicht-)Verfügbarkeit und Charakter von Beobachtungen,
- Aktionen und deren Kosten,
- aktuelle Arbeitsprozesse.

Organisatorisch gesehen hat sich die Struktur von INDIA als Verbundprojekt bewährt: 3 Säulen mit je drei Partnern aus Forschung, professioneller Softwareentwicklung und industrieller Anwendung. Vertikal ergibt sich so ein erfolgreicher Technologietransfer von der Forschung in die Anwendung. Horizontal stimuliert diese Projektstruktur einen fruchtbaren Austausch, naturgemäß am intensivsten zwischen den drei beteiligten Forschungseinrichtungen.

Die in INDIA erzielten Resultate machen das Potential modellbasierter Lösungen deutlich:

- Es wurden erfolgreich Lösungen für relevante Problemstellungen aus der industriellen Praxis entwickelt und erprobt.
- Wiederverwendung von Wissen (Modellen), Information und Lösungen und deren Austausch zwischen Arbeitsprozessen wurden erfolgreich demonstriert.

- Es wurden Wege entwickelt, auf denen modellbasierte Verfahren in den bestehenden Arbeitsprozeß und deren Softwarelandschaft integriert werden können, und damit arbeitsorganisatorische Änderungen als Hemmnisse für die Umsetzung zu reduzieren.

In den INDIA-Teilprojekten wird somit deutlich, daß Modelle ein erhebliches Potential zur Integration industrieller Arbeitsprozesse haben. Ihnen wächst damit eine wichtige Rolle im Wissensmanagement einer Firma zu. Modelle könnten in Zukunft als Dreh- und Angelpunkt firmeninterner Arbeitsprozesse dienen — Modelle z.B. als Service im firmeninternen Intranet.

Modellbasierte Verfahren liefern eine Basis für zukünftige Serviceleistungen, z.B. in der verteilten, kooperativen Ferndiagnose von Fahrzeugflotten oder im Anlagenbau. Für die weitere Integration von Arbeitsprozessen wird der Import vorhandener Modelle, z.B. aus der Anlagenprojektierung oder aus numerischen Modellen eine wichtige Rolle spielen. Hier müssen Austauschformate, z.B. auf Basis von STEP oder SGML/XML entwickelt und genutzt werden. Ein verwandtes Thema ist die Entwicklung integrierter Lösungen, d.h. die modellbasierte Kopplung von Prozeßketten. Beispiele sind Diagnose/Leittechnik, Entwurf/FMEA/Diagnose oder Diagnose/Rekonfigurierung.

Teil I

Intelligente Diagnose bei mechatronischen Systemen

Die erste „Anwendungssäule“ im INDIA Projekt bestand aus:

- der Robert Bosch GmbH als Entwickler und Anbieter von Anleitungen zur Fehlersuche in ihren Kraftfahrzeugsystemen und von kompletten Diagnosesystemen für Werkstätten
- dem Institut für Informatik der Universität München als Forschungseinrichtung mit dem Schwerpunkt Modellbasierte Diagnose und Qualitatives Schließen.

Ziel der gemeinsamen Arbeit war es daher, modellbasierte Methoden für die sog. Offboard-Diagnose am Kraftfahrzeug so weit voranzutreiben, daß ein praktischer Einsatz bei den Aufgaben in der Werkstatt ohne zusätzliche Grundlagenforschung möglich wird.

Das einleitende Kapitel I-1 dieses Teil I motiviert die Verwendung intelligenter, insbesondere modellbasierter Verfahren bei der Rechnerunterstützung für Aufgaben der Qualitätsanalyse, der Fehlersuche in der Kraftfahrzeugwerkstatt und den einhergehenden Problemen der Datenerfassung und Signalauswertung. Kapitel I-2 geht ausführlicher auf die Erarbeitung der Grundlagen und die Realisierungsschritte für modellbasierte Werkzeuge im INDIA Projekt sowohl zur Fehleranalyse (hier der sog. Fehler-Möglichkeits- und Einfluß-Analyse, kurz FMEA) als auch für die Werkstattdiagnose an Teilsystemen im Kraftfahrzeug ein. Im Kapitel I-3 werden die Anforderungen analysiert und spezifiziert, die ein Redaktionssystem zur Erzeugung von Fehlersuchanleitungen im Kraftfahrzeug erfüllen sollte, damit modellbasierte Methoden optimal zur Diagnose eingesetzt werden können. In den folgenden Kapiteln werden Anforderungen und Resultate (Kapitel I-4) und theoretische Grundlagen und praktische Aspekte (Kapitel I-5) aus einer Fallstudie für modellbasierte Werkstatt- Diagnose des Hydraulikkreises eines Anti-Blockier-Systems dargestellt. Im Kapitel I-6 wird ein Ansatz für die qualitative prozeßorientierte Modellierung und seine Anwendung für Diagnose und Überwachung diskutiert. Im Kapitel I-7 wird eine graphen-orientierte Repräsentation für dynamische Systeme vorgestellt und durch eine Anwendung aus dem Bereich Hydro-ökologischer Systeme illustriert. Kapitel I-8 beschreibt automatische Transformationen qualitativer Modelle.

1 Intelligente Diagnose bei mechatronischen Systemen im Kraftfahrzeug

Reinhard Weber - Robert Bosch GmbH

1.1 Motivation für den Einsatz von intelligenter Rechnerunterstützung

Sogenannte *mechatronische Systeme* bestehen aus mechanischen, hydraulischen, thermischen und/oder elektronischen Komponenten mit einer speziellen Steuerungssoftware; sie erfordern wegen ihrer höheren Komplexität bei der Analyse und Synthese besondere Techniken.

In Kraftfahrzeugen werden zunehmend solche komplexen Subsysteme eingesetzt, wobei insbesondere die Zahl digitaler Systeme bis hin zu Bussystemen wächst. Beispiele dafür sind elektronische Dieselregelung (EDC) und Anti-Blockier-System (ABS). Sicherheits- und Umweltgesichtspunkte stellen hohe Anforderungen an das korrekte Funktionieren, die sichere Fehlererkennung und gezielte Fehlerbehebung. Dies erhöht natürlich *die Komplexität von allen Entwicklungsschritten*, etwa von Qualitätsanalysen während der Entwurfsphase und Fertigungsvorbereitung (z.B. bei der FMEA), der Fehlererkennung während des Betriebs, der Fehlerlokalisierung und Fehlerbehebung in der Werkstatt (Werkstattdiagnose) sowie der Entwicklung von Diagnose- und Reparaturanleitungen und den notwendigen Prüfgeräten. Um die genannten Aufgaben trotzdem mit ökonomisch vertretbarem Aufwand in der notwendigen Vollständigkeit durchführen zu können, ist Unterstützung durch den Rechner erforderlich.

Im Lebenszyklus von Kfz-Systemen muß die Robert Bosch GmbH (kurz Bosch) das weite Feld vom Entwurf über die Fertigung bis hin zum Service und der Entsorgung abdecken und begleiten. Daher wird in vielen Entwicklungsbereichen an der Vision einer *durchgängigen Rechnerunterstützung* für alle Betriebsprozesse gearbeitet. Die Realisierung solcher Ideen kann eigentlich nur durch den Einsatz von wiederverwendbaren Modellen der Produkte (virtuellen Produkten) auf verschiedenen, angepaßten Abstraktionsebenen erfolgen. Für die Bearbeitung der unterschiedlichen Modellaspekte in den einzelnen Prozeßschritten sind jeweils spezielle Strategien und Algorithmen (auch: Inferenzmechanismen) zuständig. Jede Rechnerunterstützung benötigt eine für den Computer geeignete Darstellung der Objekte (Produkte) und Methoden (Prozesse). Um für die Kfz-Systeme geeignete Repräsentationen zu finden, muß man sich einerseits die Probleme für die Entwicklung, Fertigung und den Service vor Augen halten, andererseits wichtige Systemeigenschaften analysieren.

Für einen Kfz-Zulieferer wie Bosch besteht eine der größten *Herausforderungen* in der sog. Variantenproblematik, d.h. in der Vielzahl verschiedener Fahrzeughersteller, Fahrzeugtypen, Systemen und Systemvarianten den Überblick zu behalten, also gemeinsame Eigenschaften zu erkennen, so daß Synergieeffekte ausgenutzt werden können, aber auch die Verwaltung beherrscht wird. Hinzu kommt, daß Kfz-Systeme dynamisch, meist geregelt sind und Phänomene aus vielen technischen Bereichen (elektrisch, elektronisch, hydraulisch, pneumatisch, mechanisch, thermisch u.a.) darin auftreten können. Kfz-Systeme müssen aber auch in verschiedenen Betriebszuständen funktionieren und sogar noch im Fehlerfall einen (eingeschränkten) Fahrbetrieb ermöglichen. Weiterhin gibt es im Fahrzeug nur eine begrenzte Zahl von Sensoren oder vergleichbare Informationen, die für eine Überwachung herangezogen werden können, und auch in der Werkstatt sind nicht alle physikalischen Größen meßbar (hauptsächlich nur elektrische Größen).

Auf der anderen Seite besitzen Kfz-Systeme *charakteristische Eigenschaften*, die eine übersichtliche Beschreibung erleichtern können:

- In der Regel ist eine natürliche, hierarchische Zerlegung jeweils in wenige Teilsysteme, Aggregate und Komponenten möglich.
- Das Verhalten von Komponenten kann durch das Fließen und Umwandeln von Stoff, Energie und/oder Signalen beschrieben werden, also mathematisch als Relation zwischen lokalen Variablen und Parametern.
- Das Verhalten eines Systems ergibt sich aus dem Verhalten seiner Bestandteile und deren Verbindungen.
- Meist können Fehlfunktionen im Kraftfahrzeug (bzw. Subsystem) auf Komponentenfehler zurückgeführt werden.

Neben einem methodischen rechnergestützten Vorgehen bei den genannten Aufgaben aus dem Produkt-Lebenszyklus erscheinen *wissensbasierte Techniken* besonders geeignet, weil sie von vornherein eine Trennung von Wissen über die (Analyse-, Design-, Diagnose-) Aufgabe und die Struktur, Funktion und das Verhalten des

Systems vorsehen. Insbesondere geben *modellbasierte Ansätze* eine Antwort auf die Forderung nach Wiederverwendbarkeit der Modelle für ähnliche Aufgaben (Variantenproblem) und Automatisierung der Inferenzen für die Analyse und Diagnose (Verhaltensmodelle von Komponenten). Ein wichtiges Problem ist dabei der Zwang zu einer gemeinsamen Wissensbasis oder zumindest einem weitgehend automatisierten Wissenstransfer. Um die Effizienz und Wiederverwendung der Modelle noch deutlicher zu erhöhen, wurden im Projekt besonders qualitative Modellierungsansätze untersucht, bei denen von vielen speziellen Ausprägungen abstrahiert werden kann.

1.2 Industrielle Ziele im INDIA Projekt: FMEA und Diagnose bei Kfz-Systemen

Um mit der knappen Personalkapazität einen essentiellen Beitrag für die praktische Einführung von intelligenten Methoden auf das oben genannte langfristige Ziel hin leisten zu können, hat Bosch zwei Einsatzszenarien herausgegriffen und im Projekt so weit vertieft, daß sich eine industrielle Umsetzung anschließen kann. Vor allem um den Einsatz modellbasierter Techniken in verschiedenen mit Diagnose befaßten Unternehmensbereichen bei Bosch voranzutreiben, wurden im INDIA Projekt prototypische Lösungen geplant und entwickelt, die in diesem Paragraphen kurz skizziert und in den anschließenden Beiträgen ausführlicher dargestellt werden.

1.2.1 Anwendung FMEA

Zum Nachweis der präventiven Qualitätssicherung bei Komponenten und Systemen für das Kraftfahrzeug wird von den Bosch-Kunden, also nahezu allen Automobilherstellern, vor allem eine FMEA (*Fehler-Möglichkeiten und Einfluß-Analyse*) verlangt. Sie zieht alle denkbaren Fehlerarten und -ursachen während der Entwurfs-, Konstruktions- und Fertigungsphase in Betracht und versucht, ihnen Fehlerauswirkungen auf Komponenten-, Subsystem- und Fahrzeugebene zuzuordnen. So hat der Kunde auf Grund der Schwere der Auswirkungen, der Auftretens- und Entdeckungswahrscheinlichkeit des Fehlers Maßzahlen für die erreichte Qualität.

Die dazu notwendigen Informationen und Schlußfolgerungen werden auf FMEA-Teamsitzungen im „brainstorming“ Verfahren, aber nach methodischen Ansätzen (dem sog. FMEA-Formular, vgl. Abbildung 1-1) erarbeitet, gesammelt und aufgeschrieben. Meist wird noch eine Phase der Wissenserhebung vorgeschaltet, in der der Moderator bereits soviel wie möglich erledigt. Da ein solches Team sich aus Fachleuten von der Entwicklung bis hin zur Fertigungsvorbereitung zusammensetzt, ist das Verfahren extrem personalintensiv und es wird sehr viel wertvolle *Ingenieurkapazität* gebunden. Dabei werden die Ergebnisse aber noch zu wenig in späteren Entwicklungsphasen wie etwa bei der Diagnose eingesetzt.

Natürlich bringt eine Rechnerunterstützung hier Verbesserung: So sind zur Zeit schon Programme im Einsatz, die eine strukturierte *Textverarbeitung zur Dokumentation* durchzuführen erlauben. Aber auch die Anbindung an Textdatenbanken mit der Möglichkeit einer gezielten Suche nach abgespeicherten ähnlichen Analysen wurde untersucht. Allerdings konnte das dabei auftretende Matching-Problem mit Sprachelementen, die nicht standardisiert, sondern der natürlichen, aber technischen Sprache entnommen sind, nicht befriedigend gelöst werden.

Sowohl beim Wiederfinden und Wiederverwenden von abgespeichertem Wissen als auch bei der zentralen Aufgabe der FMEA, nämlich Fehler durch ein System zu propagieren, um ihre Auswirkungen zu ermitteln, liefern modellbasierte Techniken eine erfolgversprechende Alternative. Bosch untersuchte daher, wie man zur Vorbereitung durch den FMEA-Moderator, aber auch zur Verwendung während der FMEA-Sitzungen modellbasierte Techniken einsetzen kann. Offenbar wird dazu ein Werkzeug benötigt, das

- die FMEA in der bisherigen Weise als strukturierender Editor unterstützt,
- den Zugriff auf die Ergebnisse von ähnlichen Komponenten/Systemen zuläßt,
- eine *automatische Verhaltensanalyse* vornimmt durch Verfolgen der Fehlerarten zu ihren Auswirkungen.

Erreicht werden sollte dies möglichst durch Integration der modellbasierten Verhaltensanalyse in ein vorhandenes FMEA-Programm (Benutzeroberfläche, objektorientierte Datenstrukturen, Funktionalität). Darüber hinaus benötigt ein solches Werkzeug Schnittstellen zu den im Unternehmen bereits vorhandenen Informationen, wobei Standardbeschreibungen (STEP bzw. SGML/XML) der Vorzug gegenüber speziellen Lösungen gegeben wird.

In INDIA wurde ein erster *Prototyp eines solchen Assistenzsystems* spezifiziert und entwickelt, das schon wesentliche der genannten Merkmale besitzt. Es wurde aber auch ein interaktives Werkzeug entwickelt, mit dem die Modelle erstellt, verwaltet, verändert und wiederverwendet werden können. Soweit die Modellbildung nicht bereits während der Komponentenentwicklung erfolgt, bietet gerade dieses FMEA-Szenario die Gelegenheit, eine Modellbibliothek aufzubauen. Über die Untersuchungen im Rahmen des INDIA-Projekts wird in Kapitel

„Modellbasierte Verhaltensanalyse für die FMEA“ berichtet. Sie machen deutlich, daß nicht nur Komponentenmodelle aufgebaut, sondern auch klare Vorteile bei der Qualität der Analysen erreicht werden. Daher wird Bosch längerfristig die Einführung des modellbasierten Ansatzes bei der FMEA weiter vorantreiben.

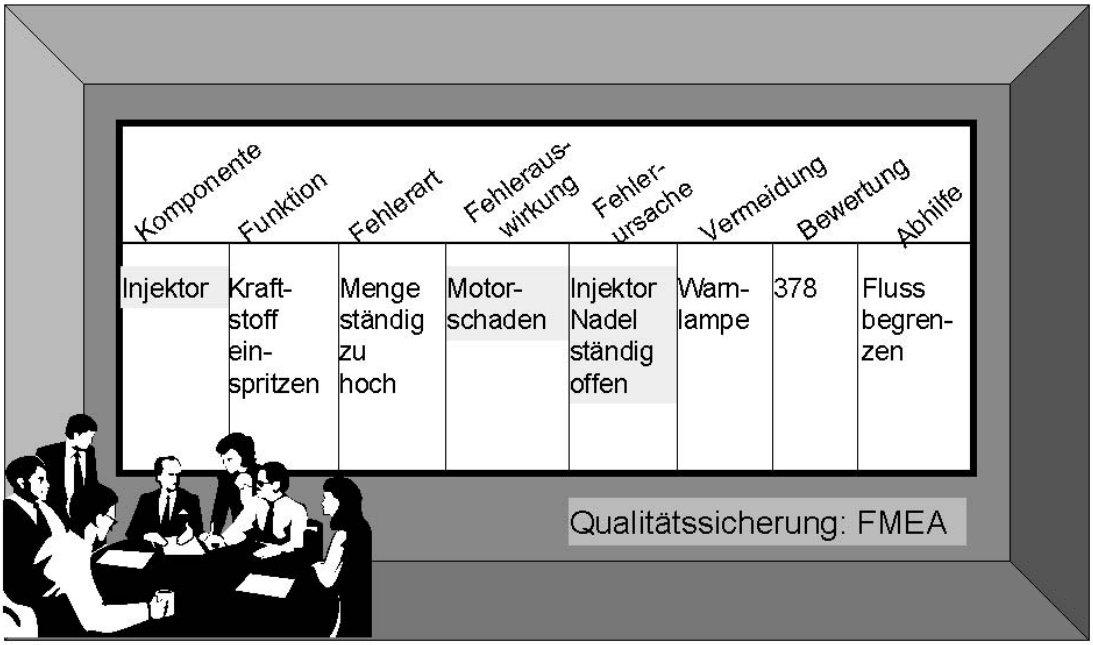


Abbildung 1-1: Typisches FMEA-Szenario mit dem sog. FMEA Formular

1.2.2 Anwendung Fehlersuchanleitung

Die meisten heutigen Diagnosesysteme für Kfz-Werkstätten arbeiten intern mit Entscheidungsbäumen und liefern Anleitungen zur Diagnose und Reparatur auf einem PC-Bildschirm. Auch das von Bosch vertriebene System ESI (Elektronische Service Information), vom Servicebereich für das weitverzweigte Netz von Bosch-Diensten geschaffen und eingeführt, arbeitet nach diesem Prinzip. Solche Fehlersuchanleitungen werden heute auf Grund von Erfahrungen am technischen Objekt in Schulungswerkstätten und den Bosch-Diensten, sowie einem sehr gründlichen Funktionsverständnis der Teilsysteme aufgebaut, wobei entweder von typischen Kundenbeanstandungen am Auto oder aber von dem aus dem Steuergerät ausgelesenen Fehlercode ausgegangen wird. Da dieses Verfahren erst nach Anlauf einer neuen Serie greift, wird es immer schwieriger, den heutigen raschen Innovationszyklen zu folgen.

Übrigens hat der bisher erreichte (und auch in den nächsten Jahren erreichbare) Stand der sog. Onboard-Diagnose (OBD), d.h. der Fehlerdiagnose durch das Steuergerät selbst, die Situation für die Werkstatt in keiner Weise entschärft. Hier sind zwei gegenläufige Entwicklungen zu beobachten: durch die OBD können Fehlerpfade diagnostiziert werden, dafür nimmt jedoch die Komplexität der Systeme zu. Nach Beobachtungen der Prüftechnik werden durch die OBD selbst die elektrischen Fehler nur grob lokalisiert, zur genaueren Lokalisierung und Identifikation müssen in der Regel zusätzliche Messungen durchgeführt werden.

Wenn auch derartige Entscheidungsbäume für die Fehlersuche immer nur eine Zwischenstufe bei der Diagnose und Reparatur bilden, da sie nur auf bestimmte Fehlerklassen zugeschnitten sind und daher einen statischen Charakter haben, bedeutet ihre automatische Generierung doch einen Zeitgewinn. Daher ist sie auch Anliegen von anderen wissensbasierten Methoden, wie regelbasierten Techniken, ohne daß diese dadurch den entscheidenden Durchbruch erzielt haben. Auch bei Bosch wurden intensive Versuche mit assoziativen Methoden und Werkzeugen für die Diagnose gemacht. Sie scheiterten an der erwähnten Variantenproblematik: Bei der stets notwendigen Systemidentifikation in der Werkstatt müssen derart viele Varianten durch unterschiedliche Automarken und -typen, sowie Systemkonfigurationen unterschieden werden, daß sowohl der Ersteller der Wissensbasen, wie auch der Benutzer des Diagnosesystems sich in der Vielfalt verliert.

Prinzipiell können diese Fehlersuch- und Reparaturanleitungen automatisch aus den Komponenten- und Systemmodellen für vorgegebene Klassen von Fehlerauswirkungen erzeugt werden. Dies ist sicherlich nur ein Zwischenschritt auf dem Weg zu einer vollständigen modellbasierten Diagnose. Er erscheint aber aus wirtschaftlichen und organisatorischen Gründen sinnvoll und liefert darüber hinaus eine Validierung der Modelle und zusätzliche Evaluierung der modellbasierten Techniken. Natürlich ist dabei der Aspekt des Wissens- und Modelltransfers von der Entwicklung und Fertigung über die Qualitätssicherung zum Service von entscheidender Bedeutung. Allerdings, auch die modellbasierte Generierung von Fehlersuchanleitungen sollte nicht der Prüfstein für derartige Techniken sein, sie könnte aber durchaus erhebliche Vorteile für den Servicebereich bringen (sie liefert in jedem Fall eine Validierung der Systemmodelle).

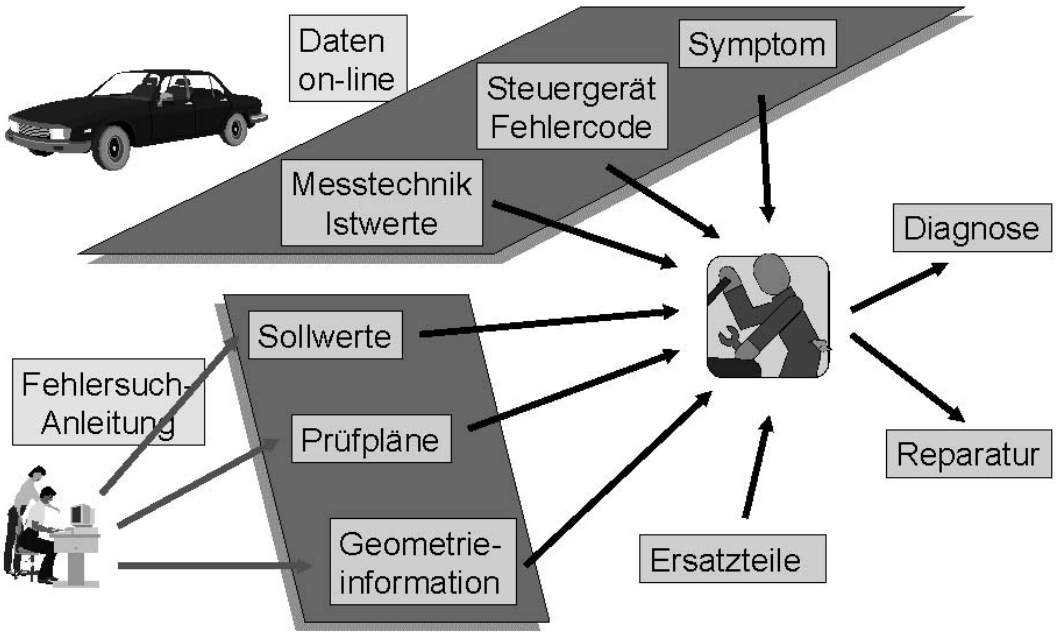


Abbildung 1-2: Diagnose in der Werkstatt mit Fehlersuchanleitung und Online-Datenerfassung

In INDIA war das Ziel von Bosch, einen ersten Prototyp eines solchen Werkzeuges zur automatischen Generierung von Entscheidungsbäumen zu entwickeln und in das vorhandene konventionelle Autorensystem zu integrieren. Bei der Analyse des heutigen Redaktionssystems für Fehlersuchanleitungen wurde schnell klar, daß sein Aufbau nach Bildschirmseiten für die Darstellung von modellbasierten Techniken nicht geeignet ist. Daher bestand die erste Aufgabe bei der Spezifikation des neuen Prototypen darin, geeignete Datenstrukturen zu definieren und zu implementieren. Es entstand das Konzept des elementaren Textbausteins mit Parametern (vgl. die Ausführungen im Kapitel „Autorensystem für Anleitungen zur Fehlersuche und Reparatur im Kfz“).

Der Prototyp wurde in aufeinander aufbauenden Stufen spezifiziert und teilweise implementiert:

- Das Basissystem enthält die für modellbasierte Ansätze geeignete Datenstruktur, den Aufbau einer Fehlersuchanleitung aus parametrierten Textbausteinen mit kontextabhängigem Angebot, kohärente Verknüpfungen mit Schaltplänen, eine standardisierte XML-Schnittstelle.
- Die weiteren Stufen umfassen die modellbasierte Generierung der Kopplungsmatrix, die Fehlersymptome und Funktionsgruppen verbindet, sowie der Prüfpläne, dann die Erfassung von Fehlercodes und zusätzlichen Meßwerten aus Prüfgeräten in der Werkstatt.

Ein Werkzeug zur modellbasierten Diagnose und Reparatur muß sich durch passende Schnittstellen zu vorhandenen Informationssystemen möglichst nahtlos in bestehende Umgebungen in der Kfz-Werkstatt einfügen lassen. In diesem Zusammenhang wurde von Bosch und TU München letztlich sogar an einem Versuchsfahrzeug implementiert und untersucht, wie Daten aus dem Fahrzeug (hier kommen vor allem Daten zur Fahrzeug- und Subsystemidentifikation, aber auch abgespeicherte „Momentaufnahmen“ zur Fehlerzeit bis hin zu online übermittelten Signalverläufe in Frage) erfaßt, aufbereitet und für die Diagnose ausgewertet werden können (vgl. Abbildung 1-2). Über die entsprechenden Versuche und Ergebnisse wird im Kapitel „Modellbasierte Unterstützung der Werkstattdiagnose“ berichtet.

2 Modellbasierte Werkzeuge für Diagnose und Fehleranalyse von Fahrzeugsubsystemen

Peter Struss, Ulrich Heller, Andreas Malik und Martin Sachenbacher – Technische Universität München

2.1 Einleitung

2.1.1 Die Zielsetzung

Modellbasierte Systeme versprechen automatische oder unterstützende Computer-Problemlösungen in Gebieten, in denen diese Lösungen wesentlich auf prinzipielles Wissen über das Verhalten und möglicherweise das Fehlverhalten komplexer technischer oder natürlicher Systeme zurückgreifen müssen. Diagnose und Fehleranalyse treten als Aufgabenstellungen in unterschiedlichen Ausprägungen während des gesamten Lebenszyklus von Produkten, in unserem Anwendungsbereich Kraftfahrzeug-Subsysteme, auf: Vom Entwurf eines Systems, das auch im Fehlerfall Risiken für Fahrzeuginsassen und Umgebung gering hält, über die Fehlererkennung in Onboard-Systemen und die Bereitstellung von Diagnose- und Reparaturanleitungen für Werkstätten bis zu Prüf- und Diagnoseabläufen auf Werkstatt-Testern. Beispiel für solche Fahrzeug-Subsysteme, die jeweils von einem elektronischen Steuergerät („Electronic Control Unit“, ECU) überwacht und geregelt werden, sind das Anti-Blockier-Systems (ABS), die Motorregelung oder das Airbag-System. Im INDIA-Projekt wurde in Kooperation zwischen der Robert Bosch GmbH und der Gruppe „Modelbased Systems and Qualitative Reasoning“ (MQM) der Technischen Universität München an Computerunterstützung für drei verschiedene Aufgaben gearbeitet, deren praktischer Hintergrund im folgenden nur kurz charakterisiert werden soll (eine genauere Analyse der Anforderungen ist in den darauffolgenden Kapiteln enthalten):

- *Fehler-Möglichkeits- und Einfluß-Analyse (FMEA)*

  FMEA (engl. „Failure Modes and Effects Analysis“) wird gemeinsam von Entwicklungsingenieuren und Diagnosespezialisten in mehreren Sitzungen während der Entwurfsphase eines Subsystems durchgeführt. Ihr Ziel ist, basierend auf dem jeweiligen Stand des Entwurfs, die möglichen Auswirkungen von Komponentenausfällen und -fehlverhalten auf das Verhalten des Fahrzeugs zu analysieren. Schwerpunktmäßig geht es um die Einschätzung der „Kritikalität“, d.h. die Frage, wie schwerwiegend die resultierenden Funktionsstörungen gemäß subjektiver oder objektiver Kriterien sind (z.B. Beeinträchtigung des Fahrkomforts, negative Umwelteinflüsse, Unfallrisiko). Außerdem werden die Auftretenswahrscheinlichkeit der Fehler sowie ihre Entdeckbarkeit abgeschätzt. Auf dieser Grundlage können ggf. Änderungen am Entwurf vorgeschlagen werden. FMEA wird in zunehmendem Maße in verschiedenen Produktionszweigen zu einer gesetzlichen oder auch Kundenanforderung. Da im Fahrzeugbereich Sicherheit und Umweltaspekte betroffen sind, muss die Analyse so vollständig wie möglich sein, d.h. nicht nur sämtliche denkbaren Komponentenfehler umfassen, sondern auch alle ihre möglichen Auswirkungen unter verschiedenen Randbedingungen (also z.B. Fahrsituationen und Wechselwirkungen mit anderen Subsystemen).

- *Werkstattdiagnose*

  Ausgangspunkt für die Fehlersuche und -behebung in der Werkstatt ist eine gewisse Menge von Symptomen eines Fehlverhaltens, die entweder als Fahrerbeanstandungen oder als Fehlercodes in den Steuergeräten der verschiedenen Subsysteme eines Fahrzeugs abgelegt sind. Abgesehen von offenkundigen Fällen gilt es, durch eine Folge geeigneter Tests und Messungen die Fehlerursache allmählich so weit einzugrenzen, daß er, gewöhnlich durch Ersetzen einer defekten Komponente, behoben werden kann. Die derzeitigen Fehlercodes stellen in der Regel keine Diagnose im Sinne einer Fehlerlokalisierung dar, sondern ein Symptom, eine von der ECU gemessene Abnormalität (etwa das Ausbleiben eines Signals oder die Überschreitung eines plausiblen Bereichs). Zusätzliche Informationen werden in der Werkstatt entweder aus manuell vorgenommenen Messungen und Beobachtungen (etwa Spannungs- und Widerstandsmessungen in der Elektrik) oder automatischen Messungen eines an die ECU angeschlossenen Werkstatt-Testers gewonnen. Solche Tests erfordern oft vor- und nachbereitende Tätigkeiten mit unterschiedlichem Aufwand (z.B. Demontage von Verkleidungen, Testfahrten, spezielle Ausrüstungen und Geräte), was die Auswahl und Reihenfolge der durchzuführenden Schritte bestimmt.

- *Erzeugung von Fehlersuchanleitungen*

Der Mechaniker in der Reparaturwerkstatt wird u.a. durch Diagnose-Manuale ausgebildet und angeleitet, die von einer zentralen Abteilung des Kundendienstes produziert und (auf Papier, CD-ROM oder künftig über das Internet) verbreitet werden. Bei der Erstellung solcher Manuale („Fehlersuchanleitungen“) führen Ingenieure Informationen verschiedener Art (Tabellen, Abbildungen, textuelle Anleitungen für Prüfabläufe) zusammen, die notwendig oder nützlich für die Durchführung der Diagnose zumindest der herkömmlichen Fehler eines Subsystems sind. Solche Dokumente müssen für jede Variante (oder manchmal Gruppe von Varianten) der verschiedenen Subsysteme produziert werden in Abhängigkeit vom jeweiligen Modell, Baujahr, speziellen Ausfertigungen etc. Vor allem für Zulieferer wie die Robert Bosch GmbH bedeutet dies einen erheblichen Aufwand für die Erstellung und Anpassung einer großen Zahl von spezifischen Fehlersuchanleitungen, wozu noch deren Übersetzung in bis zu 20 verschiedene Sprachen tritt. Der Kern des Dokuments b esteht aus einer Menge von Prüfplänen, die die Fehlereingrenzung und -behebung in einzelnen Funktionsgruppen des Subsystems ausgehend entweder von Fahrerbeanstandungen oder abgelegten Fehlercodes beschreiben. Auch diese Prüfpläne müssen natürlich die praktischen Bedingungen und Kosten der Durchführung der notwendigen Tätigkeiten in der Werkstatt reflektieren, was zum Teil durch Überprüfung an Fahrzeugen parallel zur redaktionellen Arbeit geschieht.

Normalerweise stellt keine dieser Aufgaben die entsprechenden Experten vor unlösbare Probleme und erfordert keineswegs Wissen und Kenntnisse extrem spezieller Natur. Dennoch ist ihre Erfüllung oft sehr zeitaufwendig. Dies ist zum einen der erwähnten Vollständigkeit zuzuschreiben, mit der etwa Komponentenfehler und Betriebsbedingungen in der FMEA und in den Prüfplänen abgedeckt werden müssen. Der andere Faktor für den Zeitaufwand ist durch die Vielfalt von Subsystem-Varianten gegeben, die zwar nicht immer völlig neue Analysen und Dokumente verlangen, aber dennoch in aller Regel eine Überprüfung und Anpassung. Diese Situation einer Anwendung eher routinemäßiger, wissensgestützter Tätigkeiten auf eine große Menge von Varianten begründet das starke ökonomische Interesse an Computerunterstützung. Sie ist auch die Basis für eine erfolgreiche Realisierung durch modellbasierte Systeme, weil diese die Möglichkeit bieten, das erforderliche technische Wissen im Rechner in systematischer Form zu repräsentieren und algorithmisch zu nutzen.

2.1.2 Eine Analyse der Aufgabenstellungen

Im folgenden wird versucht, den Kern der jeweiligen Problemlösung und ihre notwendigen Wissensquellen so generell zu charakterisieren, daß sie auf wissensbasierte Algorithmen und Einheiten einer Wissensbasis abgebildet werden können.

- *FMEA-Unterstützung*

  Das grundlegende Wissen, das für diese Aufgabe repräsentiert werden muss, betrifft die Wirkungsweise der in einem Subsystem verwendeten Komponenten, d.h. Verhaltensmodelle von Komponenten, die sowohl ihr nominales Verhalten als auch mögliche Fehlverhalten beschreiben. Das zweite offenkundige Element ist eine Darstellung der Struktur des Subsystems (eine „Blaupause“), die die Interaktion der beteiligten Komponenten beschreibt. Der Kern der Aufgabe besteht darin, auf der Grundlage dieser beiden Eingaben die möglichen Effekte von Fehlern der einzelnen Komponenten zu bestimmen, was als modellbasierte Verhaltensvorhersage bezeichnet wird. Die sich anschließende Bewertung stützt sich auf die Wahrscheinlichkeit der angenommenen Fehler und die „Kritikalität“ der Auswirkung. Letzteres erfordert mehr als eine bloße Beschreibung der physikalischen Effekte, nämlich Wissen darüber, ob und in welchem Ausmaß diese Effekte die intendier te Funktionalität des jeweiligen Subsystems innerhalb des Fahrzeugs beeinträchtigen (z.B. ob lediglich Geräusche entstehen, die Motorleistung reduziert ist oder die Gefahr von Feuer gegeben ist).

- *Werkstattdiagnose*

  Ziel dieser Aufgabe ist offenkundig, Komponenten zu identifizieren, deren Fehlverhalten die beobachteten störenden Effekte, d.h. Symptome verursacht haben könnten. Wiederum ist also Wissen über Verhalten und evtl. Fehlverhalten der vorhandenen Komponenten grundlegend, ebenso wie Kenntnis der besonderen Struktur des Subsystems und die auf beides gegründete Verhaltensvorhersage, die es gestattet, vom Modell impliziertes Verhalten mit dem tatsächlich beobachteten Verhalten in Beziehung zu setzen. Normalerweise kann die Fehlerursache nicht in einem Schritt ermittelt werden, so daß die Erzeugung geeigneter Tests für die Gewinnung zusätzlicher Information eine weitere Teilaufgabe ist. Deren Natur besteht darin, Aktionen vorzuschlagen, die mit Sicherheit oder zumindest gewisser Wahrscheinlichkeit Unterschiede in den Auswirkungen verschiedener (Fehl-)Verhalten beobachtbar machen. Also erfordert auch dieser Schritt eine Form der Verhaltensvorhersage und darüber hinaus Wissen über die zur Realisierung notwendigen Aktionen und den damit verbundenen Zeit- und Kostenaufwand. Soweit sich die Werkstattdiagnose auf Meßwerte stützt, die mit Hilfe eines Werkstatt-Testers aus der ECU ausgelesen werden, stellt sich ferner die Aufgabe, die Rohsignale

im Hinblick auf die Diagnoseaufgabe geeignet zu interpretieren (z.B. die Abweichung gemessener Werte von Sollwerten zu bestimmen).

- *Erzeugung von Fehlersuchanleitungen*

  Im Gegensatz zu einem (interaktiven) Werkstattdiagnosesystem, das dynamisch auf sich ändernde Situationen und darin verfügbare Information reagiert, muss eine Fehlersuchanleitung alle möglichen Diagnosesituationen abdecken, die sich während der Fehlersuche aufgrund eines Symptoms ergeben können. Demnach besteht hier das Wesen der Aufgabe in der Erzeugung eines gesamten Testplans, sinnvoll abstrahierbar zu einem mehr oder weniger komplexen Entscheidungsbaum. Auch hier wird also Verhaltensvorhersage basierend auf Wissen über Komponentenverhalten und Systemstruktur verlangt sowie Information über *Aktionen* und deren *Kosten*.

2.1.3 Architektur der Prototypen

Wenn man aus diesen Skizzen der technischen Lösungen die verschiedenen Elemente des zu repräsentierenden Wissens und der Algorithmen extrahiert, ergibt sich folgendes Bild:

- Grundlegend für alle drei Lösungen sind eine Bibliothek von ***Verhaltensmodellen*** der verwendeten *Komponententypen* und eine ***Strukturbeschreibung*** („Netzliste“) des betrachteten technischen Systems. Aus diesen beiden Elementen wird dessen Verhaltensmodell konstruiert (Systemmodell, „Device model“), was durch eine Softwarekomponente, die *Modellgenerierung* („Model composition“) automatisch geschieht (vgl. Abbildung 2-1).
- Das Verhaltensmodell des Systems stellt die Basis für die *Verhaltensvorhersage* dar, die ebenfalls zum gemeinsamen Kern der drei Werkzeuge gehört (Abbildung 2-1), wenn auch möglicherweise in verschiedenen technischen Realisierungen. Bei der FMEA und der Diagnose kommt es darauf an, aus verschiedenen angenommenen Betriebszuständen bzw. vorgenommenen Messungen und Beobachtungen möglichst viele Informationen über den zugehörigen Zustand des Systems aus dem Modell abzuleiten. Für die Testgenerierung hingegen ist wesentlich, daß Verhaltensvorhersage die Systemzustände im fehlerfreien Zustand bzw. in Fehlerfällen *für alle möglichen Eingaben und Einflüsse* in einem globalen Bild darstellt, um daraus geeignete Unterscheidungen zu ermitteln.
- Während bei der FMEA die Verhaltensvorhersage bereits den Kern der Aufgabe löst, nämlich Bestimmung der möglichen Auswirkungen von Komponentenfehlern auf bestimmte, funktional relevante Größen, sind für die anderen Aufgaben weitere Algorithmen notwendig. Ein *Diagnosegenerator* ermittelt aus den Ergebnissen der Verhaltensvorhersage (genauer: aus deren Widersprüchen zu den tatsächlichen Beobachtungen) Hypothesen über defekte Komponenten. Der *Testgenerator* hingegen macht umgekehrt zu unterscheidende Systemverhalten (Fehlerfälle) zum Ausgangspunkt und ermittelt daraus Kandidaten für diskriminierende Tests, d.h. vorzunehmende Zustandsänderungen des Systems, die für bestimmte Systemgrößen zu unterscheidbaren Beobachtungen führen. Auf diese Weise kann entweder ein gesamter Entscheidungsbaum (für die Prüfplangenerierung) oder ein Vorschlag für einen einzelnen Test (als nächstem Schritt im Rahmen der Werkstattdiagnose) erzeugt werden.
- Soweit im Werkstattdiagnosesystem Meßwerte aus einem Tester verarbeitet werden sollen, so sind die gewonnenen Rohsignale durch eine *Signalabstraktion* so zu transformieren, daß sie von dem Modell verarbeitet werden können.
- Die Generierung geeigneter Tests muss offenkundig auf eine Darstellung der praktischen Aspekte ihrer Durchführung gestützt sein, nämlich geeignete *Tätigkeiten* (Eingaben, Beobachtungen), ihrer Voraussetzungen (z.B. Geräte, Demontage) und des erforderlichen Aufwands, in Abbildung 2-1 in „Kosten“ zusammengefaßt.
- Schließlich sei noch erwähnt, daß nur für die FMEA und die Testgenerierung explizite Modelle der relevanten Komponentenfehler unabdingbar sind, während Diagnoseverfahren existieren, die Fehlerdetektion und -lokalisierung nur auf der Basis von Modellen korrekten Komponentenverhaltens vornehmen ([Dressler Struss 1996], [Hamscher et al. 1992]).

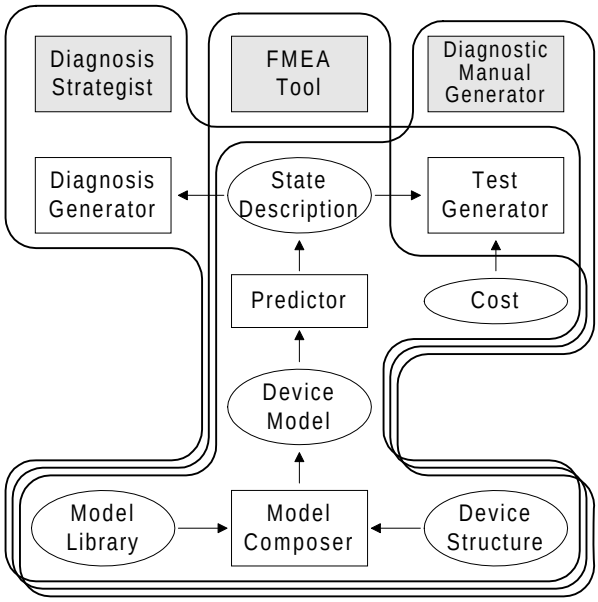


Abbildung 2-1: Zusammenhang zwischen Softwarekomponenten und Elementen der Wissensbasis

Abbildung 2-1 zeigt eine Übersicht über die erwähnten Softwarekomponenten und Elemente der Wissensbasis sowie ihren Zusammenhang untereinander und mit den verschiedenen aufgabenbezogenen Werkzeugen. Dabei wird deutlich, daß bei der vorgenommenen Strukturierung ein hoher Grad an Wiederverwendbarkeit sowohl des repräsentierten Wissens als auch der Softwarekomponenten quer zu den Einzelaufgaben erzielt werden kann. Dies betrifft zum einen natürlich die *Modellbibliothek* und die *Strukturbeschreibung*, was eine grundlegende Zielsetzung modellbasierter Systeme ist. Gemeinsam sind auch die *Modellkomposition* und *Verhaltensvorhersage*. Hier wurden die entsprechenden Komponenten wie auch der Diagnosealgorithmus aus dem Diagnose- und Modellierungssystem RAZ`R von OCC`M Software genutzt. Dies gestattet ferner eine Abtrennung der Erstellung des Systemmodells von den aufgabenspezifischen modellbasierten Problemlösungen: Erfassung und Erprobung der Komponententypmodelle und Strukturbeschreibung finden im Entwicklungssystem statt, aus dem letztlich nur das Verhaltensmodell eines Systems exportiert wird, das dann als Input für das RAZ`R-Diagnoselaufzeitsystem, den FMEA-Generator und die Testplangenerierung dient.

In den folgenden Abschnitten werden die im Projekt realisierten Anwendungsprototypen näher beschrieben. Dabei sollen nur die Grundideen der verwendeten Algorithmen und Modellierung vermittelt werden. Für Details und wissenschaftliche Grundlagen wird jeweils auf die zugehörigen Fachbeiträge verwiesen.

2.2 Modellbasierte Verhaltensanalyse für die FMEA

2.2.1 Ziele und Anforderungen

Zu den meisten Kfz-Subsystemen verlangt der Automobilhersteller heute eine ausführliche Fehler-Möglichkeits- und Einfluß-Analyse (FMEA) als Nachweis der präventiven Qualitätssicherung.

In der FMEA werden alle denkbaren Fehlerarten und -ursachen während der Entwurfs-, Konstruktions- und Fertigungsphase in Betracht gezogen und es wird versucht, ihnen Fehlerauswirkungen auf Ebene der Komponenten, Subsysteme und des Fahrzeugs zuzuordnen.

Die dazu notwendigen Informationen und Schlußfolgerungen werden auf FMEA-Teamsitzungen nach methodischen Ansätzen (Tabellenformular) erarbeitet, gesammelt und dokumentiert. Abbildung 2-2 zeigt die Grundstruktur einer so entstehenden Tabelle.

Da sich ein FMEA-Team aus Fachleuten von der Entwicklung bis hin zur Fertigungsvorbereitung zusammensetzt, ist das Verfahren sehr personalintensiv, weshalb meist noch eine Phase der Wissenserhebung durch einen

Moderator vorgeschaltet wird. Durch diese Arbeit wird insgesamt jedoch sehr viel wertvolle Ingenieurkapazität gebunden.

Der Anwender Bosch verfolgt daher das Ziel, sowohl zur Vorbereitung durch den FMEA-Moderator als auch zur Verwendung während der FMEA-Sitzungen rechnerbasierte Techniken einzusetzen.

Dazu wird ein Werkzeug benötigt, das die FMEA in der bisher gewohnten Weise (Tabellenformular) als strukturierender Editor unterstützt und das die zentrale Aufgabe der FMEA, eine Verhaltensanalyse durch Verfolgen von Fehlerarten und ihren Auswirkungen im System durchzuführen, automatisch vornimmt.

Component Process	Function Purpose	Failure Mode	Failure Effect	Failure Cause	...
...	...	...	...	...	...

Abbildung 2-2: Aufbau eines FMEA-Dokuments

2.2.2 Modellbasierte Vorhersage von Fehlerursache-Fehlerauswirkungs-Beziehungen

Der Rahmen für die Realisierung eines modellbasierten FMEA-Werkzeugs ist durch die in **Abbildung** 2-1 abgebildeten Komponenten gegeben. Die daraus für die FMEA benötigte Kernfunktionalität wird durch das Verhaltensvorhersagemodul („Predictor“) abgedeckt. Der Predictor liefert für eine Menge von Beobachtungen (sog. „Szenario“) und einen gegebenen Kontext (d.h. Normalverhalten oder Fehlverhalten von Komponenten) mit Hilfe des Modells Vorhersagen über die Werte von Modellvariablen, die unter diesem Kontext gelten.

Unter der Voraussetzung, daß Verhaltensmodelle für die relevanten Fehlerfälle von Komponenten vorhanden sind, ermöglicht dies die automatische Bestimmung von Zusammenhängen zwischen den auftretenden Fehlerursachen (d.h. Fehlermodus der entsprechenden Komponente) und deren Auswirkungen auf das System (d.h. abgeleitete Werte für weitere Systemvariablen).

Wie bei der manuell durchgeführten FMEA beschränkt sich der Prototyp dabei auf die Betrachtung von Einfachfehlern. Die auftretenden Kontexte für den Predictor können also in Form einer Schleife erzeugt werden, die alle Einfachfehlerannahmen durchläuft und diese mit den Korrektheitsannahmen für alle übrigen Komponenten kombiniert.

2.2.3 Architektur des FMEA-Prototyps

Die Basis für die automatisierte Vorhersage von Fehlerauswirkungen aus Fehlerursachen nach dem oben dargestellten Ansatz ist:

- eine Bibliothek von Verhaltensmodellen für Komponenten, die das korrekte Verhalten und das Verhalten unter relevanten Fehlern abdecken (Model Library),
- eine Beschreibung der Systemstruktur, d.h. der auftretenden Komponententypen und ihrer Verbindungen (System Model),
- eine Beschreibung relevanter Situationen (meist Worst-Case-Szenarien) in Form von Sequenzen von Zuweisungen an Variablen, die bestimmte Betriebszustände des Systems charakterisieren, z.B. die Druckaufbauphase in einem ABS (Situation Description).

Abbildung 2-3 zeigt die Architektur des FMEA-Prototyps. Es wird vorausgesetzt, daß alle aufgeführten Eingaben in Form von XML-Dateien vorliegen. Der Prototyp durchläuft anhand dieser Informationen systematisch die einzelnen Fehlermodi und Situationen und bereitet die Ergebnisse der Verhaltensvorhersage in Form einer FMEA-Tabelle auf.

Bei der Generierung der FMEA-Tabelle kann nicht davon ausgegangen werden, daß jeder für eine Variable vorhergesagte Wert eine Fehlerauswirkung darstellt und als Eintrag in die FMEA-Tabelle übernommen werden kann. Dies liegt daran, daß in der FMEA nicht das Normalverhalten, sondern nur der Unterschied von Normal- zu Fehlverhalten betrachtet wird. Außerdem geschieht dies meist noch bezogen auf eine bestimmte Auswahl von

relevanten Systemgrößen, d.h. in der FMEA-Tabelle wird meist nicht die gesamte Wirkungskette eines Fehlers wiedergegeben, sondern man beschränkt sich auf „interessante“ Auswirkungen.

In der Grundstufe des FMEA-Prototyps wird deshalb zunächst für jede Situation das Verhalten des Systems vorhergesagt unter der Annahme, daß alle Komponenten korrekt sind. Ein unter einer bestimmten Fehlerannahme vorhergesagter Wert wird dann als Fehlerauswirkung betrachtet, wenn er von dem entsprechenden Ergebnis für korrekt funktionierende Komponenten abweicht.

Die gewünschte Beschränkung auf eine Teilmenge von „interessanten“ Variablen wird dabei durch die Angabe einer zusätzlichen Fokusbeschreibung (Focus Description) ermöglicht. Diese kann z.B. Output-Variablen enthalten (etwa das Giermoment des Fahrzeugs bei der FMEA eines Anti-Blockier-Systems), wodurch der gewünschte Effekt erzielt wird, „uninteressante“ interne Variablen (z.B. hydraulischer Druck an verschiedenen Stellen im Hydroaggregat) bei der FMEA eines Anti-Blockier-Systems auszublenden.

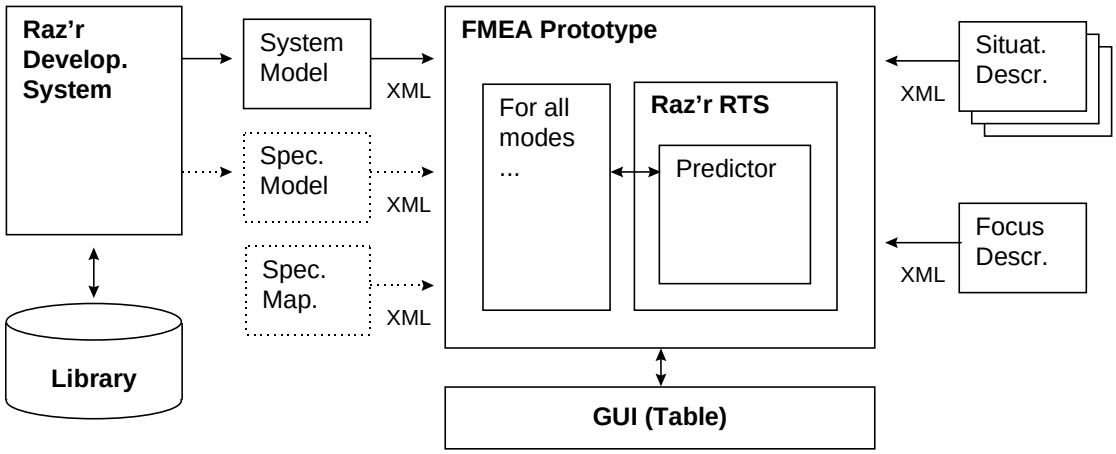


Abbildung 2-3: Architektur des FMEA-Prototyps

In diesem FMEA-Basissystem wird das Referenzverhalten des Systems also implizit durch sein Normalverhalten festgelegt. Eine Erweiterung dieser Basisfunktionalität ist in Abbildung 2-3 gestrichelt dargestellt. Bei dieser Erweiterung kann zusätzlich zum Systemmodell die explizite Repräsentation eines spezifizierten Systemverhaltens mit angegeben werden. Dieses „Specification Model“ ist im Grunde nichts weiter als ein zusätzliches Verhaltensmodell, welches nun aber dazu dient, das intendierte Verhalten des Systems zu beschreiben. Ein „Specification Mapping“ ist nötig, um die Abbildung der Variablen, die in der Spezifikation auftreten, auf die Variablen des betrachteten Systemmodells herzustellen. Dadurch wird es möglich, mehrere verschiedene Realisierungen oder Modifikationen eines Systems mit einer gemeinsamen, vorher festgelegten Spezifikation zu vergleichen.

Trotz der Beschränkung auf Einfachfehler kann die Menge der zu betrachtenden Kontexte (Fehlermoduszuweisungen) noch sehr groß werden. Die Laufzeit des Predictors würde mit jeder zusätzlichen Komponente im System multiplikativ anwachsen. Man stellt jedoch fest, daß nicht alle Vorhersagen für Modellvariablen in jedem Kontext unterschiedlich sind. Besonders wenn ein System sehr groß ist, wirkt sich ein Fehler oft nur in bestimmten Bereichen des Systems aus, während andere Bereiche des Systems von Fehlerauswirkungen unberührt bleiben. Dieser Effekt kann ausgenutzt werden, indem bereits abgeleitete Vorhersagen für weitere Kontexte wiederverwendet werden. In dem vorgestellten Prototyp wird dies durch die Verwendung eines sog. Assumptionbased Truth Maintenance Systems (ATMS) ([de Kleer 1986]) erreicht. Das ATMS ermöglicht es, daß für jeden neu zu betrachtenden Fehler nur noch die Differenzen zu bereits untersuchten Fehlern behandelt werden müssen. Bereits abgeleitete und in einem neuen Kontext noch gültige Ergebnisse werden dadurch nicht nochmals neu erzeugt. Anhängig von der Art der auftretenden Fehler und der Struktur des Systems kann dies einen erheblichen Effizienzgewinn bewirken.

2.2.4 Realisierung des FMEA-Prototyps

Im Projektzeitraum von INDIA wurde basierend auf den dargestellten Konzepten ein Demonstrator für die modellbasierte Unterstützung von FMEA spezifiziert und unter Verwendung vorhandener Softwarekomponenten des Systems RAZ'R implementiert.

Während des Projektzeitraums gab es noch keinen konkreten Anwendungspartner bei Bosch, der spezifische Anforderungen für eine Validierung bereitstellen und eine Evaluierung hätte vornehmen können. Insbesondere existierten damit für den FMEA-Prototyp noch keine Anforderungen an die Bedienoberfläche, so daß diese noch sehr rudimentär ist. Abbildung 2-4 zeigt ein Beispiel. Die zugrundeliegende Implementierung umfaßt jedoch alle in Abbildung 2-3 dargestellten Schnittstellen, inklusive der im vorigen Abschnitt beschriebenen Erweiterung um die Spezifikation eines Referenzmodells.

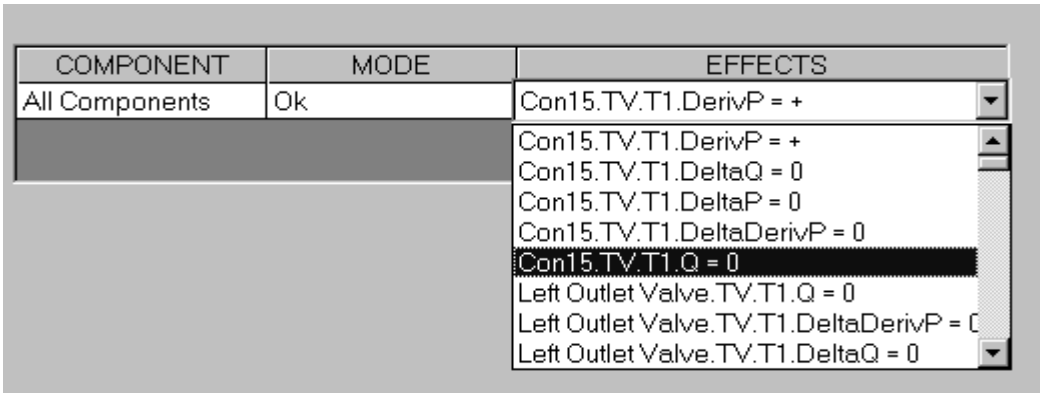


Abbildung 2-4: Beispiel für die Ausgabe des FMEA-Prototyps

2.2.5 Verwandte Ansätze

Die FMEA ist eine in der Industrie weit verbreitete Methode. Daher existieren verschiedenste Ansätze, den Prozeß der FMEA zu unterstützen. Die meisten dieser Systeme bieten jedoch nur graphische Werkzeuge und Textverarbeitungsfunktionen zur Verwaltung der Einträge in den FMEA-Tabellen an. Sie unterstützen dagegen nicht die hier zum Gegenstand gemachte zentrale Aufgabe der FMEA, die Verhaltensanalyse von Fehlern. Eine Ausnahme bildet das kommerzielle System AutoSteve ([Price 1998]), das z.B. bei den Automobilherstellern Jaguar und Ford eingesetzt wird. AutoSteve automatisiert einen Großteil des FMEA-Prozesses für eine spezielle Anwendungsdomäne, nämlich elektrische Schaltungen. Dazu wird das Verhalten des elektrischen Schaltkreises im Normalfall und im Fehlerfall simuliert. Aus dem Ergebnis werden mittels einer Auswahl vorgegebener Textbausteine weitgehend natürlichsprachliche FMEA-Einträge erzeugt.

Die wesentlichen Unterschiede zwischen dem hier beschriebenen modellbasierten FMEA-Prototyp und AutoSteve sind:

- AutoSteve testet nicht systematisch alle möglichen Einfachfehler, sondern nur eine vom Benutzer spezifizierte Auswahl. Der Grund liegt in einem Effizienzproblem des eingesetzten Simulationsverfahrens, das für jeden Fehlerfall neu angestoßen werden muss. Der hier vorgestellte Prototyp basiert dagegen auf einem ATMS, welches wie ein Cache für bereits abgeleitete Ergebnisse funktioniert. Für weitere Fehler müssen deshalb nur noch die Differenzen zu den bereits untersuchten Fehlern betrachtet werden, wodurch es möglich wird, systematisch alle Einfachfehler zu analysieren.
- AutoSteve verwendet einen speziellen Simulationsalgorithmus (QCAT), der auf Modelle elektrischer Netze spezialisiert ist. Der hier vorgestellte Prototyp basiert dagegen auf einem generischen Predictor-Modul, welches nicht auf eine spezifische Art von Modellen festgelegt ist, und kann daher beliebige Verhaltensmodelle verarbeiten.
- AutoSteve ist beschränkt auf die implizite Definition von Fehlverhalten als Differenz zum Verhalten des Systems, wenn alle seine Komponenten korrekt arbeiten. Der vorgestellte FMEA-Prototyp erlaubt dagegen den Vergleich zwischen einem Systemmodell und einer beliebigen, explizit vorgegebenen Spezifikation seines intendierten Verhaltens.

2.3 Modellbasierte Unterstützung der Online-Werkstattdiagnose

2.3.1 Ziele und Anforderungen

Die durchgeführten Arbeiten zur FMEA und zur Generierung von Fehlersuchanleitungen lassen die Verwendung der Modelle und Diagnosealgorithmen auch für Aufgaben in der Werkstattdiagnose bzw. der Onboard-Diagnose im Fahrzeug vielversprechend erscheinen. Einerseits hat diese Problemstellung einen Bezug zu den bisherigen Arbeiten, da auf Seite des physikalischen Systems die gleichen Komponenten wie in den bisher behandelten Anwendungen auftreten und damit die vorhandenen Modellbibliotheken benutzt werden können. Andererseits müssen für die Übertragung der modellbasierten Ansätze zusätzliche Probleme in Angriff genommen werden, welche in den Systemen zur Unterstützung der Verhaltensanalyse bei der FMEA bzw. Generierung von Fehlersuchanleitungen für die Werkstätten nicht auftreten.

Die entwickelten FMEA- und Testgenerierungssysteme arbeiten interaktiv. Insbesondere sind bei diesen Systemen die Beobachtungen des Systemverhaltens jeweils vom Benutzer zu spezifizieren. In der Werkstattdiagnose entspricht dies den Kundenbeanstandungen, die meist in vagen qualitativen Beschreibungen wie z.B. „Bremspedal zu weich“ vorliegen. Der in diesem Band wiedergegebene Beitrag [Struss et al. 1997] beschäftigt sich mit der Frage, wie solche Beschreibungen im Rahmen eines modellbasierten Diagnoseansatzes nutzbar gemacht werden können.

Daneben stützt sich die Werkstattdiagnose aber auch auf die mit Hilfe eines Werkstatt-Testers gewonnenen Meßwerte. Onboard-Diagnose ist sogar nur auf die laufenden Signale des Steuergeräts angewiesen. Soweit die Diagnose solche elektronisch erfaßten Meßwerte mit einbezieht, stellt sich deshalb die Aufgabe, wie zeitlich indizierte Sensorsignale in einem modellbasierten Diagnosesystem geeignet verarbeitet werden können. Die Realisierung eines entsprechenden Prototypen wird in diesem Abschnitt beschrieben. Da die Komponentenbibliothek und auch die Basis-Software-Komponenten hierfür vorhanden sind, beschränkten sich die Arbeiten für einen solchen Prototypen im wesentlichen auf zwei Komponenten: Die Erfassung und Aufbereitung von Signalen des Fahrzeug-Subsystems zur Verarbeitung in einem modellbasierten Diagnosesystem und die Transformation des Systemmodells gemäß den Anforderungen der Online-Diagnose. Die Onboard-Diagnose wirft darüber hinaus das Problem der Diagnose im Umfeld limitierter Ressourcen (Verarbeitung der Daten möglichst in Echtzeit, begrenzter Speicherplatz für das Diagnosesystem) auf.

Ein im Rahmen des Projekts betrachtetes Leitbeispiel für die Erprobung modellbasierter Online-Diagnose im Bereich mechatronischer Kfz-Subsysteme waren abgasrelevante Fehler in der Dieselregelung, die von der bisherigen Onboard-Diagnose nicht abgedeckt werden können. Für diese Szenarien standen beim Anwendungspartner Bosch entsprechende Meßdaten zur Verfügung.

2.3.2 Verfahren zur modellbasierten Online-Diagnose

Die dargestellten Anforderungen führten auf Forschungsseite zur Entwicklung eines Diagnoseverfahrens, das sich durch besondere Effizienz im Bereich der Diagnose dynamischer Systeme auszeichnet. Im folgenden Abschnitt werden die Grundlagen dieses Ansatzes skizziert. Eine ausführliche Darstellung findet sich in [Struss 1997].

2.3.3 Zustandsbasierte Diagnose dynamischer Systeme

Die Aufgabenstellung der konsistenzbasierten Diagnose ist es, zu untersuchen, ob das vom Modell vorhergesagte Verhalten mit den Beobachtungen konsistent ist. Wenn Model(Mode) das Modell eines Verhaltensmodus und Obs eine Menge von Beobachtungen beschreibt, bedeutet dies, die Konsistenz von

$$\text{Model(Mode)} \cup \text{Obs}$$

zu prüfen. Wenn dieser Ansatz für die Diagnose eines dynamischen Systems angewendet wird, ergibt sich die Frage, ob dies eine Vorhersage des Verhaltens über der Zeit, d.h. Simulation, erfordert. Häufig wird dies bei der Diagnose dynamischer Systeme sogar von vornherein als gegeben angenommen. Es zeigt sich aber, daß Diagnoseergebnisse auch auf der Grundlage eines Konsistenzchecks von Modell und beobachteten Zuständen gewonnen werden können, d.h. ohne Simulation von Verhalten über der Zeit. Unter bestimmten Voraussetzungen hat dies sogar überhaupt keinen Verlust an Diagnoseinformation zur Folge ([Struss 1997]).

Die Idee dieser sogenannten zustandsbasierten Diagnose ist es, das Modell eines dynamischen Systems in zwei Klassen von Constraints zu zerlegen. Die eine Klasse von Constraints beschränkt die möglichen Zustände des Systems zu einem bestimmten Zeitpunkt (State-Constraints). Die andere Klasse von Constraints beschränkt die möglichen Übergänge von einem Zustand zu einem anderen (Temp-Constraints):

$$\text{Model(Mode)} = \text{State-Constraints(Mode)} \cup \text{Temp-Constraints(Mode)}.$$

Für die zustandsbasierte Diagnose wird dann lediglich geprüft, ob

$$\text{State-Constraints(Mode)} \cup \text{Obs}$$

konsistent ist, d.h. es wird nur geprüft, ob die Menge der beobachteten Zustände eine Teilmenge der konsistenten Zustände des Systems ist. Da folglich die Reihenfolge der beobachteten Zustände nicht mehr in die Betrachtung eingeht, scheint dieser Ansatz auf den ersten Blick weitaus schwächer als ein simulationsbasierter Ansatz zu sein, der das beobachtete Verhalten mit dem simulierten Verhalten des Modells vergleicht. Wenn jedoch Temp-Constraints nur aus Constraints besteht, welche generell gültige mathematische Gesetze über Stetigkeit, Differenzierbarkeit etc. beinhalten, dann kann Temp-Constraints keine zusätzlichen Widersprüche im Modell liefern. Solche Constraints können deshalb von der Konsistenzprüfung ausgenommen werden, ohne das Diagnoseergebnis zu beeinflussen.

Durch zustandsbasierte Diagnose wird die Notwendigkeit vermieden, das Modell über verschiedene Zeitpunkte hinweg zu simulieren. Statt dessen kann die Diagnose auf die einfache Konsistenzprüfung von Zuständen abgebildet werden. Dies kann einen großen Effizienzgewinn bedeuten, insbesondere, wenn dadurch Fehler nicht simuliert werden müssen. Zustandsbasierte Diagnose stellt damit einen wichtigen Beitrag dar, um die geforderten Reaktionszeiten des Online-Diagnoseprototyps zu gewährleisten.

2.3.4 Automatische Modelltransformation für kompositionale Modelle

Ein anderer Schlüssel zur Erreichung der nötigen Effizienz, um modellbasierte Diagnose im Online-Betrieb einsetzen zu können, liegt auf Seiten des Modells.

Im Hinblick auf die zeit- und speicherkritische Anwendung für Online-Diagnose ist es notwendig, daß im Modell nur solche Unterscheidungen berücksichtigt werden, die zur Erreichung eines bestimmten Ziels (z.B. Diskriminierung zweier Diagnosekandidaten mit unterschiedlichen Recovery Actions) auch unbedingt nötig sind. Da das Diagnosesystem automatisch aus einer Bibliothek von Modellfragmenten konfiguriert wird, die unabhängig vom Verwendungskontext einsetzbar sein sollen, können diese Unterscheidungen nicht in den einzelnen Modellfragmenten angelegt sein. Statt dessen stellen sie eine Eigenschaft des Gesamtmodells und seines Verwendungszwecks dar.

Deshalb ist eine zwischengeschaltete Komponente zur automatischen Modelltransformation notwendig, welche die relevanten Unterscheidungen (d.h. qualitative Werte) für einen bestimmten Anwendungskontext generiert. Ziel dieses Arbeitspakets war daher die Entwicklung von Techniken zur automatischen Transformation eines kompositionalen Modells auf eine spezifische, durch den Anwendungskontext vorgegebene Granularität. Dies umfaßt die Berechnung von qualitativen Werten im Modell abhängig von dem zu erreichenden Diagnoseziel und der Beobachtbarkeit, z.B. der vorgegebenen Meßgenauigkeit oder Abtastfrequenz der Sensorsignale.

Die theoretischen Grundlagen für die aufgabenbezogene Modellabstraktion und eine Reihe von Algorithmen hierfür sind in [Sachenbacher Struss 2000] und [Struss Sachenbacher 1999] zusammengefasst.

2.3.5 Automatische Signalaufbereitung für die Online-Diagnose

Die Anbindung des modellbasierten Diagnosesystems an reelle Daten, wie sie z.B. auf dem Fahrzeug vom Steuergerät oder in der Werkstätte von Meßgeräten geliefert werden, erfordert aber auch die Transformation der Daten selbst auf eine für das zugrundeliegende Modell geeignete Darstellung und Granularität. Das Ziel eines weiteren Arbeitspakets war daher die Entwicklung einer Komponente für die automatische Signaltransformation, d.h. die Umwandlung numerischer, zeitlich indizierter Sensorsignale auf die Repräsentationsebene der qualitativen Modelle. Dies umfaßt u.a. die Abbildung quantitativer Werte auf qualitative Wertebereiche des Modells, die Berechnung abgeleiteter Werte aus den numerischen Daten (z.B. Abweichungen) und die Aggregation einzelner quantitativer Zustände zu qualitativen Beobachtungsvektoren als Input für das Diagnose-Runtimesystem.

2.3.6 Architektur des Online-Diagnoseprototyps

Die Software für den Online-Diagnoseprototypen besteht damit aus den folgenden Komponenten:

- ein Modul zur Transformation der Wertebereiche von Modellvariablen auf die erforderliche (qualitative) Granularität,
- ein Modul zur Umwandlung der Rohsignale in qualitative Beobachtungen,
- ein modellbasiertes Runtime-Diagnosesystem, das basierend auf diesen Beobachtungen und dem Modell eine zustandsbasierte Diagnose durchführt.

Der erste Punkt besteht aus einer prototypischen Implementierung der Verfahren aus [Sachenbacher Struss 2000], welche auf dem Modellformat des kommerziellen Diagnose-Frameworks RAZ'R aufsetzt.

Der zweite Punkt umfaßt eine Komponente zur Umwandlung quantitativer Signale in qualitative Werte und Abweichungen. Von diesem Modul wird jedes Mal, wenn eine Veränderung der beobachteten Variablen oder ihrer Abweichungen auf der qualitativen Ebene eintritt, ein neuer Vektor von Beobachtungen erzeugt und an die Diagnosemaschine übergeben.

Für den dritten Punkt wurden Komponenten des kommerziellen Diagnose-Frameworks RAZ'R verwendet, das ein Entwicklungssystem für Diagnosemodelle und ein Runtime-System einer konsistenzbasierten Diagnosemaschine enthält. Die Diagnosemaschine führt zustandsbasierte Diagnose mit dem Modell und den qualitativen Beobachtungsvektoren durch. Das Diagnoseergebnis erhält man schließlich aus der Kombination der Diagnoseergebnisse für die einzelnen Beobachtungsvektoren.

2.3.7 Evaluierung des Prototyps an Daten eines Versuchsfahrzeugs

Während der Laufzeit von INDIA bestand die Möglichkeit, Experimente im Zusammenhang mit der Online-Diagnose anhand der konkreten Daten eines Versuchsfahrzeug durchzuführen, das bei Bosch im Rahmen des thematisch verwandten EU-Projekts „VMBD“ (Vehicle Model Based Diagnosis) zur Verfügung stand. Es handelte sich um einen Volvo 850 TDI mit einer elektronisch gesteuerten, verteilerbasierten Einspritzung (distributor type injection, abgekürzt DTI), für die im Rahmen des Projekts VMBD ein kompositionales Modell entwickelt worden war. In dem genannten Versuchsfahrzeug konnten Fehler in verschiedenen Betriebszuständen injiziert und Messungen der Sensorsignale vorgenommen werden. Die verschiedenen im Fahrzeug eingebauten Fehler wurden hierbei durch Schalter vom Fahrzeuginnenraum aus aktiviert. Ein pneumatisches Leck wurde beispielsweise durch zusätzliche Ventile simuliert, die elektrisch geöffnet oder geschlossen werden konnten.

Für die Gewinnung der Signale zur Online-Diagnose mussten verschiedene zusätzliche Interfaces und Geräte im Fahrzeug installiert werden. Voraussetzung war dabei, daß aus Sicherheitsgründen das Seriensteuergerät und seine Diagnosefunktionen ohne Unterbrechung laufen konnten, und der modellbasierte Diagnoseprototyp Zugriff auf die gleichen Sensorsignale wie das Seriensteuergerät erhalten musste.

Momentan ist die Rechenleistung von Steuergeräten noch sehr beschränkt, so daß das modellbasierte Online-Diagnosesystem nicht direkt in die Steuergerätesoftware integriert werden konnte. Die Meßdaten für den Prototypen wurden deshalb über einen Umweg mit Hilfe eines sogenannten Applikationssteuergeräts gewonnen. Applikationssteuergeräte werden normalerweise zur Parametrisierung und Kalibrierung der Steuergerätesoftware für einen bestimmten Fahrzeugtyp verwendet und enthalten spezielle Speicherbausteine, so daß die Variablen des Steuergeräts und damit auch die Sensorsignale in Echtzeit ausgelesen werden können, ohne die normale Funktion des Steuergeräts zu beeinträchtigen. Dadurch standen die Sensorsignale dem Online-Prototyp in der gleichen zeitlichen Auflösung wie dem Seriensteuergerät zur Verfügung. Die Ergebnisse des Online-Prototyps konnten dadurch auch mit den Diagnosefähigkeiten eines normalen Seriensteuergeräts verglichen werden.

2.3.8 Diagnoseszenarien

Die in dem beschriebenen Versuchsfahrzeug eingebauten Fehler waren im Wesentlichen solche, die mit traditioneller Onboard-Diagnose (d.h. basierend auf Schwellwerten und fest eingebauten Plausibilitätsprüfungen) nicht oder nur unzureichend diagnostiziert werden können. Dazu zählen vor allem emissionsrelevante Symptome wie erhöhte Kohlenstoffemissionen aufgrund von übermäßiger Kraftstoff- oder unzureichender Verbrennungsluftzufuhr zum Motor, die durch elektrische Fehler oder undichte pneumatischen Leitungen verursacht werden.

Eines der im Versuchsfahrzeug realisierten Szenarien besteht aus einem Leck in der Verbindung zwischen dem Turbolader-Auslaß und dem Motorkrümmer. Wenn das Leck geöffnet wird, strömt - abhängig vom Betriebszu-

stand - eine signifikante Menge an Ladeluft aus, die zuvor bereits den Luftmassenmesser passiert hat. Die Einspritzmenge, die vom Steuergerät in Abhängigkeit von diesem Signal berechnet wird, ist deshalb zu hoch für die Menge an Verbrennungsluft, die in der Verbrennungskammer des Motors tatsächlich vorhanden ist. Dieser Fehler führt deshalb zu unvollständiger Verbrennung des Dieselkraftstoffs und damit zu erhöhten Emissionen und verminderter Leistung des Motors.

2.3.9 Messerfassung der Rohdaten

Der Online-Diagnoseprototyp benutzt die gleichen Signale, die auch dem Seriensteuergerät zur Verfügung stehen. Von den verfügbaren Steuergerätevariablen wurden für das beschriebene Szenario die folgenden Variablen an den Prototypen übergeben:

- Atmosphärendruck
- Ladedrucksignal
- Luftmassensignal
- Motordrehzahl
- Ansteuersignal des Ladedruckventils
- aktuelle Einspritzmenge

Die einzelnen quantitativen Meßwerte dieser Variablen liegen nicht notwendigerweise zu äquidistanten Zeitpunkten vor. Der Grund ist, daß die Frequenz, mit der das Steuergerät die Sensorwerte abliest, von der Drehzahl des Motors abhängt.

2.3.10 Beispiel für ein Diagnoseergebnis des Prototyps

In diesem Abschnitt werden die Ergebnisse eines Probelaufs des Diagnoseprototypen für das beschriebene Szenario dargestellt. Die Messung für das exemplarische Beispiel läuft über 9,75 Sekunden und umfaßt 1064 quantitative Beobachtungsvektoren. Das oben beschriebene Modul zur Signaltransformation reduziert diese Daten zu lediglich 12 qualitativen Beobachtungsvektoren. Basierend auf diesen Daten und dem Modell generiert das Diagnosemodul drei Konfliktmengen:

- {Junction1.ok, IntakeTurbine.ok, Junction3.ok, Engine.ok, AirflowSensor.ok, Junction2.ok, PressureSensor.ok} zu den qualitativen Beobachtungsvektoren 3, 5 und 6,
- {Junction1.ok, IntakeTurbine.ok, Junction3.ok, Engine.ok, AirflowSensor.ok, Junction2.ok, SpeedSensor.ok} zu den qualitativen Beobachtungsvektoren 5, 6 und 10,
- {Junction3.ok, Engine.ok, SpeedSensor.ok, PressureSensor.ok} zum qualitativen Beobachtungsvektor 10.

Die Komponentennamen stehen hier für die Annahme, daß die entsprechende Komponente korrekt funktioniert. Die drei Konflikte kombinieren sich zu zwei Einfachfehlerhypothesen und einer Reihe von Mehrfachfehlerhypothesen:

- {Junction3.ok}
- {Engine.ok}
- {PressureSensor, Junction1.ok}
- {PressureSensor, IntakeTurbine.ok}
- {PressureSensor, AirflowSensor.ok}
- {PressureSensor, Junction2.ok}
- {SpeedSensor.ok, Junction1.ok}
- {SpeedSensor.ok, IntakeTurbine.ok}
- {SpeedSensor.ok, AirflowSensor.ok}
- {SpeedSensor.ok, Junction2.ok}
- {SpeedSensor.ok, PressureSensor.ok}

Die zwei Einfachfehlerhypothesen enthalten das Verbindungsstück, wo der Fehler tatsächlich injiziert wurde (Junction3). Die Laufzeit für das Beispiel beträgt 2,87 Sekunden auf einem Pentium PC unter Windows NT. Das bedeutet, daß für dieses Beispiel die Performance des Online-Diagnoseprototyps im Bereich der Echtzeit liegt (die Messung für das Szenario lief über 9,75 Sekunden). Für das Beispiel wurde lediglich ein Modell des Normalverhaltens verwendet. Zusätzliches Wissen über die auftretenden Fehlerfälle kann verwendet werden, um die Menge der Diagnosekandidaten weiter einzuschränken, z.B. um den Motor aus den Diagnosekandidaten zu eliminieren.

Der entstandene Diagnoseprototyp hat gezeigt, daß die modellbasierte Online-Diagnose einer Motorsteuerung in Echtzeit möglich ist. Der Prototyp besitzt jedoch ebenso wie der FMEA-Prototyp noch keine Bedienoberfläche, die auf einen Anwender in der Werkstätte zugeschnitten wäre.

2.3.11 Ausblick: Jenseits komponenten-orientierter Modellierung und Diagnose

Die geschilderten Ansätze zur Werkstatt- und Online-Diagnose sind gemäß dem allgemeinen Stand der Kunst geeignet, Komponentenfehler in stabilen Strukturen zu lokalisieren. Dies ist aber bei einigen Problemen im Fahrzeug (und in stärkerem Maße in der Prozeßindustrie sowie bei der Diagnose und Therapie natürlicher Systeme) zu beschränkt. So läßt sich etwa die mögliche Ursache für erhöhte CO-Emissionen nicht nur an defekten Komponenten festmachen. Z.B. kann Treibstoff mit einem erhöhten Anteil schwerer Kohlenwasserstoffe einen veränderten Verbrennungsprozeß und damit ebenfalls erhöhte Emissionen hervorrufen.

Im Projekt wurden daher theoretische Grundlagen entwickelt und implementiert, die den modellbasierten Diagnoseansatz auf Systeme mit einer dynamischen Struktur aus Prozessen ausweiten (siehe [Struss Heller 1999], [Struss Heller 2000], in diesem Band). Die Prozeßmodellierung kann sich wiederum auf Modelle stützen, deren ursprüngliche Elemente nicht Constraints (d.h. Relationen), sondern kombinierbare Einflüsse sind (s. [Heller Struss 1996a], Nachdruck in diesem Band). Diese Erweiterungen öffnen der Anwendung modellbasierter Diagnoseverfahren eine große neue Klasse von Systemen und Fehlern.

2.4 Generierung von Fehlersuchanleitungen

2.4.1 Automatische Testgenerierung

Testen eines Systems bedeutet, es durch eine Änderung seiner Struktur und/oder einen bestimmten Stimulus (Eingabe) so zu beeinflussen, daß dadurch Beobachtungen möglich werden, die Information über den „Verhaltensmodus“ des Systems (insbesondere vorhandene Defekte) bereitstellen. Für ein System, das sich aus Komponenten in einer festen Struktur zusammensetzt, ist dies gleichbedeutend mit dem Ziel, die Verhaltensmodi der Systemkomponenten, d.h. Komponentenfehler, zu erkennen. Beispielsweise geht es beim Testen nach der Fertigung um die Feststellung, ob alle Komponenten korrekt funktionieren oder ob eine von ihnen fehlerhaft arbeitet. Dies wird normalerweise realisiert durch den Versuch, alle Einfachfehler auszuschließen. Im Diagnosekontext dient Testen dazu, einen vorhandenen Fehler oder eine Klasse von Fehlern zu identifizieren. Da es prinzipiell unmöglich ist, einen bestimmten Fehler oder eine Fehlerklasse direkt durch Beobachtungen zu bestätigen, ist die Aufgabe nur durch die Durchführung von Tests zu lösen, die das Vorhandensein aller anderen möglichen Fehler ausschließen können. Im Gegensatz zur Diagnose, die sich auch allein auf die Kenntnis des intendierten, korrekten Verhaltens stützen kann (vgl. Abschnitt I-2.3), benötigen Testgenerierung und Testen immer das Wissen über die Menge der möglichen Fehler(klassen).

Da modellbasierte Methoden Wissen über korrektes und fehlerhaftes Verhalten von Komponenten und dessen Auswirkungen repräsentieren bzw. ableiten können, bieten sie sich zur Unterstützung und Automatisierung an. In früheren Arbeiten ([Struss 1994a], [Struss 1994b]) haben wir die theoretischen Grundlagen für modellbasierte (komponentenorientierte) Testgenerierung geschaffen und deren Implementierung an einem realistischen Anwendungsproblem erprobt ([Inderst et al. 1995]). Wir verzichten an dieser Stelle auf eine formale Präsentation und beschränken uns darauf, die grundlegenden Ideen zu vermitteln und zu illustrieren.

In der entwickelten Theorie und den darauf gegründeten Algorithmen werden die Verhaltensmodelle des zu testenden Systems (etwa alle Verhaltensweisen, die sich durch Einfachfehler ergeben) als Relationen über der Menge der System- und Zustandsvariablen sowie der Parameter repräsentiert. Abbildung 2-5a zeigt zwei abstrakte Verhaltensrelationen als Venn-Diagramme, wobei die zeitlichen Aspekte des Verhaltens ignoriert werden. Das Testen zur Diskriminierung zwischen Verhaltensmodi ist dann die Aufgabe, die Unterschiede zwischen diesen Mengen zu entdecken, also Tupel, die nicht in beiden Relationen enthalten sind. Genauer gesagt

geht es darum, eine geeignete Eingabe zu bestimmen, d.h. den Wert von beeinflußbaren Variablen zu fixieren, in Abbildung 2-5a als $var_i=val_{i,1}$ auf der vertikalen Achse. Geeignete Eingaben sind solche, die unter den verschiedenen Modellen (Relationen) möglichst unterschiedliche Beobachtungen hervorrufen. Die möglichen Beobachtungen, in Abbildung 2-5a als Variable v_{obs} auf der horizontalen Achse dargestellt, sind gegeben durch die Projektionen des Durchschnitts der Eingabe $var_i=val_{i,1}$ mit den jeweiligen Relationen auf die beobachtbaren Variablen, im Beispiel v_{obs}. Offenkundig ist hier $var_i=val_{i,1}$ eine schlechte Wahl, da die resultierenden Projektionen einander sehr stark überlappen, während $var_i=val_{i,2}$ mit Sicherheit $mode_1$ von $mode_2$ unterscheidet. Wenn Fehlerwahrscheinlichkeiten und/oder Wahrscheinlichkeitsverteilungen über den Relationen gegeben sind, kann die Minimierung der Entropie als Kriterium für die Auswahl des Tests mit dem größten Informationsgewinn genutzt werden (s. [Struss 1994b])

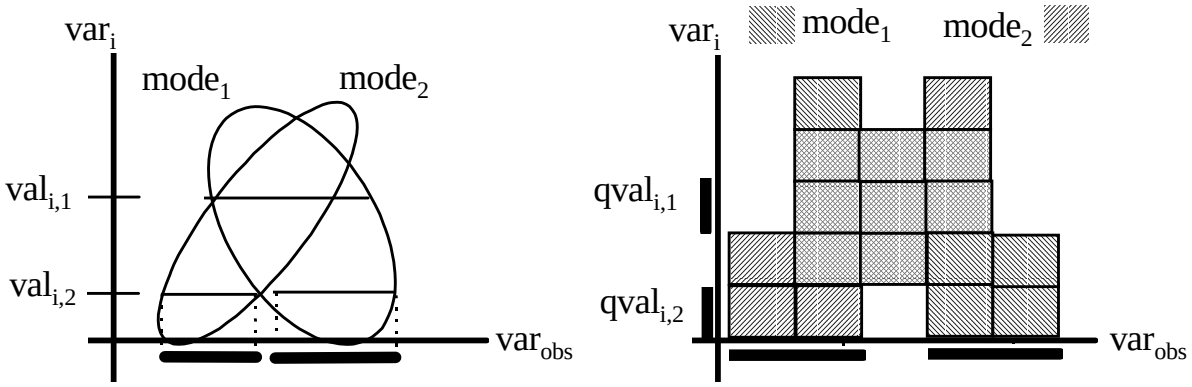


Abbildung 2-5: Zwei durch Relationen repräsentierte Verhaltensmodi
a über kontinuierlichen Wertebereichen b in abstrahierter Form
Mögliche Eingabewerte für die Inputvariable var_i auf der vertikalen Achse, die zugehörigen möglichen Werte der beobachtbaren Variable var_{obs} als Balken an der horizontalen Achse

Offenkundig ist das Berechnen der jeweiligen Projektionen für alle möglichen Inputs sehr aufwendig, wenn nicht unmöglich. Es wird aber ermöglicht, indem wir qualitative anstelle von numerischen Modellen verwenden, wie in . Abbildung 2-5b symbolisiert durch die Darstellung der Verhaltensrelationen als endliche Menge von Rechtecken (qualitativen Werten). Unter der Voraussetzung, daß die qualitativen Verhaltensrelationen die feinkörnigeren vollständig überdecken, ist es möglich, die oben genannten Berechnungen nunmehr auf endlichen Mengen von Werte-Tupeln durchzuführen. Außerdem wird durch diesen Schritt die Menge der Fehler endlich, da diese qualitative Abstraktion Mengen kontinuierlich variierender Fehler zu einer Beschreibung zusammenfaßt (z.B. unterschiedlich hohe Abweichung eines Widerstands).

Übertragen auf elektrische Schaltungen bedeutet dies beispielsweise, daß qualitative Modelle nicht exakte Strom- und Spannungswerte in Relation setzen, sondern den Wertebereich für Stromvariablen auf „negativ“(neg), „null“(zero) und „positiv“(pos) reduzieren. Entsprechend sind für den Wertebereich von Spannungsgrößen „Masse“(gnd) und „Batterie“(bat) zwei wichtige Größen, ein dritter Bereich umfaßt alle Werte dazwischen (btw).

Mit diesen Wertebereichen ergibt sich das in Tabelle 2-1 links dargestellte Modell für das korrekte Verhalten eine Temperaturfühlers. Das fehlerhafte Verhalten eines offenen Temperaturfühlers ist durch den Strom „null“ für alle Inputs gegeben (Tabelle 1, rechts). Die Inputs, die (potentiell) verschiedene Beobachtungen über den Strom produzieren, sind direkt abzulesen. Durch die qualitative Abstraktion gelten diese Modelle sogar für alle elektrischen Widerstände.

Auf dieser Grundlage wurden in vorangegangenen Arbeiten Testgenerierungsalgorithmen implementiert und auf Relais-Schaltungen angewendet, ein Beispiel aus einer Anwendung zur Programmierung von Testautomaten für Weichenschaltungen ([Inderst et al. 1995]). Das System produzierte beweisbar korrekte Mengen von Tests und identifizierte Fehlerpaare, die unter den gegebenen Voraussetzungen nicht unterscheidbar waren, für Schaltungen mit ca. 50 Komponenten. Dies liegt bereits jenseits dessen, was menschliche Programmierer garantiert erschöpfend behandeln können. Dennoch war der implementierte Prototyp noch relativ weit von einem tatsächlich einsetzbaren Werkzeug entfernt. Dies liegt an zwei grundlegenden Beschränkungen.

Zum einen betrachtet die skizzierte Lösung Testgenerierung als sehr abstrakte Aufgabe und ignoriert wesentliche praktische Aspekte des Testens, z.B. benötigte Ausrüstung und Kosten, Anordnung der Tests, um die Kosten

zu reduzieren, die Tatsache, daß einige Tests riskant oder unmöglich sind, solange nicht bestimmte Fehler ausgeschlossen sind.

Zweitens kann jedes modellbasierte System zur Testgenerierung, selbst wenn es die genannten Aspekte beinhaltet, nur ein wirksames Werkzeug werden, wenn es den tatsächlichen Arbeitsprozeß der Erzeugung und Anwendung von Testplänen berücksichtigt. In unserem Anwendungsbereich geschieht dies in den Kundendienstabteilungen. Hier werden mit relativ großem Personalaufwand sogenannte Fehlersuchanleitungen geschrieben, geprüft, angepaßt, aktualisiert und übersetzt, die dann, in unserem Fall auf CD-ROM, an die Kundendienstwerkstätten zur Anleitung der Diagnose und Reparatur ausgeliefert werden. Den Kern dieser Dokumente bilden natürlich-sprachlich beschriebene Pläne für die Durchführung von Werkstattests mit dem Ziel der Fehlerlokalisierung.

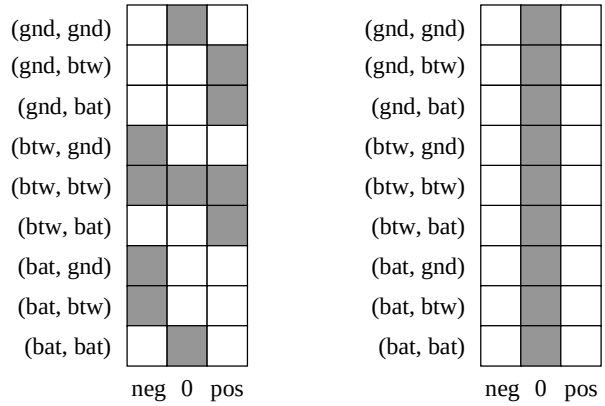


Tabelle 2-1: Qualitatives Modell des korrekten (links) und offenen Temperaturfühlers (rechts). Vertikal sind die Paare der Potentiale links und rechts aufgetragen, horizontal der Strom

Damit ist ein natürlicher Weg der Einführung modellbasierter Testgenerierung, die Autoren in den zentralen Kundendienstabteilungen bei der Erstellung von Fehlersuchanleitungen mit geeigneten Werkzeugen zu unterstützen. Wir haben dies in INDIA begonnen, indem wir einen neuen Typ von Autorensystem, genannt Genesis (Generation of Electronic Service Information Systems) spezifiziert und implementiert haben. Wir beschreiben im folgenden kurz dieses System und kommen dann auf die Nutzung modellbasierter Testgenerierung in seinem Kontext zurück.

2.4.2 Genesis - ein Autorensystem für Fehlersuchanleitungen

Genesis zielt auf die Integration automatischer, modellbasierter Testgenerierung in ein Autorensystem. Es kombiniert die Wiederverwendung von Modellen mit der verbesserten Wiederverwendung von Texten und bietet eine Ebene der Wissensrepräsentation für Daten und Informationen über die Fahrzeugsysteme.

Im Projekt wurde zunächst der gegenwärtige Arbeitsprozeß der Autoren von Fehlersuchanleitungen systematisch untersucht und formal repräsentiert, um eine Basis für die Kopplung mit der Testgenerierung und Reparaturplanung zu schaffen. Traditionelle Diagnose-Autorensysteme sind im wesentlichen Textverarbeitungssysteme, eventuell mit einer Datenbasis von Textbausteinen, die zudem Teile des weiteren Arbeitsprozesses unterstützen, wie etwa Freigabe und Übersetzung von Dokumenten. Gewöhnlich wird die vorhandene relativ starke Strukturierung des Inhalts der Diagnoseanleitungen nicht besonders ausgenutzt, und aufgrund der resultierenden reinen Textsicht gibt es keine Unterstützung für die Überprüfung und Aufrechterhaltung der Konsistenz der Anleitungen. Ferner ist es auf dieser Basis nicht möglich, semantisch annotierte Dokumentformate, z.B. sog. mark-up languages wie SGML oder XML zu erzeugen. Dies gerät zunehmend in Konflikt mit aktuellen Anforderungen wie dem in SGML formulierten Standard J2008 für Diagnosedokumente und der Präsentation der Manuale im Internet (vgl. www.w3c.org, www.sae.org).

In INDIA verfolgten wir das Ziel, die konventionellen, textorientierten Autorensysteme in drei Stufen zu verbessern und zu erweitern:

1. durch die explizite und detaillierte Strukturierung der Fehlersuchanleitungen und die semantische Annotation von Textelementen (Satzfragmenten, Phrasen, Dokumentabschnitten),

2. durch die Bereitstellung einer Ebene der Wissensrepräsentation, die es dem System ermöglicht, die Beziehung zwischen Textelementen der Fehlersuchanleitung und Struktur und Komponenten des jeweiligen Fahrzeugsubsystems herzustellen,
3. durch die Verbindung dieser Strukturdarstellung mit einer Bibliothek von Verhaltensmodellen als Basis für Anbindung und Ausnutzung modellbasierter Techniken wie etwa der Testgenerierung.

Die erste Stufe bringt bereits den gravierenden Vorteil der wesentlich weiter reichenden Wiederverwendung von Textfragmenten in verschiedenen Fehlersuchanleitungen. Dies bewirkt auch eine drastische Reduktion des Übersetzungsaufwands, der einen wesentlichen Kostenfaktor in der Bereitstellung der Fehlersuchanleitungen darstellt. Es gibt hunderte von Fahrzeugvarianten, während die Fehlersuche normalerweise aus Sequenzen von einigen wenigen stereotypen Tests und Aktionen besteht. Daher tauchen in den Anleitungen vielfache Wiederholungen bestimmter Phrasen, mit leichten Variationen, auf. Dieser Tatsache wird in Genesis Rechnung getragen, indem der Text der Fehlersuchanleitungen bis auf die Ebene kleiner kohärenter Phrasen gegliedert wird (die auch individuell übersetzt werden können). Eine derart strukturierte Fehlersuchanleitung kann damit automatisch übersetzt werden, sofern sie keine neuen Textfragmente einführt. Ein Seiteneffekt ist, daß die Manuale einheitlicher werden, selbst wenn sie von verschiedenen Autoren geschrieben werden, da sie sich auf dasselbe Repertoire von Formulierungen stützen. Die Textfragmente können gemäß ihrer unterschiedlichen Rolle in der Anleitung organisiert werden, was ihr Retrieval unterstützt. Z.B. kann ein Autor leicht zu den verfügbaren Varianten für die Formulierung von Spannungsmessungen geführt werden. Darüber hinaus ermöglicht die semantische Annotation der Textelemente die automatische Transformation in Dokumentformate wie SGML oder XML.

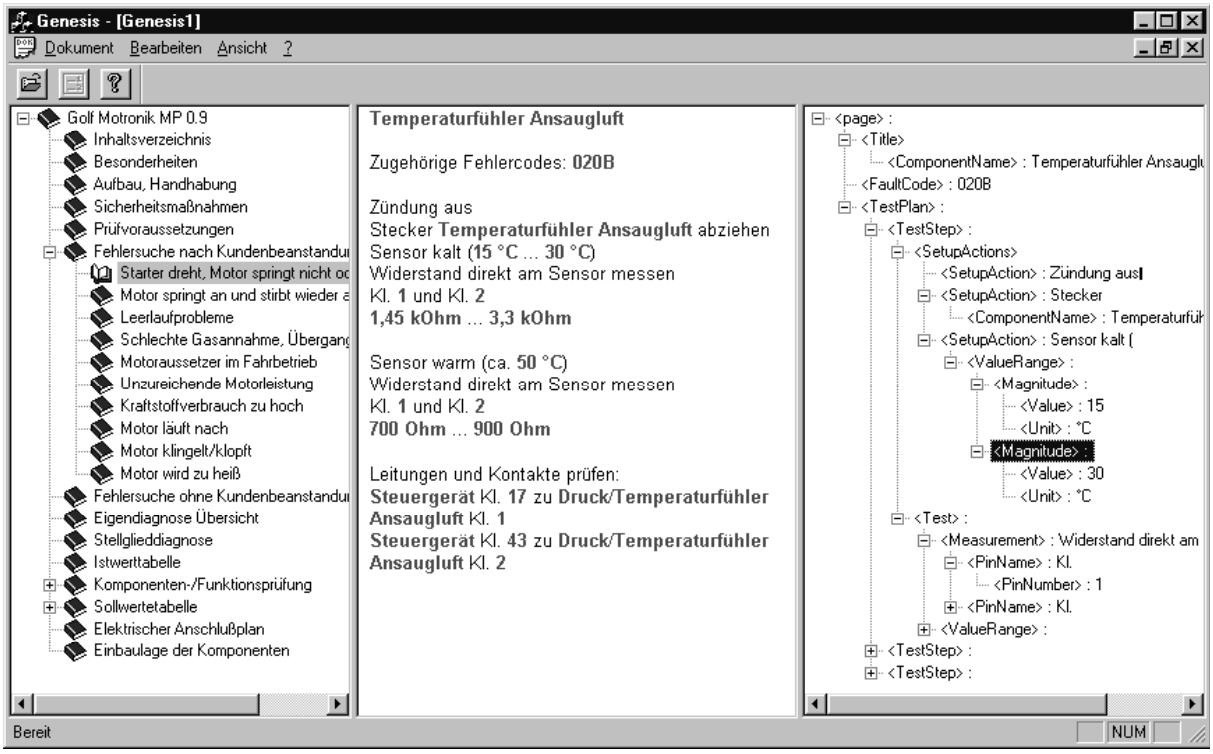


Abbildung 2-6: Bildschirmabzug des Autorensystems Genesis, in den Fenstern: links: Kapitelstruktur der Fehlersuchanleitung, Mitte: bearbeiteter Abschnitt (hier: Prüfplan), rechts: Struktur dieses Anschnitts, nur partiell entfaltet

Abbildung 2-6 zeigt einen Bildschirmabzug dieser ersten Stufe von Genesis. Im linken Fenster wird die Struktur der Fehlersuchanleitung (hier für eine Dieseleinspritzung) dargestellt, mit deren Hilfe der Autor durch die Kapitel und Seiten des Dokuments navigieren kann. Nach Auswahl einer Seite wird deren Inhalt im mittleren Fenster gezeigt. In diesem Fenster kann der Text editiert werden. Dabei sind Satzteile, die sich auf Eigenschaften von Fahrzeugkomponenten beziehen (z.B. „Temperaturfühler Kühlmittel“), farbig hervorgehoben, die Komponenten sind grün, Parameter rot. Die dritte Sicht (im Fenster rechts) zeigt die hierarchische Strukturierung der Seite in Textfragmente. Sie illustriert, wie solche Fragmente selbst aus einfacheren aufgebaut sein können. Z.B. setzt

sich ein *TestStep* aus einer Liste von *SetupActions* und einem *Test* zusammen. Der Baum Abbildung 2-6 zeigt die Seite nicht völlig entfaltet.

In der zweiten Phase von Genesis wurde diese semantische Unterlegung der Textfragmente mit einer modellbasierten Repräsentation des Fahrzeugs und seiner Subsysteme verbunden. Dieses Fahrzeugmodell besteht aus einer hierarchischen Darstellung der Komponenten und ihrer Beziehungen („Teil von“), einer Beschreibung der Gerätetopologie (Strukturdarstellung unter Benutzung der jeweiligen Komponenten) und Datenblättern, die die entsprechenden Parameter enthalten. Die Elemente dieser Darstellung, also z.B. Komponenten, Klemmen, Parameter, werden mit den entsprechenden Textfragmenten verbunden. Der Hauptvorteil liegt darin, daß der Autor in dieser Darstellung die Beschreibung des Subsystems ändern kann, ohne im einzelnen die textuelle Fehlersuchanleitung editieren zu müssen. Z.B. erscheinen im Datenblatt geänderte Parameterwerte oder in der Grafik geänderte Klemmenbezeichnungen unter allen anderen Sichten, also auch in der Textsicht und der Struktursicht auf die Seiten. Grundsätzlich (aber in Genesis nicht realisiert) besteht die Möglichkeit, elektronisch vorliegende Fahrzeugdaten aus anderen (etwa CAD-)Systemen zu importieren. Umgekehrt führen Eingaben, die etwa im Text vorgenommen wurden, auch zu den entsprechenden Änderungen in der Systemdarstellung.

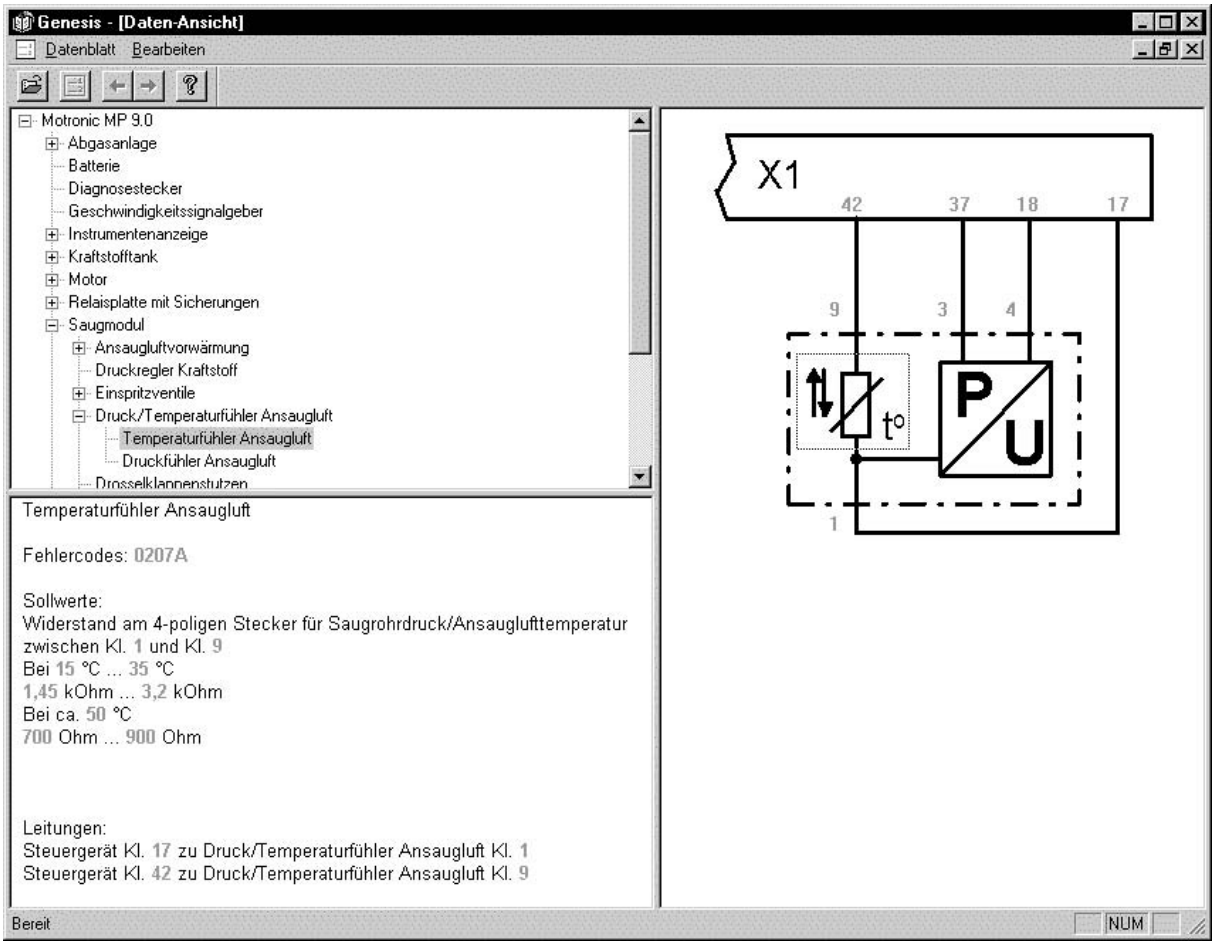


Abbildung 2-7: Bildschirmabzug: zusätzliche Views der zweiten Stufe. Links oben: Systemhierarchie, rechts: Struktur mit Eingabemöglichkeiten, links unten: Datenblatt einer Komponente

Abbildung 2-7 zeigt die zusätzlichen Sichten der zweiten Stufe. Das obere linke Fenster enthält die hierarchische Systemdarstellung. Darunter wird das Datenblatt einer ausgewählten Komponente bzw. Funktionsgruppe gezeigt, hier des Temperatursensors für die Ansaugluft. Im rechten Fenster wird der Schaltplan dieser Funktionsgruppe dargestellt. In jeder dieser Sichten kann der Autor Komponenten, Klemmen oder Parameter auswählen und die zugehörige Information editieren.

Die explizite und eindeutige Repräsentation der Information, die variieren kann, stellt automatisch die Konsistenz des Dokuments sicher und entbindet den Autor von der Notwendigkeit, gleichartige Anpassungen bei allen betroffenen Textteilen selbst vornehmen zu müssen. Etwaige Auswirkungen auf die Struktur des Prüfplans selbst können auf dieser Stufe natürlich nicht erkannt oder gar automatisch vorgenommen werden.

Dies wird in der dritten und letzten Stufe von Genesis auf der Grundlage modellbasierter Techniken möglich. In der zweiten Stufe werden durch die explizite Repräsentation der Struktur und physikalischen Eigenschaften des Systems die Grundlage dafür gelegt. Stufe 3 nutzt sie im Zusammenhang mit Bibliotheken von Verhaltensmodellen für die beteiligten Komponententypen, wie sie auch für die modellbasierte Unterstützung von FMEA und Diagnose verwendet werden. Damit werden die oben geschilderten Techniken der Testgenerierung im Kontext des Autorensystems nutzbar. Und da in ihm auch die Prüfpläne der Fehlersuchanleitung formal repräsentiert sind, wird es auch möglich, die Resultate der automatischen Testgenerierung zu importieren und als Text zu präsentieren.

Damit ist ein Weg realisiert, modellbasierte Testgenerierung ohne gravierende Änderungen im Arbeitsprozeß und in der Sicht der Autoren nutzbar zu machen, und somit eines der oben genannten Hindernisse für den Transfer dieser Technologie ausgeräumt. Hinzutreten muß aber die Erweiterung der geschilderten abstrakten Testgenerierung um die Behandlung der praktischen Dimension. Dies soll im folgenden skizziert werden.

2.4.3 Modellbasierte Erzeugung von Prüfplänen für Genesis

Abbildung 2-8 zeigt ein typisches Beispiel einer Funktionsgruppe, wie sie in der Prüfanleitung eines Motorsteuergeräts behandelt wird.

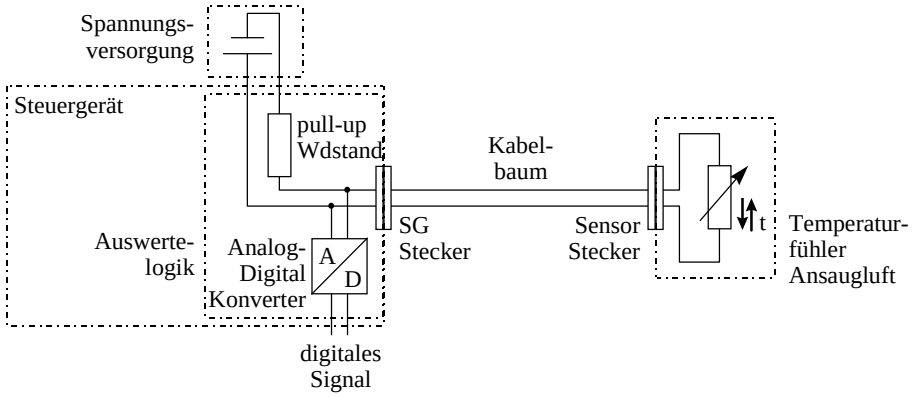


Abbildung 2-8: Funktionsgruppe „Temperaturfühler Ansaugluft“ bestehend aus Steuergerät mit Spannungsversorgung, Steckern und Sensor

Dargestellt ist der Temperaturfühler für die Ansaugluft mit seinen Steckern und Verbindungen sowie die Auswertung des analogen Sensorsignals im Motorsteuergerät. Aus der Sicht der Werkstattdiagnose ist es angemessen, zwischen den Fehlern von kleinsten austauschbaren Einheiten zu unterscheiden. Dies wäre in dieser Baugruppe ein Steuergerätefehler, ein Fehler auf der zwei-adrigen Verbindungsleitung oder ein Sensordefekt (die Steckerkontakte werden immer gemeinsam mit den Komponenten, an denen sie hängen, ausgetauscht).Typische Komponentenfehler dieser Schaltung sind eine zu niedrige Versorgungsspannung, wie auch Kurzschlüsse und Unterbrechungen des Temperaturfühlers, der Stecker und Leitungen (Kabelbaum).

Durch die Anforderung, daß die Prüfanleitung mit in der Werkstatt verfügbaren Mitteln abgearbeitet werden soll, ergeben sich Möglichkeiten und Einschränkungen der verwendbaren Beeinflussungen und Messungen: So gilt beispielsweise für den Temperaturfühler die im Steuergerät ermittelte Spannung bzw. der damit verbundene Temperaturwert als beobachtbar, da er mit Hilfe eines speziellen Testgerätes ausgelesen werden kann. Ebenso kann man ein Vielfachmeßgerät anschließen und damit Spannungen oder Widerstandswerte bestimmen. Änderungen, die an der Baugruppe vorgenommen werden können, sind im wesentlichen das Öffnen und Schließen von Steckverbindungen. An offenen Steckern kann alternativ das Vielfachmeßgerät angeschlossen werden, welches man zwischen den Betriebsmodi „Spannung messen“ und „Widerstand messen“ umschalten kann.

Diese Zusammenhänge und Informationen sind natürlich für die eigentlichen Handlungsanweisungen zur praktischen Durchführung der Fehlersuche und Reparatur das Wesentliche und legen auch Voraussetzungen, Kosten, Aufwand und damit die Reihenfolge der einzelnen Schritte fest. Z.B. ist das Ablesen des Istwertes des Sensors am Steuergerät billig, vorausgesetzt, der Tester ist bereits angeschlossen. In dem in Abschnitt 2.4.1 skizzierten Verfahren zur automatischen Testgenerierung wurde von solchen praktischen Bedingungen völlig abstrahiert. Sie liefert auf Basis des Verhaltensmodells lediglich eine abstrakte Charakterisierung von Tests. Dies ist unter Gesichtspunkten der Wiederverwendung von Software und Ergebnissen durchaus erwünscht: Wenn sich nur die praktischen Randbedingungen (wie praktischer Aufwand von Messungen, verfügbare Geräte etc.), aber nicht

die Struktur des Testgegenstandes ändern, bleiben die prinzipiell hilfreichen Tests dieselben, und lediglich ihre Auswahl oder Reihenfolge ist zu modifizieren. Im betrachteten Anwendungsbereich sind z.B. der Aufbau der Funktionsgruppen und ihre Funktionalität relativ stabil, während ihre Einbaulage und damit Zugänglichkeit und Meßmöglichkeiten abhängig vom Hersteller- oder Fahrzeugtyp sind.

Daher wurde die sehr generische Testgenerierungskomponente (im folgenden als Generator für abstrakte Tests, abstract test generator, bezeichnet) bewahrt, statt sie um die Darstellung und Berücksichtigung der praktischen, handlungsbezogenen und Kostengesichtspunkte zu erweitern, die ihrer Natur nach sehr anwendungsspezifisch sein können. Die Verarbeitung dieses Wissens zur Planung konkreter Prüfabläufe wurde in einer gesonderten Komponente, dem Aktionsplaner (action planner) realisiert. Der Einsatz dieser beiden Komponenten wird einer weiteren Komponente, dem Strategen („strategist“) kontrolliert und koordiniert (vgl. Abbildung 2-9). Die Funktion der einzelnen Komponenten und ihre Interaktion werden im folgenden anhand der Schritte in einem Zyklus der Prüfplangenerierung diskutiert. Dies ist beispielhaft zu verstehen, denn die Architektur soll es gerade ermöglichen, verschiedene Strategien unter Verwendung der Basiskomponenten zu realisieren.

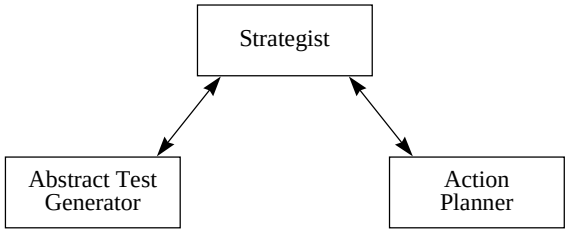


Abbildung 2-9: Die drei Komponenten der Prüfplangenerierung

Der Strategе

Diese Komponente ist Ausgangspunkt und Koordinator des gesamten Prozesses der Testgenerierung und Prüfplanung. Die Beschreibung eines Prüfplanungsproblemes als Input umfaßt, wie in Abbildung 2-10 dargestellt, zumindest

- ein Verhaltensmodell des Geräts („device model“)
- eine Beschreibung der grundsätzlich verfügbaren Aktionen und Beobachtungsmöglichkeiten („actions and observations“), sowie
- zugehörige „pragmatische“ Information über deren Kosten, Vorbedingungen etc. („costs and pragmatics“), also z.B., daß der Anschluß des Meßgeräts das Öffnen der Steckverbindungen voraussetzt und einen gewissen Zeitbedarf aufweist.
- eine Beschreibung des jeweiligen aktuellen Zustandes („current situation“).

Die Charakterisierung des Zustands wiederum umfaßt drei Aspekte:

- den Systemzustand in engeren Sinne, wie er im Verhaltensmodell ausgedrückt ist, eine (möglicherweise partielle) Beschreibung von Zustandsvariablen: Schalterstellungen, etwa „Zündung ein“ repräsentierend, „Motortemperatur betriebswarm“ etc.,
- den Zustand des physikalischen Systems als Resultat vorangegangener Aktionen, insbesondere im Hinblick auf Demontage, Anschluß von Meßgeräten etc., und
- den Zustand des Diagnoseprozesses, ausgedrückt als Menge von verbleibenden Fehlerhypothesen, die es noch zu diskriminieren gilt, ggf. mit Information über deren aktuelle Wahrscheinlichkeiten.

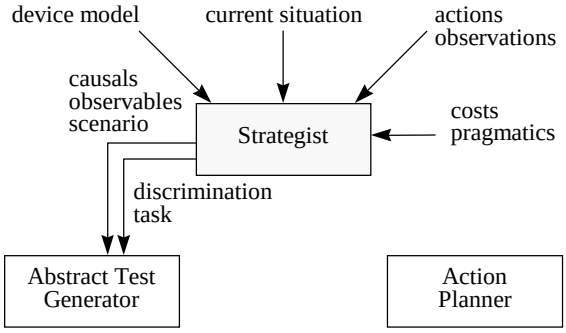


Abbildung 2-10: Input des Strategen und Aufruf des abstrakten Testgenerators

Der Stratege erstellt aus der aktuellen Situation eine (abstrakte) Diskriminierungsaufgabe, also z.B. daß der nächste Test zur Diskriminierung aller gegenwärtigen Fehlerhypothesen (oder einer Auswahl daraus) erzeugt werden soll, falls ein solcher aufgrund der gegebenen Möglichkeiten der Beeinflussung und Beobachtung überhaupt existiert. Der Stratege übergibt diese Aufgabe an den Generator für abstrakte Tests. Daneben legt er fest, welche Variablen dieser als Inputvariable („causals“) oder als beobachtbar („observables“) behandeln soll. Ferner kann der Stratege ein Szenario als Fokus festlegen, d.h. eine Menge von Systemzuständen (im engeren Sinne). Damit kann zum Beispiel kontrolliert werden, daß vorgeschlagene Tests nur beschränkte oder gar keine Veränderung des gegenwärtigen Zustands vornehmen sollen, bestimmte kritische Zustände vermieden werden u.s.w. Dies erscheint einfach als Einschränkung des Verhaltensmodells durch eine weitere Relation.

Der Generator für abstrakte Tests

Dieser im Abschnitt 2.4.1 dargestellte Modul verarbeitet die Diskriminierungsaufgabe (Abbildung 2-11) und bedient sich dabei des Gerätemodells und zusätzlich der Information über Wahrscheinlichkeiten (über Fehlerhypothesen und Variablenwerte) im aktuellen Zustand. Das Ergebnis der Testgenerierung ist eine Menge von abstrakten Tests. Jeder abstrakte Test löst zumindest eine Teilaufgabe des Diskriminierungsproblems. Wie betont, beinhaltet diese Darstellung nur die Information, **was grundsätzlich** zu tun ist, d.h. welche Variablenwerte eingestellt und welche Variablen beobachtet werden sollen, jedoch nicht, **wie** dies **in der Realität** geschehen kann. Der Testgenerator kann bereits ableiten, welche Diskriminierung wie gut durch die einzelnen Tests vorgenommen wird („discriminative power“, repräsentiert als diskriminierte Mengen von Fehlerhypothesen und Entropieänderung), jedoch noch nicht, wie teuer die Durchführung des Tests wird.

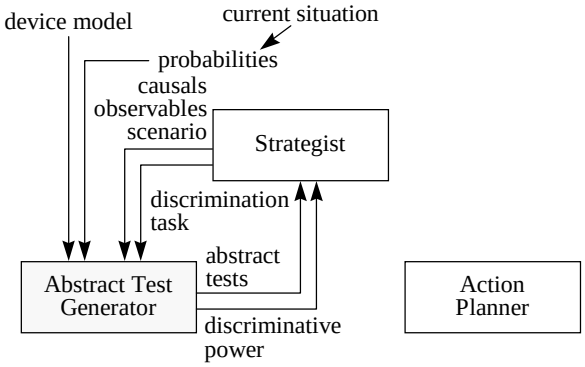


Abbildung 2-11: Input und Output des Generators für abstrakte Tests

Der **Strategе** erhält die generierten Vorschläge für Tests und deren diskriminierende Wirkung und wählt aus diesen nach statistischen oder heuristischen Kriterien eine Teilmenge aus. Sollten keine Tests möglich sein, so muß er, solange dies möglich ist, unter Variation der Diskriminierungsaufgabe, der Menge der beobachtbaren und Input-Variablen und/oder des Focus den Testgenerator erneut aufrufen. Die ausgewählten abstrakten Tests leitet er an die Planungskomponente weiter (Abbildung 2-12).

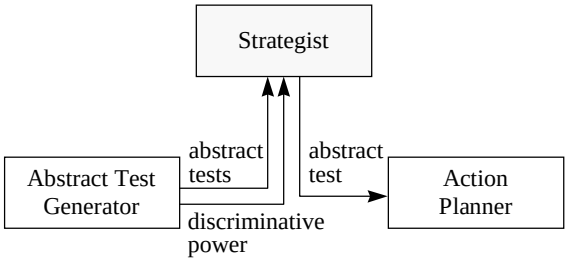


Abbildung 2-12: Der Stratege als Brücke zwischen den möglichen abstrakten Tests und deren praktischer Realisierung

Die Planungskomponente für Aktionsfolgen

Die Komponente hat die Aufgabe, einen abstrakten Test in einen konkreten Test, d.h. eine Sequenz von Handlungen umzusetzen, falls dies möglich ist (Abbildung 2-13). Die erstellte Handlungsfolge muß also zuerst die durch den abstrakten Test bestimmten Variablenwerte ausgehend vom aktuellen Zustand des Systems (einschließlich etwa der vorgenommenen Zerlegung) einstellen und dann entsprechende Beobachtungsaktionen durchführen. Die dafür zur Verfügung stehenden Handlungen und Beobachtungen und deren Kosten sind ein weiterer Input. Die Handlungsfolge sollte natürlich möglichst geringe Kosten verursachen.

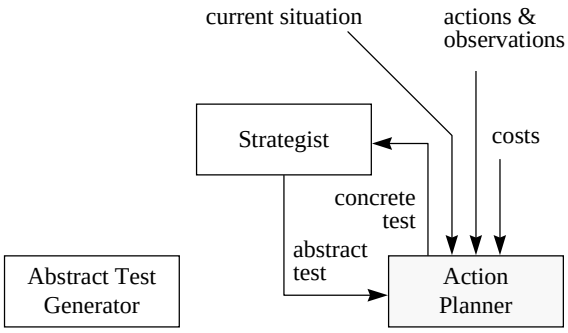


Abbildung 2-13: Input und Output des Aktionsplaners

Aktionen werden jeweils durch ihre Vorbedingungen und Nachbedingungen beschrieben. Beobachtungen sind charakterisiert durch die Variablen, deren Werte ermittelt werden und können ebenfalls Vorbedingungen enthalten. In obigen Beispiel sind alle Aktionen recht einfach gestaltet. Lediglich das Anschließen des Vielfachmeßgerätes unterliegt der Bedingung, daß Widerstände nur gemessen werden dürfen, wenn die entsprechenden Meßpunkte garantiert potentialfrei sind (d.h. keine Verbindung zur Spannungsversorgung besitzen). Andere Aktionen, wie z.B. das Zerlegen des Motors, haben komplizierte Vor- und Nachbedingungen. Im Rahmen der Arbeiten an Genesis wurde kein allgemeiner, ausgefeilter Planer entwickelt, was aber auch durch die betrachteten Prüfpläne nicht erfordert wurde. Durch den modularen Aufbau sind hier mächtigere Planungskomponenten integrierbar.

Der **Stratege** hat nun wieder die Möglichkeit, aus den von der Planungskomponente erstellten Handlungsfolgen eine auszuwählen. Bei dieser Wahl berücksichtigt er vor allem das Verhältnis von Kosten der Handlungsfolge und zu erwartendem Gewinn der Testdurchführung. Wurde kein konkreter Test als möglich zurück geliefert oder erfüllt keiner der Tests gewisse (zumeist kostenorientierte) Kriterien, so kann der Stratege weitere abstrakte Tests an den Planer übergeben oder auch alternativ Reparaturschritte einleiten (komplexere Reparaturanweisungen können dabei wieder von der Planungskomponente erstellt werden, was aber in der gegenwärtigen Implementierung nicht realisiert und hier nicht bildlich dargestellt ist).

Die vom Strategen ausgewählten Handlungen erweitern den Prüfplan, der beim ersten Durchlaufen des Zyklus nur aus der Wurzel besteht, die die anfängliche Diskriminierungsaufgabe repräsentiert. Durch Anwendung der modellbasierten Verhaltensvorhersage kann der Stratege bestimmen lassen, wie sich die Durchführung der Handlung auf den Zustand des Gerätes auswirkt (Abbildung 2-14). Je nach Beobachtungsergebnis der Tests werden dabei unterschiedliche Zustände erreicht. Jeder neue Zustand bestimmt für sich genommen wieder den

Ausgangspunkt für einen neuen Durchgang der Testgenerierung. Dabei muß für die sich neu ergebenden Szenarien nicht wieder der ganze Zyklus durchlaufen werden. Die abstrakten Tests behalten im wesentlichen ihre Gültigkeit, lediglich die Wahrscheinlichkeiten, und damit ihre Diskriminierungskraft, verändert sich. Der Stratege bricht ab, wenn alle erreichbaren gewünschten Unterscheidungen durchgeführt und die entsprechenden Reparaturschritte dem Prüfplan hinzugefügt wurden.

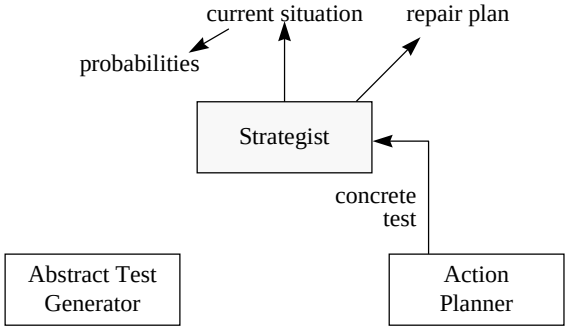


Abbildung 2-14: Der Stratege erweitert den Plan und bestimmt die aus diesem Schritt resultierenden Situationen

Wie erwähnt, sind andere Vorgehensweisen durch Austausch der Strategiekomponente realisierbar: Z.B. könnte diese jeweils vom Planer den mit den geringsten Kosten zu erreichenden nächsten Zerlegungszustand erzeugen lassen, um anschließend alle abstrakten Tests, die in diesem Zustand möglich sind, zu generieren und zu planen. Diese Flexibilität bei der Berücksichtigung anwendungsspezifischer Kostengesichtspunkte und Prüfstrategien war ein wesentliches Kriterium für die gewählte Architektur und die Entscheidung gegen die Verwendung von herkömmlichen Algorithmen zur Erzeugung von Entscheidungs- oder Klassifikationsbäumen.

Für die Anwendung des oben skizzierten Algorithmus auf das Beispiel des Temperaturfühlers an wurde zur Vereinfachung vorgegeben, daß der Stecker des Temperaturfühlers unzugänglich ist und sich somit nur der Stecker des Steuergeräts öffnen läßt. Unter diesen Voraussetzungen wird folgende Prüfplan generiert, der aus Gründen der Verständlichkeit z.T. verbalisiert wurde.

```
<TestPlan>
    <TestStep ID=1>
        <SetupAction> Schließe Steuergerätetester an Motorsteuergerät an
        <Test> Lese digitales Steuergerätesignal für Ansauglufttemperatur aus
            <TestResult NextStep=9> Signal liegt zwischen -20°C und +40°C
            <TestResult NextStep=2> Signal liegt unter -20°C
            <TestResult NextStep=5> Signal liegt über +40°C
    <TestStep ID=2>
        <SetupAction> Öffne Steuergeräte Stecker
        <SetupAction> Schalte Vielfachmeßgerät auf Spannungsmessung
        <SetupAction> Schließe Vielfachmeßgerät an Steuergerät an
        <Test> Lese digitales Steuergerätesignal für Ansauglufttemperatur aus
            <TestResult NextStep=3> Signal liegt zwischen -20°C und +40°C
            <TestResult NextStep=4> Signal liegt unter -20°C
    <TestStep ID=3>
        <RepairAction> Tausche Temperaturfühler mit Zuleitungen aus
    <TestStep ID=4>
        <RepairAction> Tausche Steuergerät aus
    <TestStep ID=5>
        <SetupAction> Öffne Steuergeräte Stecker
        <SetupAction> Schalte Vielfachmeßgerät auf Spannungsmessung
        <SetupAction> Schließe Vielfachmeßgerät an Steuergerät an
        <Test> Lese digitales Steuergerätesignal für Ansauglufttemperatur aus
            <TestResult NextStep=6> Signal liegt über +40°C
            <TestResult NextStep=8> Signal liegt zwischen -20°C und +40°C
    <TestStep ID=6>
        <RepairAction> Tausche Spannungsversorgung aus
        <Test> Lese digitales Steuergerätesignal für Ansauglufttemperatur aus
```

```
                    <TestResult NextStep=9> Signal liegt zwischen -20°C und +40°C
                    <TestResult NextStep=7> Signal liegt über +40°C
<TestStep ID=7>
        <RepairAction> Tausche Steuergerät aus
<TestStep ID=8>
        <RepairAction> Tausche Temperaturfühler mit Zuleitungen aus
<TestStep ID=9>
        <Comment> alles ok, kein Fehler gefunden
```

Dieser automatisch generierte Prüfplan beginnt, wie auch der von Menschen erstellte, mit dem Auslesen von Istwerten aus dem Steuergerät. Dieser Prüfschritt wird deshalb bevorzugt, weil er günstig auszuführen ist (wenig Vorbereitungsaktionen) und bereits im ersten Prüfschritt die am häufigsten anzutreffende Situation, nämlich die, in der kein Fehler vorliegt, von den bei weitem unwahrscheinlicheren Fehlersituationen abgrenzt. In der Regel kann man also ein Prüfergebnis zwischen -20°C und +40°C erwarten, welches auf korrektes Verhalten der Temperaturerfassung hinweist. Nach diesem Schritt bricht der Prüfplan ab.

Eine erste Überraschung tritt im Fehlerfall auf. Zwar wird erwartungsgemäß vorgeschlagen, den Steuergerätestecker zu öffnen und das Vielfachmeßgerät als Spannungsmesser anzuschließen. Abgelesen wird dann allerdings nicht das Meßgerät selbst, sonder wiederum die Anzeige des Steuergerätetesters! Diese Wahl ist zufällig, denn das Ablesen beider Meßgeräte bringt im vorliegenden Fall den gleichen Nutzen für die Diskriminierung und ist mit den gleichen assoziierten Kosten modelliert (wäre das Ablesen des Vielfachmeßgerätes gegenüber dem Ablesen des Steuergerätetesters als billiger modelliert, so würde ersteres bevorzugt). Man kann den vorgeschlagenen, unkonventionellen Test so interpretieren, daß das Vielfachmeßgerät als Prüfwiderstand fungiert, der in Ersatz für den potentiellen Fehlerkandidat Temperaturfühler eingesetzt wird. Liegt der ermittelte Digitalwert im gültigen Bereich, so ist offensichtlich der Sensor defekt und wird ausgetauscht. Ist andererseits der ermittelte Digitalwert auch bei der zweiten Messung im unerlaubten Bereich, so sind das Steuergerät und die Spannungsversorgung potentielle Fehlerkandidaten, letztere aber nur bei zu niedriger Spannung oder entsprechend zu hohem ermittelten Digitalsignal, da die Spannungsversorgung im Fehlerfall gemäß Modell nur zu niedrige Spannung, nicht jedoch zu hohe Spannung liefern kann. Bei einem Digitalsignal über +40°C wird zuerst die Spannungsversorgung ausgetauscht, da diese zum einen eine höhere Fehlerwahrscheinlichkeit aufweist, zum anderen billiger zu ersetzen ist. Erst wenn der Fehler auch nach dieser Reparatur noch vorliegt, wird das Steuergerät ausgetauscht.

Wie der dargestellte Plan illustriert, erzeugt die automatische Prüfplangenerierung eine Struktur mit Markierungen, wie sie in der Repräsentation der textuellen Prüfpläne in der ersten Stufe von Genesis verwendet werden. Dies ist die Basis dafür, den generierten Handlungsplan automatisch ebenfalls in einen Text umzusetzen. Voraussetzung hierfür ist die Existenz von Textbausteinen für die verschiedenen Typen von Aktionen, Messungen und Entscheidungen, deren sich die Planungskomponente bedient. Dies ist ein überschaubarer (und in der Basis für den Planer aufgezählter) Vorrat. Die variierenden Textbestandteile (Soll- und Istwerte, Namen von Komponenten und Meßgeräten etc.) sind bereits in der ersten Stufe von Genesis als Parameter in die Textvorlagen integrierbar.

Damit ist in diesem Anwendungsbereich erfolgreich an einem Prototypen demonstrierbar, wie modellbasierte Lösungen mit geringfügigen Änderungen im gegenwärtigen Arbeitsprozeß nutzbar gemacht werden können.

2.5 Erkenntnisse

Unsere Arbeit im INDIA-Projekt verfolgte die Zielsetzung, den Transfer von Forschungsresultaten im Bereich modellbasierter Systeme in mehrere Anwendungen in der Fahrzeugindustrie einen wesentlichen Schritt vorwärts zu bringen. Auf dem Weg von der Demonstration prinzipieller Machbarkeit zum echten Einsatz sollten prototypische Lösungen für die Unterstützung vorhandener Arbeitsabläufe unter Berücksichtigung ihrer realistischen Randbedingungen erstellt werden.

Die geschilderten Prototypen haben dies (mit unterschiedlicher Reife) geleistet. An ihnen läßt sich das Potential der modellbasierten Technologie demonstrieren:

- Es ist mit den vorhandenen Mitteln möglich, einzelne **Arbeitsschritte wirksam zu unterstützen** und sogar zu automatisieren. Letzteres gilt insbesondere für den Anteil der Arbeit, der zwar Fachgebietswissen erfordert, aber dieses in klar strukturierter Weise anwendet, wie etwa die Bestimmung der Fehlerauswirkung für eine Menge von Fehlerursachen.

- Die Kombination von kompositionaler Modellierung auf der Basis von Modellbibliotheken und generischen Problemlösungskomponenten stellt eine wirksame **Antwort auf das Problem der Variantenvielfalt** dar.
- Darüber hinaus stellt der Ansatz die entsprechenden Arbeitsprozesse auf eine **klare methodische Grundlage**, vor allem durch die Prinzipien und Organisation der Modelle und die Trennung zwischen Gegenstandswissen und Problemlösungswissen.
- Dabei lassen sich computergestützte Lösungen durch **Wiederverwendung** sowohl von Modellen als auch von Softwarebausteinen z.T. sehr effizient produzieren.

Die demonstrierbaren Möglichkeiten der Technologie beziehen sich aber nicht nur auf die Unterstützung einzelner Arbeitsprozesse:

- Modellbibliotheken, aber auch die klaren Schnittstellen zwischen Softwarekomponenten stellen die Basis für eine **Integration von Arbeitsprozessen** und den elektronischen Austausch von Information und Ergebnissen von Arbeitsschritten dar. Die entwickelten Modellbibliotheken wurden, ebenso wie die Software-Kernkomponenten in allen drei Prototypen eingesetzt.
- Noch allgemeiner gesehen, sind die Erstellung und Pflege von Modellbibliotheken als ein Beitrag zum **Wissensmanagement** in technologie-orientierten Unternehmen zu sehen.

Natürlich und durchaus beabsichtigt hat die Arbeit auch Grenzen der gegenwärtigen Technologie und Aufgaben künftiger Forschungs- und Entwicklungsarbeiten aufgedeckt oder erhellt. Zu den wichtigsten dieser Aufgabenkomplexe gehören:

- Bessere **Unterstützung und Automatisierung der Modellierung**. Dies beinhaltet insbesondere die Nutzung der im Ingenieurbereich entwickelten Modelle und die Erzeugung von Modellen, die in Effizienz und Kompetenz für die jeweiligen Systeme und Aufgaben maßgeschneidert sind.
- Systematische **Charakterisierung der Voraussetzungen und Ergebnisse** der vorhandenen Problemlösungen, vor allem im Hinblick auf die Behandlung dynamischer Abläufe und die Verwendung unvollständiger Lösungsalgorithmen (s. die Ausführungen zur zustandsbasierten Diagnose).
- Verbesserte theoretische Grundlagen und technische Lösungen, die die komplexe Natur von Arbeitsprozessen als **Handlungs**abläufe widerspiegeln und deren praktische Randbedingungen und Kosten systematisch berücksichtigen.

Das Projekt hat gezeigt, daß trotz dieser notwendigen Forschungsarbeiten vorhandene Lösungen in die industrielle Praxis überführt werden können. Dies erfordert geeignete Strategien, um die Einführung in die Arbeitsprozesse und deren Umgestaltung behutsam aber wirksam vorzunehmen. Einige Ansätze zur Lösung dieses Problems werden durch die geschilderten Arbeiten aufgewiesen.

3 Autorensystem für Anleitungen zur Fehlersuche und Reparatur im Kraftfahrzeug: Anforderungsanalyse und Spezifikation

Andreas Malik und Harald Weiler – Robert Bosch GmbH

Ulrich Heller, Jakob Mauss und Peter Struss – TU München, Institut für Informatik

3.1 Einführung

Bei komplexen Systemen, wie sie in wachsendem Maße in Kraftfahrzeugen eingebaut werden, sind Anleitungen zur Fehlersuche unerläßlich, um in der Werkstatt gezielt Diagnose und Reparatur durchführen zu können. Im folgenden Beitrag wird ein Überblick über die wesentlichen Anforderungen und Randbedingungen dieses Anwendungsfeldes für „intelligente Diagnose“ auf Grund der Erfahrungen bei der Firma Robert Bosch GmbH (abgekürzt in Bosch) gegeben und wichtige Eigenschaften für ein Redaktionssystem zur Erstellung solcher Anleitungen zur Fehlersuche in der Werkstatt spezifiziert.

3.2 Fehlersuche am Kfz in der Werkstatt

3.2.1 Ausrüstung von Werkstätten

Bosch produziert für Kraftfahrzeugwerkstätten komplette Prüfausrüstungen. Sie werden vor allem an Bosch-Dienste, aber auch an freie Werkstattpartner vertrieben. Weltweit gibt es rund 10000 Bosch-Stützstellen, die sich jedoch in ihrem Ausrüstungsgrad deutlich unterscheiden. So besitzt nur knapp die Hälfte von ihnen die nötige Ausstattung für die kompetente Diagnose von Benzin-Einspritzsystemen.

Wichtige Werkzeuge sind mobile Diagnosetester (z.B. die Pocket-Tester vom Typ KTS 300) und Motortester (z.B. MOT 501). Sie werden in der Werkstatt an dafür vorgesehene Stecker des Fahrzeugs angeschlossen und ermöglichen dann die Messung von verschiedenen Signalen und Systemparametern. Mit ihrer Hilfe lassen sich Fahrzeugsysteme sowohl von Bosch als auch anderer Hersteller diagnostizieren.

Die speziellen Fahrzeugdaten, die solch ein Tester braucht, sind in der Werkstatt auf einem Rechner verfügbar. Für häufig vorkommende Fahrzeugtypen hält der Tester die Daten im Rahmen seiner Speichermöglichkeiten lokal, für andere Typen können sie vom PC über eine RS232-Schnittstelle auf den Tester geladen werden. Wenn Serviceabkommen abgeschlossen wurden, können die Daten regelmäßig aktualisiert werden. Dies geschieht heute über Austausch von CDs, in Zukunft durch direktes Abrufen über das Internet.

3.2.2 Fehlersuche in der Werkstatt

In der Werkstatt stehen zur Diagnose von Teilsystemen, z.B. eines Anti-Blockier-Systems ABS oder einer Elektronischen Diesel-Einspritzanlage EDC, eine fahrzeugspezifische Fehlersuchanleitung und in vielen Fällen auch eine ausführliche, vom Fahrzeug unabhängig formulierte Basisanleitung auf CD-ROM zur Verfügung. Eine solche Fehlersuchanleitung läßt sich grob in sechs verschiedene Bestandteile gliedern. Die folgenden Zahlenangaben beziehen sich zwar auf eine genauer analysierte FSA zu einer verteilerbasierten EDC, weichen allerdings bei ähnlichen Dokumenten nicht entscheidend davon ab.

- **Fehlercodetabelle**: 16 verschiedene Fehlercodes, die vom Steuergerät erzeugt werden, mit Verweisen auf 30 Prüfprogramme.
- **Fehlersuchplan**: ordnet jeder der 20 sog. Fahrerbeanstandungen eine sortierte Liste von 1 bis 20 Prüfprogrammen zu, die bei einer bestimmten Kundenbeanstandung abzuarbeiten sind.
- **Prüfprogramme Eigendiagnose**: 30 verschiedene Prüfprogramme, die aus der Fehlercodetabelle oder dem Fehlersuchplan heraus angesprungen werden. Diese Prüfprogramme zum ausgelesenen Fehlercode behandeln beinahe ausschließlich die elektrische Sensorik und Aktuatorik, da in der Regel nur diese vom Steuergerät überwacht werden kann und zum Setzen von Fehlercodes führt.
- **Fehlersuchprogramm**e: 10 verschiedene Prüfprogramme, die teils elektrische Sensorik und Aktuatorik behandeln, teils aber auch pneumatische, mechanische und hydraulische Funktionsgruppen.

- **Prüfschritte**: 16 detaillierte Beschreibungen von komplexen Prüfschritten, zu deren Durchführung viel technisches Spezialwissen benötigt wird. Beispiele: Kraftstoffanlage entlüften, Filterbox erneuern oder Einspritzanlage auf Dichtheit prüfen.
- **Sonstiges**: Einleitende Funktionsbeschreibung des Gesamtsystems, Sollwerte, verfügbares Spezialwerkzeug, Sicherheitshinweise, Einbaulage der Komponenten, Bilder hierzu.

Von den 209 A4-Seiten der analysierten Basisanleitungen machen die Prüfprogramme und Fehlersuchprogramme 150 Seiten, also 75% aus. Diese Prüfprogramme beziehen sich wiederum ganz überwiegend - zu über 75% - auf die Diagnose der elektrischen Sensorik und Aktuatorik der Steuergeräte.

3.2.3 Aufbau einer Fehlersuchanleitung

Für die Strukturierung einer FSA hat sich folgende Kapiteleinteilung bewährt und wurde daher in die Arbeitsanweisung für die Autoren aufgenommen (die unterstrichenen Teile sind für die Diagnose essentiell und erfordern den größten Vorbereitungsaufwand):

- Inhaltsverzeichnis,
- Besonderheiten,
- Aufbau/Handhabung,
- Sicherheitsmaßnahmen,
- Prüfvoraussetzungen,
- Fehlersuche nach Kundenbeanstandung (evtl. mit spezifischen Abschnitten),
- Fehlersuche ohne Kundenbeanstandung,
- Eigendiagnose mit den Abschnitten
  - Eigendiagnose-Beschreibung,
  - Eigendiagnose-Tester anschließen,
  - Eigendiagnose-Blinkcodes (optional),
  - Eigendiagnose-Identifikation,
  - Eigendiagnose-Fehlercodes,
  - Eigendiagnose-Fehlerspeicher löschen.
  - Istwertetabelle (optional),
  - z.B. Stellgliedtest-Auswahltabelle (optional).
- Istwerte,
- Komponenten / Funktionsprüfung,
- Sollwerte: Liste aller Sollwerte,
- Elektrischer Anschlußplan,
- Hydraulischer Anschlußplan (optional),
- Luft- und Kraftstoffleitungsplan (optional),
- Einbaulage der Komponenten.

3.3 Erstellungsprozeß einer Fehlersuchanleitung

3.3.1 Erstellung des Dokuments

In diesem Abschnitt wird das heutige Vorgehen (bei Bosch) bei der Erstellung von FSAs beschrieben. Folgende Schritte können unterschieden werden

- **Entwicklungsbeschluß**: Der zuständige Geschäftsbereich beschließt die Entwicklung einer FSA für ein Teilsystem. Da eine stärkere Nachfrage nach Serviceleistungen bei Bosch-Diensten erst nach Ablauf der Garantiezeit einsetzt, wurde bisher eine FSA zeitversetzt nach Serienanlauf entwickelt. Zunehmend werden FSAs vom Fahrzeughersteller zusammen mit dem zu entwickelnden Teilsystem in Auftrag gegeben, so daß beides zeitgleich entstehen muß.
- **Beauftragung eines Service-Mitarbeiters**: Ein Mitarbeiter wird mit der Anfertigung der FSA beauftragt; er ist gleichzeitig für die Erstellung der Prüfsoftware des Service-Testers verantwortlich. Der Mitarbeiter sollte elektrotechnische, Kfz-technische oder maschinenbautechnische Kenntnisse besitzen und mit einer Version des Subsystems vertraut sein.
- **Informationen und Dokumente beschaffen**: Der Mitarbeiter stellt die benötigten Dokumente zusammen. Die wichtigsten sind hierbei:
  - Stromlaufplan
  - Fehlersuchplan (Tabelle der Kundenbeanstandungen und zugehörigen verdächtigen Funktionsgruppen) für ein ähnliches Teilsystem, falls vorhanden
  - Informationen über das neue Teilsystem, speziell die Änderungen gegenüber dem Vorgängersystem
  - System FMEA (Fehler-Möglichkeits- und Einfluß-Analyse) für das Teilsystem
- **Service-Tester Spezifikation**: Ungefähr die Hälfte der Arbeitszeit für eine FSA entfällt auf die Spezifikation der Software des Service-Testers, eines Geräts zum Auslesen und Löschen von Fehlercodes, zum Lesen von Sensorwerten und zur Stellgliedansteuerung. Dieser Aufwand wird hauptsächlich durch den Aufwand zur Recherche der nötigen Detailinformationen (Steuergerätesoftware, Registeradressen, Kommunikationsprotokolle, Fehlercodes usw.) verursacht.
- **Fehlersuchplan erstellen**: Der Mitarbeiter erstellt einen Fehlersuchplan für das neue Teilsystem. Die berücksichtigten Kundenbeanstandungen können oft vom Vorgängersystem übernommen werden, da sich die Funktionalität und damit die möglichen Beanstandungen nur selten ändern. Neuerungen werden nötigenfalls durch Ergänzen von neuen Beanstandungen oder durch Differenzieren bekannter Beanstandungen berücksichtigt.
- **Fehlersuchplan priorisieren**: Der Mitarbeiter sortiert die möglichen Fehlerursachen einer Beanstandung so, daß sich in der Werkstatt eine günstige Prüfreihenfolge ergibt. Er benutzt dabei vor allem Wissen über die Zugänglichkeit der Bauteile am Fahrzeug. Das Priorisieren erfolgt meist direkt am Fahrzeug, um die Zugänglichkeit der Bauteile und günstige Reihenfolgen von Prüfschritten besser beurteilen zu können.
- **Prüfprogramme erstellen**: Für jede verdächtige Funktionsgruppe des Fehlersuchplans wird ein Prüfprogramm erstellt, das die Prüfung der Funktionsgruppe schrittweise beschreibt. Bei der Erstellung der Prüfprogramme wird die Vernetzung der Teilsysteme untereinander (z.B. Motor - Getriebe) oder einzelner Funktionsgruppen innerhalb eines Teilsystems aus Komplexitätsgründen nicht berücksichtigt. Die Teilsysteme und ihre Funktionsgruppen werden also geprüft, als seien sie voneinander isoliert. Der Mitarbeiter benutzt zur Anfertigung der Prüfprogramme:
  - Stromlaufplan
  - Setzbedingungen der Fehlercodes
  - Menge der denkbaren Tests für das Teilsystem und Wissen über vorhandene Prüfmittel.
- **CD Produktion**: Ein Spezialist organisiert die Bereitstellung der Dokumente auf CD.

Der Arbeitsaufwand bei der Erstellung einer FSA für ein neues Teilsystem (Teilsystem ohne vergleichbares Vorgängersystem), ohne CD Produktion aber einschließlich Spezifikation der Software für den Service-Tester, betrug bei der analysierten EDC etwa 40 Personentage. Bei einem modifizierten Teilsystem kann natürlich viel stärker auf bestehende FSAs zurückgegriffen werden.

3.3.2 Prüfprogramme für die Werkstattdiagnose

In diesem Abschnitt wird ein typisches Prüfprogramm beschrieben, das wieder der Anleitung zu der verteilerbasierten EDC entnommen ist.

Abbildung 3-1 zeigt einen Temperaturfühler, dessen temperaturabhängiger Widerstand (siehe Kennlinie) über zwei Leitungen des Kabelsatzes vom Steuergerät SG ermittelt wird. Die Spannung V_m an Klemme 23 wird vom Steuergerät überwacht. Bei zu hoher Spannung (also Unterbrechung oder Plusschluß) oder zu niedriger Spannung (also Masseschluß) wird im Steuergerät der Fehlercode 3 gesetzt. Der gestrichelt gezeichnete Drehzahlsensor n soll andeuten, daß die Masseversorgung des Temperaturfühlers auch von anderen Funktionsgruppen genutzt wird.

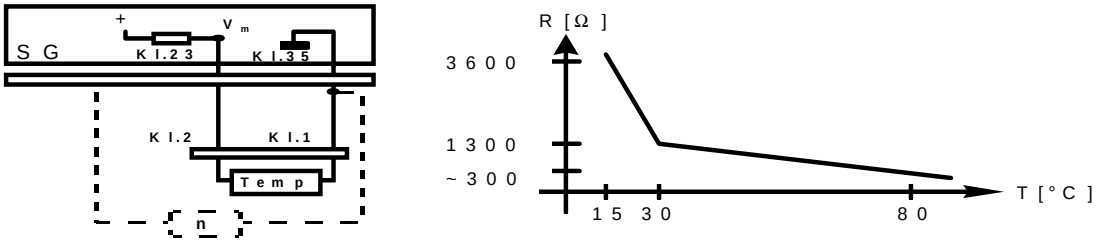


Abbildung 3-1: Funktionsgruppe Temperaturerfassung

In Abbildung 3-2 ist der entsprechende Ausschnitt aus der Serviceanleitung abgebildet. Die Funktionsgruppe zur Temperaturerfassung mag als Beispiel zu einfach erscheinen, ist aber eher typisch. 75% der Prüfprogramme in den untersuchten FSAs für ABS, EDC und Motronic befassen sich mit der Diagnose von solch einfachen Schaltkreisen der Steuergerätesensorik und -aktuatorik.

3.3.3 Das Autorensystem

Eine FSA wird heute in vier Arbeitsschritten mit dem Autorensystem IMSIS angelegt, einem für Bosch entwickelten spezialisierten Textverarbeitungsprogramm, das von einem PC bedient wird und auf einen zentralen „Datenbank-Server“ zugreifen kann.

- Anlegen der Kapitelstruktur: Die standardisierte Kapitelstruktur für Fehlersuchanleitungen wird als Vorlage kopiert.
- Ausfüllen der Kapitelstruktur: Hier muß der Autor entscheiden, ob er ganze Kapitel aus alten FSAs übernimmt und anpaßt, oder ob er nur einzelne Textbausteine kopiert und anpaßt. In beiden Fällen gehen alle Variablenwerte beim Kopieren verloren, d.h. Sprung- und Textvariablen müssen von neuem mit Werten belegt werden.
- Verifikation: Das Autorensystem bietet verschiedene automatische und halbautomatische Prüfungen an, z.B. die Prüfung, ob alle im Text verwendeten Variablen mit Werten belegt sind, ob also z.B. alle Textsprünge ein gültiges Ziel haben oder die Prüfung, ob jede Textstelle von mindestens einem Kapitel aus erreichbar ist.
- Erstellen der Austauschdateien: Das Autorensystem erzeugt für jede FSA einen Satz von Dateien (siehe Abbildung 3-3), auf deren Grundlage eine CD-ROM produziert wird.

Alternativ zu den ersten beiden Schritten kann auch eine komplette FSA als Vorlage kopiert werden, wobei alle Variablenwerte automatisch übernommen werden.

FSAs werden aus Textseiten zusammengesetzt, die einer Bildschirmseite entsprechen, d.h. jeder Satz gehört zu genau einem solchen „Textbaustein“. Das System verwaltet inzwischen weit über 100 000 Textseiten, die von jedem Autor zentral abgerufen werden können und wovon die meisten bereits in die benötigten Fremdsprachen übersetzt sind. Die Wiederverwendung der Textseiten in neuen FSAs spart also Übersetzungskosten. Jede Textseite kann nur über Schlüsselwörter gesucht werden, die in den ersten zwei der insgesamt 17 Textzeilen vorkommen. Es gibt also weder eine Volltextsuche, noch eine strukturierte Ablage der Textelemente, z.B. nach Texttyp (Prüfanweisung, Sicherheitshinweis, Reparaturanweisung, Funktionsbeschreibung usw.).

Leider verwendet in der Praxis jeder Autor meist nur die von ihm selbst angelegten Textseiten. Der Grund hierfür scheint einerseits die schwache Strukturierung der Datenbank zu sein, zum anderen arbeiten die Autoren in Teams zuständig für bestimmte Fahrzeugtypen zusammen, wobei die Fahrzeughersteller oft hauseigene Bezeichnungen verwenden, die dann oft von Bosch der Kohärenz wegen übernommen werden.

Das Redaktionssystem verwaltet für jede Fehlersuchanleitung eine Liste von Variablen, die vom Autor angelegt werden kann und in der er den Variablen Werte zuweist. Es werden zwei Arten von Variablen unterschieden

- Sprünge im Text: Es gibt interne (innerhalb der Fehlersuchanleitung) und externe Sprünge (in eine andere Fehlersuchanleitung).
- Textvariable: Hier können Platzhalter für Klemmen, Sollwerte, Bezeichner für Prüfkabel oder allgemein: unübersetzbare Textfragmente, definiert werden. Beispiele für Werte: „Kl. 17“, „10 ... 30 Ohm“. Solche Variablen werden vor allem verwendet, um Klemmen zu bezeichnen, was die Wiederverwendbarkeit der Textseiten erhöht.

Zeichnungen der Stromlaufpläne werden zur Zeit mit einem CAD-System neu erstellt, weil die entsprechenden Entwicklungsdaten entweder nicht zur Verfügung stehen oder inkompatibel sind. Aus Zeitgründen beschränkt man sich dabei auf einen einzigen Stromlaufplan, der alle Funktionsgruppen eines Teilsystems zugleich darstellt. Einbaulagen von Komponenten im Kfz werden nach Fotovorlagen gezeichnet und als Vektorgrafik benutzt.

Das Autorensystem bietet Konverter an, um FSAs, die als Word-Dokument vorliegen, in das Autorensystem zu importieren. Im Ausland werden Anleitungen für solche Fahrzeuge geschrieben, die in Deutschland nicht verfügbar sind. Diese Anleitungen werden in englischer Sprache als Word-Dokument geliefert, ins Deutsche übersetzt und mittels Konverter in das IMSIS-System übernommen. Dabei müssen Variablenwerte und Textsprünge von Hand zugewiesen werden. Ein Problem ist hierbei vor allem, den Bezug einer importierten Anleitung zu den erwähnten Textseiten herzustellen.

```
EIGENDIAGNOSE - FEHLERCODE 3

Temperaturfühler Kühlmittel

Fehleranzeigen:
* Masseschluß
* Unterbr./Plusschluß

Mögliche Fehlerursachen:
* Temperaturfühler (NTC) defekt
* Leitungsfehler

Temperaturfühler Kühlmittel
Widerstand des Temperaturfühlers Kühl-
mittel prüfen.
Messung am Steuergerätestecker zwischen
den Klemmen 53 und 13.

18...22 Grad C =
2280...2720 Ohm [?]  Ist:   Ohm
48...52 Grad C =
750...900 Ohm [?]   Ist:   Ohm
78...82 Grad C =
290...353 Ohm [?]   Ist:   Ohm

Werden die Sollwerte nicht erreicht,

Messung am NTC wiederholen.
```

Abbildung 3-2: Prüfanleitung Temperaturfühler

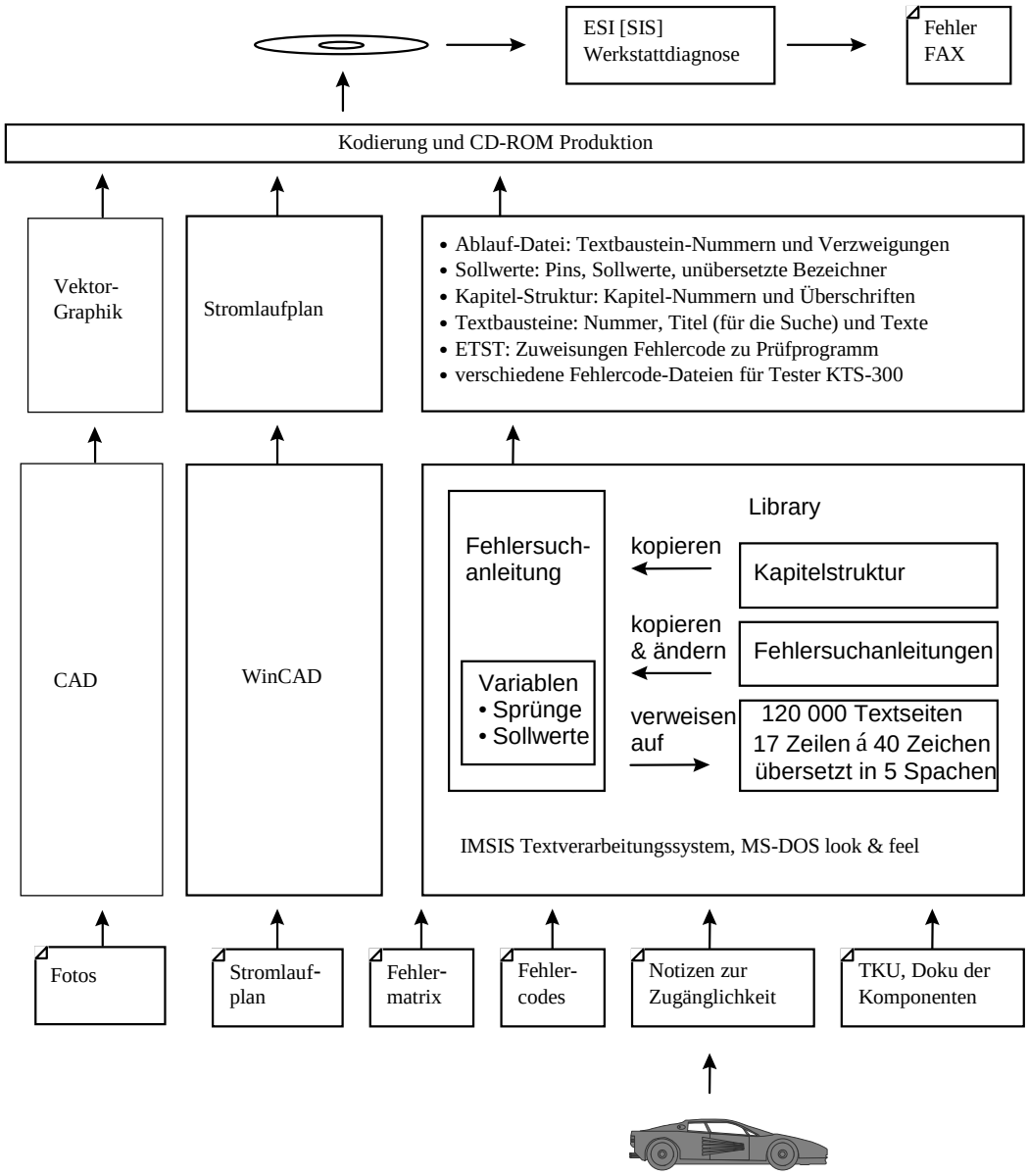


Abbildung 3-3: Produktion von FSAs mit dem Bosch Autorensystem

3.4 Anforderungen an ein Unterstützungswerkzeug

Aus der vorstehenden Analyse wurde klar, daß ein Redesign des bestehenden Autorensystems dringend erforderlich war und auch vom Geschäftsbereich gewünscht wurde. In diesem Abschnitt werden nun allgemeine Anforderungen an ein künftiges Werkzeug abgeleitet. Diese ergeben sich von verschiedenen Seiten:

- Forderungen aus Sicht des Bereichs nach Rationalisierung
- Forderungen aus Sicht der Autoren bzgl. Funktionalität und Bedienbarkeit
- Anforderungen aus der Analyse von Struktur und Inhalt der FSAs
- Allgemeine Anforderungen aus der Analyse der Arbeitsprozesse
- Anforderungen aus der bestehenden Softwareumgebung
- Anforderungen im Hinblick auf die Realisierung des Werkzeugs

3.4.1 Forderungen aus Sicht des Geschäftsbereichs KH

Die wichtigsten generellen Anforderungen an eine neue computergestützte Lösung sind:

- **Bearbeitungszeit verkürzen**. Das Werkzeug soll die Bearbeitungszeit für die Erstellung einer FSA spürbar verkürzen. Erreicht werden soll dies durch eine (evtl. teilweise) Automatisierung von abgrenzbaren, relativ aufwendigen wiederkehrenden Teilaufgaben.
- **Keine Umstellung des Arbeitsablaufs**. Zur Vermeidung von Umstellungskosten soll der bisherige Arbeitsablauf bei der Erstellung von FSAs erhalten bleiben. Das heißt, das Werkzeug soll den bisherigen Arbeitsablauf nicht ersetzen, sondern wirkungsvoll ergänzen.
- **Möglichst Verbesserung der Ergebnisse**, z.B. durch garantierte Vollständigkeit.

3.4.2 Forderungen aus Sicht der Autoren

Die Autoren von FSAs setzen bereits heute für ihre Arbeit diverse Software-Werkzeuge für die Text- und Bildverarbeitung ein. Jede Einführung eines neuen Werkzeugs bedeutet zusätzlichen Einarbeitungsaufwand und potentiell eine unerwünschte Umgewöhnung des Arbeitsablaufs. Hieraus ergeben sich folgende Forderungen:

- **Einfache Bedienung**: es sollen keine speziellen Sprachen zur Bedienung eingeführt werden. Die Konzepte und Objekte des Werkzeugs sollen sich in natürlicher Weise auf den gewohnten Arbeitsprozeß und seinen Gegenstand beziehen. Zusätzlicher Bedienungsaufwand ist nur in dem Maße vertretbar, wie er sich insgesamt in Zeitgewinn oder verbesserten Arbeitsergebnissen niederschlägt.
- **Mühelose Übernahme** erzielter Arbeitsresultate in den bisherigen Arbeitsprozeß. Das Werkzeug muß eine Schnittstelle anbieten, die den Transfer der damit erzielten Arbeitsresultate in den bisherigen Arbeitsprozeß ermöglicht. Beispielsweise soll eine automatisch generierte FSA, zumindest in der Endversion des Werkzeugs, auch automatisch entweder in das existierende Autorensystem oder an nachgeschalteter Stelle in die Prozeßkette zur Erstellung der CD übernommen werden.
- **Mögliche Korrektur automatisch erzielter Resultate**. Es kann nicht davon ausgegangen werden, daß die automatisch erzeugten Ergebnisse alle relevanten Aspekte berücksichtigen und vom Autor ungeändert übernommen werden können. Das Werkzeug muß ihm nicht nur ermöglichen, entsprechende Änderungen vorzunehmen (etwa Abarbeitungsreihenfolge von Prüfprogrammen), sondern auch möglichst robust gegenüber solchen Änderungen sein (d.h. die geänderten Ergebnisse überprüfen und weiterverarbeiten können).

3.4.3 Analyse von Struktur und Inhalt der Fehlersuchanleitungen

Die Analyse der Fehlersuchanleitung für die verteilerbasierte EDC, aber auch weiterer Dokumente für ABS und Motronic haben gezeigt, daß 75% der Seiten einer solchen Anleitung aus Prüfprogrammen für Fehlercodes und aus Fehlersuchprogrammen für Fahrerbeanstandungen bestehen. Die restlichen 25% werden von einleitenden Funktionsbeschreibungen, Informationen über verfügbare Werkzeuge, die Einbaulage der Komponenten usw. gebildet. Die automatische Erstellung dieser 25% ist nach heutigem Stand der Technik nicht mit vertretbarem Aufwand möglich. Die Erstellung der Prüfprogramme für Fehlercodes und der Fehlersuchprogramme für Fahrerbeanstandungen kann dagegen wirkungsvoll durch automatische, modellbasierte Methoden unterstützt werden. 75% dieser Prüfprogramme beziehen sich auf die Elektrikdiagnose von Aktuatoren und Sensoren. Die übrigen 25% betreffen die Hydraulik-, Pneumatik- und Mechanik-Diagnose. Vor allem die modellbasierte Automatisierung der Prüfprogrammerzeugung für Elektrik und evtl. Hydraulik scheint mit Blick auf die heute verfügbaren Techniken kurzfristig erfolgversprechend. Aus der Analyse der FSA ergeben sich zwei Forderungen:

- Konzentration auf Elektrikdiagnose in den ersten Entwicklungsstufen des Werkzeugs. Damit lassen sich 75% der Prüfprogramme mit relativ geringem Arbeitsaufwand mit bekannten Methoden aus einfachen Modellen automatisch erzeugen.
- Steuerbarer Detaillierungsgrad der Prüfanweisungen. Heutige Prüfprogramme sind oft kompakter formuliert als solche, die sich mit bekannten Methoden auf einfache Weise modellbasiert generieren lassen. Die Autoren von FSAs fassen oft mehrere Bauteile, die eine ähnliche Funktion erfüllen, zu einem Begriff zusammen (Beispiel: „Zuleitungen zum Sensor“, anstatt „Leitung Kl. 23 zu Kl. 2 und Leitung Kl. 35 zu Kl. 1“). Das geplante Werkzeug könnte diese Fähigkeit zum geschickten Zusammenfassen nachbilden und dem Benutzer auf einfache Weise eine Auswahl des Detaillierungsgrades und der Begriffe ermöglichen, mit denen ein automatisch erzeugter Prüfplan operiert.

3.4.4 Analyse des Arbeitsprozesses

Die gegenwärtige Prozeßkette zur Entwicklung von Fehlersuchanleitungen weist folgende Ansatzpunkte für eine Verbesserung durch wissensbasierte Methoden auf:

- Keine oder geringe Nebenläufigkeit: Eine FSA wird in der Regel von einem einzigen Autor erstellt.
- Keine durchgängige Verwendung elektronischer Dokumente: Maschinenlesbare Dokumente (etwa FMEA-Analysen, SGML-Lastenhefte, Stücklisten, Symptomdatenbank, CAD-Modelle usw.) gehen derzeit nicht in den Arbeitsprozeß ein.
- Übersetzungskosten: Ein wesentlicher Kostenfaktor bei der Erstellung von FSAs sind die Übersetzungskosten. Sie werden momentan durch die Wiederverwendung von übersetzten Textseiten minimiert. Jedes Werkzeug zur Unterstützung des Arbeitsprozesses muß hierzu eine mindestens ebenso ökonomische Lösung anbieten.

3.4.5 Vorhandene Softwareumgebung der Autoren

Aus der bei Bosch eingesetzten Werkzeugumgebung ergeben sich folgende Merkmale und Anforderungen:

- PC-gestütztes Arbeiten: Die Autoren verwenden heute den PC und Standardsoftware für Textverarbeitung, Tabellenkalkulation etc., auch für Aufgaben außerhalb der Autorentätigkeit.
- Hybride, lose gekoppelte Werkzeugumgebung: Die Autoren setzen für die Produktion von FSAs mehrere unabhängige Werkzeuge ein, z.B. das IMSIS Autorensystem zur Produktion der Texte und der Prüfablaufsteuerung, WinCAD zum Zeichnen der Stromlaufpläne und ein weiteres Werkzeug für die Produktion von Vektorgrafiken (extern), die zum Beispiel die Einbaulage von Komponenten illustrieren. Vektorgrafiken und Stromlaufpläne werden. vom Autorensystem nicht angezeigt, sondern nur über einen Bezeichner referenziert.

3.5 Angestrebte Funktionalität und Bedienung

Ausgangspunkt für den Entwurf eines wissensbasierten Werkzeugs für die oben beschriebene Tätigkeit ist das gegenwärtige Vorgehen bei der Erstellung. Zu einer Menge von Symptomen (Fehlercodes, Kundenbeanstandungen) erzeugt oder adaptiert der Autor unter Verwendung von Textbausteinen die Fehlersuchanleitung als verzeigerten Text mit Entscheidungspunkten und Sprungstellen, anschließend wird daraus eine CD produziert. Das heutige Autorensystem unterstützt das Zusammenstellen von Textseiten und deren Anpassung durch Belegung der Variablen, die in den Textseiten enthaltenen sind, mit Textwerten (Klemmen, Sollwerte). Das hier skizzierte System soll folgende Arbeitsschritte besser unterstützen bzw. teilweise automatisieren:

- Manueller Aufbau von Fehlersuchanleitungen aus einer kompakten Menge vorgegebener atomarer, parametrierter und semantisch indizierter Textbausteine und wissensbasierte Adaption von Prüfplänen
- Modellbasierte Adaption von Prüfplänen
- Modellbasiertes Generieren von Prüfplänen

3.5.1 Modellbasierte Methoden

Die Diagnoseproblematik im Kfz-Sektor zeichnet sich durch folgende Punkte aus:

- Hohe Geschwindigkeit technologischer Neuerungen: Auf längere Sicht ist mit der Verwendung neuer Komponententypen oder neuartiger Realisierungen zu rechnen.
- Variantenvielfalt: zum gleichen Zeitpunkt gibt es eine Fülle von Modifikationen, hervorgerufen durch unterschiedliche Einbausituationen in unterschiedlichen Fahrzeugtypen.
- Wirtschaftlichkeit: Der Druck zur Kosteneinsparung zeigt sich z.B. im sehr begrenzten Steuergerätespeicherplatz, vor allem in der begrenzten Zahl zusätzlicher Sensoren.

Aus den einzelnen Aspekten ergeben sich folgende Forderungen:

- Das Wissen muß modular repräsentiert sein in der Form, daß es abgegrenzte Komponenten bzw. Gruppen von Komponenten gibt, die leicht gegeneinander ausgetauscht werden können.

- Allgemeines Wissen über Komponenten (z.B. Funktion eines Relais), das über den Einsatz in einem speziellen Gerät hinaus Gültigkeit besitzt, muß möglichst unabhängig von dem speziellen Wissen über strukturelle Eigenheiten des Geräts gehalten werden. Dies verringert in der Folge drastisch den Aufwand für die Anpassung an strukturelle Modifikationen, weil dazu nur noch die geänderte Zusammenstellung der Komponenten beschrieben werden muß.
- Das Wirtschaftlichkeitsargument impliziert die Forderung, daß die Diagnose möglichst spezifisch innerhalb des vorgegebenen Rahmens austauschbarer Teile sein muß, um nicht unnötig verdächtigte Komponenten auszuwechseln. Ferner ist vor allem die Wiederverwendbarkeit in zweierlei Hinsicht anzustreben:
  - vorher getätigter Arbeitsaufwand (aus FMEA, Entwurf) sollte genutzt werden können
  - für die Modellierung notwendiger Aufwand sollte auch nutzbar sein für andere Zwecke (z.B. wiederum für FMEA, für Testgenerierung, zur Bestimmung optimaler Sensorplatzierung, etc.).

Die Diagnoseaufgabe bezieht sich auf ein real existierendes Subsystem und ist damit prinzipiell auch für Expertensysteme zugänglich, die sich auf eine Liste erfahrungsgestützter Symptom-Fehler-Assoziationen stützen. Dies würde voraussetzen, daß zunächst für ein neues Fahrzeug oder Subsystem praktische Erfahrungen gemacht werden müßten, bevor Computerunterstützung möglich wäre, was den angestrebten Zielen widerspricht. Ferner ist Abschätzung oder gar Garantie der Vollständigkeit dieses diagnostischen Wissens - unter Sicherheitsgesichtspunkten geboten - prinzipiell unmöglich. Aus denselben Gründen stellt die Übernahme von Diagnosewissen über ähnliche Systeme keine Lösung dar.

Die Übertragbarkeit dieses Wissens auf ein neues System bloß empirisch zu ermitteln, würde wieder die Verfügbarkeit von Computerunterstützung verzögern und ist aus Sicherheitsgründen nicht ausreichend. Schließlich verhindert in der Praxis ein ökonomischer Gesichtspunkt assoziationsbasierte Lösungen: da diese Assoziationen jeweils an einen Kontext gebunden sind, sind darauf gegründete problemlösende System für jedes System gesondert zu erstellen - bei der Variantenvielfalt und der Geschwindigkeit technologischer Neuerung im Automobilsektor kein gangbarer Weg.

Es stellt sich also die Aufgabe, nicht einfach die angestrebten Resultate (Beziehungen zwischen Fehlern und ihren Auswirkungen) zu repräsentieren, sondern das ihnen zugrundeliegende Wissen über das jeweilige technische System. Dies muß in einer modularen Form geschehen, so daß aus den Elementen das erforderliche spezifische Wissen über Systemvarianten generierbar ist. Gerade in der Diagnose ist diese Form der Repräsentation durch die modulare Struktur des technischen Systems selbst gegeben: zum einen gruppiert sich Wissen über Ursachen von Fehlverhalten um die einzelnen Systemkomponenten, zum anderen zielt Diagnose auf Reparatur, Austausch, Kontrolle oder Kalibrierung dieser Komponenten.

Das Wissen über Komponenten, ihre möglichen Fehlverhalten und ihr Zusammenwirken in einer besonderen Struktur versetzt menschliche Experten in die Lage, auch neue, nicht erprobte Gerätevarianten zu analysieren und zu diagnostizieren. Sicherlich spielt Erfahrungswissen eine wichtige Rolle und kann auch neu gewonnen werden. Es schlägt sich nieder in einer lokalen Erweiterung oder Änderung eines Komponentenmodells (etwa durch ein neu auftretendes Fehlverhalten) oder in Kontrollwissen darüber, wie das Gerätemodell für eine effektive und effiziente Problemlösung ausgenutzt werden sollte.

Häufig sind qualitative Vergleiche auf (nicht notwendig spezifizierte) Soll- und Istwerte bezogen. „Zähnezahl zu groß“ beim Impulsrad führt zu „erhöhter Frequenz“ des Signals; diese Aussage ist aus der Kenntnis der prinzipiellen Wirkungsweise des Drehzahlgebers zu gewinnen, ist unabhängig vom Sollwert und trifft für eine unendliche Menge von abweichenden Istwerten zu. Z.T. vorhandene Spezifikationen von Parameterwerten (z.B. „Widerstand zwischen 800 und 1400 Ω“) dient oft eher dem Testschritt, ohne für die Ableitung korrekter Diagnosen essentiell zu sein. Die Auswirkung eines Kurzschlusses läßt sich ohne Kenntnis des genauen Werts der Widerstände ableiten. Dies ist typisch für die Diagnosesituation, die ja in der Regel durch eine qualitative Abweichung vom spezifizierten oder gewünschten Verhalten charakterisiert ist.

Da qualitative Modelle nur die wesentlichen Unterscheidungen und Zusammenhänge formulieren, charakterisieren sie ganze Klassen von Komponentenvarianten, die sich nur durch Parametervariationen unterschieden. Die Elemente einer solchen Modellbibliothek sind also generisch und repräsentieren nicht Individuen, sondern Typen von Komponenten, was den Aufwand für die Erstellung der Bibliotheken drastisch reduziert oder überhaupt erst handhabbar macht. Auch zeitliche Aspekte des Verhaltens werden fast ausschließlich qualitativ charakterisiert und damit viele konkrete zeitliche Verläufe zusammengefaßt, z.B. wird das Auftreten fehlerhafter Signale nach „regelmäßig“, „sporadisch“ und „dauerhaft“ unterschieden oder lediglich auf die intermittierenden Varianten der konstanten Fehlerursachen und -arten eingegangen.

3.5.2 Wissensbasierte Adaption von Prüfplänen

Dies legt als 1.Stufe einer Rechnerunterstützung das modellbasierte Anpassen einer Fehlersuchanleitung (Vorlage) an eine neue Situation in folgender Weise nahe:

- Die Fehlersuchanleitung besteht aus hierarchisch gruppierten parametrierten Textbausteinen aus einer Bibliothek. Jeder Textbaustein ist dabei in Bezug auf seinen Inhalt und seine Rolle innerhalb der Fehlersuchanleitung (z.B. „Messung“, „Sicherheitshinweis“, „Reparatur“, „Klemme“ usw.) klassifiziert. Auf dieser Grundlage wird ein bestimmtes Objekt, etwa eine bestimmte Klemme, einmal und von seinen zumeist mehrfachen Erscheinungsformen in der Fehlersuchanleitung getrennt repräsentiert. Ferner erleichtert diese Typisierung die Suche.
- Suchen einer geeigneten Vorlage: Für eine gegebene Funktionsgruppenbezeichnung (z.B. „Temperaturerfassung“) werden alle bekannten Prüfpläne (Schaltplanausschnitt) angezeigt. Der Autor wählt dann die Vorlage, die seiner Einschätzung nach der zu bearbeitenden Funktionsgruppe am nächsten kommt.
- Adaptieren: Der Autor ändert die Sollwerte, Bezeichner für Klemmen und Bauteile an beliebiger Stelle im Text oder dessen strukturierter Darstellung. Diese Änderungen wirken sich sofort auf alle Erscheinungsformen dieses Objekts im zugehörigen Prüfplan aus.
- Automatisches Konvertieren des Prüfplans in eine Repräsentation, die mit dem Standard J2008 kompatibel ist, wird auf der Grundlage der semantischen Annotation möglich, ebenso die automatische Konvertierung in HTML.

Abbildung 3-4 zeigt die Architektur der Stufe 1 im Überblick.

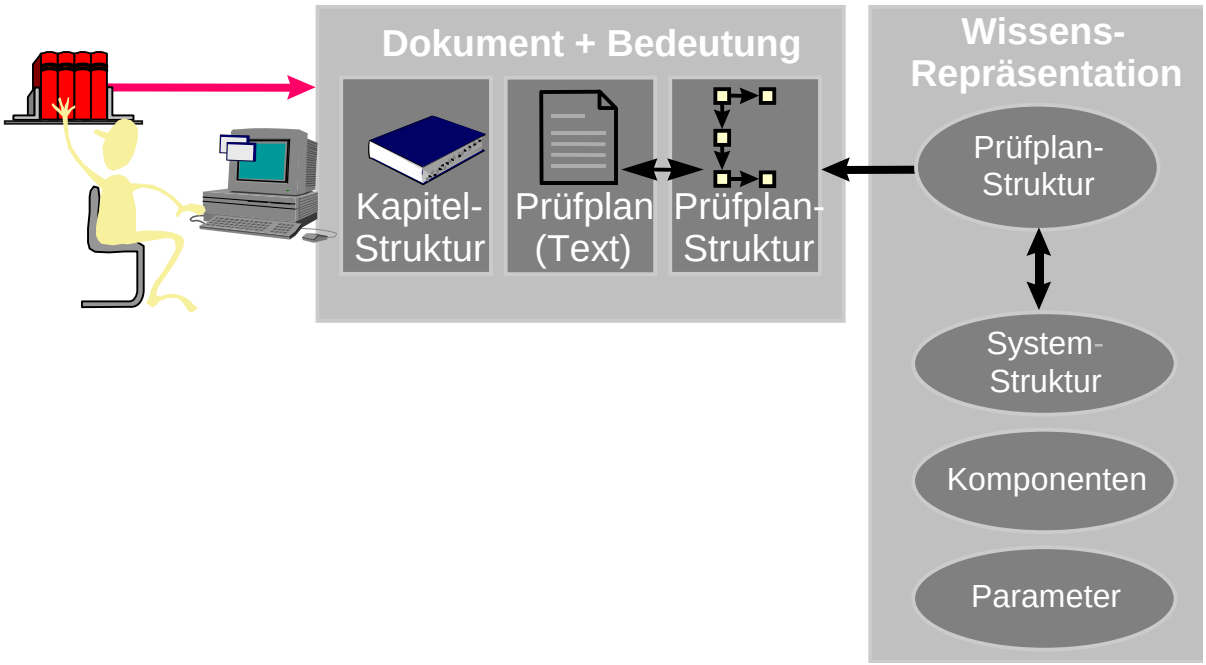


Abbildung 3-4 : Stufe 1 - Wissensbasierte Adaption von Prüfplänen

Hinter dem Konzept des wissensbasierten Adaptierens von Prüfplänen stehen folgende Überlegungen:

- Angesichts vieler verschiedener Fahrzeuge ist das manuelle Anpassen von Prüfplänen eine häufig wiederkehrende Aufgabe, Teilautomatisierung hier bringt durch den Multiplikationseffekt einen relativ hohen Gewinn
- Adaptieren auf Funktionsgruppenebene sichert einen hohen Wiederverwendungsgrad von Prüfplänen

3.5.3 Modellbasierte Adaption von Prüfplänen

In dieser Stufe wird der Zusammenhang der in der Fehlersuchanleitung repräsentierten Objekte mit den Komponenten der Funktionsgruppe und deren Eigenschaften und Beziehungen, d.h. einem Systemmodell, hergestellt und ausgenutzt.. Wie in Abbildung 3-5, dargestellt, werden Struktur, Komponenten und deren Daten repräsen-

tiert und mit der Wissensrepräsentationsebene der Fehlersuchanleitung verknüpft. Diese Information kann über eine hierarchische Darstellung des Systems, Datenblätter der Komponenten und als graphische Strukturbeschreibung visualisiert und manipuliert werden. Damit kann die Adaption von Fehlersuchanleitungen auch auf dieser Ebene vorgenommen werden, mit direkter Auswirkung auf die textuelle Darstellung der Fehlersuchanleitung. Als Schritte sind durchzuführen:

- Schaltbild zur Fehlersuchanleitung zeichnen. Dazu stehen ein komfortabler Editor und vordefinierte Symbole zur Verfügung.
- Parameter der Textbausteine mit dem Schaltbild verknüpfen. Mit der Maus werden Klemmen, Sollwerte und Bauteilbezeichner aus dem Schaltplan über die Textbausteine gezogen und dadurch deren Parameterwerte festgelegt.
- Änderungen am Schaltbild wirken sich automatisch auf den Text aus und umgekehrt, d.h. Bild und Text sind konsistent.
- Zeichnungen der Funktionsgruppen sind als zusätzliche Dokumentation im SIS verwendbar

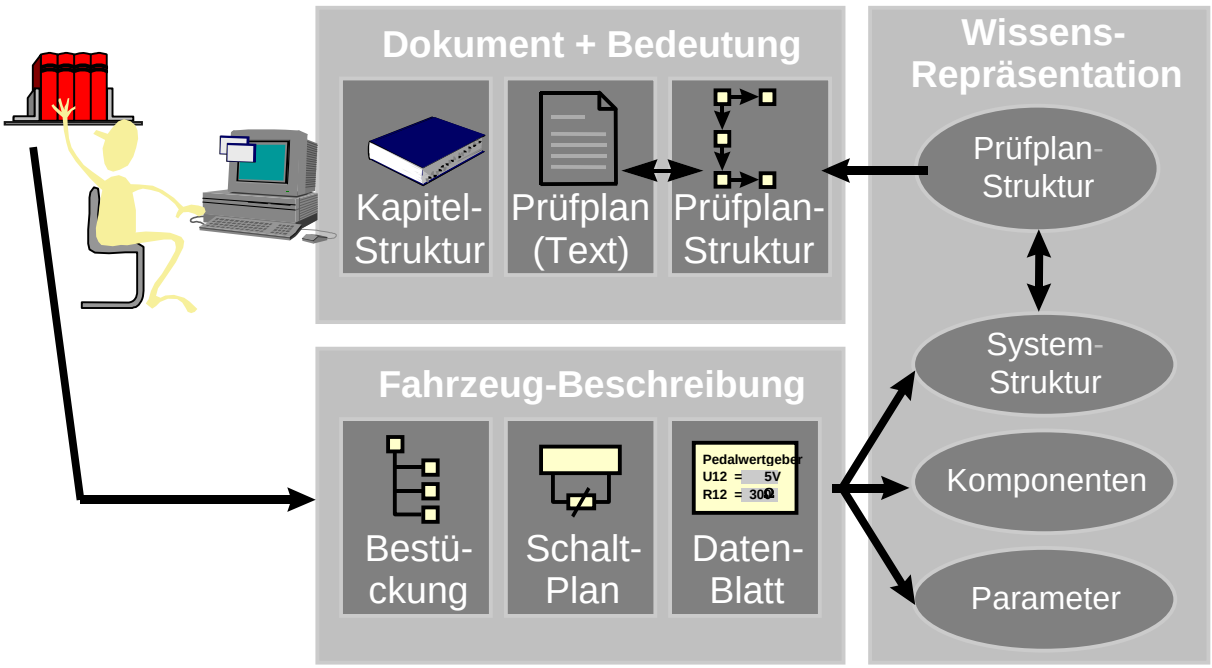


Abbildung 3-5: Stufe 2 - Modellbasiertes interaktives Erzeugen und Checken von Prüfplänen

3.5.4 Modellbasiertes Generieren von Prüfplänen

Dies zielt darauf ab, das Erstellen einer Fehlersuchanleitung modellbasiert zu unterstützen, wobei die Initiative gegenüber Stufe 2 (vgl. Abbildung 3-6) jetzt vom Autor zum System wechselt: Das Werkzeug generiert selbständig einen Prüfplan. Es liefert einen plausiblen Vorschlag, der vom Autor akzeptiert oder geändert werden kann.

Diesem Konzept liegen folgende Überlegungen und Beobachtungen zugrunde

- Der Autor erstellt den Prüfplan durch Zeichnen und Kommentieren eines Schaltplans
- Kommentare liefern die diagnoserelevante Zusatzinformation, die nicht im Schaltplan enthalten ist, z.B.
  - Relevante Fehler: Kontaktfehler und Kurzschlüsse nicht im Prüfplan berücksichtigen
  - Meßbarkeit: An Stecker 1 und 2 sind beliebige U- und R-Messungen möglich
  - Zugänglichkeit: Stecker 1 ist leichter zugänglich als Stecker 2
- Diese Eingaben reichen, um für sehr viele Funktionsgruppen einen brauchbaren Prüfplan automatisch zu generieren. Die theoretischen und technischen Grundlagen für die automatische Prüfplangenerierung werden in [Struss et al. 00] in diesem Band diskutiert.

- Der generierte Prüfplan kann vom Autor bei Bedarf überarbeitet werden

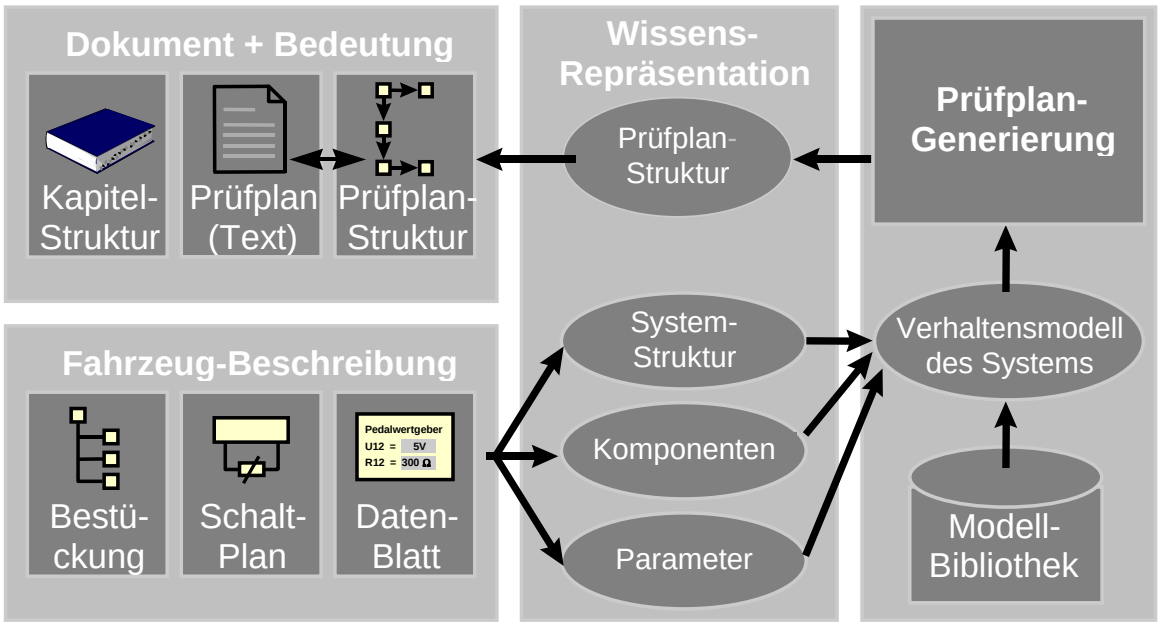


Abbildung 3-6: Stufe 3 - Modellbasiertes Generieren von Prüfplänen

3.6 Feinspezifikation eines geeigneten Basis-Autorensystems

In diesem Kapitel wird die Spezifikation des *Basis-Autorensystems* angegeben, auf dem die oben genannten Stufen aufbauen sollten, das aber bereits eine deutliche Verbesserung des heutigen Systems bietet.

3.6.1 Aufbau der Spezifikation

Die Spezifikation besteht aus folgenden Abschnitten:

- Das Fachkonzept „Komponenten“ setzt die im Rahmen einer FSA behandelten Bauteile, Funktionsgruppen, Systeme und andere relevante Fahrzeugbestandteile aus der fachlichen Sicht des Autors in Beziehung zueinander. Den Überblick über das Fachkonzept liefert ein Analyse-Klassendiagramm, das in UML-Notation („Static structure diagram“) präsentiert wird.
- Das Fachkonzept „Struktur der FSA“ stellt die Beziehung der im Rahmen einer FSA vorkommenden Elemente und vorgegebenen Gliederungen eines solchen Dokuments dar. Einen Überblick über das Fachkonzept liefert wiederum ein Analyse-Klassendiagramm.
- Der Funktionsumfang des geplanten Systems wird in Form von Anwendungsfällen beschrieben. Die Anwendungsfälle des Basisprogramms werden im „use case“ Kapitel zunächst einzelnen Benutzergruppen zugeordnet und gegenseitig in Beziehung gesetzt, um anschließend erläutert zu werden. Das Kapitel gibt Auskunft über alle Funktionen, die im Rahmen des Basisprogramms angeboten werden sollen und beschreibt, wie und unter welchen Randbedingungen die Funktionen ausgeführt werden.

3.6.2 Fachkonzept Komponenten

Folgendes Schaubild (Abbildung 3-7) zeigt die benötigten Objekte, wie sie von der FSA gefordert werden. Es wurde dabei explizit auf die genaue Anpassung an die Anforderungen der Dokumentenerstellung geachtet. Es liegt daher keine kompromißlose Abbildung der Verhältnisse der realen Systeme und Komponenten vor, sie wären an einigen Stellen zu weitreichend.

3.6.3 Fachkonzept Struktur Fehlersuchanleitung

Dieses Fachkonzept beschreibt den Aufbau einer FSA. Die Zusammenhänge der Gliederungsobjekte werden in der folgenden Abbildung 3-8 dargestellt.

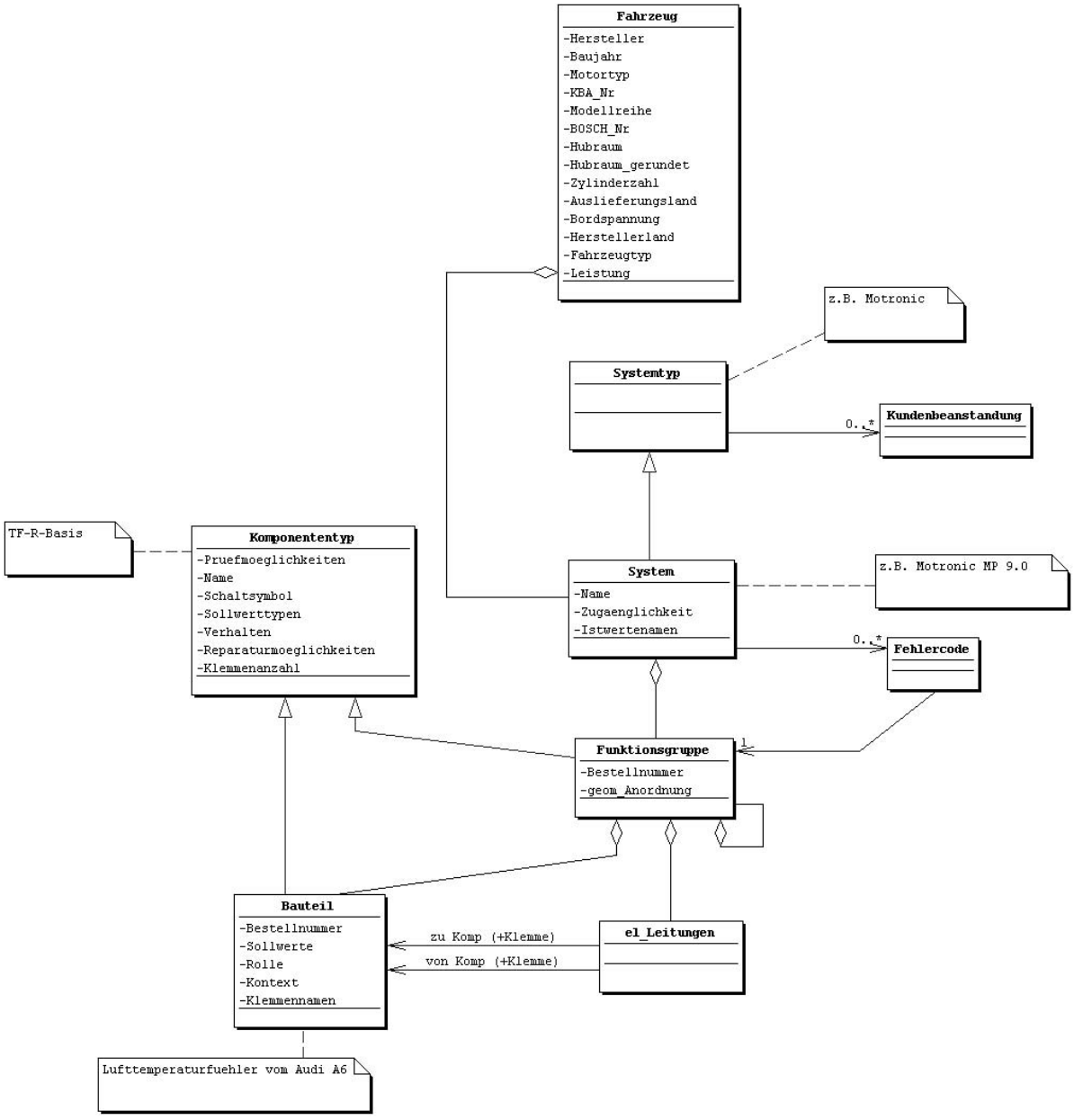


Abbildung 3-7: Fachkonzept Komponenten

Legende

- Fahrzeug: Das Fahrzeug und seine Eigenschaften sind zentral für den technischen Hintergrund einer Fehlersuchanleitung. Das Fahrzeug bringt natürlich eine Vielzahl von Eigenschaften mit.
- Systemtyp: Der Systemtyp gibt wichtige Eigenschaften für seine Unterklasse, dem System vor.
- System: Das System ererbt Eigenschaften von seiner Oberklasse, dem Systemtyp. Diese werden durch weitere Eigenschaften ergänzt, die erst auf der konkreten Ebene des Systems verfügbar sind.
- Komponententyp: Komponententypen geben viele Eigenschaften für Bauteile bzw. Funktionsgruppen vor. Sie werden in einem ersten Schritt festgelegt und an jene vererbt.

- Funktionsgruppe: Ein System besteht aus Funktionsgruppen. Eine Funktionsgruppe selbst erhält viele Eigenschaften durch Vererbung aus dem Komponententyp.
- Bauteil: Ein Bauteil ist im Gegensatz zu den meisten obigen Objekten konkret und besitzt daher auch die Attribute „Bestellnummer“ und „Sollwerte“. Ein Bauteil erbt Eigenschaften vom Komponententyp.
- Elektrische Leitungen: Elektrische Leitungen spielen eine Sonderrolle, da sie nicht unter Bauteilen eingeordnet werden sollten, sie besitzen z.B. normalerweise keine Bestellnummern.

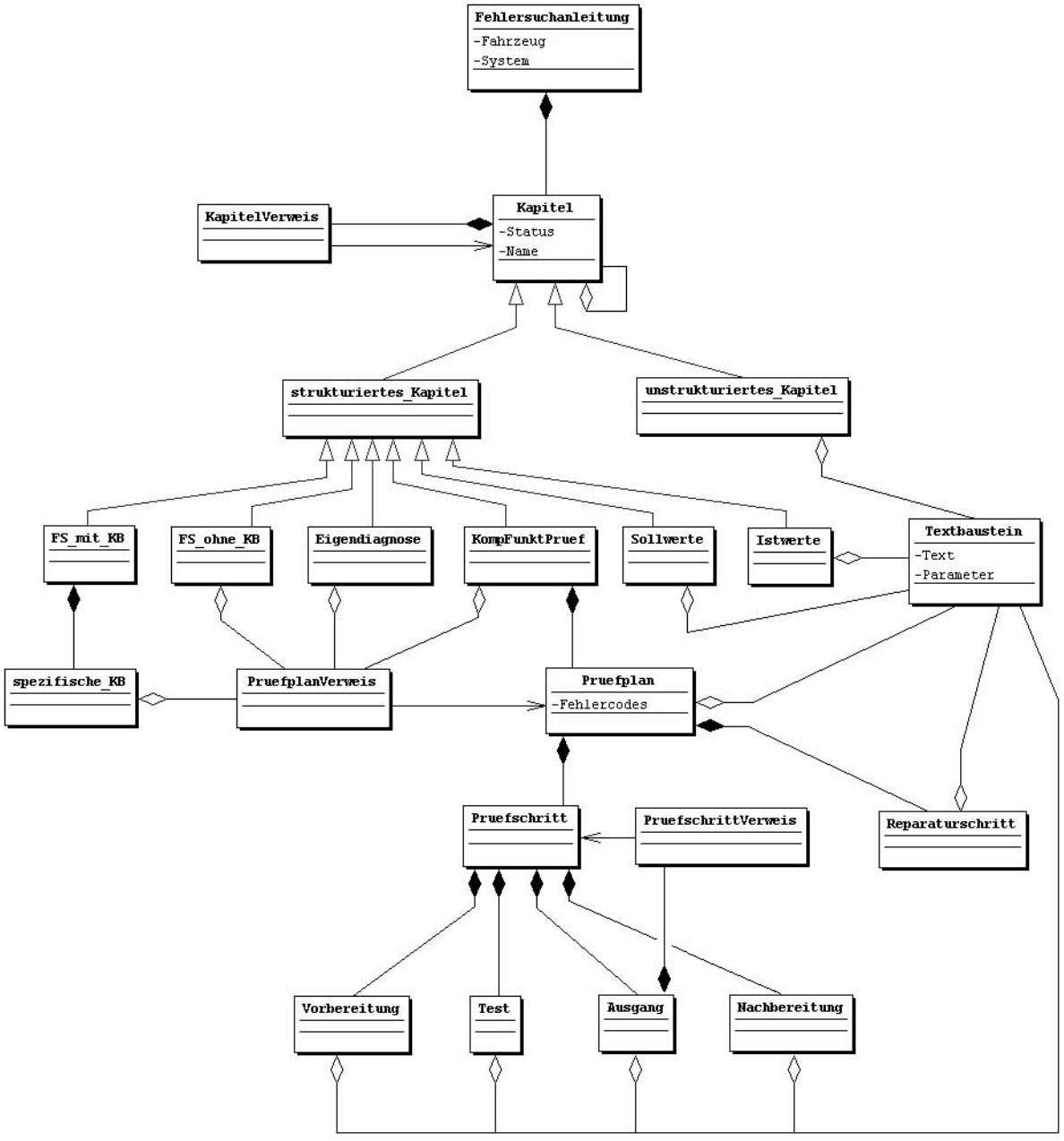


Abbildung 3-8: Fachkonzept Fehlersuchanleitung

Legende

- Fehlersuchanleitung: Eine Fehlersuchanleitung ist das oberste, hier betrachtete Objekt.
- Kapitel: Das Kapitel dient der Gliederung einer Fehlersuchanleitung
- Kapitelverweis: Der Kapitelverweis dient dem Aufbau von Inhaltsverzeichnissen.
- Spezielle Kapitel: In einer Fehlersuchanleitung ist eine Vielzahl von Kapiteln mit jeweils sehr charakteristischen Eigenschaften vorhanden. Dies wird in der hier gewählten Einteilung widergespiegelt: Sie werden ein-

geteilt in strukturierte Kapitel, deren Aufbau genau festgelegt ist, und unstrukturierte Kapitel, die vor allem unter Verwendung von Textbausteinen und Kommentaren gefüllt werden.
- Prüfplan: Der Prüfplan stellt eine Gliederungsebene zwischen Kapitel und Prüfschritt / Textbaustein dar.
- Prüfplanverweis: Prüfplanverweise werden in mehreren der strukturierten Kapitel verwendet. Sie werden dort zu Listen angeordnet.
- Prüfschritt: Ein Prüfschritt dient der weiteren Gliederung von Prüfplänen. Er wird durch einen inhaltlichen Zusammenhang und typisch aufzufindenden Teilaufgaben charakterisiert.
- Prüfschrittverweis: Prüfschrittverweise werden zur Beschreibung und Interpretation des Ausgangs eines Prüfschritts verwendet und können abhängig vom Ergebnis eines Tests zu weiteren Prüfplänen verweisen.
- Reparaturschritt: Neben Prüfschritten können in Prüfplänen auch Reparaturschritte auftreten, die auf geeignete Prüfschritte folgen (wenn eine Prüfung genügend Klarheit über einen Fehler erbringt). Ein Prüfschritt wird durch die Teilaufgaben Vorbereitung, Test, Ausgang, Nachbereitung charakterisiert.
- Textbaustein: Der Textbaustein ist das atomare Element in dieser Herangehensweise.

3.6.4 Benutzergruppen und „use cases“

Im folgenden werden die beteiligten Benutzergruppen „Administrator“ und „Autor“ kurz beschrieben. Außerdem werden die Aufgaben in „use case“ Schaubildern in UML-Schreibweise dargestellt und die Beziehung der einzelnen Aufgaben zueinander deutlich gemacht. Die Anwendungsfälle werden durch Ellipsen dargestellt, in denen der Name des Anwendungsfalls plaziert ist.

3.6.4.1 Administrator

Durch die Einführung des Benutzers „Administrator“ (vgl. Abbildung 3-9) soll deutlich werden, daß diese Trennung beim späteren operativen Einsatz eines Systems von Bedeutung sein wird. Änderungen von Daten, die dem Administrator unterliegen, führen nicht rückwirkend zu Änderungen in FSAs. Beispielsweise werden die Symptome bei Auswahl des Systems für eine FSA eingefroren.

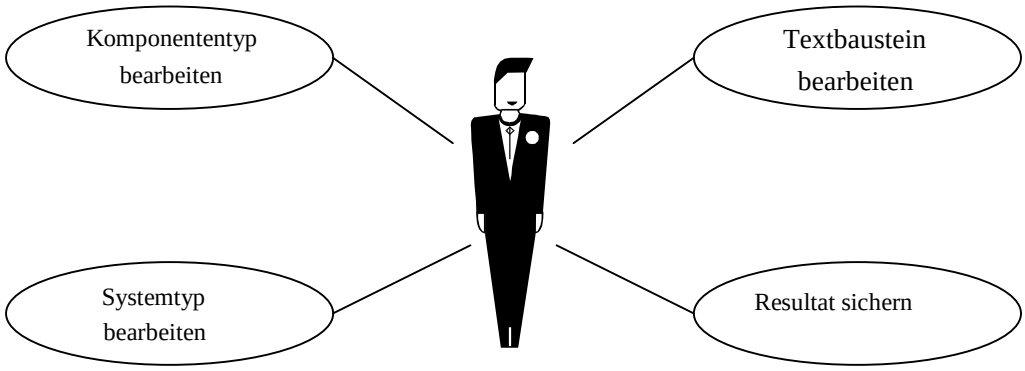


Abbildung 3-9: Aufgaben des Administrators

Die dem Administrator zugeordneten **„use cases“** sind:

- Komponententyp bearbeiten

  In diesem Anwendungsfall wird dargestellt, wie ein Komponententyp bearbeitet wird, was auch die Erzeugung und Löschung einschließt.

- Systemtyp bearbeiten

  Dazu gehört die Eingabe von Symptomen, die charakteristisch für die jeweilige Art von Systemen sind.

- Textbaustein bearbeiten

Textbausteine stellen die Grundlage für die Erstellung einer FSA dar. Sie müssen durch einen Administrator bestätigt werden, damit in den vorhandenen Textbausteinen ein einheitlicher Stil sichtbar wird und die Gesamtzahl der vorhandenen Textbausteine möglichst klein bleibt.

3.6.4.2 Autor

Der Autor hat die Aufgabe, zu einem bestimmten System eine FSA zu erstellen (vgl. Abbildung 3-10). Da dies eine sehr komplexe Aufgabe ist, sind im Rahmen der Bearbeitung einer FSA eine große Anzahl von Anwendungsfällen vorhanden.

Die einzelnen **„use cases“** für den Autor werden im folgenden beschrieben.

- FSA bearbeiten

  Dies ist die zentrale Aufgabe, die der Autor ausführt und der sich eine Vielzahl der anderen Anwendungsfälle unterordnen. Ziel ist es, die FSA für ein System eines bestimmten Fahrzeugs zu erstellen, zu ändern oder zu löschen.

- FSA kopieren

  Ein Autor kann eine komplette FSA kopieren, um sie mit einigen Änderungen an ein sehr ähnliches Fahrzeug anzupassen und mit einer anderen Nummer abzuspeichern.

- Kapitel bearbeiten

  Die Reihenfolge der Kapitel ist vorgegeben (vgl. Abschnitt 3.3), hier werden vor allem die Kapitel mit freien Texten bearbeitet.

- Istwert-Kapitel bearbeiten

  Für dieses Kapitel soll lediglich die Istwertetabelle aufgebaut werden. Die Anlage von einzelnen konkreten Istwerten, also die Sprungziele aus der Istwertetabelle, sowie die Anlage der konkreten Verweise in der Istwertetabelle, ist nicht Gegenstand dieser Spezifikation, da diese Funktionalität für das Basisprogramm nicht vorgesehen ist.

- Generierte Kapitel anzeigen

  Kapitel, die generiert werden und nicht vom Autor erstellt werden müssen, können angezeigt werden. Für diesen „use case“ kommt im Basisprogramm nur das Sollwerte-Kapitel in Frage, das aus den Datenblättern generiert wird.

- Textbaustein wählen

  Da die Anzahl der Textbausteine im Laufe der Zeit anwachsen kann, ist es nötig, dem Autor eine geeignete Möglichkeit an die Hand zu geben, in der Menge von Textbausteinen einen für den aktuell editierten Kontext geeigneten Textbaustein zu finden.

- Parameter belegen

  Textbausteine dienen dazu, Text schneller und in einem einheitlichen Stil aufzubauen. Um sie an den jeweiligen Kontext anpassen zu können, werden Parameter verwendet. Dieser Anwendungsfall beschreibt alle Aktivitäten des Autors, um passende Textbausteine zu finden und Parameter gefundener oder vorhandener Textbausteine zu belegen.

- Sequenz von Verweisen bearbeiten

  Für einige Kapitel wird eine Liste von Verweisen auf Prüfpläne benötigt. Dieser Anwendungsfall beschreibt, wie solch eine Liste aufgebaut werden kann. Folgende Kapitel fordern dieses Vorgehen:

  – Komponenten- / Funktionsprüfungstabelle.

  – Fehlersuche nach Kundenbeanstandung, insbesondere die Abschnitte, die spezifische Kundenbeanstandungen enthalten.

- Fehlersuche ohne Kundenbeanstandung.
- Eigendiagnose-Fehlercodes.

Es wird hier davon ausgegangen, daß keines dieser Kapitel vollständig automatisch generiert werden kann. Vielmehr muß der Autor in allen Fällen noch eingreifen, um eine geeignete Füllung der Kapitel zu gewährleisten. Der folgende Ablauf gilt für alle obigen Kapitel außer dem Kapitel „Fehlersuche nach Kundenbeanstandung“, das über einen alternativen Ablauf entsteht, der mit der Eingabe einer Fehlersuchmatrix beginnt.

- Komponente bearbeiten

  „Komponente“ umfaßt in den hier dargestellten Anwendungsfällen sowohl „Bauteil“ als auch „Funktionsgruppe“ (siehe auch: Fachkonzept). Eine Komponente wird im Rahmen einer FSA durch ihren Namen identifiziert. Dieser Anwendungsfall beschreibt alle Manipulationen einer Komponente von ihrer Erzeugung bis zur Löschung.

- Komponente kopieren

  Dieser Anwendungsfall beschreibt die Möglichkeit des Autors, aus einer vorhandenen FSA ein geeignetes Bauteil auszuwählen und als Kopie im Rahmen einer FSA zu verwenden. Funktionsgruppen können nicht kopiert werden.

- Datenblatt bearbeiten

  Das Datenblatt enthält die Sollwerte einer Komponente, die sich aus ihrer Spezifikation ergeben. Bei Vorliegen dieser Daten in normierter elektronischer Form kann die Erstellung des Datenblatts automatisch ausgeführt werden, was jedoch nicht im Umfang des Basisprogramms enthalten ist. Ein Datenblatt enthält außerdem Informationen wie z.B. Bestellnummer, das Schaltsymbol eines Bauteils oder den Stromlaufplan einer Funktionsgruppe.

- Datenblatt kopieren

  Dieser Anwendungsfall kopiert das bereits vorhandene Datenblatt einer Komponente und ermöglicht es, dieses Datenblatt für eine andere, der vorhandenen Komponente ähnlichen Komponente einzusetzen.

- Strukturplan bearbeiten

  Der Strukturplan stellt den Aufbau einer Komponente dar. Für die Erstellung durch einen Grafik-Editor ist die Eingabe von Knoten, Kanten, Namen und Typen nötig. Ein Strukturplan kann sein: Stromlaufplan, hydraulischer Plan. Beim Stromlaufplan ist eine automatische Generierung aus der Stück- und Netzliste der Komponente vorstellbar.

- Prüfplan bearbeiten

  Ein Prüfplan beschreibt das Vorgehen beim Prüfen einer Komponente auf Fehler. Der Prüfplan besteht in den meisten Fällen aus einer Sequenz von Prüfschritten, seltener aus baumförmig organisierten Prüfschritten. Der normale Ablauf stellt zunächst das Editieren eines aus einer Sequenz von Prüfschritten aufgebauten Prüfplans dar.

- Prüfplan kopieren

  Dieser Anwendungsfall beschreibt eine Situation, in der eine Komponente einen ähnlichen Prüfplan benötigt, wie er schon bei einer beschriebenen Komponente vorhanden ist.

- Prüfschritt bearbeiten

  Prüfschritte sind Aktionen, wie Vorbereitung, Test, Ausgang, Nachbereitung. Jeder dieser einzelnen Aufgaben läßt sich aus Textbausteinen zusammensetzen. Im Falle trivialer Aktivitäten („Motorhaube öffnen“,...) kann eine Aufgabe leer bleiben. Es muß mindestens eine Aufgabe vorhanden und mit Text gefüllt sein.

- FSA überprüfen

  In einem solch umfangreichen Prozeß, wie ihn die Erstellung einer FSA darstellt, können Parameter aus verschiedenen Gründen unbelegt bleiben: Sie könnten z.B. noch nicht vorliegen, sie könnten jedoch auch unbe-

legt bleiben, da der Autor die Arbeit an dem Dokument nicht in einem Arbeitsgang ausführt. Dieser Anwendungsfall beschreibt die Möglichkeit des Autors, die Anleitung auf solche Lücken überprüfen zu lassen.

- Resultat sichern

  Die Ergebnisse der verschiedenen Schritte müssen vom Autor zu durch ihn bestimmten Zeitpunkten gesichert werden können, z.B. durch das Ausführen von Operationen auf einer Datenbank.

- Sprache der Fehlersuchanleitung umschalten

  Ein wichtiger Effekt, der durch den Einsatz von Textbausteinen erreicht werden soll, ist die Vereinfachung bei der Übersetzung einer FSA in andere Sprachen. Bei diesem „use case“ wird eine FSA in einer anderen Sprache dargestellt.

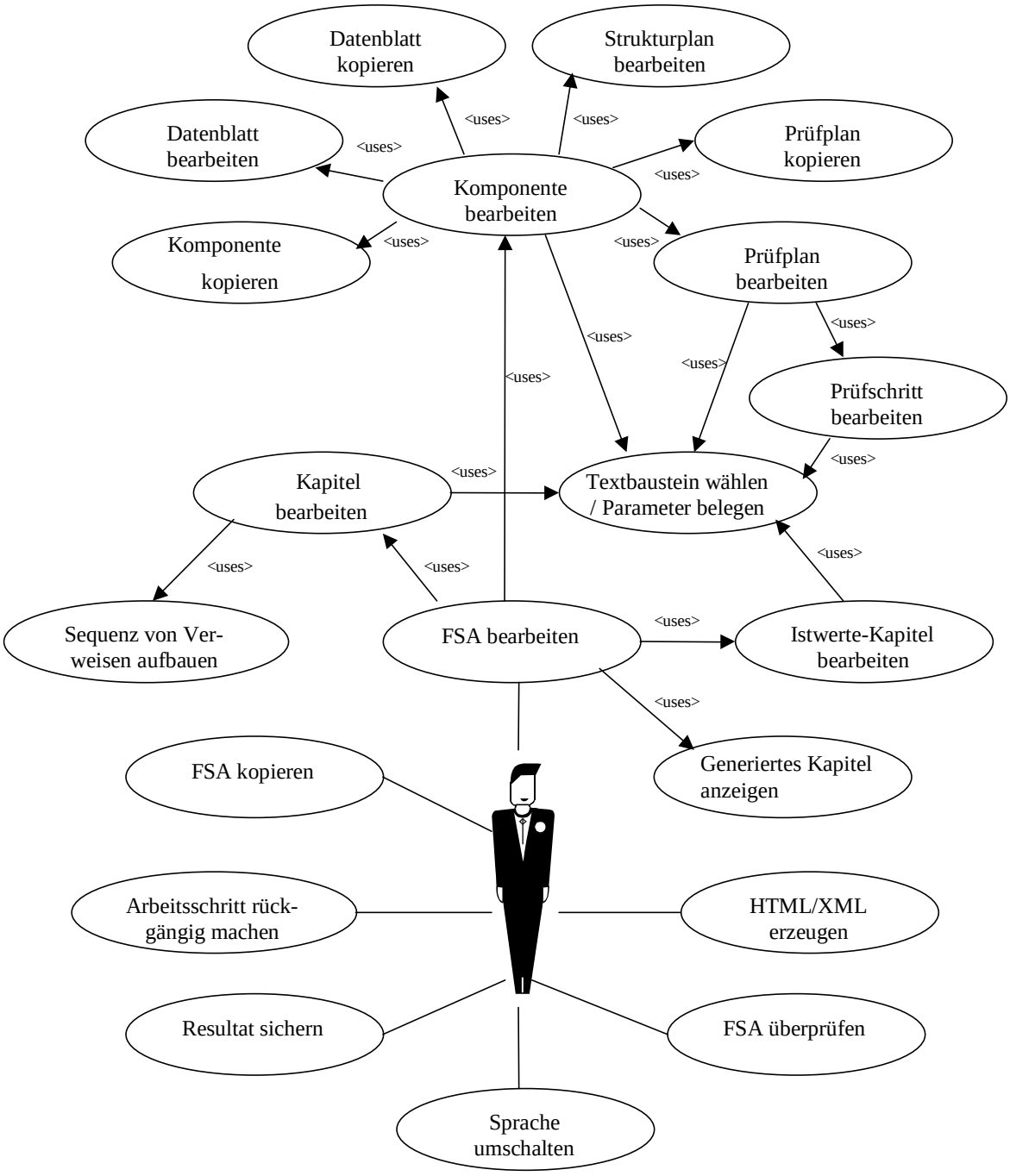


Abbildung 3-10: Aufgaben des Autors

- Arbeitsschritt rückgängig machen

  Der „Undo“ Anwendungsfall beschreibt die Aktionen, die stattfinden, um die letzten Arbeitsschritte zurückzunehmen.

3.7 Zusammenfassung und Ausblick

Der zentrale Kundendienst von Bosch versorgt die Bosch-Dienste weltweit und auch viele freie Werkstätten mit Anleitungen zur Fehlersuche und Reparatur von Subsystemen im Kraftfahrzeug. Diese sog. Fehlersuchanleitungen (FSA) werden heute mit einem einfachen Redaktionssystem erstellt. Wegen der großen Zahl solcher Dokumente für die verschiedenen Systeme und ihrer Ähnlichkeit untereinander ist eine Unterstützung der Erstellung bis hin zur teilautomatischen Erzeugung aus Modellen wirtschaftlich vielversprechend.

Im Rahmen des INDIA Projekts wurde der derzeitige Arbeitsprozeß bei der Entwicklung der FSAs analysiert mit dem Ziel, eine bessere Unterstützung durch moderne Informationstechnik und intelligente Diagnose zu erreichen. Neben funktionalen Anforderungen an ein neues, geeignetes Redaktionssystem ist dabei besonders die Kontinuität des Arbeitsprozesses zu gewährleisten, um die Akzeptanz bei den technischen Autoren zu sichern. Dazu bietet sich ein Vorgehen in mehreren, aufeinander aufbauenden Stufen der Rechnerunterstützung an. Basis ist die Einführung systemunabhängiger, elementarer, parametrierter Textbausteine, aus denen die Fehlersuchanleitungen mittels des Autorensystems zusammengebaut werden können.

Nach intensiven Versuchen mit einer prototypischen Implementierung für ein solches Redaktionssystem wurde im Projekt ein Basissystem spezifiziert, das in diesem Beitrag vorgestellt wird. Mit der Implementierung des operativen Programms wird der Bosch-Geschäftsbereich später (nach dem Ende von INDIA) ein Software-Haus beauftragen. Für die weiteren Stufen der Rechnerunterstützung werden Konzepte zur Adaption von Prüfplänen, sowie zum interaktiven bzw. automatischen Erzeugen und Testen von Prüfplänen dargelegt. In den Konzepten wurde darauf geachtet, daß der Erstellungsaufwand reduziert, eine bessere Nachvollziehbarkeit der Prüfpläne und eine Darstellung der Dokumente in unterschiedlichen Ausgabemedien sichergestellt wird. Voraussetzung für den Einsatz der avisierten modellbasierten Methoden in der Praxis ist neben einem geeigneten Diagnosewerkzeug die Verfügbarkeit von Modellbibliotheken, die die in den Anwendungen auftretenden Komponenten und Aggregate abdecken. Zum Aufbau von Modellbibliotheken mit vertretbarem Aufwand muß u.a. die Austauschbarkeit von Modellen mit standardisierten Schnittstellen zwischen den Unternehmensbereichen, die für verschiedene Abschnitte im Produkt-Lebenszyklus zuständig sind, gesichert werden (von der Entwicklung zum Service).

4 Diagnosing a Dynamic System with (almost) no Observations[1]

Peter Struss und Martin Sachenbacher – TU München, Institut für Informatik

Zusammenfassung

Wir stellen Anforderungen und Resultate aus einer Fallstudie für modellbasierte Werkstatt-(d.h. Offboard-) Diagnose des Hydraulikkreises eines Antiblockiersystems dar. Die auftretenden Probleme sind ziemlich grundlegender Natur: es ist praktisch nicht möglich, das Verhalten des Systems vorauszusagen, und außerdem sind keine Beobachtungen des tatsächlichen Verhaltens verfügbar. Beides zusammen könnte als fatal für das Verfahren der modellbasierten Diagnose gelten. Wir lösen dieses Problem, indem wir Modelle anwenden, die qualitative Abweichungen von Variablen bzw. Parametern von deren nominalen Verhalten erfassen. Diese Repräsentation erlaubt es, vage beschriebene Symptome wie z.B. „Bremspedal zu weich“ ausnutzen. Die Modelle werden in einem zustandsbasierten Diagnoseansatz benutzt, d.h. es wird nur jeweils ein beobachteter Zustand auf Konsistenz mit dem Systemmodell geprüft, aber keine Simulation des dynamischen Systemverhaltens durchgeführt. Der entscheidende Schritt, um dieses Verfahren anwendbar zu machen, ist dabei, grundlegende mathematische Beziehungen über Stetigkeit und Differenzierbarkeit auszunutzen, welche die direkt beobachteten Werte durch zusätzliche Informationen über deren Ableitungen ergänzen. Experimentelle Ergebnisse werden mit Expertenwissen aus FMEA-Dokumenten verglichen.

Abstract

We present requirements and results of a case study in model-based (off-board-)diagnosis of the hydraulic circuit of an anti-lock braking system. The primary problems to be addressed are quite fundamental: it is impossible to predict the behavior of the system and, additionally, there are no measurements of the actual behavior available. Both might be considered fatal for model-based diagnosis. We tackle these problems by applying models that capture qualitative deviations of variables and parameters from nominal behavior. They allow to exploit vaguely described symptoms such as “brake pedal too soft”. The models are used in a state-based diagnosis framework, i.e. only the observed states are checked for consistency with the model, and no simulation of the dynamic behavior is required. The crucial step in making the approach work is to exploit basic constraints on continuity for complementing the directly obtained observations by information about derivatives. Experimental results are compared to expert knowledge represented in existing failure mode and effects analysis (FMEA) documents and prove to be adequate.

4.1 Introduction

Commonly, the principle of consistency-based diagnosis of a device is described as follows:

- Take measurements of the actual device behavior.
- Predict the expected behavior of the device based on its model.
- Infer potential diagnoses from discrepancies derived from predictions and measurements.

But what do you do if

- you are not able to predict the device behavior?!

And, on top of it, if

- you have no measurements?!

We describe a case study that demonstrates how and to what extent (consistency-based) diagnosis can be performed even under these conditions. The problem was encountered in our work on (off-board) diagnosis of anti-lock braking systems (ABS). For a complete treatment of this car subsystem, we had to model and diagnose

[1] Dieser Beitrag ist erschienen im Tagungsband des 11. International Workshop on Qualitative Reasoning (QR97), Cortona, Italien, 1997.

its hydraulic circuit, in addition to the electrical circuit and the speed sensor which were covered by previous work ([Struss et al. 1995]).

Behavior prediction for the hydraulic circuit is a problem for two reasons:

- It is a **controlled** dynamic system. Model-based prediction would require to include a model of the complex behavior of the electronic control unit (ECU) of the ABS.
- Crucial **contextual influences**, such as road conditions, are **not measurable**, and, hence there is no sufficient input to prediction.

This leads to the second problem, measurability:

- There are **no sensors** in the hydraulic circuit.
- The only **observations** available are related to the behavior of the **wheels** and the **pedal**, and they are inherently **vague and qualitative** ('The pedal feels soft when pushed', 'The left front wheel tends to lock up'), in particular when reported by the driver.
- The occurrence of such symptoms is not temporally specified.

A description of the system and the diagnosis scenario is given in Section 4.2 of this chapter. Section 4.3 describes how the model addresses the problems:

- A qualitative model is required to enable the exploitation of the qualitative observations.
- The relative nature of the observations ('pedal too soft') and the lack of behavior predictions in absolute terms, lead us to the use of deviational models which capture how deviations in system parameters relate to deviations from some reference behavior, independently of a specification of this reference behavior.

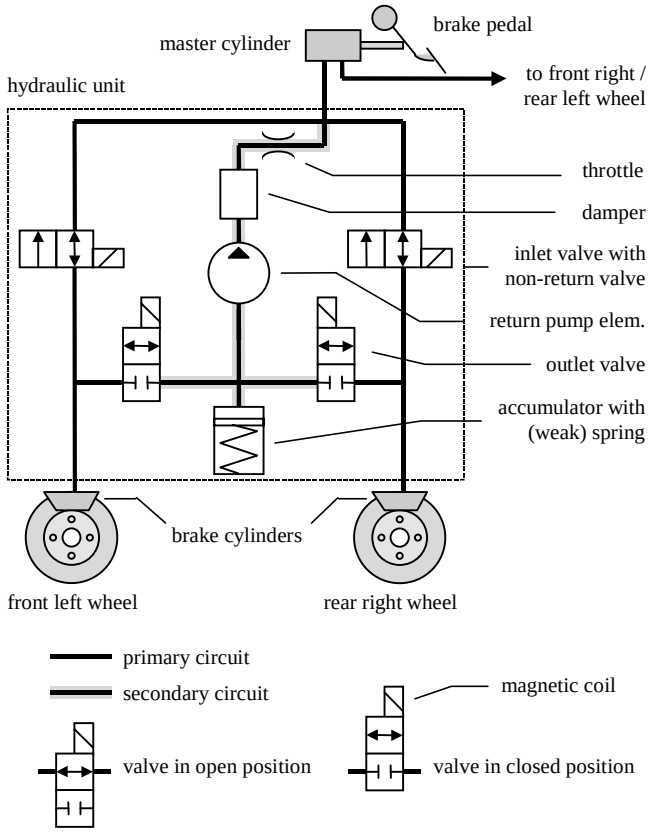


Figure 4-1: Hydraulic circuit of the ABS (diagonal distribution pattern)

The features of the utilization of this model by the diagnostic algorithm are discussed in Section 4.4:

- We apply **state-based diagnosis** which checks consistency of observed and modeled (qualitative) states only, taking into account that there is hardly any information about the dynamics in the observations and that predicting it is also infeasible.
- General (device-independent) constraints are exploited to **infer derivatives** of variables which turns out to be crucial for having state descriptions that are strong enough for state consistency checking.

We summarize results of an experiment in which the approach described above was applied to several fault scenarios extracted from a failure mode effects analysis document which represented available expertise and was used to provide success criteria. The results were basically positive, but depend on a number of assumptions and decisions. These assumptions can be considered reasonable in this application, but are far from being generally met preconditions which is why we discuss them at the end.

4.2 Application Domain

4.2.1 Anti-lock Braking System Hydraulics

The purpose of the anti-lock braking system (ABS) [Bosch 1996] is to prevent the wheels from locking up and, thus, to maintain the steerability and stability of the car while braking. The ABS consists of an

- electronic control unit (ECU),
- wheel-speed sensors and
- pressure-modulation valves.

The rotational speed of the wheels is measured and serves as an input to the control unit, which governs the valves and pump elements inside a hydraulic unit to reduce or increase the pressure exerted on the wheel brake cylinders.

The vehicle speed is estimated based on the wheel speeds of two diagonally opposite wheels. From this reference speed and the individual wheel speeds the ECU calculates the brake slip for each wheel, and, by combining this value with the (de-)acceleration of the wheel, it determines whether a wheel has a tendency to lock up. If this is the case, the control unit energizes the magnets of the pressure-modulation valves which control the brake pressure in the respective brake cylinders.

A typical ABS consists of two subsystems, each one operating on a pair of diagonally opposite wheels. As shown in Figure 4-1, the hydraulic circuit of each diagonal comprises

- four valves,
- two brake cylinders,
- a return pump element,
- an accumulator,
- a damper with throttle.

The pump elements of the two diagonals share one common drive motor. The hydraulic circuit is connected to the master cylinder that transforms a force acting on the brake pedal into increased pressure. To ensure that the pressure in the brake cylinders is never higher than the actual pressure in the master cylinder, inlet valves have built-in non-return valves. If the ABS is inactive, the braking system acts in the regular manner, maintaining the pressure on the brake cylinders while the pedal is pushed. In this mode, only the so-called primary circuit (see Figure 4-1) is active with the outlet valves closed. If the ABS is activated, reduction of pressure on the brake cylinders involves also the secondary circuit.

Control of the brake cylinders' pressure is achieved by stepping through different operation modes, as shown in Figure 4-2:

- **pressure-buildup**: for each wheel, an increase in pressure is achieved by an open inlet valve and a closed outlet valve, as in the regular braking mode.
- **pressure-holding**: the inlet valve is closed.

- **pressure-reduction**: the outlet valve is opened, and the accumulator fills quickly. Also, the return pump starts immediately to transport the fluid back towards the main cylinder.

If necessary, the brake pressure is then increased again to ensure that the wheel is not under-braked, and the next cycle may start.

- **pulsed pressure-buildup**: in some cases it might be useful to quickly interleave pressure-holding and buildup mode (the inlet valve receives a pulsed signal), to achieve a more smoothly raise of pressure.

The finite state machine shown in Figure 4-3 models in more detail the modes of the ABS control and the transition conditions which lead from one state to another. The essential condition is given by the wheel acceleration *a*, which is computed from the measured wheel speed v_w, and thresholds $a_1 < 0 < a_2 < a_3$. Additional variables are the vehicle speed v_{veh} and the pedal position s_{ped}, with thresholds v_0 and s_0, respectively, which serve as termination conditions to switch off the ABS control (because the driver stopped emergency braking or the vehicle speed has been reduced appropriately). In fact, this is still a description of a rather simple or simplified ABS control. It should be noted that the above-mentioned parameters are not available under workshop conditions, nor would they be particularly useful for diagnosis, as the behavior of the control loop depends to a large extent on (unknown) factors like the adhesion of the road surface, tire condition and vehicle load.

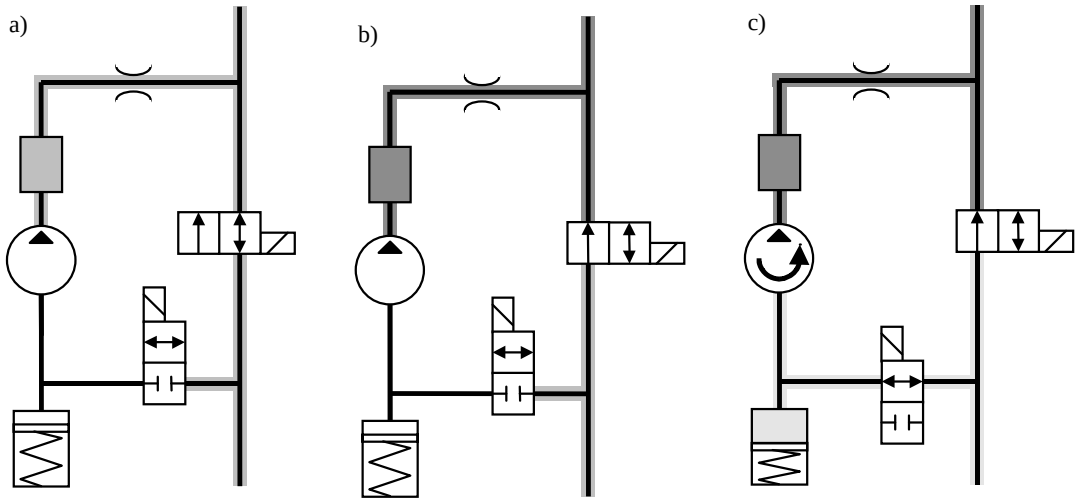


Figure 4-2: Operation modes of the ABS: a) pressure-buildup, b) pressure-holding and c) pressure-reduction

4.2.2 The Diagnostic Problem

The problem we address in our work is to support detection and localization of faults in the hydraulic circuit under workshop conditions. [Struss et al. 1995] reports our first results in model-based automation of the generation of diagnosis guidelines for the ABS. Usually, off-board diagnosis starts by reading information off the ECU. The ECU is equipped with built-in monitoring capabilities and produces error codes if it detects implausible signals. It already performs fault detection and a weak form of fault localization. However, this only applies to the electrical parts of the system. The reason is that there are simply no sensors, e.g. for pressure, in the hydraulic circuit (except for a sensor indicating that the brake fluid level has dropped below a critical threshold, which is of no help for immediate detection of misbehaviors and irrelevant for fault localization). Even in the garages, there exist no specific test-benches or analyzers that check the function of the entire ABS. Instead, information about pressures inside of the hydraulic circuit can only be obtained indirectly from observing the (de-)acceleration of the wheels.

However, it is not realistic that the driver or even a mechanic can exactly measure wheel acceleration or deceleration. As a result, diagnosis of the hydraulic subsystem has two major sources of information:

- **Symptoms reported by the driver**. Except for a lit control lamp, all a driver can perceive is some unexpected behavior of the vehicle w.r.t. braking and steering. This bears a chance of being translated into features of the individual wheels, perhaps a suspect response by the brake pedal, and possibly some sounds. For instance, typical observations could be that a wheel tends to lock up (indicating too high pressure in the respective brake cylinder) or that the brake pedal is too soft (as a result of an unusually low pressure in the

main cylinder). We should emphasize however, that it is not guaranteed that an ordinary driver is able to provide even this kind of information, simply because most drivers do not gain extensive experience with braking in ABS mode. It can be produced more reliably by the mechanic in the workshop.

- **Tests** under defined operation modes **in the workshop**. With the same tool that is used to read off the error codes of the ECU, each operation mode of the ABS can be activated individually. The test e.g. for the pressure-holding phase consists of pushing the brake pedal while the pressure-holding mode is activated and checking if the respective wheel can still be moved freely, indicating that the system could indeed maintain the low brake cylinder pressure.

A typical diagnosis scenario which we will use for illustration in the following sections is that when braking,

- the car is yawing to the right, while
- the brake pedal feels somewhat harder than normal.

We assume that the first symptom can be refined to

- under-braking at the left-hand and
- over-braking at the right-hand side.

Actually, the symptoms are taken from a failure mode and effects analysis for the ABS. This analysis is carried out during the design of a system and lists a number of possible component faults such as clogged or enlarged valve profiles, valves stuck open or punctured, a defective pump element or air included in the circuit, together with their (potential) effects (e.g. the symptoms stated above).

We emphasize that observations like the ones mentioned above

- are **qualitative** in nature,
- are sparse, and only **indirectly related** to the important **internal system variables**, and
- have an **unspecified temporal extent**.

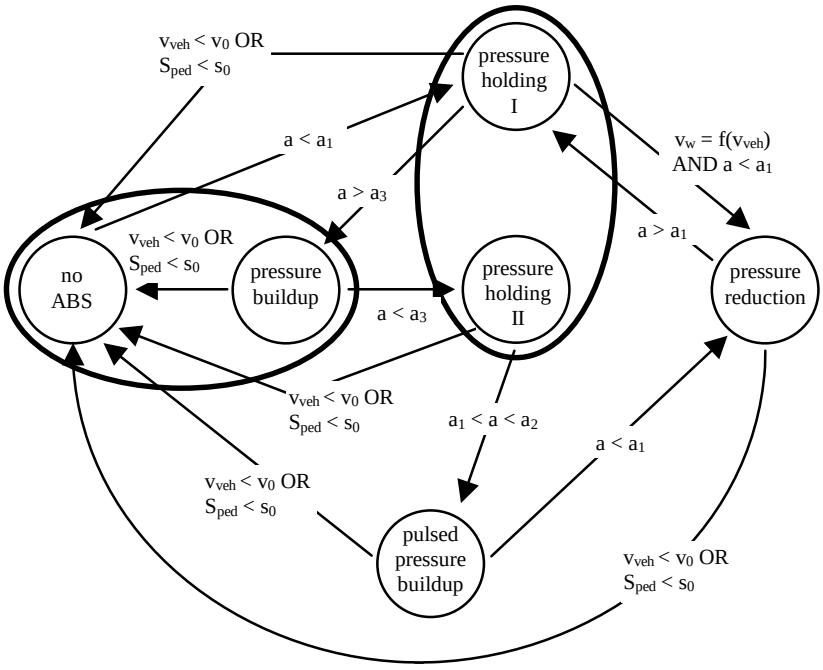


Figure 4-3: States and transition conditions of the ECU

For instance, under-braking is a phenomenon that characterizes the behavior of a wheel over the entire period of braking, not even related to a particular phase. Only under the described testing conditions in the workshop, observations can be associated with the operation mode of the test, but even then, no detailed temporal aspects can be measured.

This fact contrasts sharply with the dynamic behavior of the device, thus making the diagnosis problem a real challenge for modeling and automated diagnosis. However, a human observer who is familiar with the components of the circuit and has a basic understanding of the functionality of an ABS as given in this section, is able to come up with reasonable diagnostic hypotheses; e.g. based on the symptoms of our example scenario:

Under-braking on the left-hand side indicates insufficient pressure. This could be due to a clogged inlet valve or an open outlet valve. The former would also explain why too high pressure remains in the master cylinder (hence the hard pedal) and in the primary circuit of the right-hand wheel (possibly causing over-braking). So, this seems to be a plausible diagnosis.

The question is what is required to perform this kind of reasoning in an automated diagnosis system. The following sections present our answer to this question, first, regarding the modeling formalism and, second, the diagnostic procedure.

4.3 Qualitative Deviation Models

4.3.1 Models of Hydraulic Components

In response to the nature of the observations, we adopted an approach that states models in terms of qualitative deviations of variables and parameters from some unspecified and even potentially changing nominal value.

The above-mentioned failure causes and effects qualitatively describe deviations of component parameters from such a nominal value (e.g. "inlet valve profile enlarged") or deviations of system variables from values one would expect normally (e.g. "over-braked"). The successful use of models which capture the qualitative relations between such discrepancies has already been presented in [Malik Struss 1996]. We briefly summarize the foundations of this approach.

Each variable domain is limited to signs

$$[x] := \text{sign}(x),$$

especially derivatives of time

$$\partial x := [dx / dt],$$

and the deviation of an actual value from its reference value

$$\Delta x := x_{act} - x_{ref}.$$

In this case, the reference value is defined as the value that would occur under normal behavior of the ABS given the same situation in terms of road condition, force exerted on the pedal etc. Obviously, there is no way of explicitly specifying these values of a nominal behavior and their changes over time because of the unmeasurable or even unknown context. There are two basic insights: first, even under these conditions, it can be possible to predict the effect, or, likewise, the potential cause of a deviation in one system variable, and, second, many possible faults can be characterized in terms of parameters deviating from their nominal values. For instance, a clogged valve can be described by its profile, A, being smaller than normal: $\Delta A<0$. Models capturing the relationships of such qualitative deviations can be generated from the equations that describe the normal behavior of the respective components.

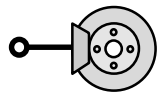
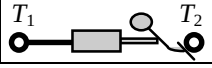
Component	Symbol	Constraints
wheel (with brake cylinder)		$T.[\Delta p] = -\partial \Delta v_w$
brake pedal		$T_1.[Q] = T_1.[p] \ominus T_2.[f]$; $T_1.[Q] \oplus T_2.\partial s = 0$ $T_1.[\Delta Q] = T_1.[\Delta p] \ominus T_2.[\Delta f]$; $T_1.[\Delta Q] \oplus T_2.\partial \Delta s = 0$

Figure 4-4: Qualitative model fragments for the brake pedal and wheels

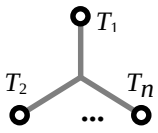
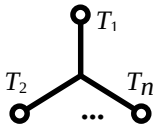
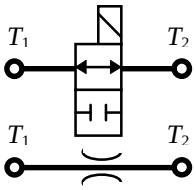
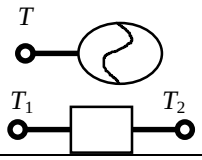
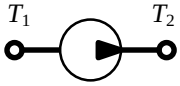
Component	Symbol	Constraints
n-node		$T_1.[p] = T_2.[p] = \ldots = T_n.[p]$; $\sum_{i=1}^{n} T_i.[Q] = \partial p \otimes [\beta_T]$ $T_1.[\Delta p] = T_2.[\Delta p] = \ldots = T_n.[\Delta p]$; $\sum_{i=1}^{n} T_i.[\Delta Q] = \partial p \otimes [\Delta\beta_T] \oplus \partial\Delta p \ominus \partial\Delta p \otimes [\Delta\beta_T]$
n-node (with invariant compressibility)		$T_1.[p] = T_2.[p] = \ldots = T_n.[p]$; $\sum_{i=1}^{n} T_i.[Q] = \partial p$ $T_1.[\Delta p] = T_2.[\Delta p] = \ldots = T_n.[\Delta p]$; $\sum_{i=1}^{n} T_i.[\Delta Q] = \partial\Delta p$
resistive element e.g. valve, throttle		$T_1.[Q] = [A] \otimes (T_1.[p] \ominus T_2.[p])$; $T_1.[Q] \oplus T_2.[Q] = 0$ $T_1.[\Delta Q] \oplus T_2.[\Delta Q] = 0$ $T_1.[\Delta Q] = [A] \otimes (T_1.[\Delta p] \ominus T_2.[\Delta p])$ $\oplus [\Delta A] \otimes (T_1.[p] \ominus T_2.[p])$ $\ominus [\Delta A] \otimes (T_1.[\Delta p] \ominus T_2.[\Delta p])$
volume element e.g. accumulator, damper		$T.[p] = [p]$; $T.[Q] = \partial p$ $T.[\Delta p] = [\Delta p]$; $T.[\Delta Q] = \partial\Delta p$ $T_1.[p] = T_2.[p]$; $T_1.[Q] \oplus T_2.[Q] = \partial p$ $T_1.[\Delta p] = T_2.[\Delta p]$; $T_1.[\Delta Q] \oplus T_2.[\Delta Q] = \partial\Delta p$
pump element		$T_1.[p] = [+] \Rightarrow T_1.[Q] = [D]$; $T_1.[p] = 0 \wedge [D] \neq [-] \Rightarrow T_1.[Q] = 0$ $T_1.[p] = 0 \wedge [D] = [-] \Rightarrow T_1.[Q] = [-]$ $T_1.[Q] \oplus T_2.[Q] = 0$

Figure 4-5: Qualitative model fragments for basic hydraulic components

Component	Quantitative equations
conduit with zero resistance	$p_1 = p_2$ $Q_1 + Q_2 = 0$
resistive element	$Q = kA\sqrt{\lvert p_1 - p_2 \rvert}\ \mathrm{sign}(p_1 - p_2)$ $Q_1 + Q_2 = 0$
volume element with compressibility	$Q = k\frac{dp}{dt}\beta_T$
volume element without compressibility	$Q = k\frac{dp}{dt}$

Table 4-1:Basic equations for hydraulic elements

Table 4-1 lists a number of basic equations for types of hydraulic elements. They are meant to model basic aspects of hydraulic components and can be combined to describe a particular component type. Using the operators, [.], ∂, and Δ, defined above, qualitative deviation models of components can then be derived from these equations. Figure 4-5 shows the resulting qualitative models of basic hydraulic component types, whilst Figure 4-4 lists additional ones for ABS-specific components. In the notation, Q stands for flow, p for pressure, A for profile area and D for the pump delivery rate, whereas k and β_T are (material-dependent) factors. Thus, in the notation of the model fragments, e.g. $T_1.[p]$ denotes qualitative pressure at terminal T_1.

4.3.2 Coding of Observations

The inherently vague and qualitative observations of the ABS behavior can now be captured by our modeling formalism. For the symptoms of the scenario described in Section 4.2, we obtain the following translations:

- under-braked left-hand wheel, i.e. it rotates faster than under normal conditions: $[\Delta v_L] = [+]$.

- over-braked right wheel, i.e. it rotates slower than expected: $[\Delta v_R] = [-]$.
- too hard brake pedal, which, given the usual pedal force, moves a shorter distance than normally: $\partial\Delta s_{PED} = [-]$.

Together with the characterization of the operation modes of the hydraulic circuit given in terms of states of valves and pump, these observations represent the only directly and somewhat reliably available input for model-based prediction and consistency-checking.

4.4 Using the Model for Consistency-based Diagnosis

4.4.1 State-based Diagnosis

Consistency-based diagnosis requires checking whether observations about the actual device behavior are consistent with the behavior predicted by a model:

$$\text{model} \cup \text{OBS} \overset{?}{\vdash} \perp .$$

For fault detection, the system checks the model of correct behavior. Fault localization is based on identifying inconsistent subsets thereof. Fault identification is done by checking models of faulty behavior for consistency with observations.

Since, in our domain, we have neither a chance to predict the dynamics reliably, nor a way to observe changes over time, we cannot perform what is often associated with the task of diagnosis of dynamic systems: tracking of the actual behavior over time and simulation of the modeled behavior. In previous work ([Malik Struss 1996], [Struss 1997]), we have shown that, in theory and practice, checking only the observed states (rather than the temporal behavior) for consistency with the device model often suffices to obtain the desired diagnostic results and that, under certain conditions, these results are even equivalent to the ones generated by simulation-based approaches. In our case, we have no choice but trying to apply **state-based diagnosis**.

Stated more systematically, this means that we ignore part of the model, namely the part that captures the laws of evolution over time (which, in practice, is often implicit in the predictive engine): if the model is divided into a set of constraints on the permissible states and a set of constraints expressing rules of continuity, integration, and derivatives („CID“),

$$\text{model} = \text{state-constraints} \cup \text{CID-constraints},$$

then we confine the consistency check to

$$\text{state-constraints} \cup \text{OBS} \overset{?}{\vdash} \perp .$$

It turns out that the observations together with the model do not suffice to generate appropriate conflicts. The "measurements" characterized above, enable the models to infer deviations in the pressure in different parts of the circuit. This, trivially, suffices to establish measurability for fault detection, but not measurability for fault identification and localization. The reason lies in the lack of information about the derivatives of pressures, which cannot be provided by the observations and the constraints of the model fragments alone. Basically, this information would help to detect significant inconsistencies because resistive elements like valves relate flow to pressure, whereas pipes and other containers link flow and derivatives of pressure and, respectively, their deviations.

In our example („yawing to the right“), the observation about the pedal, $\partial\Delta s_{PED} = [-]$, allows to infer a positive deviation of the pressure in the master cylinder (see Figure 4-6 for reference),

$$[\Delta p_{MC}] = [+],$$

and from the under-braked left-hand wheel, we obtain a lower pressure in the respective wheel brake cylinder,

$$[\Delta p_{WBC}] = [-].$$

From this information, the model left inlet valve can predict an increased flow across the valve:

$$[\Delta Q_{LIV}] = [+],$$

which does not establish any contradiction. What is it that makes us not feel comfortable with this situation? Well, the state description obtained may be consistent with the (part of the) model. However, an increased flow across the valve causes the pressure in the wheel brake cylinder to **rise** which conflicts with the **reduction** in pressure in this component. In other words, we squeeze more information out of the observations of the variables, namely information about their derivatives.

4.4.2 Adding CID-Constraints

If we would like our system to perform this kind of reasoning, we have to exploit additional knowledge which can compensate for this limited measurability. This is actually implied by the constraints we dropped in the previous subsection, *CID-constraints*. However, they are not used for simulation of correct or faulty behavior modes (for integration), i.e. by drawing inferences based on

state-constraints ∪ CID-constraints.

Instead, they are applied to complement the observations with derivative information, i.e. we combine

CID-constraints ∪ OBS.

More specifically, a version of the following theorem is applied:

Theorem 1

> *Let f(t) be a continuously differentiable function and $t_0<t_e$. If $f(t_0)=0$ and $f(t)>0$ for $t\in(t_0, t_e)$ then $\exists\, t_1$ such that $t_0 < t_1< t_e$ and $df(t)/dt > 0$ for $t\in(t_0, t_1)$.*

Or, stated in its qualitative version,

CID_1

> *If $[f(t_0)]=[0]$ and $[f(t)]=[+]$ for $t\in(t_0, t_e)$ then $\exists\, t_1$ such that $t_0 < t_1< t_e$ and $\partial f(t)=[+]$ for $t\in(t_0, t_1)$.*

Informally, this says: if a variable is zero initially and then becomes positive in an interval, there must exist an initial (but potentially shorter) time interval during which both the variable and its derivative are positive (no matter what happens after this interval). This rule and other variants of it can be encoded as constraints and used to create state descriptions that contain information about derivatives, in our case about qualitative derivatives of deviations. Because the observations themselves are not explicitly related to specific time periods, the same holds for this derived information. We need to state more properties of the problem domain, or introduce more assumptions.

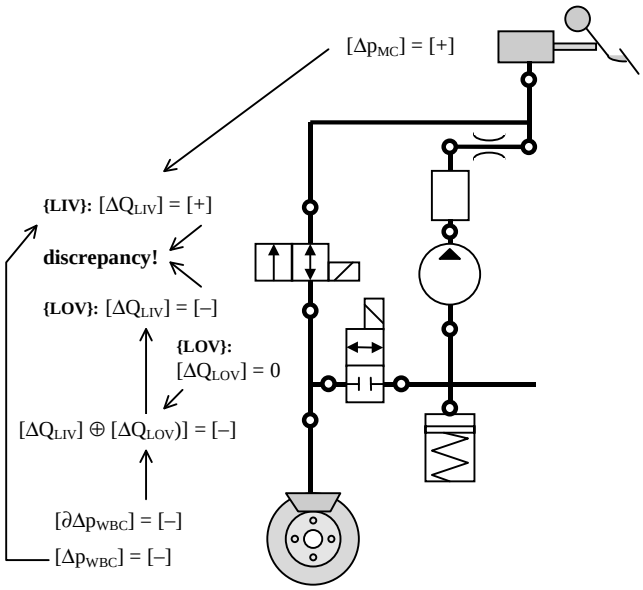


Figure 4-6: Inferences at left wheel brake cylinder

4.4.3 Adding Assumptions

Checking consistency of the set of observations with the *state-constraints* makes only sense if the individual observations refer to the same state. This means we need to assume that those **observations** that, together with a part of the model, establish a **discrepancy**, actually occur during **overlapping time intervals**.

The intervals (t_0, t_1) introduced by the CID_1 rule for the different deviations must have a non-empty intersection. Note, that we do **not** have to postulate that **all** existing deviations have to occur at the same time, but only those that are used to detect one discrepancy. In our application, we make an assumption that entails the first one, namely that the related effects of faults occur at the beginning of some phase (operation mode of the ABS), which provides the „synchronizing" initial time point t_0. This means, the phase starts with no deviation : $[\Delta f] = 0$ at t_0.

Furthermore, in this case study, we make the assumption that only valves, throttles with dampers or pump elements can be faulty.

4.4.4 Using Models and CID_1 for Prediction

In this subsection, we illustrate how the approach described, namely

- state-based diagnosis with
- qualitative deviation models and
- observations extended through CID-constraints,

works on our example (see Figure 4-6 for reference). Recall that the initial observations were:

- under-braked left-hand wheel: [ΔvL] = [+].
- over-braked right wheel: [ΔvR] = [–].
- too hard brake pedal: ∂ΔsPED = [–].

Under the assumption that the pressure-buildup phase started at t_0 with no deviation, CID_1 yields that

$$[\Delta v_L] = 0 \text{ at } t_0 \wedge [\Delta v_L] = [+] \text{ after } t_0$$
$$\Rightarrow [\partial \Delta v_L] = [+] \text{ after } t_0,$$

From this, the model of the left wheel brake cylinder infers

$$[\partial \Delta v_L] = 0 \text{ at } t_0 \Rightarrow [\Delta p_{WBC}] = 0 \text{ at } t_0$$
$$[\partial \Delta v_L] = [+] \text{ after } t_0 \Rightarrow [\Delta p_{WBC}] = [-] \text{ after } t_0.$$

By applying CID_1 again, we establish a link between this deviation of the brake cylinder pressure and the deviation of its derivative:

$$[\Delta p_{WBC}] = 0 \text{ at } t_0 \wedge [\Delta p_{WBC}] = [-] \text{ after } t_0$$
$$\Rightarrow [\partial \Delta p_{WBC}] = [-] \text{ after } t_0.$$

The model of the node that joins the terminals of wheel brake cylinder, left outlet valve and left inlet valve then derives

$$[\partial \Delta p_{WBC}] = [-] \text{ after } t_0$$
$$\Rightarrow [\Delta Q_{LOV}] \oplus [\Delta Q_{LIV}] = [-] \text{ after } t_0.$$

The correct behavior of the left outlet valve in the pressure-buildup mode states that

$$[\Delta Q_{LOV}] = 0 \text{ at } t_0 \text{ and after } t_0.$$

This, together with the model of the node, yields a negative deviation of the flow through the left inlet valve:

$$[\Delta Q_{LIV}] = [-] \text{ after } t_0.$$

Note that we cannot determine the actual direction of flow, i.e. it is not possible to distinguish whether there is a flow from the master cylinder to the wheel brake cylinder which is smaller than usual, or there exists an increased fluid flow in the opposite direction.

From the observation at the brake pedal, the pedal model infers that the pressure in the master cylinder deviates in positive direction:

$$[\partial\Delta s_{PED}] = [-] \text{ after } t_0 \wedge [\Delta f_{PED}] = 0 \text{ after } t_0$$
$$\Rightarrow [\Delta p_{MC}] = [+] \text{ after } t_0.$$

The increased pressure of the master cylinder and the decreased left wheel brake cylinder imply an increase in pressure drop across the left inlet valve:

$$[\Delta p_{MC}] \ominus [\Delta p_{WBC}] = [+] \text{ after } t_0.$$

With the information that the left inlet valve is opened in the pressure-buildup phase, the model of the valve predicts a positive deviation of flow through the component:

$$[\Delta Q_{LIV}] = [+] \text{ after } t_0.$$

This contradicts the negative deviation of flow that has been inferred first, and a discrepancy is detected with the underlying conflicting correctness assumptions

{left inlet valve, left outlet valve},

i.e. one of these components must be broken. The observations for the right-hand wheel brake cylinder can be processed in a similar manner and reveal a second conflict

{left inlet valve, right inlet valve, right outlet valve, throttle, pump element}.

These two conflicts suffice to produce the **left inlet valve** as the **only** possible **single fault** candidate and a number of potential double faults.

4.4.5 Adding Domain Axioms

In principle, to further refine conflicts, we could use fault models for the components in the style of GDE$^+$ ([Struss Dressler 1989]). The problem is that the models stated above can only derive deviations, but not the direction of flow through a component. However, meaningful fault models would require actual directions of pressure drops and flow. For example, a valve with no deviation of the pressure drop but a flow which is too low is consistent both with a too low and too high valve profile, if the direction of flow is unknown. Deriving this information would need a richer domain than just signs.

Instead, we adopted an approach using domain axioms to further refine the conflicts. The domain of the profile A of a valve is {0, [+]}, and its deviation ΔA can either be negative, zero or positive. We make model-based prediction more complete by adding the disjunctions of values local to components occurring in a conflict. It turns out that in our example, the right inlet valve then does not contribute to the second conflict in the sense that for each combination of values for A and ΔA, we obtain an inconsistency with the rest of the components in the conflict. Therefore, the conflict is reduced to

{left inlet valve, right outlet valve, throttle,
pump element}.

With this reduced conflict, we get

{left inlet valve}

as the only single fault candidate, as before, and

{left outlet valve, right outlet valve},
{left outlet valve, throttle},
{left outlet valve, pump element}

as the possible double faults. Indeed, the system successfully inferred a fault in the left inlet valve from the failure effects listed for a clogged left inlet valve in the FMEA.

4.5 Empirical Results

We carried out a number of experiments for several relevant failures (Table 4-2). They were selected based on an existing failure mode and effects analysis of the system. The guiding criteria which led to this selection were on the one hand the estimated probabilities of occurrence (as stated in the FMEA), and on the other hand concrete experience of workshop technicians. In addition, wrong mounting of the device or leaks are also relevant in practice. However, most likely, leaks would trigger the switch for the level of brake fluid and activate a warning lamp before affecting the functionality of the ABS.

Failure cause	**Failure effect**	
inlet valve profile clogged	pressure increase rate too small	under-braking of the respective wheel, over-braking of other wheels possible, hard braking pedal, worst case: car yawing
inlet valve stuck open or punctured	pressure retaining not possible	too high retardation on one wheel due to pressure on main cylinder, wheel tends to lock up
outlet valve stuck open or punctured	pressure retaining not possible	accumulator gets filled, pedal has to be moved a greater distance, braking less effective on diagonally opposite wheels
pump element defective	low pressure level not achieved	affected wheels tend to lock up
hydraulic unit not properly vented	air in primary circle	under-braking on the affected diagonally opposite wheels, pedal soft

Table 4-2: Failure effects for the hydraulic unit from a system FMEA

For evaluation of the models and the approach, the symptoms of a particular failure cause listed by the FMEA were fed into the diagnosis procedure, and the success criteria was whether the respective cause occurred in the candidates generated and how well it could be isolated. The approach turned out to be fairly successful: for each sample of failure effects, fault localization was successful in the sense that the respective component failures were included in the set of single fault diagnoses, sometimes being the only possible single fault (Table 4-3).

Observation: Failure effects for fault	**Candidates generated**
left inlet valve profile clogged	**{left inlet valve}**, {left outlet valve, right outlet valve}, {left outlet valve, pump element}, {left outlet valve, throttle}
left inlet valve stuck open or punctured	**{left inlet valve}**, {left outlet valve, right inlet valve}, {left outlet valve, right outlet valve}, {left outlet valve, pump element}
left outlet valve stuck open or punctured	**{left outlet valve}**, {right outlet valve}, {throttle}, {pump element}
pump element defective	**{pump element}**, {left outlet valve}, {right outlet valve}, {right inlet valve}, {damper}
hydraulic unit not properly vented	**{compressibility}**, {throttle}, {pump element}, {left outlet valve}, {right outlet valve}

Table 4-3: Candidates generated with the failure effects as observations

4.6 Discussion

This case study extends the list of pieces of evidence that state-based diagnosis can very well suffice to diagnose dynamic systems (related work is described in [Dressler 1996], [Chantler et al. 1996], [Malik Struss 1996]). In [Struss 1997], we present a more formal analysis of preconditions and limitations to this approach. Much more

work is needed to develop good designs and criteria for it to be advantageous. This will require a more detailed analysis of the form and contents of the *CID-constraints* and their possible applications and relating their results to various limitations in measurability.

Our example also demonstrates that very weak qualitative observations can be exploited to get close to human diagnostic results under such conditions. This is a benefit of qualitative modeling combined with deviation models. However, we have to make a number of fairly strong assumptions to make diagnosis work, in particular, to compensate for the unspecified temporal scope of the observations. The assumption that the occurrence of symptoms is synchronized appears questionable, especially if we take multiple faults into consideration. Also, in our example, we presented diagnosis only for the pressure buildup phase. This is reasonable, but, in principle, deviating pressure could also result from a malfunction that affects the pressure-reduction phase. A diagnostic system would either need to exhaustively perform the analysis for all phases or require some (model-based) reasoning to pick the informative phases.

Acknowledgments

We would like to thank the members of the model-based systems and qualitative reasoning group Munich, O. Dressler, U. Heller, A. Malik and J. Mauss for their valuable contributions and T. Beschta, G. Biswas, M. Chantler, P. Nayak, and B. Williams, as well as many participants of the QR-96 and DX-96 workshops for interesting discussions and contributions. This work was supported in part by the Commission of the European Union (Project VMBD, #BE 95/2128) and by the German Ministry of Education and Research (# 01 IN 509 41).

5 Fundamentals of Model-Based Diagnosis of Dynamic Systems[2]

Peter Struss – TU München, Institut für Informatik

Kurzfassung

Dieser Artikel diskutiert theoretische Grundlagen und praktische Aspekte der Anwendung modellbasierter Diagnose (insbesondere konsistenzbasierter Diagnose) auf dynamische Systeme. Viele Ansätze dafür nehmen als gegeben an, daß dies die Simulation des zu diagnostizierenden Systems erfordert. Wir zeigen Voraussetzungen dafür auf, den oft unverträglich hohen Aufwand des Simulationsschritts zu umgehen. Diese Voraussetzungen sind Anforderungen an das Modell und die modellbasierte Verhaltensanalyse, sowie die Beobachtbarkeit des Systems. Die Resultate liefern Entwurfskriterien für Modelle und Diagnosesysteme und eine Basis für die Behandlung neuer, wesentlicher Anwendungsklassen. Dies wird illustriert durch eine Fallstudie zur Diagnose der Hydraulik eines Antiblockiersystems (ABS).

Abstract

The chapter discusses theoretical foundations and practical aspects of applying model-based diagnosis (particularly consistency-based diagnosis) to dynamic systems. Many approaches to this task take it for granted that it requires simulation of the system being diagnosed. We present conditions for avoiding the often prohibitively expensive step of simulation, which are stated as properties of the model and the predictive algorithm and the observability of the system. The results provide design criteria for models and diagnostic systems and a foundation for tackling new significant types of applications. This fact is illustrated by a case study on diagnosis of the hydraulic circuit of an anti-lock braking system.

5.1 Introduction

Model-based diagnostic systems are becoming fairly successful and starting to address industrial applications. Many, if not most, systems can be regarded as some variant of the *General Diagnosis Engine* (GDE), following the principle of *consistency-based diagnosis* ([Dressler Struss 1996]):

- In order to perform fault detection and fault localization, check whether the observations about the actual device behavior are consistent with the behavior predicted by a model of the correct device (or some part of it).
- For fault identification, check consistency of observations with models of faulty behavior.

For static systems (or, rather, systems represented by a static model), such as combinatorial circuits, this amounts to checking satisfiability of a set of constraints, representing the observed state(s) and the state constraints of the device components.

In principle, of course, consistency-based diagnosis also applies to dynamic systems, but it requires checking the consistency of device **behaviors over time** with behaviors allowed by a dynamic model of the device. At a first glance, this inevitably demands

- tracking of the actual behavior over time and
- simulation of the modeled behavior

in order to check consistency of the results. This idea underlies several approaches (e.g. MIMIC, [Dvorak Kuipers 1992]) and has, on the other hand, probably prevented practical solutions to challenging problems. The reason for the latter is that the simulation task and the comparison of behavior sets can often be prohibitively expensive. This holds, in particular, when qualitative simulation yields ambiguous results and when fault identification requires simulation of many fault scenarios.

---

2 Dieser Beitrag ist erschienen in: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan, August 23-29, pp. 480-485, 1997.

Other approaches, such as [Dressler 1996], avoid simulation and generate diagnostic candidates based on checking consistency of the model with observed **states** only, (as opposed to observed **behaviors**, i.e. sequences of states); but so far a formal analysis of their preconditions and consequences is lacking.

Consistency of observed states with the model is a necessary condition for the observed behavior to be consistent with the model. Are there conditions under which this is **sufficient**? [Malik Struss 1996] states a condition for the equivalence of state-based and simulation-based diagnosis without giving a proof.

This chapter discusses theoretical foundations of model-based predictors and consistency-based diagnosis (Section 5.2) in order to derive criteria for the utility and equivalence of the approaches. This is done taking **practical** aspects of diagnosis into account, in particular measurability (Section 5.3), and with the goal of designing diagnostic **algorithms** appropriately (Section 5.4). This is illustrated by a case study on diagnosis of the hydraulic subsystem of an anti-lock braking system (Section 5.5).

Many of the results are fairly fundamental. In fact, so fundamental that it is surprising they have never been spelled out in the literature, even more since they have a considerable **practical** impact on diagnostic systems.

5.2 Theoretical Foundations

5.2.1 Consistency-based diagnosis

As pointed out in the introduction, the key lies in checking whether or not a set of observations, OBS, contradicts the models of certain *behavior modes* of the device, the correct behavior (to perform fault detection) or faulty behaviors (for fault identification) which result from certain component faults and/or structural faults in the device. If such a behavior model of a mode, *model(mode)*, and the set OBS are considered as logical theories, the task is to check their joint consistency:

$$\textit{model(mode)} \cup \text{OBS} \overset{?}{\vdash} \bot .$$

If the behavior model does not capture temporal aspects, it is a set of constraints, *state-constraints*, that restricts the set of states possible under the respective mode. Alternatively, we can think of it as being represented by this set, *STATES(mode)*. For our purposes, we have to extend the concepts.

5.2.2 Model-based prediction of behaviors

To characterize the evolution of a behavior over time, the behavior model does not only have to constrain the states, but also **relations between states across time**:

$$\textit{model(mode)} = \textit{state-constraints(mode)} \cup \textit{temp-constraints(mode)}.$$

The domain, DOM($\underline{v}$), of the vector $\underline{v}$ of all variables describes the set of all (theoretically) possible states. We call the triple

$$(v, \text{DOM}(\underline{v}), T) ,$$

where T is a universe of time instances, a *representation*(al space) for modeling. A *behavior* is seen as specifying the state at each time instance and, hence, can be defined in this representation as a mapping

$$\textit{behvr}: T \rightarrow \text{STATES} \subseteq \text{DOM}(\underline{v}),$$

or, alternatively, as the graph of this mapping

$$\{(t, \underline{v}(t)) \mid t \in T\}.$$

Then *state-constraints* restricts STATES, whilst *temp-constraints* restricts the possible mappings, *behvr*, or their graphs.

We need a clear understanding of what kinds of constraints go into the different parts of the model. For instance, if we choose a set of ordinary differential equations (ODEs) to model a behavior mode, then this constitutes the *state-constraints*. This may seem surprising, since, after all, ODEs are supposed to express temporal behavior. However, the equations themselves only restrict the values of the involved variables and their derivatives at each time point. Restrictions on the temporal evolution of the described system are based on rules that capture properties of **continuity, integration, and derivatives** (CID). Usually, these restrictions are only implicitly

represented in procedures, e.g. for numerical integration. For our analysis, we have to make them explicit in a set which we will call *CID-constraints*. In qualitative simulation, e.g. the system QSIM used in MIMIC, the qualitative versions of these restrictions are often captured by so-called transition rules which list the admissible pairs of neighboring states.

It is important to characterize the different form and contents of the different parts of the model: *state-constraints* captures the **specifics of the device** under a certain behavior **mode**. In contrast, *CID-constraints* comprises **general** constraints that are **independent** not only of the mode, but even **of the device** (and of time). Regarding the form, the former constraints relate **different variables** (including derivatives) at **one time** instance, whereas the latter, at least in their pure form, constrain values and derivatives of **only one variable across time**.

This splitting of dynamic models into two orthogonal sets of constraints that limit the possible states and the possible evolution of states, respectively, suggests that consistency of a behavior with a model can be checked by checking both aspects independently (as indeed done in QSIM) and motivates the following definition.

Definition (Separable Model)

A dynamic model *model(mode)* is called separable, if the following holds:
A behavior, *behvr*, is an admissible behavior under the model
if and only if
its states are admissible under *state-constraints* and
it satisfies its *temp-constraints.*

Formally:

$\{behvr\} \cup model(mode) \nvdash \bot$ iff

$\{behvr\} \cup state\text{-}constraints(mode) \nvdash \bot$ and

$\{behvr\} \cup temp\text{-}constraints(mode) \nvdash \bot$.

A behavior model's *temp-constraints* can contain more than the *CID-constraints*. This is the case when it specifies changes of a variable over time which are independent of the *CID-constraints* or even contradict them. The former case is illustrated by a valve's state switching between *OPEN* and *CLOSED*, the latter happens when the same situation is modeled by the valve's opening area discontinuously changing from 0 to a positive value and vice versa. The *CID-constraints* for the directly or indirectly affected variables have to be suspended while such a change occurs. Such constraints on transitions which are specific for the device and usually also vary with the behavior mode will be denoted

trans-constraints(mode),

and we obtain a partitioning of the dynamic part of a model:

temp-constraints(mode) = trans-constraints(mode)
$\cup$ *CID-constraints,*

where the latter possibly hold only with exceptions introduced by the *trans-constraints.*

Often, *trans-constraints* will violate the conditions of the above definition of a separable model, because they may mix state restrictions and temporal restrictions. For instance, difference equations, which belong to *trans-constraints*, usually do so. We will define a continuous behavior model through the absence of such constraints.

Definition (Continuous Behavior Model/System Description)

A behavior model *model(mode)* is called strictly continuous, if
it is separable and

trans-constraints(mode) $= \varnothing$, i.e.
model(mode)=
state-constraints(mode) $\cup$ *CID-constraints*.

A system description {*model(mode)*} is called strictly continuous, if all its models are.

From the above discussion, it follows that we consider the condition of a separable model in this definition as naturally satisfied if the dynamic part is confined to the *CID-constraints*. Strictly continuous system descriptions have an interesting property: since the *CID-constraints* do not depend on the mode, all models share the same *temp-constraints*. We call this property homogeneity.

Definition (Homogeneous Dynamic Models/System Description)

Two models

$$model(mode_i) = state\text{-}constraints(mode_i) \cup temp\text{-}constraints(mode_i),\ i=1,2,$$

are called homogenous, if

$$temp\text{-}constraints(mode_1) = temp\text{-}constraints(mode_2).$$

A system description {*model(mode)*} is called homogeneous, if any two models are homogeneous.

This property turns out to have an important impact on discrimination among modes and, hence, diagnostic algorithms. This is indicated by the following proposition.

Proposition (State and Behavior Equivalence)

Two homogeneous models, and, hence also two strictly continuous models in the same representation,

$$model(mode_1),\ model(mode_2),$$

share all states if and only if they share all behaviors:

$$STATES(model(mode_1)) = STATES(model(mode_2)) \text{ iff}$$
$$BEHAVIORS(model(mode_1)) = BEHAVIORS(model(mode_2)).$$

The proof is trivial: if the STATES, i.e. the logical models of $state\text{-}constraints(mode_i)$, are equal, then

$$state\text{-}constraints(mode_1) \Leftrightarrow state\text{-}constraints(mode_2),$$

and, due to homogeneity,

$$model(mode_1) \Leftrightarrow model(mode_2),$$

which implies equal sets of behaviors. The other direction is obvious.

Consistency-based Diagnosis of Dynamic Systems

The proposition states a fundamental property of many model-based predictors and, in particular, qualitative simulation systems which, to the best of my knowledge, has not yet been pointed out in the literature. This is worth while, because the proposition indicates the possibility we are interested in: in order to discriminate between different modes (that is what diagnosis is about), we need not check for different behaviors; it suffices to check for the existence of different states.

However, we do not want to compare entire sets of states and behaviors. Using a few behaviors, or even only one (the observed one), should do. The foundation for this is provided by the following proposition and theorem.

Proposition (Model Discrimination by State Checking)

Let $model(mode_1)$, $model(mode_2)$ be two strictly continuous models in the same representation (or; more generally, separable and homogeneous models).
If a behavior, *behvr*, is admissible under $model(mode_1)$, then
it is admissible under $model(mode_2)$ iff it contains only states admissible under $state\text{-}constraints(mode_2)$:

$$\text{if } \{behvr\} \cup model(mode_1) \nvdash \perp$$

then $\{behvr\} \cup model(mode_2) \nvdash \perp$

iff $\{behvr\} \cup state\text{-}constraints(mode_2) \nvdash \perp$.

This follows directly from the definitions:

$\{behvr\} \cup model(mode_1) \nvdash \perp$

implies

$\{behvr\} \cup temp\text{-}constraints(mode_1) \nvdash \perp$.

Because the models are homogeneous,

$temp\text{-}constraints(mode_1) = temp\text{-}constraints(mode_2)$,

and $model(mode_1)$ is separable, the conclusion is obtained.

We are still comparing behaviors under different **models**. However, in a diagnostic setting, we have to check behaviors of the **real device** for consistency with one or more models. Especially for fault detection, the only model we want to use is the model of correct behavior. But the models of faulty behavior do not really have to be present and used in the diagnostic system. If *behvr* is the description of a real behavior under a particular behavior mode in a representation ($\underline{v}$, DOM($\underline{v}$), T) and satisfies the *CID-constraints* for this representation, this suffices to exploit a separable model we compare *behvr* with. With this step, the above proposition yields the following theorem.

Theorem (Mode Discrimination by State Checking)

Let *model(mode)* be a strictly continuous model and *behvr* the description of a real behavior in the same representation, satisfying the *CID-constraints.*
Then *behvr* is admissible under *model(mode)*
iff it contains only states admissible under *state-constraints(mode)*.

This means, all the diagnostic system has to do is to perform a consistency check of all states of the real behavior with the *state-constraints* of the model. The temporal constraints can be ignored, and, in particular, **no simulation of the modeled behavior is required. It simply could not reveal additional contradictions**. Also, it suffices to detect **a single state** to be inconsistent with the model, which means we do not have to rely on a temporally dense description of the behavior. All this sounds too wonderful to be true and to contradict many practical experiences. Indeed, we have to consider a pragmatic precondition for diagnosis that has been ignored in our theoretical considerations, so far.

5.3 Temporal Constraints and Measurability

Our definition of states and behaviors or, more precisely, of their description in a particular representation requires completeness: a state assigns a value to each variable (and derivative) that occurs in the representation. In many applications, limited measurability of the device or process to be diagnosed provides us only with a partial description of states. A dynamic system, by definition, has internal states that depend on previous input and states we may have no information about, and it is likely that the state descriptions are fairly incomplete.

There are three relevant limitations to measurability:

- The measurements of variables have limited precision.
- Only a subset of the variables in the respective representation can be measured, i.e. only a subvector $\underline{v}_{obs} = p_{obs}(\underline{v})$, where p_{obs} is a projection.
- We can obtain measurements only for a subset $T_{obs} \subset T$ of the temporal universe.

While the first limitation indeed affects both state checking and behavior checking, there is a chance that using *CID-constraints* may help to detect more inconsistencies. Let $behvr_{obs}$ be the partial description of an actual behavior:

$$behvr_{obs} = p_{obs} \circ behvr \mid T_{obs} : T_{obs} \rightarrow \mathrm{DOM}(\underline{v}_{obs}) ,$$

where „∘“ denotes composition of mappings and „$|$“ restriction of the mapping. But how should simulation be able to overcome limitations imposed by measurability? If

$$\{behvr_{obs}\} \cup state\text{-}constraints(mode) \nvdash \bot,$$

how can joining *CID-constraints* change the situation to

$$\{behvr_{obs}\} \cup state\text{-}constraints(mode) \cup CID\text{-}constraints \vdash \bot \ ?$$

Basically, the *CID-constraints* can do so by improving the behavior description in two respects:

- Since they constrain variable values and derivative over time, they could complement the existing **partial state descriptions**. This is particularly true for derivatives which are difficult to measure, but might be determined (or estimated) based on variable values of adjacent states.
- They may complement the **partial behavior description** by inferring (partial) descriptions of states that were not directly observed, i.e. by extending T_{obs}. The mean value theorem can derive information about intermediate, unobserved states from values measured at time instances of T_{obs}.

In this sense, **exploiting the temporal constraints can compensate for limited measurability** of a device and be superior to simple consistency checking of states. However, it is necessary to ensure for any class of devices and measurement conditions whether or not this is actually true. In [Malik Struss 1996], we state a sufficient condition for the case where *CID-constraints* will not improve the diagnosis result.

The first part of this condition states that the sampling rate suffices to guarantee that no state in the evolution of a behavior is missed by the observation ("observations without gaps"). This may be fulfilled for qualitative behavior descriptions. Obviously, this precondition denies that the *CID-constraints* can reveal inconsistencies by providing information about unobserved states.

The second part formulates that measurements need not provide a complete state description, but only have to be "complete enough" to make the important distinctions between different modes visible. We restate and reformulate the definition.

Definition (Measurability for Fault Detection)

Let $\{model(mode)\}$ be a strictly continuous system description. Let the measurability of a device be characterized by

$$p_{obs}: DOM(\underline{v}) \rightarrow DOM(\underline{v}_{obs}) .$$

The condition of measurability for fault detection is satisfied if at least one distinctive state can be measured: For any behavior behvr under a fault mode the following statement holds

If $\{behvr\} \cup (state\text{-}constraints(mode_{correct})) \vdash \bot$

then $\{behvr_{obs}\} \cup (state\text{-}constraints(mode_{correct})) \vdash \bot$.

(A similar condition can be stated for fault identification). If this precondition is satisfied, application of *CID-constraints* would be prevented from detecting additional inconsistencies based on completion of partial state descriptions. Thus, we obtain the following proposition stated in [Malik Struss 1996].

Proposition (Fault Detection by Checking Measured States)

Let $model(mode_{correct})$ be a strictly continuous model and *behvr* the description of a real behavior in the same representation, satisfying the *CID-constraints*. If the observations are without gaps and the condition of measurability for fault detection is satisfied then

behvr is admissible under $model(mode_{correct})$

iff $behvr_{obs}$ contains only states admissible under $state\text{-}constraints(mode_{correct})$.

(Again, a respective proposition can be stated for the problem of fault identification.) In other words, under the conditions of this theorem, **diagnosis based on checking state consistency yields results equivalent to diagnosis based on checking state and temporal constraints**, which means simulation is needless.

An important question is whether we have a chance to determine if the condition of measurability for fault detection holds. We are able to formulate valid models of the relevant fault behaviors in the representation used by $model(mode_{correct})$, we can establish a necessary condition, namely whether or not p_{obs} preserves the **distinctive states between the models:**

Measurability for fault detection is not satisfied, if there exists a *mode* such that the measurements of all states that are admissible under *model(mode)* but not under $model(mode_{correct})$ are consistent with $model(mode_{correct})$, i.e.

$$\forall s \in STATES(model(mode)) \setminus STATES(model(mode_{correct}))$$
$$p_{obs}(s) \in p_{obs}(STATES(model(mode_{correct}))).$$

This provides a criterion that can be checked by analyzing sets of states and applying a projection. Note that the models of fault modes are only used for this analysis and need not be represented in the diagnostic system. Because it cannot be excluded that a fault model contains tuples whose projection to observables is inconsistent with the model of correct behavior but which does not occur in any real behavior, the proposition yields only a necessary condition.

Yet, there are fast processes that may not allow observations without gaps, and measurability may be bad enough to violate the second criterion. In such cases it is still worth while to check whether exploiting the *CID-constraints* actually improves results of consistency checking. But even if there is evidence of this possibility, there are different ways to achieve it. Performing simulation is not the only one.

5.4 State-based vs. Simulation-based Diagnosis

So far, we analyzed when and why behavior models can omit temporal constraints without impairing the results of the diagnosis. Now, let us assume we are certain that we have to deal with a situation that forces us to exploit the temporal constraints. Here, we consider strictly continuous systems again. The task of the diagnostic algorithm to be devised is to check consistency of observations with both *state-constraints* and *CID-constraints* :

$$state\text{-}constraints(mode) \cup CID\text{-}constraints \cup \mathrm{OBS} \overset{?}{\vdash} \bot ,$$

possibly for different modes if we are interested in fault identification. There are two extreme ways for performing this task, and, certainly, a number of mixed forms. Both ways could start by pruning the modes through

$$state\text{-}constraints(mode) \cup \mathrm{OBS} \overset{?}{\vdash} \bot .$$

Then, we can first compute the results of

$$state\text{-}constraints(mode) \cup CID\text{-}constraints ,$$

which basically means simulation (through integration) or constructing an envisionment for one or more modes. This corresponds to **extending information about the possible behaviors** which can then be checked for consistency with the observed behaviors in OBS. This can be expensive for several reasons, for instance, if we have to consider many modes, or if there is no metric information about time and it is not obvious how many simulation steps have to be carried out. Also, comparing behaviors involves both checking states and state transitions.

Alternatively, we can use *CID-constraints* to **extend information about the actual behavior** by computing deductions from

$$CID\text{-}constraints \cup \mathrm{OBS}$$

mainly by estimating derivatives and applying rules like the mean value theorem and then check with the newly derived information for states inconsistent with *state-constraints(mode)* for the respective modes. The advantage lies in applying *CID-constraints* only once and in the restricted context of OBS, as opposed to many modes. Also, no transition checking is required. This has an intuitive appeal, because, after all, it is the limitations in OBS that lead us to seeking the help of the *CID-constraints*. Hence, using them to enhance OBS seems appropriate. A potential source of problems is a situation where values of a variable vary strongly across neighboring samples and combination of results of the mean value theorem for different variables creates many intermediate states. Also,

temporal information about derived intermediate states will tend to be weak. As stated above, there are mixed versions, for instance, in guiding the simulation task tightly by the incoming observations etc.

Although there is some evidence that the second scheme is advantageous for many situations, we are far from suggesting one single best approach to this task. We present the discussion to show that the often advocated simulation-based approach is by no means compelling, but, on the contrary, highly questionable. Much more work is needed to develop good designs and criteria for their utility. This will require a more detailed analysis of the form and contents of the *CID-constraints* and their possible applications and relating their results to various limitations in measurability.

5.5 Case Studies

The motivation for this work and the expectations are generated by the attempt to build systems that tackle industrial applications beyond the scope of previous model-based systems. One example is (off-board) diagnosis of the hydraulic circuit of an anti-lock braking system (ABS) used in cars (see Figure 5-1). In this section, we first try to summarize key features of this problem, convey the basic ideas underlying our solution, relate them to the issues raised in this chapter, and report some experimental results obtained. Details are presented in [Struss et. al. 1996].

The purpose of an ABS is to prevent the wheels of the vehicle from locking up in order to enable the driver to be able to steer the car while using the brakes. This is achieved by controlling the pressure which is exerted on the wheel brake cylinders by pushing the pedal via the hydraulic circuit. The speed of each wheel is measured, and when the measurements indicate a tendency of a wheel to lock up, because the (negative) acceleration is too strong, the Electronic Control Unit reduces the pressure for some time, before increasing it again for the next deacceleration phase. Fig. 1 shows one subsystem of the hydraulic circuit which typically affects two diagonally opposite wheels. For each wheel, an increase in pressure is achieved by an open inlet valve and a closed outlet valve. For maintaining pressure level, the inlet valve is closed, and for reduction of the pressure the outlet valve opened. The latter step is supported by a reservoir chamber that fills quickly in this phase. Also the pump starts immediately to transport the liquid back towards the main cylinder, and the next cycle may start, if necessary.

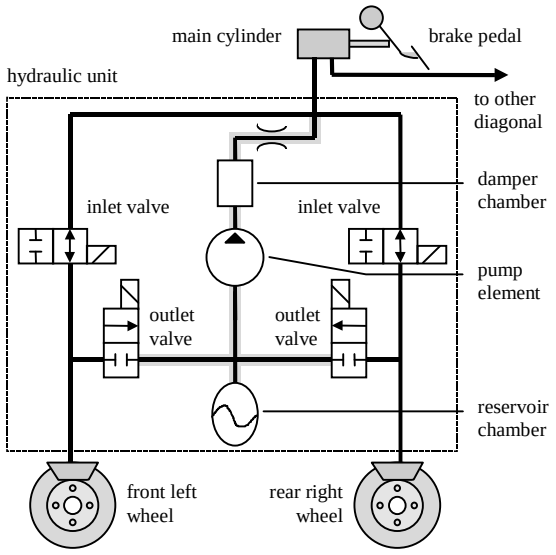


Figure 5-1: Hydraulic Circuit of ABS

Diagnosing this subsystem requires solving a number of challenging problems:

- Obviously, this is a system with **complex dynamics**.
- Observations are sparse: there is **not a single sensor** in the hydraulic circuit (the level sensor for the braking liquid is of no help). Information about pressures can only be obtained indirectly from the (de-)acceleration of the wheels. However:
- For **off-board diagnosis**, available observations are inherently **qualitative** in nature **and not temporally specified**. Typical observations would be "one wheel tends to lock up" (indicating too high pressure in the respective brake cylinder) or "the brake pedal is too soft" (as a result of an unusually low pressure in the

main cylinder). Even more precise values of speed and acceleration present on-board do not help too much, since:

- **Measurement** of strongly **influential exogenous factors** (e.g. the state of the road surface) is **impossible**.

In response to these conditions, we adopted an approach described in [Malik Struss 1996] using models that are stated in terms of qualitative deviations of variables and parameters from some unspecified and potentially changing nominal value.

It turns out that the "measurements" characterized above, enable the models to infer deviations in the pressure in different parts of the circuit. This trivially suffices to establish measurability for fault detection, but not measurability for fault identification and localization. The reason lies in the lack of information about the derivatives of pressures which cannot be provided by the observations and the state-constraints. Basically, this information would help to detect significant inconsistencies because resistive elements like valves relate (deviations of) flow to (deviations of) pressure, whereas pipes and other containers link (deviations of) flow and (deviations of) derivatives of pressure.

The solution was indeed provided by exploiting *CID-constraints*. However, they were not used for simulation of correct or faulty behavior modes, but to complement the observations in the following manner:

For each phase in the cycle (determined by valve positions), strict continuity can be assumed. Furthermore, we assume that there were no deviations when the phase was entered (this limits the applicability of the solution). If a deviation of a variable v was zero initially and different from zero after a while, then the *CID-constraints* tell us that there must exist a time interval during which both the deviation of v occurred and the derivative of v has a deviation with the same sign. In other words, *CID-constraints* were combined with OBS to deliver information about derivatives.

This turned out to be fairly successful: for a sample of component faults, such as clogged or punctured valves, defect pump, and air included, fault localization was successful: the respective faults were included in the set of single fault diagnoses, sometimes as only possible ones.

Other case studies that use related approaches and shed a light on its utility, are reported in the literature: [Dressler 1996] empirically discovered the possibility of fault detection via state checking in a prototype for diagnosing ballast water tank systems in off-shore platforms (a fairly sensor-rich system). [Chantler et al. 1996] reports results on equivalence of integration-based and differentiation-based algorithms for the special case of numerical models. [Williams Nayak 1996] pursues a transition-oriented, as opposed to simulation-oriented, approach to diagnosis and reconfiguration applied to the propulsion system of a space craft. [Malik Struss 1996] covers fault detection and identification in a simplified system (a controlled electric motor) with limited observability, but measurable derivatives based only on state checking.

5.6 Summary

As [Malik Struss 1996] states, "Diagnosis of dynamic systems does not necessarily require simulation". This chapter presented theoretical foundations and practical considerations supporting model-based diagnostic systems that are confined to using state constraints for consistency checking and do without temporal behavior constraints, or use them for purposes other than simulation. There are empirical results that provide evidence of the utility of such approaches, but also attempt steps towards identifying their limitations, particularly limitations that result from limited observability of the system. We hope that our analysis

- discourages overly "straightforward" approaches to model-based diagnosis,
- encourages more steps towards a systematic and thorough investigation leading to the design of diagnosis systems that are both well-founded and efficient,
- shows that such steps require a specific and detailed analysis of properties of models and predictive engines on the one hand and their interrelationship with observability on the other hand.

The benefit of these efforts will be beyond theoretical insight and result in widening the scope of feasible industrial applications which often involve dynamic systems.

6 An Approach to Consistency-based Process Diagnosis

Peter Struss und Ulrich Heller – TU München, Institut für Informatik

Kurzfassung

Es wird oft argumentiert, daß gewisse technische Prozesse oder Anlagen sich der Anwendung modellbasierter Ansätze für Diagnose oder andere Aufgaben entziehen, weil es keine geeigneten Modelle dafür gebe. Der Verbrennungsprozeß in einem Motor könnte als ein typisches Beispiel dafür angesehen werden. Es trifft zu, daß es natürliche und technische Systeme gibt, für die derzeit keine Modelle in den klassischen mathematischen Formalismen existieren. Ferner ist richtig, daß ein komponenten-orientierter Modellierungsansatz, wie er den herkömmlichen modellbasierten Diagnosealgorithmen in der Künstlichen Intelligenz zugrunde liegt, den spezifischen Anforderungen von Diagnose und Überwachung solcher Systeme nicht gerecht wird. Wir stellen die These auf, daß es - unter einem entsprechend erweiterten Modellbegriff - durchaus möglich ist, das Verhalten dieser Systeme formal zu modellieren, etwa durch qualitative prozeß-orientierte Modellierung, und ferner, daß dies eine ausreichende Basis für die Anwendung modellbasierter Techniken für Diagnose und Überwachung darstellt - allerdings ebenfalls nur durch wesentliche Erweiterung der zugrunde liegenden Theorien und Techniken. Es wird sichtbar, daß die herkömmlichen Theorien und Systeme für die Diagnose technischer Systeme(implizit) maßgeschneidert sind für die Anwendung komponenten-orientierter Modellierung.

Wir stellen eine Weiterentwicklung der traditionellen Theorien modellbasierter Diagnose vor. Sie verfolgt das Ziel, diese Theorien allgemeiner zu machen im Hinblick auf die zu lösenden Problemklassen und zugleich spezifischer durch die Ausnutzung einer verfeinerten Charakterisierung der Systembeschreibung.

Abstract

Often, it is argued that there are technical processes and plants that are not amenable to model-based diagnostic or other systems because there exist no proper models, and combustion processes may be considered as a typical example. It is true that there are natural or technical systems for which no models represented in classical mathematical formalisms exist, and also that component-oriented modeling, the basis of classical model-based diagnosis in AI, is inappropriate for addressing the specific requirements of diagnosis and monitoring of such systems. We argue that, with a broader concept of modeling, useful formal models can be derived, especially through process-oriented modeling. This enables the exploitation of model-based techniques for diagnosis and monitoring, but this does also require significant extensions to the underlying theories and techniques: It becomes evident that the current theories and systems are (implicitly) tailored for diagnosing artifacts based on component-oriented modeling.

We have developed a revision of the traditional theories of diagnosis from first principles. The goal is to make it more general in terms of the class of problems to be addressed and more specific by proposing and exploiting a refined representation of the system description.

6.1 Introduction: Towards a Theory of Diagnosis from First Principles

Model-based diagnosis and, more specifically, consistency-based diagnosis has matured to a point where commercial tools and significant industrial applications appear. Despite this significant progress, a broad range of diagnostic tasks and potential application domains have not been addressed with consistency-based diagnosis methods equally successful, so far. Examples are diagnosis in process industry and diagnosis of disturbances in ecological systems. We argue that this is due to implicit assumptions and a very specific view underlying up-to-date consistency-based diagnosis and the limitations resulting from them. This view can be characterized in a nutshell as follows:

- The entities relevant to diagnosis are **components**, which can be associated with different behavior modes: the correct one and at least one **fault mode** (possibly with unspecified behavior).
- A system to be diagnosed consists of a **given set of** such **components** which interact in a way determined by the **fixed structure** of the system (its blueprint) and are to be scrutinized for faulty behavior.
- The result of the diagnosis is **an assignment of actual behavior modes** to all these components.

- The criterion for a proper diagnosis candidate is that the respective mode assignment **is consistent with the observations** of the actual system behavior.

This can be illustrated by an example taken from the area of car diagnosis. If a diesel injection engine produces increased carbon emissions in its exhaust gas, there are a number of potential causes for this. This includes, for instance, delayed injection or insufficient air supply to combustion, faults that can be related to malfunctions of certain components, such as the distributor or a leakage in the intake manifold Standard component-oriented diagnosis is able to generate diagnostic hypotheses (see e.g. [Sachenbacher et al. 2000a]). However, it might be the case that all components of the vehicle are working properly and that the problem lies somewhere else, namely in „bad fuel“, i.e. an increased amount of heavy hydrocarbon. Since it is not very intuitive to model the fuel as a system component, standard approaches to model-based diagnosis would miss this explanation. They might blame the combustion chamber which would be quite „unfair“. After all, the combustion chamber is not broken, what is happening inside is the important issue, namely the combustion of heavy hydrocarbon. To insure proper diagnostic reasoning, the model supporting it would have to include this process. Although one might be tempted to call this a „faulty combustion“, this is actually not very appropriate either. First, it is not a „damaged version“ of the expected combustion of light molecular weight hydrocarbon. It is simply a different one, the one that is happening when heavy hydrocarbon are present. Second, this would prevent the diagnosis process from tracking the origin of the problem which lies in the conditions for this process to occur.

While all this distinguishes our diagnosis task from standard component-centered consistency-based diagnosis, there is another difference which concerns foundations of the theory and the starting point of diagnostic algorithms: the detection of inconsistencies. In the traditional approach, they arise from observations contradicting the system model:

$$\mathrm{MODEL} \cup \mathrm{OBS} \vdash \bot$$

In our example, the model that includes the proper cause of the fault is not at all in conflict with the observations. However, it contradicts some expectations or legal restrictions. While the original theory implicitly assumed that the system model carries the "gold standard" ("if all components work properly, the overall behavior is the desired one"), we now realize: what is crucial is the inconsistency of the observations with the **intended function or goals**, i. e. something that is external to the model.

Hence, this example violates all assumptions listed above as basic to standard consistency-based diagnosis. We have to notice that the "classical" theories and systems of consistency-based diagnosis are too narrow and, as a result, fail to provide a solution to many diagnostic problems.

- the appropriate diagnosis result is **not a component faul***t*,
- even if we would treat the processes occurring (such as combustion) like components, they are **not** in **a „fault mode“**,
- the cause of the fault is not to be sought in the set of entities that are part of some initial system model (like the components), but an **additional** object which gives rise to some disturbances, and
- that this disturbance does not occur as a **contradiction** between the model and what we observe, but as a violation of some **goals** which are imposed on the system. This means: diagnosis based on inconsistencies between the model and the observations is not the proper perspective.

As a consequence, a theory of diagnosis from first principles has yet to be developed. This chapter attempts to contribute to this goal by proposing a revision and extension to consistency-based diagnosis that preserves the principled approach while expanding the scope of the underlying modeling paradigms and diagnosis tasks and algorithms

In the following section, we re-state the formal foundation of consistency-based diagnosis and its implementation. Section 6.3 outlines a general formalism for composing system descriptions which accommodates, in particular, component-oriented and process-oriented modeling. On this basis, the following section defines and distinguishes different tasks, including a more general notion of the “classical” diagnosis task. In the following two sections, we describe one way to implement the compositional modeling scheme and how to perform diagnosis in a manner that overcomes the limitations of the accepted wisdom of consistency-based diagnosis.

6.2 Consistency-based Diagnosis

The accepted wisdom of consistency-based starts from a point where a set of observations, OBS, is inconsistent with a system description, SD, and an assignment of correct behavior modes to all components (the set COMPS):

$$SD \cup OBS \cup \{OK(C_i) \mid C_i \in COMPS\} \vdash \bot \quad . \quad (1)$$

The diagnosis procedure is then organized as a search for revised mode assignments to the components that eliminate inconsistency (see e.g. [Reiter, 1987]):

$$SD \cup OBS \cup \{mode_i(C_i) \mid C_i \in COMPS\} \nvdash \bot \, . \quad (2)$$

Implementations such as GDE (modeling correct behavior only, [de Kleer Williams 1987]) and GDE+ (exploiting also fault models, [Struss Dressler, 1989]) employ an Assumption-based Truth-Maintenance System (ATMS) in order to record inferential dependencies and to determine (minimal) sets of mode assumptions that conflict with the observations and compute diagnosis candidates as (minimal) revisions of the assignment of correct behavior to all components.

Obviously, any attempt to overcome the limitations shown above requires a more general and flexible scheme for providing the system description, SD, which is what we outline first.

6.3 What's in SD?

We follow the principles of structure-to-behavior reasoning and compositional modeling and provide a generalization of both component-based and process-based modeling paradigms. According to this view, the system description, SD, consists of four parts: the domain theory, the system specification (structure and parameters), the situation specification3 and basic laws. The diagram in Figure 6-1 provides an overview, and we briefly discuss each part. Section 6.5 provides a more formal and algorithmic view.

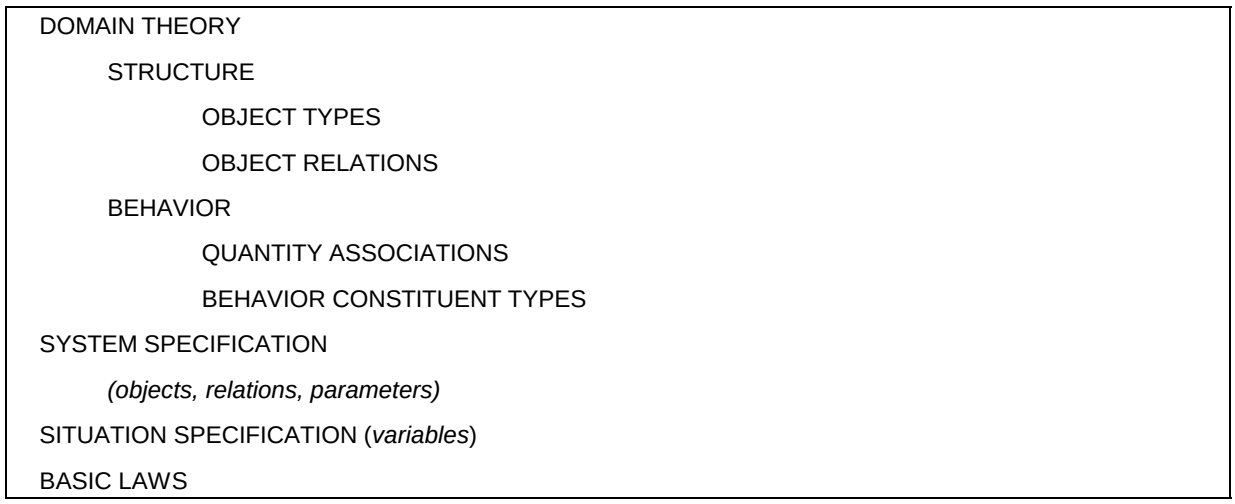


Figure 6-1: The Structure of SD

Domain Theory

The domain theory captures what we know about the domain, i. e. all systems of a certain class (e. g. combustion or the drive train of a vehicle). We distinguish *structure constituents* ("objects") from *behavior constituents* (which might be processes or other model fragments). The structural ontology consists of

3 We use the term "situation" rather than "state" in order to avoid confusion with the concept an (internal) state, e. g. in control theory. A "situation" is characterized by all variables (including state variables).

- *object types* which occur in structural descriptions, for instance types of components in a device (resistor, broken wire), spatially distinguished entities (combustion chamber, pipes, tanks), etc. Object types can be structured hierarchically.
- *object relations* for characterizing "configurations" of objects. Examples are spatial relationships (contained-in, below), connectivity of components, etc. Some properties of relations (like uniqueness) can be specified.

The domain theory also has to provide a vocabulary for behavior descriptions and the inferences that derive behavioral constituents from a structural description. It introduces

- *quantity associations.* Parameters and state variables will be associated with instances of object types, e. g. resistance, voltage drop, and current for a resistor and amount of heavy carbon-hydrates. For simplicity of presentation, we associate quantities only with single object instances.
- *behavior constituent types*. These are physical phenomena which are considered to contribute to the behavior of the overall system. They can represent basic component laws (Ohm's Law, logical-or) or processes in QPT ([Forbus, 1984]), such as combustion, or partial behavior models like in [Bredeweg, 1991]. They occur deterministically under certain conditions, and their occurrence generates particular effects. Applying the distinction between structural aspects and the characterization of behavior through quantities to both conditions and effects, we obtain

  $$\text{STRUCT-CONDS} \wedge \text{QUANT-CONDS} \Rightarrow$$
  $$\text{STRUCT-EFFECTS} \wedge \text{QUANT-EFFECTS}$$

  as the abstract form of a behavior constituent type. Here, STRUCT-CONDS are assertions about the existence and relations of objects (e. g. existence of heavy carbon-hydrates and QUANT-CONDS are statements about values of quantities (e. g. a minimum concentration of oxygen). STRUCT-EFFECTS can specify the creation and elimination of objects and their relations (e. g. the generation of some substance, such as carbon emissions), and QUANT-EFFECTS can be expressed as restrictions on variables (e.g. the amount of carbon emissions in dependence of oxygen concentration and amount of heavy carbon-hydrates).

  More precisely, we state that for each constellation of objects satisfying the structural and quantity conditions, an instance of the behavior constituent is occurring - and imposes the respective effects on the constellation. An important point is that the quantity effects can be partially specified w. r. t. the combination with other behavior constituents. We use a formalism of "influences" for this purpose.

At this point, we make almost no commitment w. r. t. the quantity domains (symbolic, qualitative, real, ...), the formalism for specifying the quantity effects (constraints, differential equations, ...), and the expressiveness of structural conditions and effects (e. g. non-existence of certain objects as condition or destruction of objects as a structural effect). In general, conditions and effects are assumed to be local and compositional. Some requirements for model formation and prediction will be discussed in Section 6.5.

System Specification

A particular system under consideration is characterized by its *object structure*, i. e. instances of the object types and individual tuples of object relations (for instance the components and the connection structure of a device) and *parameter values* for objects involved in the physical system.

Situation Specification

A particular situation of the system is characterized by *variable values*. Dependent on the task and context they may represent actual measurements (e.g. an increased amount of carbon emissions), specification of goals (a certain maximal amount of carbon emissions), mere hypotheses, etc.

Basic Laws

Additionally, we include a part for the fundamental laws that determine the mechanisms of model formation, how influences combine and prediction over time (continuity, integration etc.). They cannot be specified arbitrarily by the modeler - but rather represent the logical equivalent of procedural aspects of model composition and prediction software components.

This way of modeling, (like QPT which can be regarded as a specialization of it), allows for dynamic changes in the set of active processes and, thus, distinguishes from approaches that represent systems as a pre-defined sequence of processes which are then considered to possibly fail very much like components (see e.g. [Guckenbiehl et al. 1999]). Unlike the QPT-based work of [Collins 1993], our approach includes structural effects and, hence, facilitates diagnosis w.r.t. changes in the (object-related) structure (e.g. due to unanticipated objects or interactions).

6.4 Characterizing Different Tasks

With the structure of the system description as a background, we can characterize problem solving tasks concerning the system. We do so using the core of consistency-based diagnosis: *Given some initial description of a system that is inconsistent with information external to the model (e.g. observations), determine a revision of this model such that consistency is achieved.*

To find out what to revise, we distinguish what we *know* about the domain, the system under consideration and its present situation, and what we *assume* about the system and the state. In particular we make the assumption explicit that we have modeled all relevant aspects of the system at hand -the closed-world assumption. In other words, this initial system description, SD_{init}, can be split into the fixed part, SD_{fix}, and the revisable part, SD_{rev}.

$$SD_{init} = SD_{fix} \cup SD_{rev}$$

where

$$SD_{rev} = STRUCT_{rev} \cup PAR\text{-}SPEC_{rev} \cup VAR\text{-}SPEC_{rev}$$

From this initial description, the domain theory and the basic laws generate a behavior model as part of some complete system description, SD_0, possibly under extension of the structure specification and completion of the parameter and variable specification. SD_0 can be inconsistent in itself or with some expectations or goals.

We can try to ask and answer different questions and characterize the task by what is considered fixed and revisable, respectively (and by specifying what establishes inconsistencies) :

- What's going on, anyway? System identification[4] takes observations for granted:
  $OBS \subset VAR\text{-}SPEC \subset SD_{fix}$

  and attempts to find appropriate system models. If model composition yields an inconsistency, one needs to find a consistent one
  $SD_0 \vdash \perp \quad \rightarrow \quad SD_1 \nvdash \perp$

  by revising some of the initial assumptions about the structure and parameter values:

  $SD_{rev} = STRUCT_{rev} \cup PAR\text{-}SPEC_{rev}$, or just

  $SD_{rev} = PAR\text{-}SPEC_{rev}$

  for **parameter identification** for a system with known structure.

- What's **going on right now? Situation assessment** treats the system as fixed and tries to determine variable values starting from observations and (possibly revised) assumptions about values:

  $OBS \cup STRUCT \cup PAR\text{-}SPEC \subset SD_{fix}$

  This may boil down to prediction of values, but can also include revision of assumptions about the situation and doubtful observations:

  $SD_{rev} = VAR\text{-}SPEC_{rev}$

- What's going wrong? Diagnosis: What is "wrong", is determined by some goal, rather than physics itself. This means inconsistencies and the necessity for diagnosis are caused by criteria that are external to the physical system and its model (e. g. a limit on the amount of carbon emissions). We have to make these goals explicit. If they are violated by the system description (possibly SD_1 obtained in the previous step - if

[4] Precisely what is happening is **model identification**. But note that, in contrast to system identification in engineering, the model generated is not merely a mathematical model, but stated in terms of physical phenomena!

we assume it represents the actual system sufficiently well) one can ask how SD_1 has to be revised in order to no longer contradict the goals:

$$SD_1 \cup GOALS \vdash \bot \rightarrow SD_2 \cup GOALS \nvdash \bot.$$

The "difference" between SD_1 and SD_2, i. e. the revised elements of SD_1, which eliminate the contradiction can be considered to have "caused" the trouble. Diagnosis may focus on ultimate or external causes such as unexpected objects being present or try to identify values that can be influenced for treatment or control. We assume that GOALS are stated as a characterization of healthy or desirable situations, i. e. as variable specifications:

$$GOALS \cup OBS \subset VAR\text{-}SPEC \subset SD_{fix}$$

$$SD_{rev} = STRUCT_{rev} \cup PAR\text{-}SPEC_{rev}$$

- What **can be done? Control and therapy proposal** has to look for revisions that establish consistency with the goals and can be achieved by real *actions*:

  $$SD_1 \cup ACTIONS \cup GOALS \nvdash \bot,$$

  or, stronger, to establish them:

  $$SD_1 \cup ACTIONS \vdash GOALS.$$

  In some cases, the actions correspond to the revisions of the diagnosis task, e. g. removing a disturbing object, replacing an identified faulty component by a correct one, manipulating a deviating parameter etc. This could be considered as a "real cure", which eliminates the ultimate causes of a disturbance. Then a (complete) diagnosis uniquely determines the therapy. This might be impossible and the proposal of structural changes by adding objects and object relations and/or modify parameters and variables might be the only control option. This corresponds to real "therapy" and is the interesting case considered in this chapter.

  Introducing some chemical treatment, which might not be prepared in the initial system model, but is rather selected from the domain theory as a possible action, illustrates the case

  $$SD_{rev} = STRUCT_{rev}$$

  whereas some reconfiguration of a system means

  $$SD_{rev} = VAR\text{-}SPEC_{rev}.$$

  These examples also highlight that the possible means for therapy influence or even determine the formulation of the diagnosis task and that the framework includes the possibility of a "therapy without diagnosis". However, elaboration of this is beyond the scope of this chapter.

  If some GOALS are defeasible, they may be considered for revision, as well.

6.5 SD in Action: Model Formation, Prediction and Revision

The reasoning tasks discussed above include important non-monotonic steps. This is a consequence of the usage of the closed-world assumption to calculate quantities, which may in turn have effects on the occurrence of other behavior constituents. This makes implementations difficult and often inefficient. However, for certain domain theories, we have been able to implement a model composition algorithm that employs monotonic reasoning and some form of constraint satisfaction.

6.5.1 Instantiation and Activity

In the following, we consider the case, where we have only positive structural effects, i. e. objects (and relations) are only generated, but not destroyed, and only positive structural conditions (i. e. no conditions in terms of non-existence of objects or relations). If we assume, furthermore, that we can determine the identity of any generated object (e. g. using some relation like spatial locatedness by "contained-in"), then we can monotonically construct all objects that can possibly be generated by any instance of a behavior constituent, without taking the quantity values into account.

This corresponds to the common distinction between "instantiation" and "activation", as described in the Qualitative Process Theory ([Forbus 1984]), which does not allow for structural effects in the domain theory. Technically, we separate the structural conditions and structural effects from the quantity conditions and quantity effects.

In an instantiation phase, also called model composition (see Figure 6-2), all potentially active behavior constituents are assembled, i. e. for each set of objects matching the structural conditions of a behavior constituent type, we create an instance of a behavior constituent, bci, plus all objects and relations created in the associated structural effects.

Some guard variables are introduced along with constraints that block structural effects from becoming effective before the respective quantity conditions are met. Namely, for each object instance and each relation tuple, we introduce an additional boolean quantity exists, representing whether it is present or not. A behavior constituent does have a similar boolean variable $active_{bci}$, encoding whether both its structural and quantity conditions are met. A set of constraints encodes the following dependencies:

$$\text{STRUCT-CONDS} \wedge \text{QUANT-CONDS} \Leftrightarrow active_{bci} = true$$

$$active_{bci} = true \Rightarrow \text{STRUCT-EFFECTS}$$

Where STRUCT-CONDS and STRUCT-EFFECTS are expressed using the exist quantities of the respective objects and relations.

In the second phase (constraint generation in Figure 6-2), the behavior constituent instances are turned into a behavior model that can be used for prediction. For quantity effects, locally specified as constraints,

$$constraint(x_1, x_2, ..., x_n)$$

(where xj represent quantities) we add a condition, so that their effects can be switched on and off according to the current value of the active variable. The resulting constraint is

$$active_{bci} = true \Rightarrow constraint(x_1, x_2, ..., x_n)$$

For partially specified quantity effects, i. e. influences, we carry out influence resolution. Here, we have to take into account that the influences are only potentially active, and we further explicitly assume that the collected influences are the only ones acting on the influenced variable (local closed-world assumption). For a set of influences from quantities x_1, ... x_n on variable y, this is achieved by duplicating the x_j yielding x_1', ... x_n', with additional constraints

$$active_{bcji} = true \Rightarrow x_i' = x_i, \quad \forall i$$

$$active_{bcji} = false \Rightarrow x_i' = 0, \quad \forall i$$

$$y = sum(x_1', x_2', ..., x_n')$$

where $active_{bcji}$ are the activity variables of the behavior constituent instances creating the influences from x_i to y. The constraints are written for the (standard) case of additive influence combination, so that a neutral contribution is generated for inactive influences. The third one is the resolution constraint, which is supported by the local closed-world assumption for y, CWA_y. With this step, the influences have been combined and converted into sets of constraints.

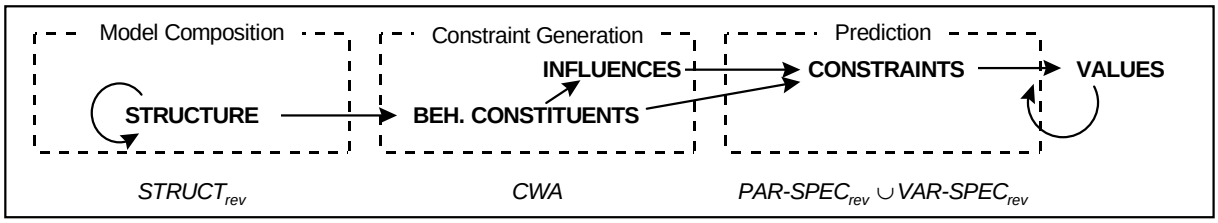


Figure 6-2: Phases and Assumptions for Model Formation and Prediction

As a third phase, the system performs *prediction* in its general sense, using the known and assumed quantity values. This means evaluating the quantity effects plus the newly introduced constraints for controlling the effects: behavior constituents can become "active" or "inactive", which triggers or suspends both their structural and quantity effects.

This way, we can compute a representation of the extensions of the non-monotonic theory (maximum consistent sets of consequences, see [Reiter 1980]) by prediction. A constraint propagator can create (most of) the intersection of all extensions, while a complete constraint filtering algorithm could even generate all extensions.

When the identity of newly created objects cannot be determined, we face unification problems with existing objects. So we have to construct multiple worlds - and can rely solely on the predictions that are identical in all worlds.

Further note, that the approach can fail entirely, if model formation produces infinitely many potentially active behavior constituents, even if only a finite number could become active under the given quantity specification.

6.5.2 Assumption Tracking

Our main objective in prediction is the detection of conflicts, i.e. sets of assumptions that are inconsistent with the system description, observations, and goals. For this purpose, we have to keep track of the

- structural assumptions (STRUCT_{rev}) leading to some quantity effect being present in the model at all (or, rather, being active at a particular moment),
- of the closed-world assumptions used in combining influences (CWA) and all
- quantity value assignments that are considered revisable ($\mathsf{PAR\text{-}SPEC}_{rev} \cup \mathsf{VAR\text{-}SPEC}_{rev}$).

Thus, we have requirements for assumption tracking in all three phases (see Figure 6-2).

6.5.3 Searching for Revisions

For performing one of the tasks defined in Section 6.4, the system starts off with the constraint network generated during model formation. Observations or goals are added in the form of variable specifications. Hence, consistency checking is a matter of further prediction only, and conflicts, i. e. inconsistent assumption sets can be generated as in component-oriented diagnosis. The candidates generated from these conflicts (also as usual) include assumptions to be revised. They fall in one of three categories: variable assignments (elements of $\mathsf{PAR\text{-}SPEC}_{rev} \cup \mathsf{VAR\text{-}SPEC}_{rev}$), existence of objects or relation tuples (elements of STRUCT_{rev}), and closed-world assumptions (See Figure 6-3).

The structural and quantity assumptions can immediately be tested for consistency by prediction with the revised system description (all effects of the objects or relation tuples now revised to be non-existent are constructed with conditional constraints, so that they will be switched off consistently as by the constraints described in Section 6.5.1).

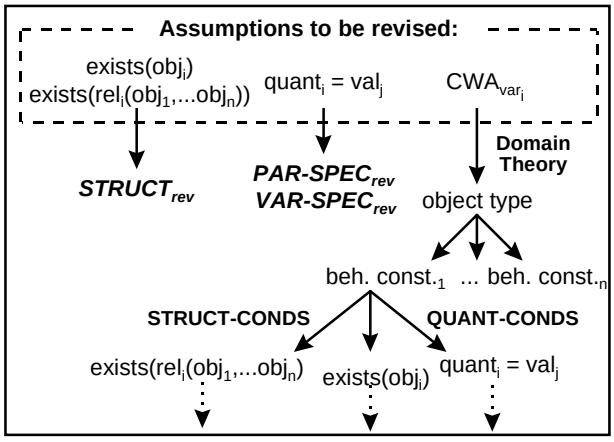


Figure 6-3: Searching for Revisions

Closed-world assumptions are of a different nature, for their retraction does not give a clue what the revised model should look like. It is only the starting point for a search. We are looking for a revision (in terms of objects, relation tuples and quantity values) that violates the local closed-world assumption by hypothesizing behavior constituents whose influences could remove inconsistencies. This revision will be a candidate in meaningful, i. e. ontological, terms.

The domain theory provides the necessary background knowledge to determine the behavior constituent types, which could influence a quantity of the given kind (associated object-type) at all. Each behavior constituent instance in question would require to have both its structural and quantity conditions satisfied. Each possibility can be subject to further search, e. g. the structural conditions of one behavior constituent instance could be established by the structural effects of another one with even simpler conditions.

Of course, in carrying out the search within the space of the domain theory, we rely on the domain theory to be complete w. r. t. to the phenomena of interest.

[Collins 1993] is solving the analogous problem for QPT models. He proposes an analogous distribution of the global closed-world assumption and is using an abductive backchaining algorithm to generate explanations implying the violation of the local closed-world assumption. But since he is not considering structural effects of processes, structural revisions are taken as primitives.

Generally, focusing the search is an important problem to be solved, since standard minimality principles can be misleading. Even in the initial candidate generation, where it is common to prefer the candidates with the minimal number of elements, it is possible that a larger candidate with a number of (violations of) local closed-world assumptions can be explained by a smaller revision, e. g. by introducing a single object triggering a range of additional processes achieving the deviating behavior.

This problem recurs with every intermediate search result, so that in general we have no guarantee that there is no simpler candidate. One solution is to distinguish a class of revisions as "ultimate causes".

The search triggered by hypothesized violations of the closed-world assumptions provides significantly more power than traditional, component-oriented diagnosis, but, obviously, has its price in terms of complexity. One should notice, however, that there is a class of diagnostic cases that require the introduction of additional objects, but not necessarily the retraction of the closed-world assumption and the costly search: if the respective object has been created by the model formation step, but the system description implies its non-existence (exists = false), due to the given specification of quantities, this is directly reflected by "classical" candidates that are subsets of VAR-SPEC_{rev}.

6.6 Perspectives and Discussion

The described approach to different tasks of diagnostic problem solving attempts to preserve the formal and rigorous foundation, while overcoming the specific conditions and limitations of state-of-the-art consistency-based diagnosis. It should help to expand the scope of applicability significantly.

Classical component-oriented diagnosis is a (very) special instance of this scheme. However, our extension offers a principled approach to diagnosis of structural faults. Furthermore, it allows for appropriate diagnostics in cases where component replacement is not a means for repair, but we rather aim at reconfiguring the system. We also feel that extended consistency-based diagnosis can complement fault diagnosis and identification (FDI) methods developed in control engineering by providing the search for model revisions at the level of physical phenomena rather than mathematical expressions.

We have implemented model formation as described and plan to use a commercial constraint-based diagnosis tool that provides an efficient ATMS, a predictor and a candidate generator. We have experimented with this system as a core of an integrated decision support system for environmental applications and water treatment problems, but we also see a potential for applications in the process industry.

Among the unsolved problems is the handling of unification problems in object creation (if two objects of the same type are created, they could be distinct or identical, which spans two "worlds"). It is not yet clear, how predictions in one world could be shared in another.

Some issues of guiding and focusing the search for revisions when retracting a closed-world assumption have been pointed out. However, it is not easy to derive useful heuristics for the general case. Solutions could come from exploiting certain structures of specific domain theories.

7 Transformation of Qualitative Dynamic Models – Application in Hydro-Ecology[5]

Ulrich Heller und Peter Struss – TU München, Institut für Informatik

Kurzfassung

Hydro-ökologische Systeme sind gekennzeichnet durch komplexe Interaktion physikalischer, chemischer und biologischer Prozesse. Kompositionale Modellierung, d.h. die Erzeugung eines Verhaltensmodells für ein System durch Aggregation von Modellen seiner Bestandteile, ist eine entscheidende Voraussetzung dafür, die Modellierung komplexer Systeme mit vertretbarem Aufwand bewältigen zu können. Jedoch ist ein kompositionales Modell oft zu feinkörnig im Hinblick auf eine bestimmte Aufgabenstellung, etwa weil es zu viele unwichtige Zwischenvariablen enthält oder weil es die grundlegenden Zusammenhänge verdeckt. Aus diesem Grunde muß ein solches Modell oft transformiert und vereinfacht werden. Dieser Artikel stellt eine Repräsentation für dynamische Systeme vor, die graphen-orientiert und existierenden prozeßorientierten Modellierungssprachen nahesteht, sowie eine Menge syntaktischer Operatoren, die ein entsprechend formuliertes Modell unter Bewahrung bestimmter Eigenschaften transformieren. Der Formalismus wird motiviert und illustriert durch ein Beispiel aus unserer Arbeiten über die Modellierung Hydro-ökologische Systeme. Wir demonstrieren aber auch seine Tauglichkeit für Anwendungen auf technische Systeme.

Abstract

Hydro-ecological systems comprise complex interaction among physical, chemical, and biological processes. Compositional modeling, i. e. creating a system's behavior model by aggregating models of its constituents, is crucial for making the modeling task feasible. However, the composed model is often too fine-grained for a particular task, for instance, in containing too many irrelevant intermediate variables or obscuring the basic interdependencies. For this reason, the model may have to be transformed and simplified. The chapter presents a graph-oriented representation for dynamic systems closely related to existing process languages, and a set of syntactic operators that transform such a model while preserving certain properties of the model. The formalism is motivated and illustrated by an example taken from our work on modeling hydro-ecological systems, but we also demonstrate its utility for technical applications.

7.1 Introduction

In our work on modeling complex ecological systems for decision-support systems, a number of important challenges arises. In an effort to obtain prediction models for algal blooms in a specific river in Southern Brazil, we relied upon the following methodologies:

- **Compositional modeling,** i. e. generating a behavior model of a complex system through aggregation of models of its elementary constituents taken from a library.
- **Modeling of dynamic systems.**
- **Multiple modeling,** i. e. the task of creating different models of one system dependent on the problem and situation, relating them in a systematic manner and potentially switching between them.
- **Qualitative modeling.**

Quite powerful methods have been developed to address these issues. However, they are limited, partly conflicting with respect to their goals and results, and raise new problems:

- Models of complex systems obtained by **compositional modeling** bear the **potential of being too detailed**. This is because the constituent models in the library have to be stated in terms of local variables and parameters only which may appear irrelevant from the perspective of the entire system or the particular task

[5] Dieser Beitrag ist erschienen im Tagungsband des 10th International Workshop on Qualitative Reasoning (QR-96), TR-96-01, AAAI Press, Stanford, 1996. (pp. 83-92).

and conceal the elementary influence structure of the system. Besides, some reasoning tools exhibit exponential behavior in the number of variables, so it is desirable to keep the model small.

- **Models of dynamic systems** can be **too fine-grained** with respect to their temporal resolution, if they capture all aspects of the dynamics. This can be inappropriate, for instance, when the rapid but minimal fluctuations of the concentrations of some substances that are basically held in chemical equilibrium complicate the long term prediction of the behavior of the entire system.

The solution is to replace the detailed, fine-grained models by appropriate structural or behavioral abstractions of themselves which is obviously the subject of multiple modeling. However,

- relationships among **multiple models** are often **too informal**, lacking a systematic way of generation. But a formal characterization of model relations and transformations is required a) for an automated solution to the problem and b) for determining the impact of the transformation, i. e. the properties gained and the properties preserved by the transformation.

As it turns out, the result of such transformations may suffer from another, more general problem:

- Current languages for **qualitative modeling** can be **too weak** to derive all possible conclusions. If the qualitative description of functional dependencies is limited to monotonic functions, the analysis of their counteraction or comparison may lead to spurious results.

In order to improve both the predictive power of the models and the results of the operators, the confinement of qualitative functional relationships to monotonic dependencies only has to be overcome. A common answer to ambiguity and insufficient distinctions in quantitative models is "Hybrid models by integrating quantitative information!". We believe that this seemingly obvious solution is often inappropriate and obscures the fact that the expressive power of the models can be extended **without having to leave the realm of qualitative descriptions.**

The work described in this chapter is a contribution to the solution of these problems. We present a graphical representation of a dynamic system's model relying upon four classes of functions to characterize influences and different schemes for combining them (usable in a process-oriented style á la QPE/QPC [Forbus 1984], [Crawford et al. 1990]) and a set of syntactic operators that transform and simplify such representations. This set includes generation of strict abstractions of a model as well as the approximation of dynamic relationships through functional dependencies (sometimes termed "temporal abstractions").

We illustrate the resulting description language (which can be regarded as an extension of the cited process formalisms) and the transformations on a subproblem taken from a project of modeling hydro-ecological systems ([Heller et al. 1995], [Heller Struss 1996b]), namely the interaction of transport processes and chemical reactions.

This problem domain is briefly described in the following section. Section 7.3 introduces the graphical notation and points out the formal semantics. An overview of the abstraction and approximation operators is given in Section 7.4. Then the application of the operators to the interaction of advection with ammonia dissociation is illustrated (Section 7.5). and finally some perspectives for applications in other areas and for further development of the approach are given (Section 7.6).

7.2 The Problem Domain

In an international collaboration between researchers of Brazil, France and Germany, we have been examining a specific ecosystem, namely the Rio Guaíba in Southern Brazil, with the objective of analyzing and predicting undesirable occurrences of algal blooms. The modeling of the complex hydrodynamics and the various chemical and biological processes involved provided us with important challenges for our modeling and reasoning techniques.

The Rio Guaíba

The Rio Guaíba, located in Rio Grande do Sul, the southernmost state of Brazil, is heavily polluted by organic matter. The municipal department for water and sewage (DMAE) is alarmed by the occurrence of local algal blooms, that are to be taken as an indication of ecological instabilities. Because of the undesirable effects for both the use of the water (mainly as a source of drinking water) and for the ecosystem in itself, there is an effort to build models for the analysis and prediction of algal blooms.

Among the elementary conditions for the possibility of algal blooms is the availability of nutrients, which is influenced primarily by distribution and transformation processes. In this chapter, we will examine a typical example of an interaction of two such processes.

Advection

Advection is the transport of matter by directed flow of water. The complex hydrodynamics in the bays of the Rio Guaíba prevent us from using a linear water flow model and we had to chose a flexible representation of spatial distributions and water transportation. By using compartments, the elements of a topological partitioning of the water body (described in more detail in [Heller et al. 1995] and [Heller 1995]), and by locating the transport processes between adjacent compartments, we also gain more generality.

The advective effect on the concentration of some specific chemical constituent in two adjacent compartments can be easily determined if the volumes are assumed constant (requiring the net flow for each compartment to be zero). A simple model under this assumption is discussed in Section 7.3.

Ammonia Dissociation

One of the most important constituents is ammonia, appearing both in free (NH_3) and ionized form (NH_4). Both forms can act as nutrient, but free ammonia in high concentrations can also exhibit toxic effects, so we have to study the chemical equilibrium ($NH_4 + OH \leftrightarrow NH_3 + H_2O$) established by the counteracting reactions of ionization and dissociation.

Both reactions are strongly influenced by the pH of the location. To put it more precisely, if the ratio between (the molar concentrations of) NH_3 and NH_4 is below $10^{(pH - 9.26)}$, then the dissociation reaction dominates ionization. Above the given reaction constant, the ionization is predominant. Both reactions will be modeled as a single process with a rate that is linearly dependent on the difference between the ratio NH_3/NH_4 and the reaction constant (modeled as positively monotonic in the pH, see Section 7.3).

7.3 A Graphical Notation for Models of Dynamic Systems

In order to model these processes, we introduce a graphical notation for qualitative dynamic models, called *influence diagrams*, that enable the modeler to express the available knowledge about parameter relationships. On the one hand, it provides more specific means of expression than just the monotonic functional dependency, on the other hand we allow for more general functions depending from multiple variables than given by the linear combination assumption implicitly used in various other modeling languages (e.g. in QPC, see [Crawford et al. 1990]).

Characterization and Combination of Influences

A system is represented by a finite set of variables, with continuous real-valued functions as the domain (describing the temporal development of some parameter value). The variables are depicted as boxes containing a meaningful variable name. Additionally, there is a set of constraints on these functions, namely the existence of some "influence function" specifying the dependency of a variable on a set of other variables, which is shown as labeled arrows and a specific "combination information".

In particular, we want to express that a variable A depends monotonically on a set of other variables $\{B_1, B_2, ..., B_n\}$ (possibly with different direction coefficients $S_1, S_2, ..., S_n \in \{+1, -1\}$), i.e.

$$\exists f \in Mon_{(S_1,S_2,...S_n)} \ \forall t \in \mathbb{R} \quad A(t) = f(B_1(t), B_2(t), ..., B_n(t)),$$
where $Mon_{(S_1,S_2,...S_n)} := \{ f : \mathbb{R}^n \to \mathbb{R} \mid \forall i \in \{1, 2, ..., n\} \ \forall x_j, x_j' \in \mathbb{R} \ (1 \le j \le n)$
$(((x_i > x_i') \wedge (j \ne i \Rightarrow x_j = x_j')) \Rightarrow S_i \cdot f(x_1, x_2, ..., x_n) > S_i \cdot f(x_1', x_2', ..., x_n')) \}$
for $S_i \in \{+1, -1\}$ $(1 \le i \le n)$ (simply written "+" or "−")

In Figure 7-1, the corresponding influence diagram is shown. The small box to the left of B contains the combination information with the direction indicators $S_1, S_2, ..., S_n$.

There are several ways to represent additional information about the influence function. The most important one is a further restriction by a Lipschitz condition or even linearity in one of arguments. A Lipschitz condition in the i-th argument is given by

$$\exists M \in \mathbb{R}^+ \ \forall x_j, x_j' \in \mathbb{R} \ (1 \le j \le n) \ \ ((x_i' \ne x_i) \wedge (j \ne i \ \Rightarrow \ x_j' = x_j')) \ \Rightarrow$$
$$|f(x_1', x_2', ..., x_n') - f(x_1, x_2, ..., x_n)| \ \le \ M \cdot |x_i' - x_i|$$

and linearity in the i-th argument can be expressed as

$$\forall a, x_j, x_j', x_j'' \in \mathbb{R} \ (1 \le j \le n) \ ((x_i = a \cdot x_i' + x_i'') \wedge ((j \ne i \ \Rightarrow x_j'' = x_j' = x_j')) \ \Rightarrow$$
$$f(x_1, x_2, ..., x_n) \ = \ a \cdot f(x_1', x_2', ..., x_n') + f(x_1'', x_2'', ..., x_n'')$$

In the influence diagram (Figure 7-1), the arrows are labeled with the restriction of the functional dependency ("Mon" for monotonicity, "Lin" for additional linearity and "Lip" for the Lipschitz condition). For a function with a single parameter, we use also strict identity ("Id") and the following (proper) inclusions are valid: Id $\subset$ Lin $\subset$ Lip $\subset$ Mon.

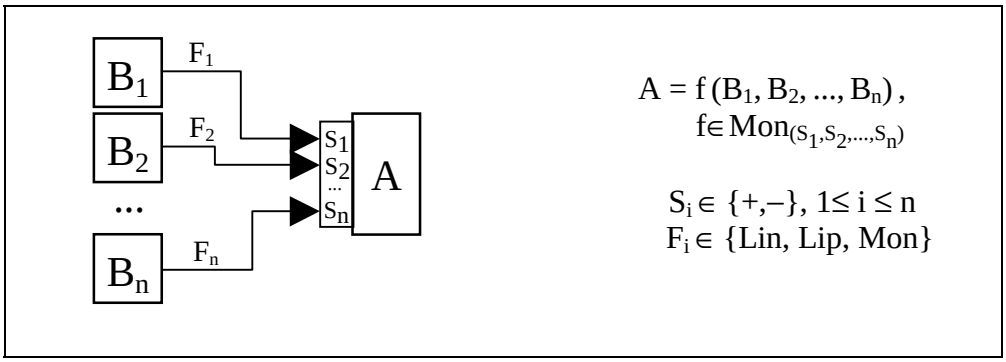


Figure 7-1: The basic elements of the influence diagrams

Decomposable Influences

Furthermore, certain influence functions can be decomposed in the sense that they are known to consist of groups of influences combined additively or multiplicatively. More precisely, a function $f \in \mathrm{Mon}_{(S_1,S_2,...,S_n)}$ is said to be decomposable additively, iff

$$\exists i \in \{1, ..., n-1\} \ \exists f_1 \in \mathrm{Mon}_{(S_1,...,S_i)} \ \exists f_2 \in \mathrm{Mon}_{(S_{i+1},...,S_n)} \ \forall x_1, ..., x_n \in \mathbb{R}$$
$$f(x_1, x_2, ..., x_n) = f(x_1, ..., x_i) + f(x_{i+1}, ..., x_n)$$

A special case is the complete decomposition into single influences, which corresponds to the assumption of linear combination. We depict decomposed influences by separating the combination information. Compare Figure 7-2 for the notation for completely decomposed influences. The restrictions given at the arrows refer only to the respective group of influences, thus making e. g. linearity a weaker condition in the decomposed case.

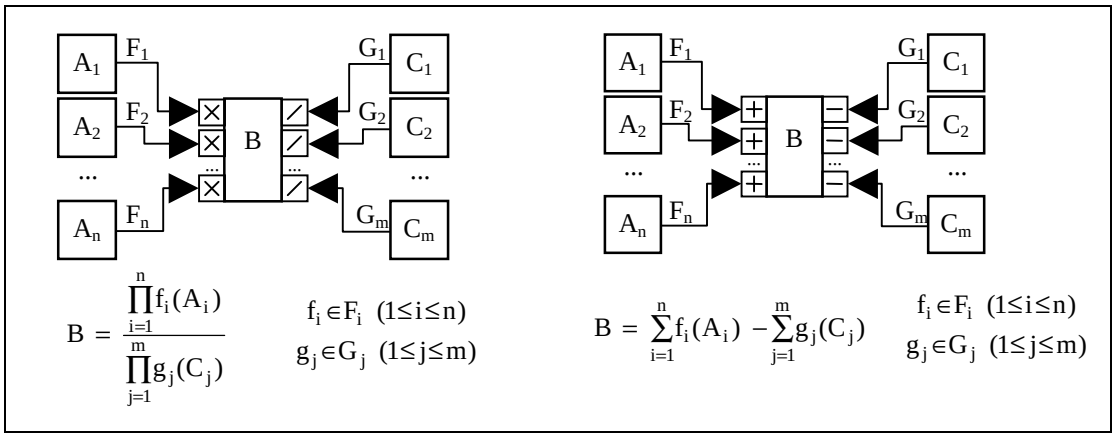


Figure 7-2: (Completely) multiplicatively and additively decomposed influences

The multiplicative decomposition requires two new combination symbols ("×" and "/"). For now, it refers only to the product or quotient of single influences. For more sophisticated constructions, intermediate variables have to be used. The syntax is also shown in Figure 7-2.

Integrative Influences

The discussed influences correspond to the qualitative proportionalities used in QPT ([Forbus 1984]). To represent the so-called "direct influences" of QPT, we need an integrative influence, expressing that the derivative of a variable A is (monotonically) dependent on a group of other variables B_1, B_2, ..., B_n:

$$\frac{dA(t)}{dt} = f(B_1(t), ..., B_n(t)), \quad f \in Mon_{(S_1,S_2,...S_n)}$$

We use all of the constructions discussed above, but enclose the combination information in a circle or a rectangle with rounded edges, as is shown in Figure 7-3.

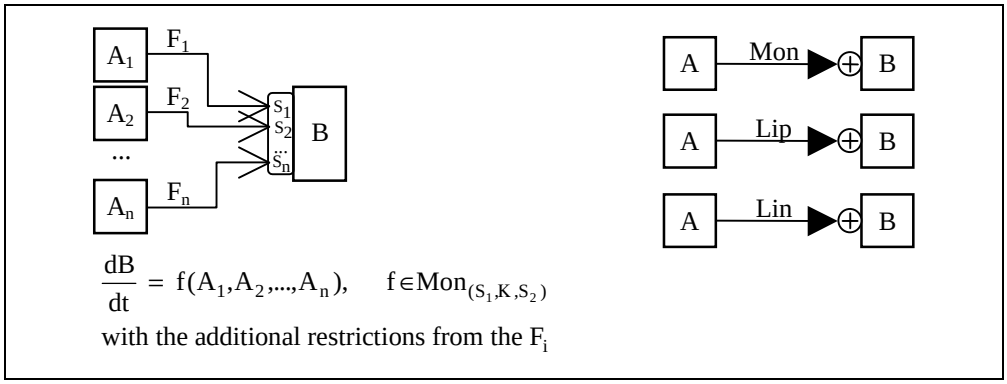


Figure 7-3: Integrative influences

In this way we can display a qualitative abstraction (with respect to the functions involved) of an ordinary differential equation as an influence diagram and in turn extract a partially specified differential equation from a diagram. Together with a mechanism for instantiating and composing model fragments, we can also visualize models written in QPE/QPC notation.

Process Models for the Domain Problem

Processes are described by partial influence diagrams (possibly with parameters) and additional information about how to compose them with other processes acting on common variables. A process is instantiated by giving the parameters defined values that can be obtained from the system description and aggregating the partial diagrams into the system model. The formal semantics described in Section 7.3 depends on the closeness of the model. In the cases discussed here (transport and chemical transformation), we have apparently additive combination of influences. We developed models for prediction tasks for both the short and the long term behavior of hydro-ecological systems. Here we will present only two simple ones to study their interaction.

A simple generic process description for advection for some constituent (e. g. ammonia) using influence diagrams is shown in Figure 7-4. Unlabeled arrows are to read as bearing the identity label "Id". The boxes with a black shadow denote important state variables. They represent concentrations. Thus, the transported amount of matter is obtained by multiplication of the source compartment concentration with the (absolute) flow between the compartments. The loss respectively gain in concentration is then calculated as a linear function (the linearity factor being in either case the reciprocal of the volume of the respective compartment, which is assumed constant):

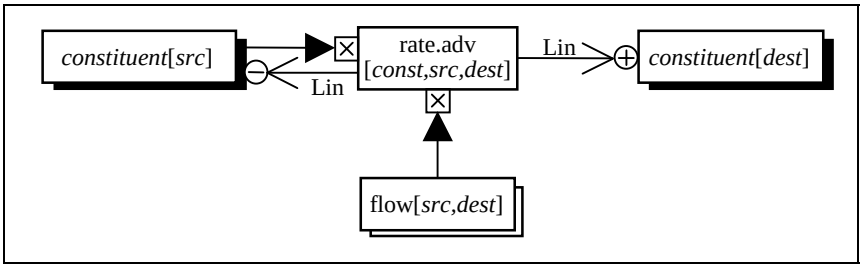


Figure 7-4: The advection process (simple version)

It will instantiated for various chemical constituents, *const(ituent)*, and locations, *src* and *dest*, (given in square brackets). Note that the semantics in the strict sense given in Section 7.3 will be valid for the complete (composed) model only. However, the combination of the influences on the concentration in the destination compartment are assumed to be additively decomposable from other influences.

Furthermore, a version of the dissociation process without feedback will be used. The concentration of NH_4 will be treated as equaling the total ammonia concentration. Thus, we can neglect the loss of NH_4 by the transformation. The resulting influence diagram is presented in Figure 7-5:

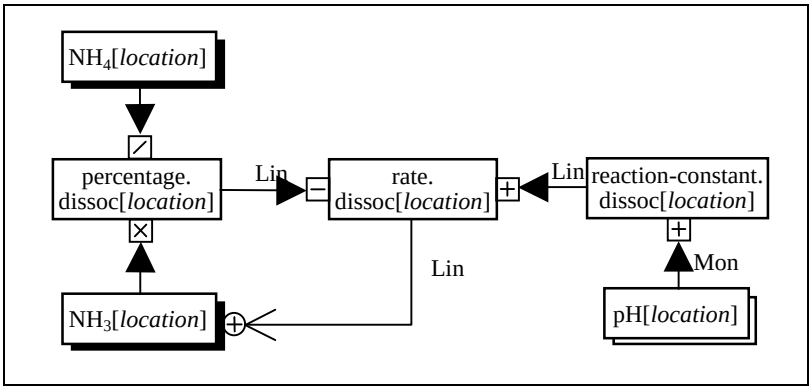


Figure 7-5: Dissociation process without feedback

We compose the advective transport of ionized ammonia (which we treat as total ammonia, so that NH_3 is assumed not to be subject to advection) from compartment "In" into a specific compartment, X, and from X to compartment "Out", with the dissociation taking place inside compartment X, we obtain the following diagram (Figure 7-6).

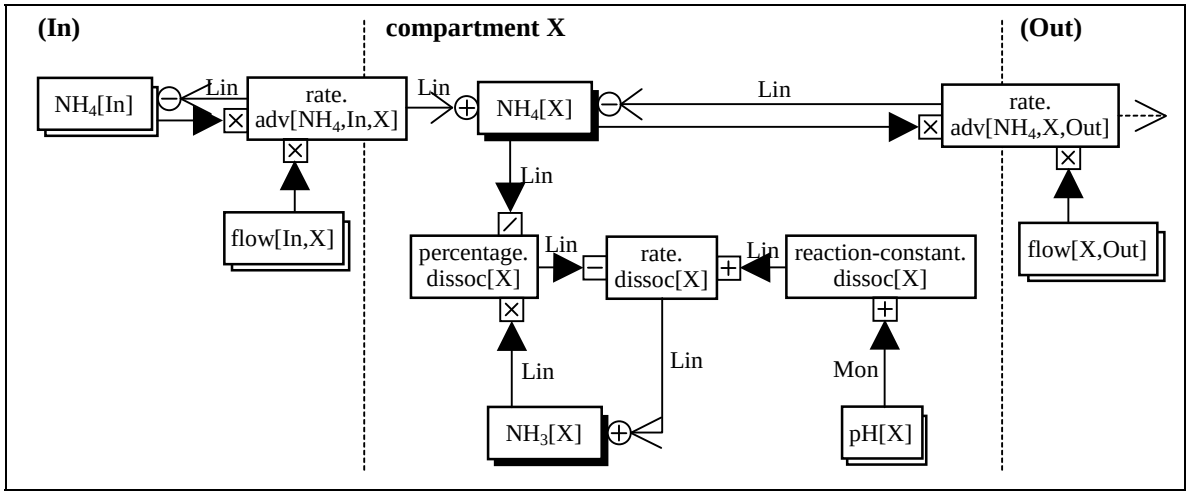


Figure 7-6: The interaction of dissociation and advection

So we benefit from being able to compose the system model from a simple structure description and a library of generic process descriptions (both described in detail in [Heller 1995]), but the simulation of the resulting model is unnecessarily complicated by the large difference in the strength of the integrative influences. Some qualitative simulation frameworks lack a way to express the different orders of magnitude and produce therefore spurious solutions by erroneous assessment of the counteraction. For the illustration of this effect, an extended example is given in in [Heller 1995].

The solution is to identify and approximate the fast mechanism, which is done by a set of transformations operating locally on the influence diagram.

7.4 Model Transformations

We developed a set of transformation operators to simplify influence diagrams and to identify the basic influence structure in more complex interactions. The goal is to examine in a formal way the applicability of the so-called time-scale abstraction. Time-scale abstraction, as introduced by B. Kuipers ([Kuipers 1987]), will formally be treated as an approximation. In the formal framework of model relations developed in [Struss 1991] and [Struss 1992], abstraction transforms a model into a strictly weaker version, whereas approximation replaces one model by another one that may violate validity.

Abstraction Operators

If a variable specified in a model fragment is assumed constant in the context of the complete model, we can eliminate it, because it unnecessarily complicates the reasoning task. The **elimination of constants** is achieved by the following operator (Figure 7-7):

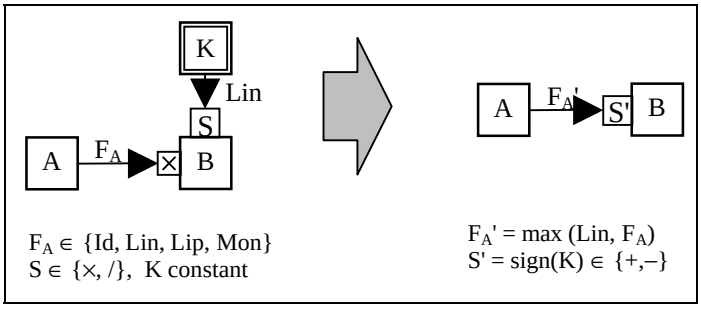


Figure 7-7: Elimination of multiplicative constants

The class of functions on the right hand side is obtained by taking the maximum with respect to set inclusion (remember that Id ⊂ Lin ⊂ Lip ⊂ Mon). The resulting model transformation is an abstraction (even more precisely, a "view" as defined in [Struss 1992]). The proof for this operator and for the following ones can be found in [Heller 1995]).

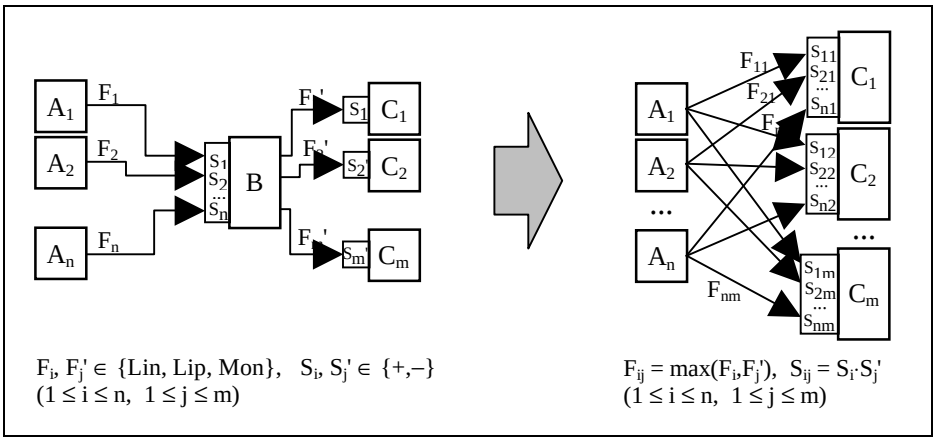


Figure 7-8: Elimination of intermediate variables with multiple influences

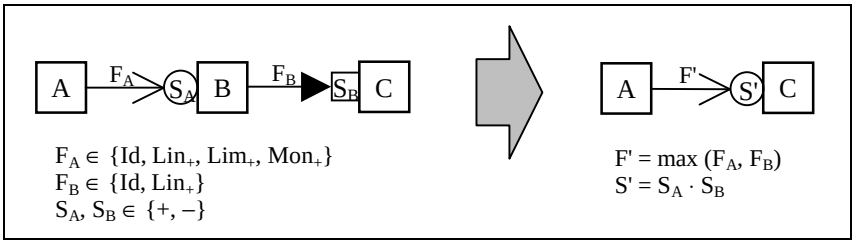


Figure 7-9: Elimination of intermediate variables, shown for a single integrative influence

Some variables might be irrelevant, e. g. because they are not observable. The **elimination of intermediate variables** for multiple influences is shown in Figure 7-8.

Analogous operators exist for integrative influences, on some variable C_i. For an integrative influence on B there is a restriction (at least in the semantics used): B can only be eliminated, if all of the influence originating from B are linear (Figure 7-9).

Partial decomposition of influences can be preserved, if the relating function is linear. For completely decomposable influences, the operator has the form shown in Figure 7-10.

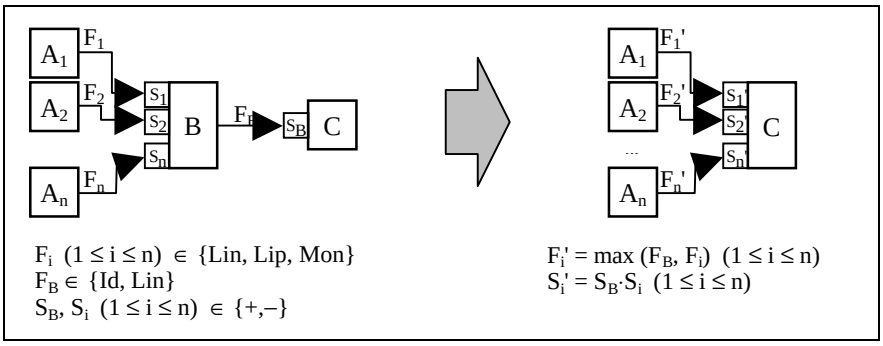


Figure 7-10: The variant for completely decomposed influences

Various cases with additional influences on C, not originating from the intermediate variable B, are considered in [Heller 1995], but will not be discussed here.

Another class of operators achieves the **subsumption of parallel influences**, i. e. of influences with the same source and destination and the same combination symbol (either "+" or "–"). Figure 7-11 shows the decomposed case, which is the simplest one.

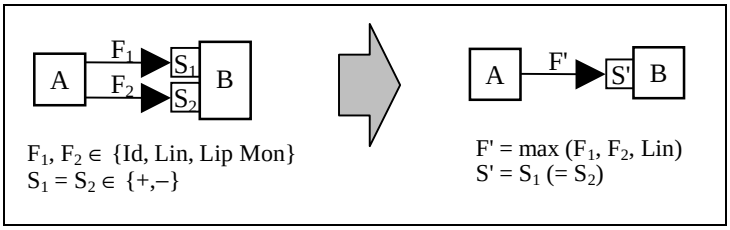


Figure 7-11: The subsumption of parallel influences (decomposed case)

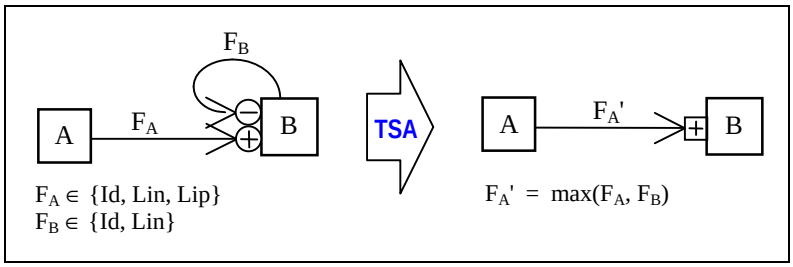


Figure 7-12: Time-scale abstraction for linear self-stabilization

Time-Scale Abstraction as Approximation

To cope with widely separated time-scales and to make the reasoning task feasible in cases where "fast" and "slow" processes interact, we intend to identify subsystems (by employing the operators introduced above) that can – under certain conditions – be substituted by functional dependencies with neglectable deviation. This corresponds to the technique of "abstraction by time-scale" as defined in [Kuipers 1987]. If the elementary influence structure has one of the following forms, we use the solution of the equilibrium equation as substitute.

We show two operators acting on closely related structures, namely on direct linear self-stabilization (Figure 7-12) and on multiplicatively mediated linear self-stabilization (Figure 7-13). Both are discussed in detail in [Heller 1995].

In this case it is even possible to derive precise bounds on the approximation error committed, by analysis of the underlying ordinary differential equation. In general the quality of the approximation increases with the linearity factor of the stabilizing function (class F_B) and decreases with the Lipschitz coefficient of the transfer function (class F_A) and the maximum variation of the derivative of A.

An instance of the latter occurs in our example, where NH_3 takes the role of B, A stands for NH_4, P for pH and Q for the ratio NH_3/NH_4 ("percentage", see Figure 7-14 below)

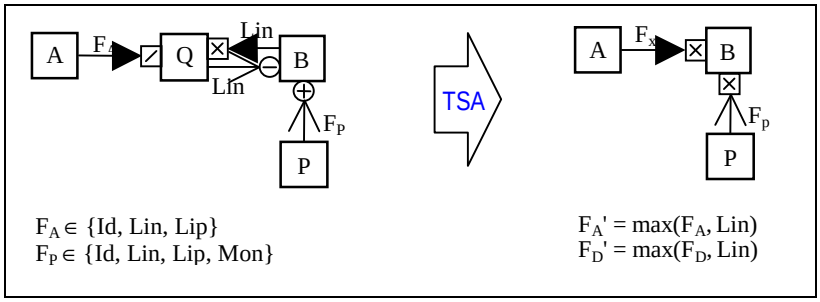


Figure 7-13: Time-scale abstraction for multiplicatively mediated linear self-stabilization with an additional influence

7.5 Application to the Ammonia Dissociation Model

To use this kind of approximation for the model given in Figure 7-6, we have to identify the elementary influence structure of the faster subsystem. Therefore, the operator from Figure 7-8 is applied first to reaction-constant.dissoc[A] and then to rate.dissoc[A], which yields the following influence diagram (Figure 7-14):

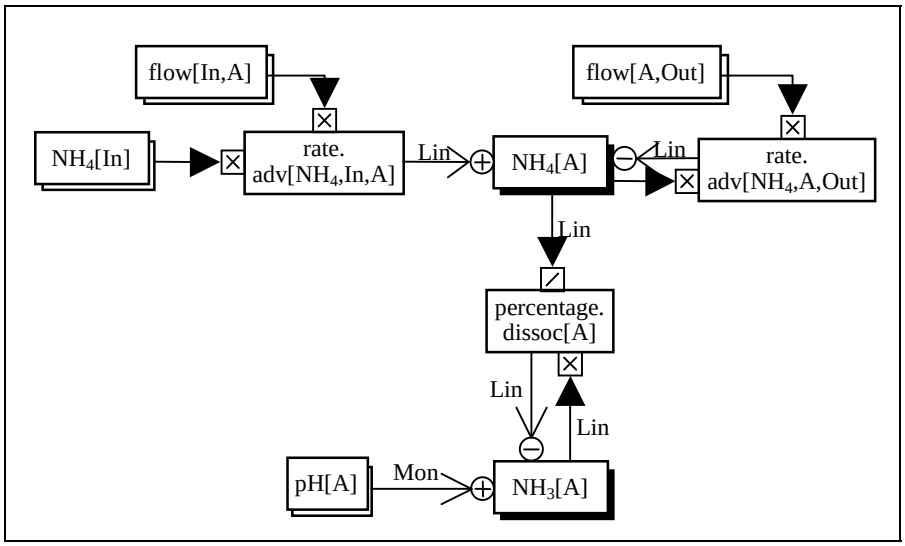


Figure 7-14: After the elimination of the intermediate variables

The influence structure that appears now in the lower part of the figure is a case of a multiplicatively mediated linear self-stabilization. It will be approximated by using the operator shown in Figure 7-13, which is justified by the strong stabilization by the chemical reaction and the comparatively slow changes in NH_4. The background knowledge about the orders of magnitude of the influences can be attached to the model fragments by the modeler (and propagated consistently through all abstraction operations), so the decision about the application of the approximation operator can be taken by formal reasoning about local information.

So the simpler model in Figure 7-15 can be used, which shows the right level of detail, the right granularity for long-term predictions and therefore produces more focused predictions:

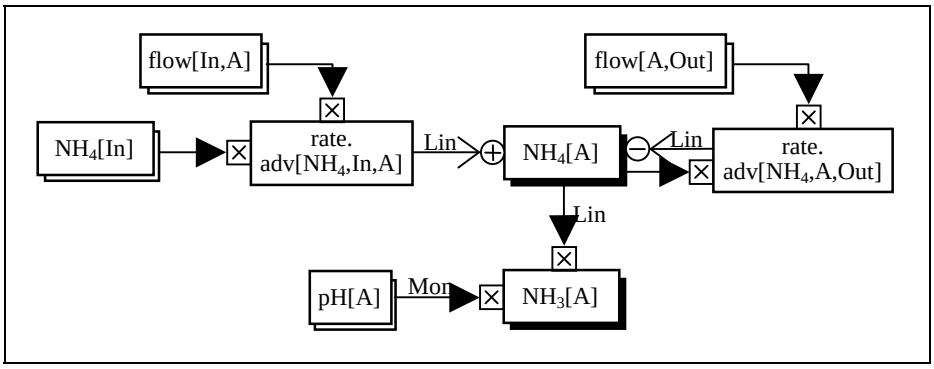


Figure 7-15: The resulting model for the interaction of dissociation and advection

7.6 Outlook

We give a short example of the application of the modeling and transformation formalisms in the technical domain. Then we discuss some issues about the implementation of the operators and about the specification of local model fragments.

Another Example: Motor with Control Circuit

The modeling formalism and the abstraction operators have also been successfully applied to a technical example, namely to a direct current motor with control circuit (described in more detail in [Malik Struss 1996]). The influence diagrams of the components were derived directly from the following differential equations (for the parameter descriptions refer to the table on the right):

$$v = \frac{2d - \omega_m}{c_C} \quad \text{(controller)}$$

$$\omega_m = (1 - S) \cdot \omega \quad \text{(sensor)}$$

$$\frac{d\omega}{dt} = \frac{c_M \cdot v - \omega}{T} \quad \text{(motor behavior)}$$

v	driving voltage
ω	rotational speed (of the motor axis)
ω_m	measured rotational speed
d	desired rotational speed
c_C	controller constant
c_M	motor constant
T	motor inertia
S	slip (of the measuring pulse wheel)

From a simple structure description, the following influence diagram will be derived (Figure 7-16):

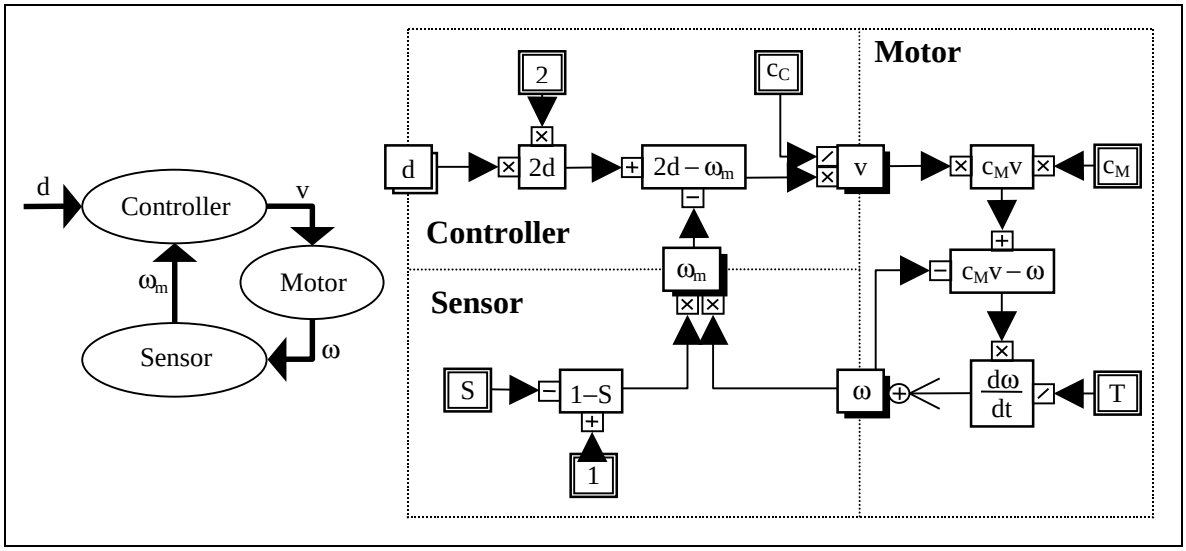


Figure 7-16: Structure description and influence diagram of a motor with control circuit

Application the developed operators eliminates the constants (inclusively the derived constant 1-S) from the model (operator from Figure 7-7) and also successively the intermediate variables 2d, $2d-\omega_m$, $c_M v$, $d\omega/dt$, and $c_M v-\omega$ (operators shown in the Figures 7-8 through 7-10) yields the simpler model in Figure 7-17 (left hand side). On the right hand side the result of further elimination (of ω_m and then v) is shown:

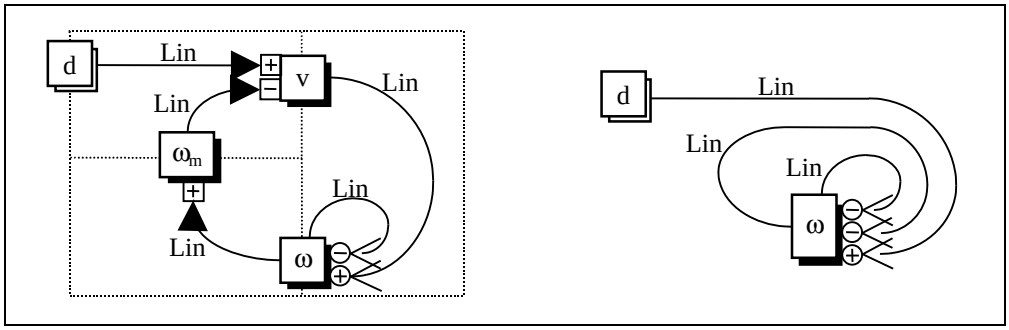


Figure 7-17: The motor model after the first and the second simplification

Finally, subsumption of the resulting parallel influences identifies the elementary influence structure as a (direct) linear self-stabilization that can be approximated by a functional dependency (see Figure 7-18, TSA-operator from Figure 7-12):

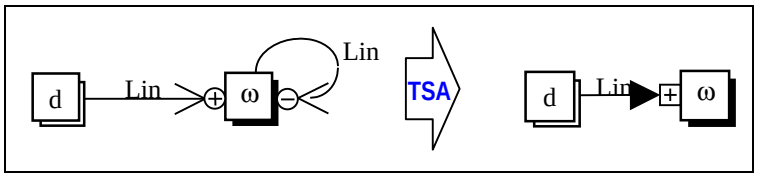


Figure 7-18: The final time-scale abstraction of the motor model

In the desired case (no slip, S = 0, controller constant equals motor constant, $c_C = c_M$), the approximation error can be bounded by $\frac{T}{2}\cdot c$, c being a bound on the derivative of d, thus showing the response of the motor being dependent solely on the inertia. For details refer to [Heller 1995].

Issues of Implementation

So far, the operators are applied only manually to models that have been formalized in other qualitative modeling languages (for the hydro-ecological models we have been using QPC, see [Crawford et al. 1990]). Nevertheless, the formal graphical notation will serve as a basis for an implementation, possibly using a transformational graph grammar approach.

The set of operators has to be expanded to achieve completeness, in the sense that from a given influence diagram some standard form can be reached in a finite number of steps. Finally, a reasoning mechanism relying directly upon models in the developed notation could exploit more of the contained information. The function classes could be extended by some subgroups like "over-linear growth" to determine a dominant influence in a counteraction presently ambiguous.

Issues of Compositional Modeling

Some fundamental issues about the specification of model fragments have to be examined further. So far, the semantics are defined only for the completely composed (and closed) model. However, compositional modeling requires a formulation of isolated fragments in the first place, and it is not obvious what such a "context-free" model fragment "knows" about the appropriate way of combining with other fragments.

For some class of processes, like transportation and transformation processes, it is evident, that the effects combine additively with other processes. Possibly, the ontology has to be extended by a classification of processes which, based on their physical nature, uniquely determines the correct type of combination. Studies in other domains and with different examples might require different mechanisms for combining processes influencing shared variables.

A task for future work is the development of a framework with a graphical model fragment editor and algorithms for model composition, transformation, and simulation.

Acknowledgements

We like to thank for the valuable collaboration with François Guerrin and Waldir Roque. This work was partially supported by the Brazilian Research Council (CNPq) and the German Ministery of Education and Research (BMBF). We benefited from discussions with Paulo Marcos Amaral Alves, Elenara Correa Lersch and Maria Mercedes Bendati from the research department of DMAE.

8 Automated Determination of Qualitative Distinctions: Theoretical Foundations and Practical Results[6]

Martin Sachenbacher and Peter Struss – TU München, Institut für Informatik

Zusammenfassung

Die automatische Transformation qualitativer Modelle für einen bestimmten Anwendungszweck ist wichtig für den praktischen Einsatz modellbasierter Techniken, weil dort die Wiederverwendbarkeit von Modellen eine wichtige Anforderung darstellt. Wird diese Aufgabe nicht gelöst, dann sind aus Modellbibliotheken zusammengesetzte Modelle meist entweder ineffektiv, weil sie zu grob für das gegebenes Problem sind, oder ineffizient, weil sie unnötig fein sind. Die Frage lautet daher: Welches sind die Unterscheidungen in den Wertebereichen der Modellvariablen, die sowohl notwendig als auch hinreichend sind, um ein bestimmtes Ziel für einen bestimmten Anwendungskontext unter bestimmten Randbedingungen zu erreichen? In dem vorgestellten Ansatz wird das Ziel durch eine Menge von Partitionierungen auf den Wertebereichen von Variablen vorgegeben, der Anwendungskontext ist durch die Struktur des Modells festgelegt, und die Randbedingungen werden durch mögliche initiale Unterscheidungen auf den Variablen definiert. Die zu lösenden Aufgaben umfassen beispielsweise die Bestimmung geeigneter qualitative Werten für die Verhaltensvorhersage oder die Diagnose, die Entscheidung, ob Zustandsübergänge kontinuierlich oder nichtkontinuierlich modelliert werden können, und die Festlegung, ab wann die Abweichung eines Parameters von seinem Normalwert als signifikant gewertet werden muß. Wir haben dies für den Fall relationaler Verhaltensmodelle analysiert, formalisiert, einen (unvollständigen) Algorithmus implementiert und erste Experimente damit durchgeführt. Dieses Kapitel vervollständigt zunächst bestehende theoretische Grundlagen. Darauf aufbauend wird eine formale Definition des Ziels und eine Spezifikation der algorithmischen Lösung entwickelt. Wir beschreiben dann den implementierten Algorithmus und diskutieren erste Ergebnisse, die aus der Anwendung dieses Prototypen gewonnen wurden. Wir schließen mit einigen offenen Problemen und einer Diskussion möglicher Erweiterungen.

Abstract

Automating the generation of qualitative models at a level that is tailored to support a particular task is crucial to the deployment of model-based systems technologies in practical applications, because reusability of models is of vital importance. If this task cannot be solved, models in a library will either be ineffective, because they are too coarse for solving a particular problem, or inefficient, because they are too fine-grained. The key question to be answered is, "*what are the distinctions in the domains of the system variables that are both necessary and sufficient to achieve a particular goal in a certain context and under given conditions*?". In our approach, the goal is defined by a set of target partitions of the domains of selected variables (e.g. output variables), the context is given by the structure of the modeled system, and the conditions are represented by a set of initial variables and their possible distinctions (e.g. possible observations). The task includes problems such as determining the appropriate qualitative values of variables in order to enable prediction at the desired level or discrimination for diagnosis, deciding whether or not changes can be modeled as discontinuous ones, and determining when a deviation of a parameter can be considered significant. We have analyzed and formalized the task for relational behavior models, implemented an (incomplete) algorithm, and carried out first experiments. The chapter first elaborates on previous theoretical foundations, defining the goal and the specification of algorithmic solutions. We then outline the implemented algorithm and present and discuss some experimental results of applying this prototype. We conclude with some open problems and discuss alternative approaches.

8.1 Introduction

Automated modeling, model abstraction and model composition has been studied in the area of qualitative modeling for quite some time (e.g. [Addanki et al. 1991], [Iwasaki 1992], [Falkenhainer Forbus 1991], or [Struss 1992]). Very often, the motivation for this work was mainly academic, and there are hardly any tools that are designed and implemented to serve serious applications. With model-based systems technology intruding

[6] Dieser Beitrag ist erschienen im Tagungsband des 14. International Workshop on Qualitative Reasoning (QR00), Morelia, Mexiko, 2000.

industrial applications, the creation of appropriate models gains practical importance, and strong support to this task or automating it may well be decisive to the success of model-based systems.

For instance, in our work on applying model-based diagnosis, prediction, and fault analysis to industrial problems (prominently car subsystems, see [Sachenbacher et al. 2000b]), we are thrown back to very fundamental tasks and theoretical problems the closer we get to the stage of actual use of the technology in industry.

Where do the models come from? is the most frequently asked question, because most of our solutions are based on qualitative models which look quite different from the existing (differential) equation models used in engineering, e.g. for numerical simulation. *Can't we use these models or, at least, start from them?* Our answer is, in principle, yes, because qualitative models are abstractions of the equations. But, "in principle" is not enough for convincing management - it has to be for real, well-supported, and preferably automated.

What is the appropriate level of a qualitative model? is a second question, which we often ask ourselves, and experience says "it depends". It depends on what you need the model for, and what problem you want to solve. This is anything but surprising, but it is very critical: Much of the feasibility and economic attractiveness of model-based solutions stems from the extensive re-use of model fragments taken from a domain-specific library across different systems and tasks. The contradiction between the genericity of models (to be re-usable) and their task-dependent specificity (to be effective and efficient) can be fatal for the industrial success of model-based systems. If we use too coarse-grained models (e.g. with signs only), a diagnostic system may be unable to detect certain symptoms. On the other hand, if the model is too fine-grained, it may be inefficient w.r.t. time and space requirements. For instance, the real-time performance of our on-board diagnosis system described in [Sachenbacher et al. 2000b] crucially depends on the qualitative abstraction of the models and the observed signals. Consequently, unless we find ways to automatically transform a generic model from a library into a model tailored to a particular task, the applicability of model-based techniques will be limited.

8.2 A Tiny Example

In order to illustrate the problem and the key ideas of our work, we introduce a simple example. The system comprises a reservoir (which is assumed to be never empty and not shown in Figure 8-1), filled with liquid with pressure $T_1.p$. It is connected via a valve with maximal diameter A_{max} to an outlet pipe that fills a container with bottom area B and vertical walls (see Figure 8-1). The task is to use a model in order to design the control scheme that opens and closes the valve in order to fill the container up to a given height h with a precision of $\Delta h > 0$.

The example does not appear to be an industrial application at first glance. However, consider it to be a simplification of a controlled injector (i.e. a valve) that is to supply a certain amount of diesel fuel to the combustion chamber (i.e. a container) of a car engine. Below h, the fuel mixture will be too lean, above $\Delta h + h$, there will be too much fuel in the cylinder to burn it completely, a situation which should be avoided in any case.

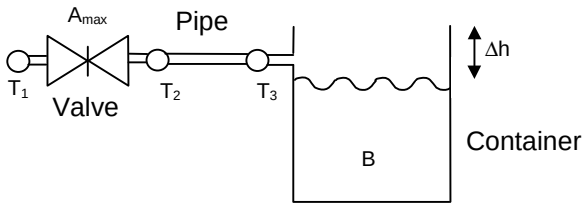


Figure 8-1: Filling a container

All the components in the system are fairly standard, and we expect to find their behavior model fragments in a library in order to compose a model of the example system. For instance, the valve model will have to associate the obvious equation

$$T_1.q = A \cdot sgn(T_1.p - T_2.p) \, (| T_1.p - T_2.p |)^{0.5}$$

with the states open and closed of the valve. But what about the transitions between the states? Do we have to explicitly model the opening and closing state, during which some amount of liquid is pouring into the container? Or can we model the transition from $A = A_{max}$ to $A = 0$ as a discontinuous change, i.e. neglect the duration of the valve closing operation? The answer, of course, depends on the targeted precision Δh and on the

characteristics of the entire configuration, namely T_1.p, T_2.p, A_{max}, and B. Perhaps, the container is so large that the increase of the height during closing of the valve is negligible. Perhaps, the time required for closing the valve is quite significant, because the required precision Δh is tight and/or because the pressure difference T_1.p - T_2.p is high. Given the contextual conditions of the task, we are able to compute a boundary for durations of the opening that are insignificant w.r.t. the required precision Δh. Based on the result, we can decide whether or not it is appropriate to approximate the valve model by one with discontinuous transitions. Note that the result will be influenced by another factor: the precision of the inputs to the calculation. For instance, if we just know that the diameter is between zero and A_{max} during opening of the valve, we obtain a boundary on the duration which is smaller than the one calculated for an opening that is described as linear within a certain tolerance. In the same way, the precision of the pressure values influences the distinctions on the duration. Finally, note that the decision about the appropriate distinctions in the valve model affects other component models as well. If the opening has to be modeled in a detailed way, then this requires the pipe model and the container model to include certain distinctions of the flow value and of the volume and the level, respectively.

Let us summarize the key insights from this example that form the starting point for our work:

- The distinctions to be made within a (component) model (w.r.t. system variables, time, and states) depend on the task, the context, and conditions on the inputs.
- The **task** was characterized by some **target distinctions** to be made (in our case, Δh).
- The **context** is given by the structure of the system (e.g. a second valve and inlet would influence the model of the first one).
- The **conditions** appear as certain **initial distinctions** that are possible (reflecting, for instance, how precisely certain variables can be measured).
- Resulting from this are certain **induced distinctions** on other variables (in our example, whether the finite duration of the opening has to be distinguished from a zero duration).

The thrust of the work reported here is to turn these intuitions into a formal theory and algorithms in order to automate the task of goal-dependent qualitative abstraction. Although our results are still preliminary, they extend, refine, and partially correct the theoretical foundations presented in [Struss Sachenbacher 1999], including now a prototype system and results carried out based on its implementation.

The following section summarizes the formal foundations and extends the notion in [Struss Sachenbacher 1999] by including the role of initial distinctions and by analyzing the impact of different purposes of the model (consistency check vs. prediction), the uniqueness of solutions, and completeness of proposed algorithms. Then we describe our current implementation (of an approximation algorithm) and present experimental results on the introductory example and the pedal position sensor example that was outlined in [Struss Sachenbacher 1999]. Finally, we discuss possible modifications and extensions of the current algorithms as well as alternative approaches.

8.3 Foundations

8.3.1 Basic Definitions and Concepts

In this section, we present the theoretical foundations of our work, trying to turn the intuitive ideas extracted from the example into formal concepts. The theory we presented in [Struss Sachenbacher 1999] will be summarized as briefly as possible. We will indicate when we omit the repetition of a formal definition from that chapter, and we try to highlight the extensions and modifications.

Behavior Models. Our theory is based on **relational** models, i.e. the behavior of a component or system, S, is given by a relation

$$R_S \subset DOM(\underline{v}_S) ,$$

where $\underline{v}_S = v_1 \times v_2 \times ... \times v_n$ is the vector of all parameters and variables in the system. This allows us to treat symbolic values, discrete states, behavior modes, and non-deterministic models rather than only real-valued functions which are subsumed as a special case. The relation R_S can be given implicitly as the composition of several component models, as in our example.

Domains. We assume that there exists a base domain $DOM_0(v_i)$ for each variable v_i, from which the desired task-specific domains can be obtained by means of abstraction, and a base model

$$R_{S,0} \subset DOM_0(\underline{v}_S) = DOM_0(v_1) \times ... \times DOM_0(v_n).$$

Distinction. This central concept of the problem is defined as a partition of the base domain, i.e. a set of non-empty disjoint subsets

$$\Pi_i = \{P_{i,k}\} \subset P(DOM_0(v_i))$$

that together cover the entire domain (P(X) is the power set of X). Each $P_{i,k}$ is a qualitative value which represents a set of values that are not distinguished from each other.

Target distinctions. As pointed out in the discussion of the introductory example, the target distinctions characterize the goal of a particular task to be performed based on a model and are represented as a set of target partitions[7]

$$\Pi_{targ,i} = \{P_{targ,i,k}\} \subset P(DOM_0(v_i))$$

If we are not interested in any distinctions of a variable vi, then we can express this by the trivial partition

$$\Pi_{targ,i} = \{DOM_0(v_i)\}.$$

Initial Distinctions. Analogously, the conditions of the model-based task are expressed as a set of initial partitions $\Pi_{obs,i}$. They represent what will be the granularity of inputs to the model, e.g. observations or partially described hypothetical situations. All the other variables receive the trivial partition as their initial one.

Qualitative Domain Abstraction. Finding qualitative values for all variables which can then replace the original, fine-grained domain is the goal of our enterprise. Given a set of qualitative values (i.e. a partition), each value of the fine grained domain can be mapped to its respective qualitative value. We call such a mapping .

$$\tau: DOM_0(v_i) \rightarrow DOM_\alpha(v_i) \subset P(DOM_0(v_i))$$

a domain abstraction. Dependent on the kind of partition that induces a domain abstraction, we will give τ an index "targ" for target distinctions or "obs" for initial distinctions. There is a dualism between the domain partitions and qualitative abstractions, because each domain abstraction that produces mutually exclusive values induces a partition, which is given by

$$\Pi_{\tau,i} = \tau(DOM_0(v_i)) .$$

We can apply the concepts of **refinement** of a partition (i.e. a partition that makes further distinctions compared to the elements of another partition) and **merge** of partitions (i.e. the maximal partition which is a refinement of both) to domain abstractions, as well. For a rigorous treatment and definitions of these operations, see [Struss Sachenbacher 1999].

In this formalism, we can define our goal precisely. Whatever model we use, the only thing that matters is the information it can infer about the target partitions from the possible initial partitions. We can express this interest by abstracting each result of inference to the level of target partitions by the respective domain abstraction. Let

$$\tau_{targ} = (\tau_{targ,1}, \tau_{targ,2}, ... \tau_{targ,n}) : DOM_0(\underline{v}_S) \rightarrow \Pi_{targ} := \Pi_{targ,1} \times \Pi_{targ,2} \times ... \times \Pi_{targ,n} .$$

If we supply the base model, $R_{S,0}$, with some information, i.e. some restrictions on system variables expressed in terms of initial partitions,

$$R_{obs} \subset \Pi_{obs} := \Pi_{obs,1} \times \Pi_{obs,2} \times ... \times \Pi_{obs,n},$$

then we are interested in

$$\tau_{targ}(R_{obs} \cap R_{S,0}) ,$$[8]

[7] We changed terminology compared to [Struss Sachenbacher 1999], because we felt that the term "primary partitions" used in this chapter is misleading as it suggests they would be the starting point of the task.

[8] To simplify the notation, here and in the following we implicitly map the initial partitions to DOM_0 whenever necessary, i.e. we identify $R_{obs} \subset \Pi_{obs}$ with $R'_{obs} = \tau^{-1}_{obs}(R_{obs})$. Furthermore, we implicitly extend each mapping, τ, defined on some set, S, to the power set P(S) by

and our goal is to find qualitative domain abstractions that do not change this result. Hence, we obtain the following definition of our goal concept:

Definition (Distinguishing Qualitative Domain Abstraction)

Let $R_{S,0} \subset DOM_0(\underline{v}_S)$ be the base model of a system S, and Π_{targ} and Π_{obs} the target partitions and initial partitions, respectively. A qualitative domain abstraction

$$\tau: DOM_0(\underline{v}_S) \rightarrow P(DOM_0(\underline{v}_S))$$

is distinguishing w.r.t. $\{\Pi_{targ}, \Pi_{obs}\}$ iff it is a refinement of τ_{targ} and

$$\forall R_{obs} \subset \Pi_{obs}:$$
$$\tau_{targ}(\tau(R_{obs}) \cap \tau(R_{S,0})) = \tau_{targ}(R_{obs} \cap R_{S,0}). \quad (1)$$

A distinguishing abstraction, τ, is **maximal** iff there exists no other distinguishing abstraction, τ', that is a refinement of τ.

Condition (1) requires that the abstraction preserves information about the **tuples** of target partitions. However, a weaker and often sufficient goal for some applications is to postulate that **for each single target variable**, the possible distinctions are maintained under the abstraction. If $proj(v_1, v_2, ..., v_k)$ denotes projection on a set of variables $v_1, v_2, ..., v_k$, then this can be expressed as

$$\forall R_{obs} \subset \Pi_{obs}, \forall v_{targ}:$$
$$proj(v_{targ})(\tau_{targ}(\tau(R_{obs}) \cap \tau(R_{S,0}))) =$$
$$proj(v_{targ})(\tau_{targ}(R_{obs} \cap R_{S,0})). \quad (1')$$

8.4 The Scope of R_{obs}

In contrast to the corresponding definition in [Struss Sachenbacher 1999], we will capture the set of initial restrictions, i.e. the starting point of the model-based computation, and their impact on the resulting qualitative abstractions in a more rigorous and appropriate way. In that chapter, we allowed arbitrary initial restrictions $R_{obs} \subset DOM_0(\underline{v}_S)$, called "$R_{ext}$". But in practice, the initial restrictions are limited in basically two ways:

- only a subset of the variables form a potential input to the model (e.g. because only certain variables are observable),
- the input values usually have limited precision which can be coarser than DOM_0. This may be due to the precision of a sensor or a partial specification of some input to the model (e.g. when analyzing a circuit's behavior "with a voltage source of more than 6V").

8.4.1 Uniqueness Properties of Maximal Distinguishing Abstractions

It turns out that without further restrictions on R_{obs}, maximal distinguishing abstractions are not uniquely defined. Consider a very simple example that consists of two equality constraints

$$x = y = z .$$

Let us assume that for all variables, the base domain is the set of integers, that x and z are observable at this granularity, and that the only target distinction is to determine the sign of y:

$$\Pi_{targ,y} = \{\{(-\infty, 0)\}, \{0\}, \{(0, \infty)\}\} .$$

For unique initial distinctions (i.e. there is always a value given for both x and z), there are obviously two maximal distinguishing abstractions, given by the partitions

$$\Pi_{\tau,x} = \{\{(-\infty, 0)\}, \{0\}, \{(0, \infty)\}\} , \Pi\tau,y = \{\{(-\infty, \infty)\}\}$$

and

---

defining $\forall X \subset S \;\; \tau(X) := \cup \{\tau(x) \mid x \in X\}$.

$$\Pi'_{\tau,x} = \{\{(-\infty, \infty)\}\} , \Pi'\tau,y = \{\{(-\infty, 0)\}, \{0\}, \{(0, \infty)\}\},$$

which simply reflects the fact that one of x and z is not needed for determining the target distinction. In contrast, if the initial restrictions of x and y are made independently of each other, this includes situations in which one of the variables is not restricted, leading to a unique distinguishing abstraction

$$\Pi_{\tau,x} = \{\{(-\infty, 0)\}, \{0\}, \{(0, \infty)\}\},$$
$$\Pi'\tau,y = \{\{(-\infty, 0)\}, \{0\}, \{(0, \infty)\}\} .$$

To reflect this difference w.r.t. restrictions on R_{obs}, we further refine our definition. A distinguishing abstraction τ is called **maximal for independent initial restrictions**, if R_{obs} in (1), (1') is of the form

$$R_{obs} = R_{obs,1} \times R_{obs,2} \times ... \times R_{obs,n} \subset \Pi_{obs}.$$

The case of independent initial restrictions captures a very common case of model-based reasoning, namely to perform prediction (or consistency checking) after restricting a number of variables to a certain value (set) independently of each other. This is the case, for instance, if the input is the result of measuring certain system variables. The only important exception we can think of is the case when time is included as a variable: if the input is a time-stamped set of variable values, one has to use the general definition.

A still further specialization is obtained, when there are unique initial distinctions, i.e. each R_{obs} is a tuple of single qualitative values:

$$R_{obs} \in \Pi_{obs} .$$

An example is on-board diagnosis, where at each instance in time, unique values (at the level of Π_{obs}) are present for a fixed set of variables. In contrast, the general case of independent initial distinctions captures the situation where there is a set of variables that are observable, but we are looking for an abstraction that allows to draw the strongest conclusions also if only a subset of the variables have actually been measured (the non-measured ones receive no restriction).

8.4.2 Different Tasks: Consistency Check vs. Prediction

The definition of distinguishing abstractions nicely expresses that we are looking for qualitative abstractions that do not reduce the information about the target variables. In [Struss Sachenbacher 1999], we have shown that sometimes, every abstraction entails a loss, and one is stuck with the base domain. We discuss one fundamental reason for this.

Condition (1) includes the case where R_{obs} is **inconsistent** with the model, $R_{S,0}$. This could well be a situation in practice; e.g. in model-based diagnosis or in design verification, we are interested in determining whether or not a set of observations is in conflict with a behavior model. In this case,

$$R_{obs} \cap R_{S,0} = \varnothing$$

holds, and condition (1) implies

$$\tau(R_{obs}) \cap \tau(R_{S,0}) = \varnothing \Leftrightarrow R_{obs} \cap R_{S,0} = \varnothing.$$

Note that this imposes a condition on τ which is **independent of the target partitions** and reflects only $R_{S,0}$ and the initial distinctions, represented by R_{obs}. If the latter have the granularity of the base domain, the very "shape" of $R_{S,0}$ may exclude an abstraction, because the above implies

$$\underline{v}_0 \in R_{S,0} \Leftrightarrow \tau(\underline{v}_0) \subset \tau(R_{S,0}) ,$$

and for any $\underline{v}_1 \in R_{S,0}$ with

$$\tau(\underline{v}'_0) = \tau(\underline{v}_0) ,$$

we obtain

$$\begin{aligned} \underline{v}_0 \in R_{S,0} \Leftrightarrow \tau(\underline{v}_0) \subset \tau(R_{S,0}) \\ \Leftrightarrow \tau(\underline{v}'_0) \subset \tau(R_{S,0}) \\ \Leftrightarrow \underline{v}'_0 \in R_{S,0} , \end{aligned}$$

This means all values that are mapped onto $\tau(\underline{v}_0)$ are in $R_{S,0}$, and since $\tau(\underline{v}_0)$ contains exactly these values, we have

$$\underline{v}_0 \in R_{S,0} \Leftrightarrow \tau(\underline{v}_0) \subset R_{S,0}.$$

Hence, a distinguishing abstraction can never "cross the boundaries" of $R_{S,0}$. Since

$$\tau(\underline{v}_0) = \tau(v_1) \times ... \times \tau(v_n) \in R_{S,0}$$

τ is limited by the existence of "rectangular building blocks" within $R_{S,0}$. Figure 8-2 provides an abstract illustration: $R_{S,0}$ allows an abstraction in case a), but not in case b).

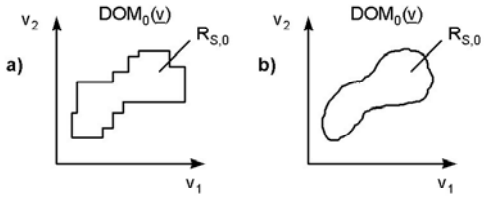


Figure 8-2: Different $R_{S,0}$ with different potential for abstraction

In other applications, the model is not intended to be used for consistency check with $R_{S,0}$, but only to compute consequences of the initial restrictions which are assumed to be consistent with the model. Under this assumption, condition (1) can be weakened as follows

$$\forall R_{obs} \subset \Pi_{obs}: \\ R_{obs} \cap R_{S,0} \neq \varnothing \Rightarrow \tau_{targ}(\tau(R_{obs}) \cap \tau(R_{S,0})) = \\ \tau_{targ}(R_{obs} \cap R_{S,0}). \quad (2)$$

This means that τ, based on initial values that are consistent with the model, manages to predict the target distinctions properly, but it is also allowed to map certain inconsistent tuples to consistent ones. However, also in this case, we will sometimes not be able to move away from the real numbers as a base domain, e.g. for a simple multiplication constraint on real numbers, whereas we obtained a nice abstraction when starting with a discretized domain (see [Struss Sachenbacher 1999]). This highlights the importance of a definition and algorithms that reflect initial partitions (e.g. granularity of observations), because they introduce a discretization. Also discrete states and operating modes may induce partitions on continuous variables, as will be illustrated in the results section.

8.5 Characterizing Distinguishing Domain Abstractions

While (2) defines the desired property of the abstractions, i.e. our goal, we need some constructive characterization as a starting point for designing algorithms to compute them.

Exploiting the duality between domain abstractions and domain partitions pointed out in the previous section, we will characterize τ by constructing the appropriate partition of the observable domains.

To simplify the presentation, we restrict ourselves in the following to the special situation where the observed (or otherwise given) initial restrictions have a granularity equal to the base domain DOM_0, i.e. $\Pi_{obs,i}$ is the identical mapping. We will return to the more general case later.

The question can be stated as "*Which values can be joined in a partition (i.e. abstracted to the same qualitative value under τ) without violating (2)*?". The intuitive answer, based on (2) is: If two values $v_{j,1}$, $v_{j,2}$, of a variable v_j, when combined with at least one consistent initial restriction for other variables, are consistent with different sets of target values, then we cannot join them without losing information on the level of the target distinctions. Otherwise, we can, because it is then guaranteed that there exists no initial restriction that leads to different predictions on the level of target distinctions if we drop the distinction between $v_{j,1}$ and $v_{j,2}$. Formally, we can introduce an equivalence relation for $v_{j,1}, v_{j,2} \in proj_j(R_{S,0})$ by:

$$v_{j,1} \approx_j v_{j,2} :\Leftrightarrow \\ \forall R_{obs,-j} = R_{obs,1} \times ... \times R_{obs,j-1} \times R_{obs,j-+1}. \times ... \times R_{obs,n} \subset R_{S,0}:$$

$$\tau_{targ}(\text{select } (v_j = v_{j,1})\ (R_{obs,-j} \cap R_{S,0})) =$$
$$\tau_{targ}(\text{select } (v_j = v_{j,2})\ (R_{obs,-j} \cap R_{S,0}))\quad (3)$$

Here, select$(v_j = v_{j,1})(R)$ is the restriction of a relation R to the set of tuples that satisfy the condition $v_j = v_{j,1}$.

Definition (Induced Partitions)

Let $\approx_j$ be the equivalence relation on $DOM_0(v_j)$ defined by (3). The sets of partitions $\Pi_{ind,j}$ for $DOM_0(v_j)$ given by the equivalence classes of the relations $\approx_j$,

$$\Pi_{ind,j} := DOM_0(v_j) \,|\, \approx j\ ,$$

are called the partitions induced by the target distinctions.

This defines a distinguishing domain abstraction τ_{ind}:

Lemma (Characterization of distinguishing domain abstractions)

The induced partition

$$\Pi_{ind} := \times \Pi_{ind,j}$$

defines a distinguishing qualitative abstraction τ_{ind} w.r.t. Π_{targ} .

Conjecture (Maximality of the induced partition)

The induced partition is a maximal distinguishing qualitative abstraction.

Analogously, we obtain a characterization of distinguishing qualitative abstractions under the weaker condition (1') which states that determining the distinctions for each single target variable suffices:

$$v_{j,1} \approx_j v_{j,2} :\Leftrightarrow$$
$$\forall\, R_{obs,-j} = R_{obs,1} \times ... \times R_{obs,j-1} \times R_{obs,j-+1}. \times ... \times R_{obs,n} \subset R_{S,0}\,, \forall\, v_i:$$
$$\text{proj}(v_i)\ \tau_{targ}(\text{select } (v_j = v_{j,1})\ (R_{obs,-j} \cap R_{S,0})) =$$
$$\text{proj}(v_i)\ \tau_{targ}(\text{select } (v_j = v_{j,2})\ (R_{obs,-j} \cap R_{S,0}))\quad (3')$$

8.6 Approximations to Induced Partitions

Conditions (3) and (3') formulate that a distinction between values of v_j is made if and only if this leads to a target distinction, potentially in conjunction with other observations.

From a computational point of view, this is problematic, since it requires to consider mutual combinations of values, plus all the possible (combinations of) observations, to determine if they have to be distinguished.

In this section, we develop a characterization that avoids these difficulties by starting from the target distinctions instead. As we will see, this characterization is more amenable to computation, but at the price that the resulting partitioning represents only a subset of the necessary distinctions.

Definition (Approximation of Induced Partitions)

For $v_{j,1}, v_{j,2} \in \text{proj}(v_j)(R_{S,0})$, let

$$v_{j,1} \approx_4 v_{j,2} :\Leftrightarrow$$
$$\forall\, R_{targ} := R_{targ,1} \times ... \times R_{targ,n} \subset \tau_{targ}(R_{S,0}):$$
$$v_{j,1} \in \text{proj}(v_j)(R_{targ,} \cap R_{S,0}) \Leftrightarrow v_{j,2} \in \text{proj}(v_j)(R_{targ,} \cap R_{S,0})\quad (4)$$

The partition defined by the equivalence relation (4) is called approximation of the induced partition, and τ^*_{ind} denotes the corresponding domain abstraction.

Again, we can formulate a corresponding approximation for the weaker conditions (1'), (3').

The approximation τ^*_{ind} is not necessarily a distinguishing abstraction. Since (4) is only a necessary condition of (3), it can fail to preserve all possible target distinctions. However, the distinctions of the approximation are

necessary ones, and, hence, all distinguishing abstractions are refinements of the approximation. Formally, this is captured by the following proposition.

Proposition

τ_{ind} is a refinement of τ^*_{ind} .

Proof:

Assume that $v_{j,1}$, $v_{j,2}$ are distinguished in τ^*_{ind} , i.e. $v_{j,1} \approx_4 v_{j,2}$ does not hold. Then, there exists at least one initial restriction $R_{targ,}$ such that either $v_{j,1} \in proj(v_j)(R_{targ,})$ and $v_{j,2} \notin proj(v_j)(R_{targ,})$, or $v_{j,1} \notin proj(v_j)(R_{targ,})$ and $v_{j,2} \in proj(v_j)(R_{targ,})$. Since $R_{targ,} \subset \tau_{targ}(R_{S,0})$, $v_{j,1}$ combined with R_{targ} yields a restriction on the level of target distinctions that is different from the restriction resulting from combining $v_{j,2}$ with R_{targ}, i.e. τ_{targ} (select $(v_j = v_{j,1})$ $(R_{targ} \cap R_{S,0})) \neq \tau_{targ}$ (select $(v_j = v_{j,2})$ $(R_{targ} \cap R_{S,0})$). From condition (3), it follows that $v_{j,1}$, $v_{j,2}$ must also be distinguished in τ_{ind}.

So far, we have assumed that all distinctions in DOM_0 can be observed. Now, we consider the case that Π_{obs} is not given by the identical mapping.

8.7 Observable Distinctions

Obviously, what distinctions can be provided by the initial (observable) variable crucially affects the distinctions that can be derived for other variables. It does not make sense to introduce distinctions for some intermediate variables that would help to determine the target distinctions, but that never become effective because the initial distinctions are not precise enough to obtain them. Intuitively, two values of a variable v_j cannot be distinguished by the observable distinctions, if, for any observable tuples, they are either both consistent or both inconsistent with this tuple, or, stated differently, if they co-occur in the projections of the observable tuples.

Definition: Induced observability

For $v_{j,1}, v_{j,2} \in proj(v_j)(R_{S,0})$, let

$v_{j,1} \approx_5 v_{j,2} :\Leftrightarrow$
$\forall R_{obs} := R_{obs,1} \times ... \times R_{obs,n} \subset \tau_{obs}(R_{S,0})$:
$v_{j,1} \in proj(v_j)(R_{obs,} \cap R_{S,0}) \Leftrightarrow v_{j,2} \in proj(v_j)(R_{obs,} \cap R_{S,0})$ (5)

Let $\tau_{ind,obs}$ denote the domain abstraction corresponding to the partitions induced by (5).

The abstraction $\tau_{ind,obs}$ captures all distinctions that can in principle be revealed by initial restrictions, for example, be observed. The general case of arbitrary initial partitions $\Pi_{obs,i}$ is thus captured by replacing $R_{S,0}$ with $\tau_{ind,obs}(R_{S,0})$ in definitions (3) to (4).

Note that (5) has the same form as (4), with $R_{targ,}$ replaced by R_{obs}. The partition on v_j that results from τ^*_{ind} and $\tau_{ind,obs}$ can be reformulated as a merge operation

merge (proj(v_j) (select($\underline{v}_S = P$)$R_{S,0}$)

running over all tuples

$P := P_{targ,1} \times ... \times P_{targ,n} \in \tau_{targ}(R_{S,0})$

in the case of τ^*_{ind} , and

$P := P_{obs,1} \times ... \times P_{obsn} \in \tau_{obs}(R_{S,0})$

in the case of $\tau_{ind,obs}$. This means that both τ^*_{ind} and $\tau_{ind,obs}$ can be computed using the same basic algorithm, as just the different sets of tuples used in the select statement account for the difference.

Now, computing distinguishing domain abstractions under a given granularity of initial distinctions has reduced to computing and intersecting projections. Yet, they are still described as global computations on the system relation $R_{S,0}$. This will be the topic of the following section.

8.8 Computing Induced Abstractions for Composed Relations

The problem of determining significant distinctions arises, in particular, due to composing generic behavior models taken from a library of model fragments. Therefore, we extend the concepts of computing induced distinctions to relations that are composed of several individual behavior model fragments. In the following, the relation $R_{S,0}$ is considered to be a composed relation of the form

$$R_{S,0} = R_{c1,0} \cap R_{c2,0} \cap ... \cap R_{cn,0} ,$$

where the $R_{ci,0}$ represent the relational model for component C_i. If necessary, the $R_{ci,0}$ include also mode variables representing different behavior modes or fault modes of the components.

The interaction between a variable v_j and targeted or initial distinctions in other variables v_i is then not direct, but instead mediated by a sequence of variables in a set of component relations. This leads to the idea of determining the induced qualitative values from the primary distinctions by propagation through the structure of the model. In [Struss Sachenbacher 1999], we described the foundations for an approach to propagate distinctions to other parts of the system and then to compute the distinctions for one component relation locally.

For this, two problems have to be overcome that were not covered in detail in [Struss Sachenbacher 1999].

First, it is possible that tuples occurring in the relations $R_{ci,0}$ are inconsistent with the aggregate system model $R_{S,0}$. The problem is that the presence of inconsistent tuples can both lead to results which are too coarse and to results which are too fine-grained. Intuitively, inconsistent parts of the relation could make it appear either more "rectangular" or less "rectangular" (in the sense of Figure 8-2), which either enables or blocks possibilities for domain abstractions. Thus, in order to ensure correctness of the results, a complete elimination of inconsistent tuples is required. This shows that determining significant distinctions is an inherently hard problem, because as a consequence of the above, the problem of determining whether a given distinction is significant or not w.r.t. the target distinctions is as least as hard as checking satisfiability of the involved behavior constraints, which is NP-hard in general.

The second issue is that the components C_i have to exchange information about their distinctions, since a distinction in the domain of one components' variable could be necessary to make a desired distinction in another component. In order to avoid cycles in the local computation (which would mean determining distinctions in one variable which aim at target distinctions for the same variable), it is necessary to provide additional information about the origin of each propagated distinction. We accomplished this by creating a label for each partition element, which contains a reference the target partition element it was introduced for. Propagation rules then ensure that loops in the computation are eliminated and the introduced induced distinctions reflect indeed only the target distinctions (details are described in [Kutscha 00]).

Furthermore, it is quite clear that the local propagation algorithm is incomplete, with the consequence that it will in general not compute τ^*_{ind} and $\tau_{ind,obs}$, but approximations thereof. For the result, however, the proposition stated in the last chapter still holds: a distinction that has been derived by the propagation algorithm is guaranteed to be a necessary distinction, although not all such distinctions might be found.

The next section describes the implemented prototype in more detail.

8.9 A Prototype System for Automated Qualitative Model Abstraction

Following the theory outlined above, we implemented a prototypic system that automatically computes approximations of maximal distinguishing abstractions for a given compositional system model.

The prototype (described in detail in [Kutscha 00]) uses software components from the commercial model-based systems framework Raz'r ([Iacucci Struss 1998]). The Raz'r development system allows to compose a system model from a library of model fragments. The prototype reads in the resulting system description, which consists of a definition of domains, constraint types and behavior models.

The task-dependent initial and target partitions Π_{obs}, Π_{targ} are given as XML documents. Alternatively, they can be defined interactively using a domain partition editor.

In a first step, tuples inconsistent with $R_{S,0}$ will be eliminated from each component model. Based on this, the prototype uses the propagation algorithm outlined in the previous section to iteratively compute the abstraction

$\tau_{ind,obs}(R_{S,0})$. In a third step, this result is used as a starting point for the computation of qualitative distinctions based on the definition of τ^*_{ind}.

The local computation steps for the $R_{ci,0}$ are based on a constraint system that represents the relational behavior models (more precisely, their characteristic functions) as ordered binary decision diagrams (OBDDs, see [Bryant 1992]). This data structure allows for fairly efficient realization of the required manipulations of the constraints, such as join or projection on a subset of variables. However, this also results in the limitation that the system can currently only transform behavior models that have finite domains.

The resulting qualitative domain abstraction τ^*_{ind} can be applied to the system model, leading to a transformed system description. Variables in the model that are found to have no induced distinction after computation (i.e. whose domain consists of one element only) can optionally be eliminated form the model. If all variables of a component have been eliminated, this may eventually lead to the elimination of a component from the model.

The transformed system description can be fed back into the Raz'r framework and be used for task such as behavior prediction, diagnosis, or test generation. Additionally, it is possible to specify domain abstractions that link the elements of DOM_0 to finer domain, in particular to a subset of the real numbers. By concatenating such domain mappings, this allows to maintain a mapping to the real numbers for the induced qualitative values. This information is used by a signal transformation component that accomplishes the transformation of (real-valued) sensor readings to the abstraction level of the qualitative model.

The next section presents results of running the implementation on two small examples. Note that since τ^*_{ind} characterizes an upper approximation of the induced partitions only, the computed sets of distinctions will necessarily be incomplete.

8.10 Computational Results

8.10.1 Container Filling Example

We first present the application of our theory and implementation to an instance of the container filling problem that was presented in the introduction.

Initial distinctions: The domain for all pressure, flow and parameter variables were chosen to be a 19-valued domain that consists of open intervals and points between them:

Π_{obs} = {{(-∞, -4)}, {-4}, {(-4, -3)}, {-3}, ..., {(3,4)}, {4}, {(4, ∞)}}.

The model formulated in this domain consists of 12 variables and 2 mode variables, which means it has a tuple space (i.e. size of $DOM_0(\underline{v})$) of $19^{12} \cdot 2^2 \approx 8.9 \cdot 10^{15}$.

Target distinction: There are two behavior modes for the container component, one that captures the situation where the container level is within the specified range (h + Δh) and one that represents overflow. Thus, the target distinction for the example can be stated by giving a distinction for the mode variable of the tank component.

What we are after is an answer to the question whether or not we have to include the transition delay of the valve's closing operation, denoted Δt, in the model.

Induced distinctions. The following result is obtained for parameters A_{max} , Δh, B, and T_1.p set as

Amax = {(0,1)},
Δh = {(2,3)},
B = {1},
T1.p ∈ {{(2,3)}, {3}, {(3,4)}}.

After computation of the composed relation RS,0, 21036 tuples remain consistent for the valve component. The algorithm determines the following induced distinctions for the valve delay parameter:

$\Pi_{ind,\ valve.\Delta t,\ 1}$ = {(-∞,-4), -4, (-4,-3), -3, (-3,-2), -2, (-2,-1), -1, (-1,0), 0, (0,1), 1}
$\Pi_{ind,\ valve.\Delta t,\ 2}$ = {(1,2), 2, (2,3), 3, (3,4), 4, (4, ∞)}

The first partition element corresponds to situations where overflow is not possible. The second partition element corresponds to situations where overflow of the container is possible (though will not necessarily

occur). This means that all situations where the delay during closing the valve is part of the first partition element (i.e. is equal to or smaller than 1) are equivalent to no (zero) delay. In other words, the transition delay can be neglected in the model of the valve if it is part of the first qualitative value, and must be considered in the model if it is part of the second qualitative value.

To give an idea of the problem size, a multiplication constraint that uses the base domain consists of 713 tuples out of a tuple space of $19^3 = 6859$ (which amounts to a ratio of 10.4% consistent combinations of values; for a domain that consists just of signs, the respective constraint would have had 9 tuples out of a tuple space of $3^3 = 27$, i.e. a ratio of 33.3% consistent tuples). The constraint describing the valve equation mentioned in the introduction, for instance, has 23075 tuples. Compared to $|DOM_0(\underline{v}))| \approx 8.9 \cdot 10^{15}$ for the original behavior model, the abstracted behavior model has a tuple space $|\tau^*_{ind}(DOM_0(\underline{v})|$ of only $1.2 \cdot 10^4$.

We can also start with a given parameter for the delay and ask for a partition for the mode variable of the valve component, given the same target distinction for the container. Depending on the magnitude of the delay, it will turn out to be necessary or unnecessary to explicitly distinguish the two behavior modes of the valve ("closing" and "closed"). For the given parameters, we obtain one partition element,

$$\Pi_{ind,\ valve.mode} = \{closing, closed\}$$

i.e. the distinction between the two behavior modes becomes irrelevant, if and only if the parameter for delay is restricted to the first qualitative value.

8.10.2 Pedal Position Sensor Example

As a second example, we list results for a problem that was originally presented in [Struss Sachenbacher 1999]. The device in Figure 8-3 shows a pedal position sensor in a passenger car. Its purpose is to deliver information about the position of the accelerator pedal to the electronic control unit (ECU) of the engine management system. The position is sensed in two ways, via the potentiometer as an analogue signal, v_{pot}, and via the idle switch as a binary signal, v_{switch}. The idle switch changes its state at a particular value of the mechanically transferred pedal position. The reason for the redundant sensing of the pedal position is that the signals from the potentiometer and the switch are cross-checked for plausibility by the on-board control software of the ECU. The two possible values of v_{switch} correspond to two ranges of v_{pot} separated by a particular voltage value.

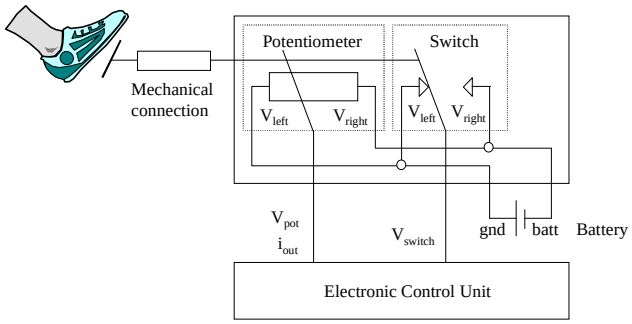


Figure 8-3: The Pedal Position Sensor

If we use qualitative models which distinguish only e.g. between voltage "gnd", "between" and "batt" as convenient for many applications, we are unable to perform tasks such as diagnosis or design verification, because these tasks refer to the redundancy which purposefully has been implemented in the system. The problem is that the particular distinction in the domain of v_{pot} cannot be anticipated in a generic model of the potentiometer, because the voltage landmark would not make any sense in a different structure.

We would like to have a composition of component models that make just the right distinctions required by the other components and the task the model is used for.

To this end, we can use approach and the software component outlined above to compute an approximation of the necessary qualitative distinctions.

Initial distinctions: We chose

$\Pi_{obs,\ *.voltage}$ = {{[0,.2)}, {[.2,.4)}, {[.4,.6)}, {[.6,.8)}, {[.8,1.0)}}

for variables involving voltage and likewise to values

$\Pi_{obs,\ *.position}$ = {{0}, {.2}, {.4}, {.6}, {.8}, {1.0}}

for variables involving position. The switch-over parameter of the idle switch component is specified as the value {.4}.

Target distinction: This is determined by the goal to distinguish between the ground voltage, corresponding to partition element {[0,.2)}, and the rest of the domain for terminal variable v_{switch} (termed switch.output.voltage in the model) of the control unit component:

$\Pi_{targ,\ switch.output.voltage,\ 1}$ = {[0,.2}
$\Pi_{targ,\ switch.output.voltage,\ 2}$ = {[.2,.4), [.4,.6), [.6,.8), [.8,1.0)}

Induced distinctions. The algorithm first determines that it is necessary to distinguish between the two values

$\Pi_{ind,\ switch.state,\ 1}$ = {right}
$\Pi_{ind,\ switch.state,\ 2}$ = {left}

for the state of the switch, and then induces two qualitative values

$\Pi_{ind,\ pedal.position,\ 1}$ = {.6, .8, 1.0}
$\Pi_{ind,\ pedal.position,\ 2}$ = {0, .2, .4}

for the domain of the pedal position. The resulting partition for v_{pot} (termed potentiometer.output.voltage in the model) consists of three partition elements:

$\Pi_{ind,\ potentiometer.output.voltage,\ 1}$ = {[0,.2), [.2,.4)}
$\Pi_{ind,\ potentiometer.output.voltage,\ 2}$ = {[.4,.6)}
$\Pi_{ind,\ potentiometer.output.voltage,\ 3}$ = {[.6,.8), [.8,1.0)}

The first qualitative value corresponds to the situation where v_{switch} equals ground voltage, the second qualitative value corresponds to the situation where v_{switch} equals battery voltage, and the third qualitative value corresponds to the situations where the position of the switch and thus the voltage of v_{switch} is ambiguous.

The resulting abstracted behavior model that achieves the same target distinctions as the original model has a tuple space of 9216. The original model, in contrast, had a tuple space of $5.6 \cdot 10^7$.

8.11 Discussion and Future Work

The work presented here has provided us with a theoretical foundation to analyze the fundamental problem of automated qualitative modeling in the context of model-based systems, and a first implementation of an algorithm to compute an approximate solution. The first experimental results were encouraging, as even the incomplete algorithm produced useful model abstractions. More experiments will have to be done, and some will aim at producing models for tasks and problems in our application domains of car subsystems and process-oriented modeling and diagnosis.

Answering the question to what extent we can scale up requires, on the one hand, analysis and improvement of the operations on the underlying OBDD encoding of the models. On the other hand, further improvements of the propagation algorithm need to be explored. For instance, rather than starting with the granularity of the base domain, there could be an iterative, top-down approach for determining qualitative distinctions which starts with some small set of partitions and refines them if there is evidence for its utility. This evidence could be provided by the labels of the induced qualitative values. They indicate which partition elements might be the best candidates for "splitting" (e.g. qualitative values with maximum label size, since this means that a large number of target partition elements is consistent with this value, i.e. the value lacks "discriminating power").

Our deeper theoretical analysis compared to [Struss Sachenbacher 1999] has shown that different tasks impose different requirements and definitions, e.g. consistency check vs. prediction, independent observations vs. observed tuples, and target distinctions vs. initial distinctions.

Another important technical aspect has not been discussed in this chapter, namely the relation between an algorithm for computing significant distinctions and the algorithm that uses the resulting model for prediction or diagnosis. For instance, if we had an algorithm that would actually obtain a maximal abstraction based on performing global consistency checking, a run time system that is performing value propagation on the resulting abstract model only is likely to be unable to take advantage of it and might miss some distinctions.

Acknowledgments

We would like to thank Alexander Kutscha, who worked on the implementation of the prototypic software in Visual Basic under Windows NT, and Oskar Dressler for helpful discussions and collaboration. This work was supported in part by the German Ministry of Education and Research (#01 IN 509 41).

9 Literatur Teil I

[Addanki et al. 1991]

Sanjaya Addanki, Roberto Cremonini and J. Scott Penberthy: Graphs of models. *Artificial Intelligence*, 51 (1-3) (1991) pp. 145-177.

[Bredeweg 1991]

Bredeweg, B., *Expertise in Qualitative Prediction of Behaviour*. Doctoral thesis, Faculty of Psychology, University of Amsterdam, 1991.

[Bosch 1996]

Robert Bosch GmbH (ed.): Automotive Handbook (4th edition). *Society of Automotive Engineers (SAE)*, Warrendale, 1996

[Bryant 1992]

Randal E. Bryant: *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*. Carnegie Mellon University Technical Report CMU-CS-92-160, 1992.

[Chantler et al. 1996]

M. J. Chantler, S. Daus, T. Vikatos and G. M. Coghill, The Use of Quantitative Dynamic Models and Dependency Recording for Diagnosis. *Workshop Notes of the 7th International Workshop on Principles of Diagnosis (DX-96)*, Montreal, 1996.

[Collins 1993]

Collins, J. W., *Process-based Diagnosis: An Approach to Understanding Novel Failures*. Doctoral thesis, Institute for the Learning Sciences, Northwestern University, 1993.

[Crawford et al. 1990]

James Crawford, Adam Farquhar, Benjamin Kuipers: QPC: A Compiler from Physical Models into Qualitative Differential Equations. In*: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-90)*, 1990.

[de Kleer 1986]

de Kleer, J.: Problem Solving with the ATMS, *Artificial Intelligence*, 28(2), pp. 197-224, 1986.

[de Kleer Williams 1987]

de Kleer, J. and Williams, B. C., Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97-130, 1987.

[Dressler 1996]

Dressler O., On-line Diagnosis and Monitoring of Dynamic Systems based on Qualitative Models and Dependency-based Diagnosis Engines. In: *Wahlster, W., (ed.), Proceedings of the European Conference on Artificial Intelligence (ECAI-96)*, John Wiley & Sons, 1996

[Dressler Struss 1996]

Dressler, O., Struss, P.: The Consistency-based Approach to Automated Diagnosis of Devices. In: Brewka, G. (ed.), *Principles of Knowledge Representation*, *CSLI Publications*, Stanford, ISBN 1-57586-057-0, pp. 267-311, 1996.

[Dvorak Kuipers 1992]

Daniel Dvorak and Benjamin Kuipers, Model-based Monitoring of Dynamic Systems, in: W. Hamscher et al. (eds.), *Readings in Model-based Diagnosis*, Morgan Kaufmann Publishers, 1992.

[Falkenhainer Forbus 1991]

Brian Falkenhainer, Kenneth D. Forbus: Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence* 51(1-3): 95-143, 1991.

[Forbus 1984]

Kenneth Forbus, Qualitative Process Theory. *Artificial Intelligence*, 24(1-3), 1984.

[Guckenbiehl et al. 1999]

Guckenbiehl, T., Milde, H., Neumann, B., Struss, P., Meeting Re-use Requirements of Real-life Diagnosis Applications. Puppe, F. (ed.), XPS99: Knowledge-based Systems. *Lecture Notes in Artificial Intelligence* 1570, Springer Verlag, Berlin 1999, pp. 90-100, 1999.

[Hamscher et al. 1992]

Hamscher, W., Console, L. and de Kleer, J. (eds.): *Readings in Model-Based Diagnosis*, Morgan Kaufman Publishers, 1992.

[Heller 1995]

Ulrich Heller: *Temporale Verhaltensabstraktion am Beispiel von Regelkreisen und hydro-ökologischen Systemen*. Master thesis, Department of Computer Science, Technical University of Munich, Germany, 1995. (in German)

[Heller et al. 1995]

Ulrich Heller, Peter Struss, Francois Guerrin, Waldir Roque: A Qualitative Modeling Approach to Algal Bloom Prediction. In: *Workshop Notes of the International Workshop on "Artificial Intelligence and the Environment"* (at IJCAI-95), Montreal, Canada, 1995.

[Heller Struss 1996a]

Heller, U., Struss, P.: Transformation of Qualitative Dynamic Models - Application in Hydro-Ecology. In: *10th International Workshop on Qualitative Reasoning* (QR-96), AAAI Press, pp. 83-92, 1996. Nachdruck in diesem Band.

[Heller Struss 1996b]

Ulrich Heller, Peter Struss: Qualitative Modeling for Environmental Decision Support. 10. Symposium Informatik für den Umweltschutz, Hannover. In: *Umweltinformatik Aktuell*, Band 10, Metroplis Verlag, Marburg, Germany, pp. 358-367, 1996.

[Iacucci Struss 1998]

Iacucci, E., Struss, P.: *Raz'r User Manual, Version 1.0*, Occ'm Software GmbH, Deisenhofen, Germany, May 1998 (see http://www.occm.de).

[Inderst et al. 1995]

Inderst, R., Struss, P. and Dressler, O.: Automatische Testgenerierung für Weichenschaltungen auf der Basis qualitativer Modellierung, *Tagungsband zum Treffen der GI-Fachgruppe 1.5.2 „Diagnostik und Klassifikation“*, Goslar, 1995.

[Iwasaki 1992]

Iwasaki, Y.: Reasoning with Multiple Abstraction Models. In Faltings, B. and Struss, P. eds*.: Recent Advances in Qualitative Physics*, Cambridge, Mass., MIT Press, 1992.

[Kuipers 1986]

Benjamin Kuipers: Qualitative Simulation. In: *International Journal of Artificial Intelligen*ce 29(3), 1986.

[Kuipers 1987]

Benjamin Kuipers: Abstraction by Time-Scale in Qualitative Simulation. In*: Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87),* 1987.

[Kutscha 2000]

Alexander Kutscha: *Design and Implementation of a Component for Automated Abstraction of Domains in Qualitative Models* (in German), Master Thesis, Technische Universität München, Department of Computer Science, February 2000.

[Malik Struss 1996]

Malik, A., Struss, P., Diagnosis of Dynamic Systems Does Not Necessarily Require Simulation. *Workshop Notes of the 10th International Workshop on Qualitative Reasoning (QR-96),* AAAI Press, 1996.

[Price 1998]

Price, C.: Function-directed Electrical Design Analysis, *AI in Engineering* 12(4), pp. 445-456, 1998.

[Reiter 1980]

Reiter, R., A logic for default reasoning. *Artificial Intelligence*, 13:81-132, 1980.

[Sachenbacher Struss 2000]

Martin Sachenbacher, and Peter Struss.: Automated Determination of Qualitative Distinctions: Theoretical Foundations and Practical Results, *Working Papers of the 14th International Workshop on Qualitative Reasoning (QR-00)*, Morelia, Mexico, 2000. Nachdruck in diesem Band.

[Sachenbacher et al. 2000a]

Sachenbacher, M., Struss, P., and Weber, R., Advances in Design and Implementation of OBD Functions for Diesel Injection based on a Qualitative Approach to Diagnosis, *SAE 2000 World Congress*, Detroit, USA, 2000 (SAE SP-1501), ISBN 0-7680-0551-5, p. 23-32

[Sachenbacher et al. 2000b]

Martin Sachenbacher, Peter Struss, Reinhard Weber,: Advances in Design and Implementation of OBD Functions for Diesel Injection Systems based on a Qualitative Approach to Diagnosis, *SAE 2000 World Congress*, Detroit, USA, 2000.

[Struss Dressler 1989]

Struss, P., Dressler, O., Physical Negation - Integrating Fault Models into the General Diagnostic Engine. *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, Morgan Kaufmann Publishers, San Mateo, CA, 1989

[Struss 1991]

Peter Struss: A Theory of Model Simplification and Abstraction for Diagnosis. In*: Working Notes of the Fifth International Workshop on Qualitative Reasoning (QR-91)*, Austin, Texas, 1991.

[Struss 1992]

Peter Struss. 1992. What's in SD? Towards a Theory of Modeling for Diagnosis, In Hamscher, W., Console, L., and de Kleer, J. eds. 1992. *Readings in Model-based Diagnosis*. Morgan Kaufmann Publishers, pp. 419-449.

[Struss 1993]

Peter Struss: On Temporal Abstraction in Qualitative Physics - A preliminary report. In*: Working Notes of the Seventh International Workshop on Qualitative Reasoning (QR-93)*, Seattle, Washington, 1993.

[Struss 1994a]

Peter Struss: Testing Physical Systems. In: *Proceedings of AAAI-94*, Seattle, USA, 1994.

[Struss 1994b]

Peter Struss: *A Theory of Testing Physical Systems Based on First Principles*. Technical Report CD-TR 94/63 of the Christian Doppler Laboratory for Expert Systems, Vienna University of Technology, 1994.

[Struss et al. 1995]

Struss, P., Malik, A., Sachenbacher, M., Qualitative Modeling is the Key. *Workshop Notes of the 6th International Workshop on Principles of Diagnosis (DX-95)*, Goslar, Germany, 1995.

[Struss 1997]

Peter Struss: Fundamentals of Model-Based Diagnosis of Dynamic Systems. *International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 23-29, pp. 480-485, 1997.

[Struss et al. 1997]

Peter Struss, Martin Sachenbacher, Fabian Dummert: Diagnosing a Dynamic System with (almost) no Observations. *Workshop Notes of the 11th International Workshop on Qualitative Reasoning, (QR-97)* Cortona, Italy, June 3-6, pp. 193-201, 1997. Nachdruck in diesem Band.

[Struss Heller 1999]

Peter Struss and Ulrich Heller: Model-based Support for Water Treatment. In: Milne, R. (ed.): Qualitative and Model Based Reasoning for Complex Systems and their Control, *Workshop KRR-4 at the 16th International Joint Conference on Artificial Intelligence*, Stockholm, 1999, pp. 84-90.

[Struss Sachenbacher 1999]

Peter Struss, Martin Sachenbacher: Significant Distinctions Only: Context-Dependent Automated Qualitative Modeling. In: *13th International Workshop on Qualitative Reasoning* (QR-99), Loch Awe, Scotland, 1999 Also in: *10th International Workshop on Principles of Diagnosis* (DX99), Loch Awe, Scotland, 1999.

[Struss Heller 2000]

Peter Struss and Ulrich Heller: An Approach to Consistency-based Process Diagnosis. In diesem Band.

[Struss et al. 2000]

Peter Struss, U. Heller, A. Malik und M. Sachenbacher, Modellbasierte Werkzeuge für Diagnose und Fehleranalyse von Fahrzeug-Subsystemen, in diesem Band.

[Williams Nayak 1996]

Brian Williams and Pandurang Nayak, A Model-based Approach to Reactive Self-Configuring Systems, in: *Workshop Notes of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, Montreal, 1996.

Teil II

Modellbasierte Generierung von Fehlerbäumen

In diesem Teil stellen die Partner STILL, ServiceXpert und Labor für Künstliche Intelligenz aus Hamburg den in INDIA verfolgten Ansatz zur Diagnose von Gabelstaplern vor. Dabei werden aus Modellen von Schaltungen Fehlerbäume automatisch abgeleitet. Fehlerbäume dienen der Diagnose von Schaltungen. In Diagnosesystemen wie DiaMon können Fehlerbäume weiter verarbeitet werden, um sie für den Einsatz in der Wartung von Gabelstaplern vorzubereiten.

Zunächst werden die Anforderungen aus Anwendersicht genannt. Im Kapitel II-2 wird die Modellierung, Analyse und automatische Erzeugung von Fehlerklassenstrukturen beschrieben. Im Kapitel II-3 wird die Integration der Fehlerklassenstrukturen in das System DiaMon und dessen Einbindung in den betrieblichen Prozeß dargestellt. Im letzten Kapitel II-4 wird der Ansatz bewertet.

1 Anforderungen an ein Diagnosesystem aus Anwendersicht

Stefan Volke – STILL GmbH Hamburg

1.1 Hintergrund

Zum Lieferumfang der STILL GmbH (kurz: STILL) als Hersteller von Flurförderzeugen zählen Gabelstapler mit Elektro-, Diesel- und Treibgasantrieb. Die Geräte werden am Standort Hamburg gefertigt und weltweit vertrieben. Vertrieb und Service werden in Deutschland über eigene Werksniederlassungen und Vertragswerkstätten durchgeführt. Im Ausland ist STILL durch Tochtergesellschaften und Vertretungen präsent. Der Service wird in Europa mit über 1100 Kundendienstwagen durchgeführt.

Mit der Einführung von Mikrocontrollern als zentralen Elementen in der Steuerungselektronik der Gabelstapler im Jahre 1995 wurde der Ruf nach einem leistungsfähigen Diagnosesystem laut. Während Fehler in den mechanischen und hydraulischen Systemen des Fahrzeugs relativ leicht identifiziert werden können, wird die Fehlersuche in der Fahrzeugelektrik durch den wachsenden Komplexitätsgrad in diesem Bereich immer schwieriger und ist ohne intelligente Hilfsmittel kaum noch möglich. Insbesondere die Verzahnung von Sensoren, Aktoren und dem Programm des Mikrocontrollers, das neben der eigentlichen Fahrzeugfunktion zusätzliche Sicherheits- und Komfortfunktionen realisiert, erschwert die Identifikation einer Fehlerursache ausgehend von der beobachteten Symptomatik ungemein.

Aus diesem Grund wurde von der Mikroelektronik Anwendungszentrum Hamburg GmbH (kurz: MAZ) basierend auf den Anforderungen der Firma STILL das PC-gestützte fehlerbaumorientierte Diagnosesystem DiaMon entwickelt. Mit diesem wird ein Servicetechniker interaktiv durch eine Diagnosesitzung geführt, wobei das System auch selbständig Daten mit dem zu diagnostizierenden Fahrzeug austauscht. Eine ausführlichere Beschreibung von DiaMon findet sich im Kapitel II-3.2.

Die Erstellung der Fehlerbäume in der derzeitig bei STILL verfügbaren Implementierung von DiaMon ist mit erheblichem zeitlichen Aufwand verbunden. Dieser entsteht nicht nur bei der eigentlichen Erzeugung der Fehlerbäume in der Entwicklungsumgebung von DiaMon, sondern auch beim Erwerb von Wissen über die Eigenschaften des zu diagnostizierenden Fahrzeugs. Aus diesem Grund bestand der Wunsch nach einer Erleichterung bzw. (teilweisen) Automatisierung der anfallenden Tätigkeiten.

Zusammen mit dem Labor für Künstliche Intelligenz der Universität Hamburg (kurz: LKI) beteiligten sich MAZ und STILL deshalb am Verbundprojekt INDIA. Die am MAZ begonnenen Arbeiten wurden im Verlauf des Projekts von der ServiceXpert GmbH (kurz: SXP) weitergeführt (s. hierzu Kapitel II-3.1).

Im Projekt fielen STILL im wesentlichen die folgenden Aufgaben zu:

- Formulierung der Anforderungen aus Anwendersicht (nachstehend beschrieben)
- Bereitstellung von Leitbeispielen als Entwicklungsgrundlage und zur Verifikation der Forschungsergebnisse (s. Kapitel II-2)
- Bewertung der Ergebnisse (s. Kapitel II-4)

1.2 Anforderungen an Diagnose

Bei der Formulierung von Anforderungen aus Anwendersicht sind durchaus unterschiedliche Aussagen möglich, je nachdem, welchen Anwenderkreis man zu Grunde legt. Es ergeben sich hieraus zwar keine widersprüchlichen Forderungen, aber es existieren verschiedene Schwerpunkte.

Unabhängig vom Anwenderkreis lassen sich jedoch die folgenden Forderungen formulieren:

1. Die Diagnose muß **korrekt** sein. Es darf nicht vorkommen, daß Fehlerursachen nicht gefunden werden oder, was noch schlimmer ist, daß falsche Ursachen gefunden werden, die zum Austausch von fehlerfreien Teilen am Fahrzeug führen, ohne daß damit der eigentliche Fehler beseitigt wurde.
2. Die Diagnose muß **vollständig** sein. Alle Ursachen von Fehlern müssen gefunden werden, damit eine Instandsetzung des Fahrzeugs innerhalb einer einzigen Diagnosesitzung möglich ist.

3. Die Diagnose muß **kostengünstig** sein. Dies erfordert im wesentlichen, daß die Ursache eines Fehlers mit möglichst wenigen Diagnoseschritten gefunden wird. Außerdem sollen während der Diagnosesitzung keine umfangreichen Einstellungen oder Montagen am Fahrzeug durchgeführt werden. Das entscheidende Kriterium ist also immer die Zeit, die für eine Diagnosesitzung aufzuwenden ist. Auf der einen Seite wird durch eine möglichst kurze Fehlersuche die Ausfallzeit des Fahrzeugs minimiert, auf der anderen Seite fallen weniger Lohnkosten an und der Servicetechniker ist schneller wieder für weitere Kunden verfügbar.

An dieser Stelle werden nun die vier Anwendergruppen behandelt, die mit der Diagnose in unmittelbaren Kontakt kommen.

1.2.1 Anforderungen aus Sicht des STILL-Kunden

Der Endkunde ist vom Standpunkt der Firma STILL als Hersteller von Gabelstaplern der wichtigste Anwender der Diagnose. „Anwender“ ist hier im übertragenen Sinne zu verstehen: Der Kunde arbeitet nicht selbst mit dem Diagnosesystem, ist aber als Betreiber von Fahrzeugen unmittelbar von den Auswirkungen einer guten oder auch schlechten Diagnose betroffen. Eine Ausnahme stellen hier einige Großkunden dar, die eine eigene Gabelstapler-Reparaturwerkstatt betreiben und dadurch im doppelten Sinn von einer guten Diagnose abhängig sind, nämlich sowohl als Betreiber wie auch als Instandsetzer des Fahrzeugs.

Der Kunde als Betreiber von Gabelstaplern hat das Ziel, Transportaufträge zu erfüllen. Um dieses Ziel termingerecht erreichen zu können, sind die Standzeiten für Fahrzeuge im Fehlerfall möglichst kurz zu halten. Dies betrifft im besonderen Fahrzeuge, die wegen spezifischer Ausstattungsmerkmale nicht ohne weiteres durch andere Geräte aus dem Fuhrpark ersetzt werden können, z.B. Fahrzeuge mit Druckklammern zum Transport von Papierrollen. Kurze Reparaturzeiten verringern für den Kunden auch die Kosten für den Arbeitslohn des STILL-Servicetechnikers.

Eminent wichtig ist auch die Zuverlässigkeit der Diagnose. Nach erfolgreicher Reparatur soll die Zeit bis zum nächsten Ausfall eines Fahrzeuges natürlich so groß wie möglich sein. Idealerweise sollten außer den regelmäßigen Wartungen überhaupt keine Standzeiten zu verzeichnen sein. Es ist daher von Vorteil, wenn während einer Diagnosesitzung Ergebnisse der auf dem Fahrzeug zusätzlich vorhandenen Online-Diagnose ausgewertet werden können, die Hinweise auf in Kürze zu erwartende Ausfälle liefern. In diesem Fall könnten präventive Maßnahmen ergriffen werden.

Dem Kunden ist also im wesentlichen daran gelegen, daß die Standzeiten seiner Gabelstapler und somit die Ausfallkosten minimiert werden. Die Art und Weise, wie dies erreicht wird, das heißt mit welchen Hilfsmitteln der Servicetechniker die Reparatur durchführt, ist dabei nur in den seltensten Fällen von Interesse. Hinzu kommt, daß mit dem Verkaufsargument „fortschrittliche Diagnose“ nur sehr beschränkt geworben werden kann. Während z.B. der Hinweis auf große Wartungsintervalle positiv aufgenommen wird, kann die Hervorhebung einer besonders leistungsfähigen Diagnose unter Umständen den Eindruck erwecken, daß diese auch notwendig sei, weil die Fahrzeuge unzuverlässig seien.

1.2.2 Anforderungen aus Sicht des Servicetechnikers

Für den Servicetechniker ist eine unkomplizierte Bedienung des Diagnosesystems während des Einsatzes beim Kunden von großer Wichtigkeit. Insbesondere bei rechnergestützten Systemen muß die Bedienung so einfach und unmißverständlich sein, daß es zu diesem Zweck keines PC-Spezialisten bedarf.

Während einer Diagnosesitzung darf der Servicetechniker vom System nicht in die Irre geleitet werden, das heißt, der Fehler ist zielgerichtet und in möglichst wenigen Schritten zu lokalisieren. Dauert die Fehlersuche zu lange oder werden Einstellungs- und Montageschritte erforderlich, die gegenüber der konventionellen Fehlersuche „mit dem gesunden Menschenverstand“ keine Erleichterung bringen, wird die Akzeptanz des Systems beim Bediener sinken. Schlimmstenfalls wird das System nicht verwendet.

Erleichtert wird der Einsatz des Servicetechnikers vor Ort auch durch die Möglichkeit, vom Diagnosesystem aus auf andere serviceunterstützende Programme zuzugreifen. Als Beispiel seien hier elektronische Ersatzteilkataloge oder Servicedokumente (Werkstatthandbuch) genannt.

1.2.3 Anforderungen aus Sicht der zentralen Serviceabteilung

Die zentrale Serviceabteilung ist bei STILL zuständig für die Betreuung der Servicetechniker und damit auch für die Einführung und Pflege eines Diagnosesystems, das von diesen im Rahmen ihrer Reparatureinsätze bedient wird.

Ein Diagnosesystem soll den Servicetechniker vor Ort schnell und sicher die Ursache von Fehlverhalten erkennen lassen. Gleichzeitig ist der zeitliche Aufwand für die Erstellung der Fehlersuchlogik in der Serviceabteilung möglichst klein zu halten. Dies bedeutet, daß die hierzu erforderlichen Informationen bereits in einer Form bereitgestellt werden, die eine unkomplizierte Bearbeitung ermöglicht.

Ein wesentliches Kriterium für die Güte eines Diagnosesystems ist die Beherrschung von Produktvarianten, von denen im Falle der Gabelstapler von STILL eine größere Anzahl innerhalb einer Baureihe existiert. Idealerweise wird die Fehlersuchlogik für eine Baureihe nur einmal erstellt und deckt auch die Varianten ab. Die umfassende Bearbeitung jeder neu auftretenden Variante inklusive aller gleichbleibenden Anteile verbietet sich wegen des damit verbundenen Aufwands.

Mit der gleichen Begründung ist auch die Forderung nach einer einfachen Pflege der Datenbestände eines Diagnosesystems zu verstehen. Weiterentwicklungen in einer Produktreihe dürfen nicht dazu führen, daß die bisherigen Daten unbrauchbar werden. Neue Funktionen müssen vielmehr in die vorhandene Fehlersuchlogik integriert werden können. Auch die Verteilung aktualisierter Datenbestände an die Servicetechniker und deren Installation auf den Zielrechnern muß einfach zu handhaben sein.

1.2.4 Anforderungen aus Sicht der Entwicklungsabteilung

In der Entwicklungsabteilung stehen alle Informationen über den Aufbau und die Funktionen eines Gabelstaplers zur Verfügung, die im Vorfeld bei der Konzeption und während des eigentlichen Entwicklungsprozesses anfallen.

Bezüglich der Diagnose liegt der Schwerpunkt hier eindeutig im Bereich der Online-Diagnose, um Fehler, die im laufenden Betrieb des Fahrzeugs auftreten, erkennen und ggf. sofort Maßnahmen ergreifen zu können. Als Beispiel seien hier Fehler im Umfeld des Fahrantriebs genannt, die ohne Gegenmaßnahmen zu unkontrollierten Bewegungen des Fahrzeugs führen könnten. Im Bereich solch sicherheitskritischer Baugruppen ist eine ständige Überwachung im laufenden Betrieb erforderlich. Im Fehlerfall wird das Fahrzeug auf jeden Fall in einen sicheren Zustand überführt. Beim betrachteten Beispiel des Fahrantriebs kann dies je nach Art des Fehlers z.B. bedeuten, daß das Fahrzeug nur noch mit reduzierter Geschwindigkeit oder gar nicht mehr bewegt werden kann.

Zu diesem Zweck werden sowohl in der Hardware der Fahrzeugsteuerung als auch in der Software des zentralen Mikrocontrollers im Rahmen des Entwicklungsprozesses Funktionen realisiert, die folgendes ermöglichen:

- sicheres Erkennen von Fehlerzuständen (insbesondere bei sicherheitskritischen Funktionen)
- sofortige Reaktion im Fehlerfall, Überführung des Fahrzeugs in einen sicheren Zustand
- Ausgabe entsprechender Hinweismeldungen an den Fahrer
- Speichern von Informationen über den eingetretenen Fehler für eine sich anschließende Post-Mortem-Diagnose

Zusätzlich werden Testroutinen implementiert, die im wesentlichen in der Entwicklungsphase zur Inbetriebnahme und zum Test von Baugruppen und Fahrzeugfunktionen dienen. Zum Teil können diese aber auch innerhalb einer Post-Mortem-Diagnose Verwendung finden, da auf diese Art Zustände hergestellt werden können, die im laufenden Betrieb des Fahrzeugs nicht möglich sind, aber zur Identifikation von Fehlerursachen wertvolle Hinweise liefern können.

Es ist jedoch anzumerken, daß die angesprochenen Überwachungsfunktionen, Testroutinen und gespeicherten Fehlerinformationen nicht unbedingt auf die Anwendung in einer Post-Mortem-Diagnose zugeschnitten sind. Das Hauptaugenmerk wird während der Entwicklung auf eine zügige Realisierung und auf einen sicheren Betrieb des Gabelstaplers gelegt. Die Fähigkeit, Informationen für eine Post-Mortem-Diagnose bereitzustellen, entsteht quasi als Nebenprodukt mit den damit verbundenen Schwächen.

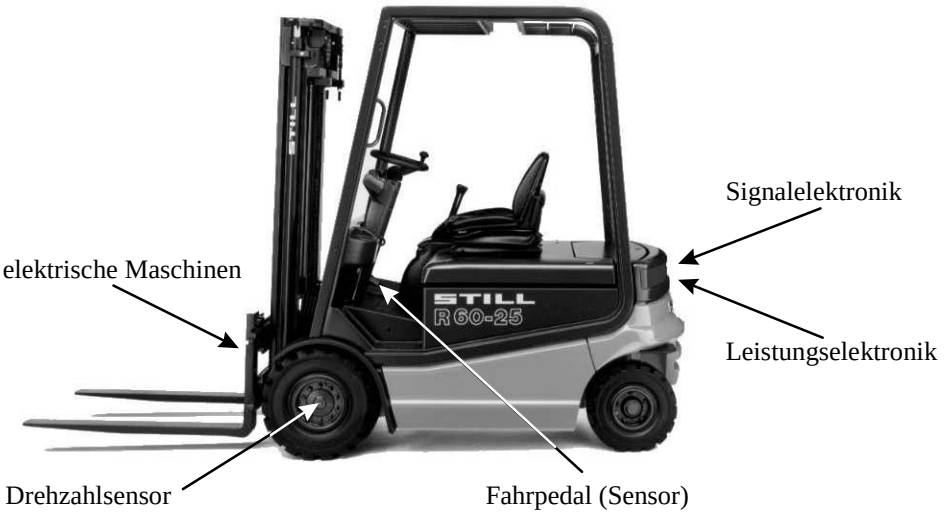


Abbildung 1-1: räumliche Verteilung der Komponenten für die Funktion "Fahren"

Das Fahrzeug wird baugruppen- und softwareübergreifend als eine Ansammlung von Funktionen (z.B. Fahren, Heben, Lenken etc.) angesehen. Für die Realisierung von komplexen Funktionen sind neben den steuernden Programmteilen des Mikrocontroller-Programms Hardwarekomponenten beteiligt, die über mehrere Baugruppen des Fahrzeugs auch räumlich verteilt sind (s. Abbildung 1-1 als Beispiel für die Funktion „Fahren"). Im Falle eines Fehlers wird festgestellt, welche **Funktion** (oder Teilfunktion) gestört ist. Dieses ist aber aus den angegebenen Gründen nicht zwangsweise identisch mit dem Wunsch, die defekte **Baugruppe** identifizieren zu können. Hierzu ein Beispiel:

In der Abbildung 1-2 ist schematisch die Schaltung des Fahrgebers dargestellt. Beim Fahrgeber selbst handelt es sich um ein Potentiometer, das die Stellung des Fahrpedals als ein elektrisch auswertbares Signal abbildet. Versorgungsspannung und Bezugsmasse werden vom Steuergerät bereitgestellt, das sich im allgemeinen im Heck des Fahrzeugs befindet. Über Leitungen und Steckverbinder werden diese zum Fahrgeber geführt. Umgekehrt wird der Mittelabgriff des Potentiometers (Nutzsignal des Fahrgebers) ebenfalls über Leitungen und Stecker zurück zum Steuergerät geführt und dort in der Signalelektronik aufbereitet. Das aufbereitete Signal wird schließlich auf den Analog-Digital-Umsetzer des Mikrocontrollers geschaltet und kann danach auch programmtechnisch verwertet werden, um z.B. aus der Stellung des Fahrgebers die gewünschte Geschwindigkeit des Fahrzeugs abzuleiten.

Wird nun während einer Diagnosesitzung festgestellt, daß die im Steuergerät gemessenen Spannungswerte des Fahrgeber-Nutzsignals nicht in den erwarteten Bereichen liegen, können diesem Symptom diverse mögliche Ursachen zu Grunde liegen. Hier einige Beispiele:

- Versorgungsspannung zu gering
- Abriß der Masseverbindung
- Unterbrechung von Leitungen im Signalpfad (Leitungen 4 - 6)
- defektes Potentiometer
- Fehler in der Signalanpassung im Steuergerät
- defekter Analog-Digital-Umsetzer
- etc.

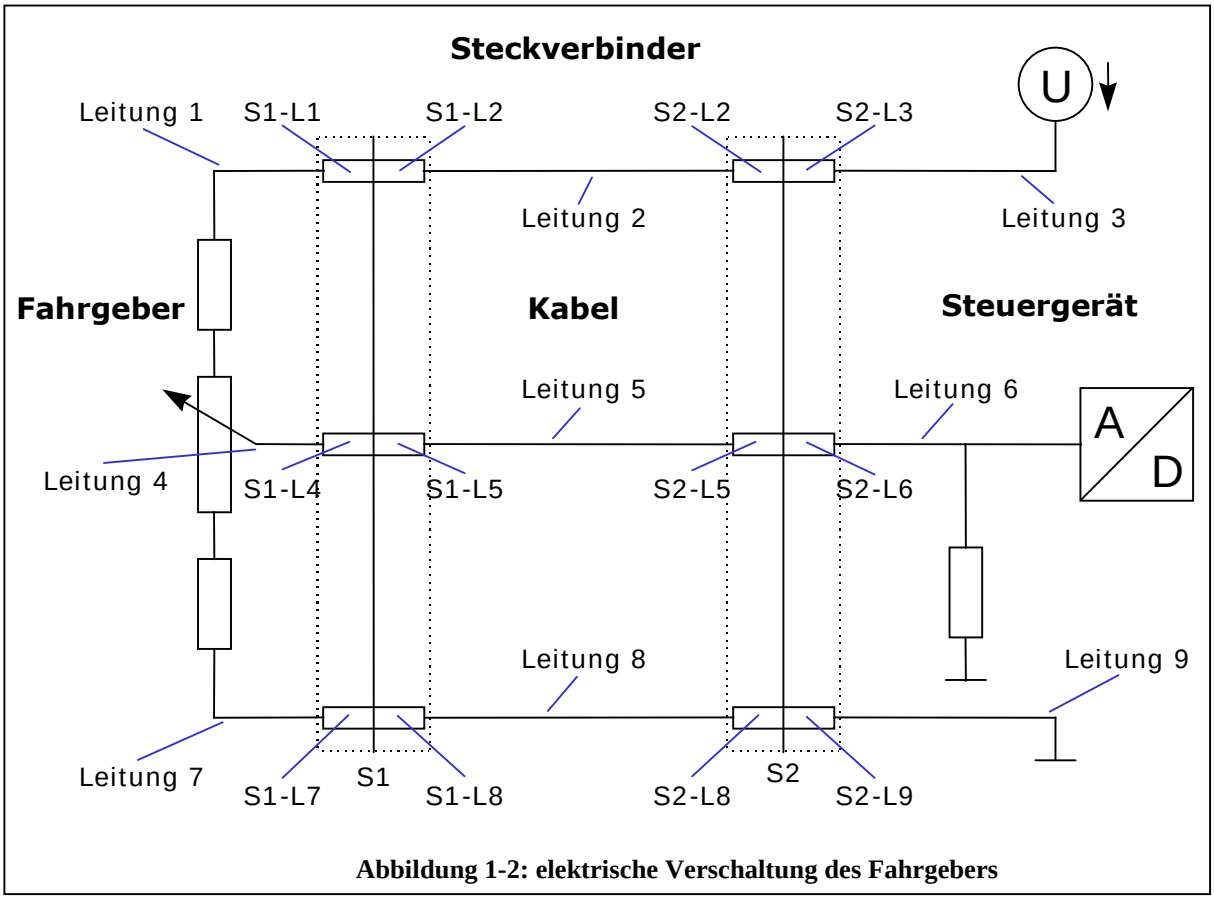


Abbildung 1-2: elektrische Verschaltung des Fahrgebers

In der Online-Diagnose des Fahrzeugs wird jedoch nur vermerkt, welcher Art die aufgetretene Störung ist, ob z.B. der Zahlenwert, der hinter dem Analog-Digital-Umsetzer dem Mikrocontroller zur Verfügung steht, zu niedrig oder zu hoch ist. Während des Betriebs ist diese Information ausreichend für eine sichere Funktion. Der genaue Ort der Störung kann allerdings hiermit nicht festgestellt werden.

Aus dem bisher Gesagten entstehen die folgenden Forderungen seitens der Entwicklungsabteilung:

1. Mit Hilfe der Diagnose sollen defekte Baugruppen erkannt werden. Hierbei kann es sich sowohl um komplette Funktionseinheiten, einzelne Leiterplatten bis hin zu peripheren Komponenten (z.B. Sensoren, Bedienelemente) handeln. Entscheidend für die Diagnosetiefe ist dabei die Tauschbarkeit eines Einzelteils. Als Maß sind dabei die Fähigkeiten der Servicetechniker zu Grunde zu legen, fehlerhafte Teile am Fahrzeug aus- und Ersatzteile einbauen. Dabei darf weder der zeitliche Aufwand für den Austausch der Teile zu groß sein noch darf die Gefahr bestehen, daß Ersatzteile bereits beim Einbau erneut beschädigt werden. Dies könnte z.B. bei elektronischen Komponenten eintreten, die empfindlich auf statische Elektrizität reagieren.
2. Um Informationen für die Post-Mortem-Diagnose zu erhalten, die an die Serviceabteilung zur Verwendung in einem Diagnosesystem weitergeleitet werden, darf nur wenig Zusatzaufwand erforderlich sein. Idealerweise können Daten verwendet werden, die im laufenden Entwicklungsprozeß ohnehin anfallen, wie Schaltpläne elektronischer Baugruppen, CAD-Daten etc.
3. Ergebnisse aus den Diagnoseeinsätzen beim Kunden sind der Entwicklungsabteilung zur Verfügung zu stellen, um diese zur Verbesserung des Produktes verwenden zu können.

1.3 Erfahrungen mit dem vorhandenen System DiaMon

Das seit 1995 im Einsatz befindliche Diagnosesystem DiaMon erfüllt bereits wesentliche der oben genannten Forderungen. Es basiert auf einer fehlerklassenorientierten Repräsentation mit einem darauf zugeschnittenen

Schlußfolgerungs- und Dialogsteuerungsverfahren (s. Kapitel II-3). Stärken und Schwächen des Systems werden im folgenden beschrieben, wobei die Darstellung der Schwächen ausführlicher ist, da sich aus diesen letztlich die Zielvorgaben für die Arbeiten im Projekt INDIA ergeben.

1.3.1 Stärken des Systems

Das gesamte Diagnosewissen steht an zentraler Stelle in der Serviceabteilung zur Verfügung. Der Bearbeiter versucht bei der Erstellung der Fehlerbäume, die möglichen Fehler und ihre Zusammenhänge, die zur Bestimmung von Fehlern notwendigen Überprüfungen sowie die Reparaturmaßnahmen so weit wie möglich im Vorfeld zu erkennen. Bei der Modellierung fließen funktionelle, strukturelle, quantitative wie auch qualitative Betrachtungen mit ein. Die geschickte Kombination dieser Kriterien führt zu einem effektiven und effizienten Diagnoseablauf. Alle Objekte, durch die dieser Ablauf beschrieben wird, werden in einer Datenbank abgelegt. Für die unterschiedlichen Baureihen von Fahrzeugen existieren jeweils eigene Datenbanken.

Es ist möglich, sowohl Varianten von Baureihen als auch Unterschiede innerhalb einer Variante, die im Rahmen der Produktpflege nach Markteinführung entstehen, korrekt zu behandeln. Komponenten, die in verschiedenen Baureihen in gleicher oder ähnlicher Form eingesetzt werden, können aus einer bereits vorhandenen Datenbank kopiert werden. Dies vermindert den Erstellungsaufwand beträchtlich.

Während der eigentlichen Diagnosesitzung werden dem Servicetechniker genaue Anweisungen erteilt, welche Vorbereitungen und Einstellungen er durchzuführen hat, wo ggf. Meßgeräte anzuschließen sind und wie die Reparatur zu erfolgen hat. Er kann zusätzlich auf unterstützende Hilfetexte, Zeichnungen und Servicedokumente zugreifen, die bei der Erstellung der Fehlerbäume in die Datenbank aufgenommen wurden. Durch die exakte Benutzerführung kann es kaum zu Fehlbedienungen kommen.

1.3.2 Schwächen des Systems

Den genannten Vorteilen stehen auch einige Nachteile gegenüber. Diese sind nicht so gravierend, daß die Akzeptanz des Diagnosesystems sowohl bei der Erstellung der Fehlerbäume als auch beim Einsatz des Servicetechnikers gefährdet wäre. Bestimmte Eigenschaften des Systems behindern jedoch die Ausnutzung der vollen Leistungsfähigkeit. Diese beziehen sich zum einen auf das Softwarepaket DiaMon selbst, zum anderen auf den Informationsfluß und die Informationsverfügbarkeit innerhalb des Unternehmens:

1. Manuelle Erstellung der Fehlerbäume in DiaMon: Die Modellierung von Hand ist anfällig für alle Arten von Fehlern, die bei dieser Bearbeitungsmethode zu verzeichnen sind, z.B. Schreibfehler, Fehler beim Kopieren und Umbenennen von Objekten, versehentliches Löschen von Objekten, Verwendung falscher logischer Verknüpfungen etc. Zudem ist die Bearbeitung sehr zeitaufwendig, da die Anzahl der im Fehlerbaum enthaltenen Objekte sehr groß wird (zu den einzelnen Objektarten s. Kapitel II-3.2.1). Ein typischer Fehlerbaum enthält zur Zeit knapp 3000 Objekte (Fehler, Meßgrößen, Beschreibung der Messungen, Benutzeranweisungen etc.). Die manuell modellierten Fehlerbäume sind ggf. nicht vollständig: Da keine zwingende Systematik der Vorgehensweise während der Modellierung existiert, verwenden die Entwickler unterschiedlichste Kriterien bei der Strukturierung der Fehlersuche. Ein möglicher Weg ist, von der Struktur der zu diagnostizierenden Komponente auszugehen und unter der Annahme von defekten Teilen (also Fehlern) die hieraus folgende Symptomatik zu ermitteln. Der umgekehrte Weg ist aber ebenso denkbar. In diesem Fall legt man typische Symptome zugrunde und ermittelt die hierfür verantwortlichen Fehler. Der zweite Weg entspricht in seiner Strategie eher der späteren Abarbeitungsreihenfolge des Fehlerbaums im Diagnosesystem. Oft ist es jedoch einfacher, den ersten Weg zu beschreiten. In vielen Fällen ist die Bestimmung der Wirkung eines angenommenen Fehlers leichter als die der Ursache eines angenommenen Symptoms. In der Praxis werden meist beide Strategien nebeneinander verwendet. Aus diesem Grund ist durchaus möglich, daß nicht alle denkbaren Fehler erfaßt werden. Weiterhin kann es vorkommen, daß unbemerkt widersprüchliche Verknüpfungen in der Suchlogik formuliert werden.
2. Berücksichtigung von Meßkosten: Für eine kostengünstige Diagnose ist es wünschenswert, wenn die Fehlersuche so weit wie möglich auf Daten basiert, die mit wenig Aufwand zu gewinnen sind. Dies sind zum Beispiel im Fahrzeug gespeicherte Fehlerinformationen oder im weiteren Sinn alle Daten, die direkt aus der Fahrzeugsteuerung gelesen werden können. Aufwendige Messungen, die ggf. sogar Eingriffe in die Struktur des Fahrzeugs erfordern (z.B. Abklemmen von Leitungen, Einschleifen von Meßadaptern etc.), sind so weit wie möglich zu vermeiden. Bei der manuellen Fehlerbaumerstellung liegt die Berücksichtigung dieser Forderung im Ermessen des Bearbeiters. Nicht immer wird dabei die optimale Reihenfolge der erforderlichen Messungen gewählt, sondern diejenige, die von der Reihenfolge der Suchschritte am schnellsten zum Ziel

führt. Dabei ist es durchaus vorstellbar, daß für das Bestimmen eines Fehlers das Auslesen von einer größeren Anzahl von Daten aus dem Fahrzeug und deren Bewertung innerhalb der Fehlersuchlogik weitaus schneller ist als das manuelle Vorbereiten des Fahrzeugs für eine spezielle Messung, mit deren Hilfe der Fehler sofort bestimmt werden kann. Voraussetzung ist hierbei natürlich, daß der Fehler mit beiden Verfahren identifiziert werden kann.

3. Ein weiteres Problem besteht in der Tatsache, daß an der Erstellung der Fehlerbäume nicht nur die Serviceabteilung, sondern indirekt auch die Entwicklungsabteilung beteiligt ist. Das vollständige Wissen über das Fahrzeug und dessen Eigenschaften ist nur in der letzteren verfügbar. Die Schwierigkeit besteht für die Serviceabteilung nun darin, aus dieser Vielfalt von möglichen Informationen die für die Diagnose relevanten auszufiltern. Hierzu ist ein erheblicher zeitlicher Aufwand zur Abstimmung zwischen den Abteilungen erforderlich, um das zur Erstellung der Fehlerbäume erforderliche Wissen über Struktur und Funktionen des Fahrzeugs in die Serviceabteilung zu transferieren. Auch tritt hier der schon im Abschnitt 1.2.4 genannte Konflikt zwischen der Diagnose von Funktionen und Baugruppen zu Tage: In der Entwicklung werden vorrangig Daten im Fahrzeug verwendet, die für den sicheren Betrieb und für Tests im Rahmen der Inbetriebnahme, also für Funktionen des Fahrzeug erforderlich sind. Für die Serviceabteilung sind aber Daten von Interesse, die auf defekte Komponenten hinweisen. Natürlich werden im Fahrzeug auch Daten speziell für Diagnosezwecke bereitgestellt. Um welche Daten es sich dabei handelt, hat allerdings häufig erst die Erfahrung mit den Anforderungen aus der Serviceabteilung gezeigt. Diese werden immer dann gestellt, wenn mit den bereits zur Verfügung gestellten Daten Fehlerursachen nicht eindeutig oder eventuell noch gar nicht bestimmt werden können. Dieser ständige Abgleich zwischen den verantwortlichen Abteilungen führt zu zeitlichem Mehraufwand auf beiden Seiten.

1.3.3 Ziele für das Projekt INDIA

Die im vorigen Abschnitt genannten Schwächen des verwendeten Diagnosesystems führten seitens STILL zur Formulierung der Aufgaben, die im Rahmen des Projektes INDIA zu bearbeiten waren. Dabei sollten die Untersuchungen schwerpunktmäßig an Komponenten der elektrischen Systeme eines Fahrzeugs erfolgen, da zum einen hier auch bisher der Hauptanwendungsbereich der Diagnose bei STILL lag und zum anderen durch die Anbindung an den Mikrocontroller des Fahrzeugs eine einfache Möglichkeit bestand, eine relativ große Anzahl von Informationen wie Zustandsdaten und Fehlerinformationen ohne manuellen Aufwand zu gewinnen. Diese Ziele sind im folgenden aufgeführt:

- Die Entwicklungen im Projekt INDIA sollten dazu führen, den Erstellungs- und Pflegeaufwand des Diagnosewissens zu reduzieren. Nutzeffekte waren vor allem durch den Einsatz modellbasierter Techniken zur Erstellung von Fehlerbäumen zu erwarten. Ziel war, durch die Verwendung qualitativer Modelle elektronischer Schaltungen und vergleichbarer Komponenten des Fahrzeugs in die Lage versetzt zu werden, eine vollständige und korrekte Beschreibung aller relevanten Fehlerzustände erzeugen zu können.
- Weiterhin sollte untersucht werden, in wie weit im Unternehmen bereits vorhandene Daten (CAD-Daten, Dokumentationen, Bauteilbibliotheken etc.) für ein Diagnosesystem verwendet werden können. Auch die Anbindung an andere den Servicetechniker unterstützende Softwarekomponenten (Servicedokumentation, Ersatzteilkataloge etc.) war Gegenstand der Betrachtungen.
- Es sollte eine prototypische Realisierung erfolgen mit der Möglichkeit, die aus der modellbasierten Analyse gewonnenen Daten in das vorhandene DiaMon-System zu importieren, aus diesen Fehlerbäume zu erzeugen und diese ggf. nachträglich zu bearbeiten.

2 The Modelling, Analyzing, and Diagnozing System (MAD)

Heiko Milde, Lothar Hotz, Jörg Kahl, Bernd Neumann und Stephanie Wessel –

Universität Hamburg, Labor für Künstliche Intelligenz

In industrial practice, computer-based approaches are applied to support service technicians in their diagnosis task. Using fault trees is the prevailing method due to its simplicity. For fault identification, fault-tree-based computer diagnosis systems instruct service technicians to perform sequences of tests and measurements. Unfortunately, determining an optimal fault tree considering costs of tests and measurements is a difficult and time consuming task if it has to be done manually. Our approach realized in a system called MAD (Modeling, Analyzing and Diagnosing) allows automated fault tree generation based on qualitative models of the technical systems. MAD's fault trees can be edited manually.

This chapter describes a second prototype of MAD and it is structured as follows. In the introduction, in Section 2.1, we present the accelerator pedal circuit which is a simple example showing basic characteristics of diagnosis in our application. In Section 2.2, we enumerate development goals for MAD's modeling module. Section 2.3 describes our new method for qualitative electrical circuit analysis. Circuit modeling and behavior prediction using MAD is described in Section 2.4 and 2.5. In Section 2.6, we discuss basic characteristics of the modeling module of MAD. In Section 2.7 we describe the diagnosis part of MAD, i.e. the automated generation of fault trees.

2.1 Introduction

More than 100.000 forklifts made by the German company STILL GmbH Hamburg are in daily use all over Europe. In order to reduce forklift downtimes, approximately 1100 STILL service workshop trucks utilize fault tree-based computer diagnosis systems for workshop diagnosis. In case of a malfunction, service technicians attach a computer diagnosis system to a forklift. Then the diagnosis system performs automated testing and it also instructs service technicians to carry out manual tests. Test sequences are specified by fault trees which are diagnostic decision trees allowing fault identification.

Due to the complexity of the electrical circuits employed in forklifts, fault trees may consist of more than 5000 nodes. When forklift model ranges are modified or new model ranges are released, the central service division manually generates or adapts fault trees. Dealing with this task, service engineers apply detailed expert knowledge concerning faults and their effects. Adapting fault trees to new model ranges can take a service engineer several months. As described in Chapter II-1, this practice is costly and quality management is difficult. Furthermore, fault trees are not optimized and fault identification cost is unnecessarily high. Hence, there is a need for computer methods to systematically support the design, modification, and optimization of fault trees. The introduction of new diagnosis techniques, however, raises challenges.

- As described in Chapter II-1, a new diagnosis system has to be integrated into existing STILL workflow such that the current service and diagnosis processes do not have to be restructured. A new diagnosis tool must be easily accepted by service technicians and service engineers. Hence, doing fault identification with a new diagnosis system has to be intuitive for service technicians which are familiar with the current system. The STILL central service division must not have any difficulties getting used to a new system. Long terms of training for service technicians and service engineers are not acceptable

In Chapter II-1, the STILL GmbH defines further requirements for the introduction of a new computer diagnosis system which can be summarized as follows.

- Cost of diagnosis equipment generation, modification, and maintenance has to be reduced. In particular, cost-efficient dealing with product variants is essential. That is, generating diagnosis equipment for new forklift model ranges as well as adapting diagnosis equipment to forklift updates has to be carried out cost-efficient.
- The current diagnostic performance should be exceeded. As a minimal requirement, a new diagnosis system has to be able to identify all faults handled by the current system. Incorrect fault identifications cannot be accepted.
- Average fault identification cost has to be reduced.

The Laboratory for Artificial Intelligence of the University of Hamburg (LKI) develops concepts of innovative computer diagnosis techniques for the STILL application scenario. Facing the above-described requirements for the introduction of a new diagnosis techniques, we consider innovative model-based techniques to be advantageous because they provide a systematic way for design, modification, and optimization of diagnosis equipment. In our application, consistency-based techniques, such as GDE for instance [de Kleer and Williams 1986], are not helpful because they would comprise a complete reorganization of the current service and diagnosis process. In particular, service technicians spread all over Europe would have to get used to the new system in expensive preparatory training. We figured out that, for STILL, completely replacing fault trees is not an immediate option. Hence, in principle, automatically generating fault trees from device models is a promising strategy.

In our application, nodes of fault trees represent fault sets. Edges are labeled by the tests (involving measurements, observations, display values and error codes) which must be carried out to verify the corresponding child node. Although the basic concepts of model-based fault tree generation are already described in [Cascio et al. 1999] and [Faure et al. 1999], for the reader's convenience, we briefly outline the main ideas of the approach in the following.

- The first step to model-based fault tree generation is to model a device. This step is supported by component libraries and a device model archive (see Figure 2-1). Design data and knowledge from the design process (knowledge concerning intended device behavior, expected faults, available measurements) are integrated into the device modeling process.
- In a second step, behavior predictions are automatically computed from the device model and stored in the so-called fault relation.
- The third step is to build fault trees from the fault relation. This step is supported by a fault tree archive and a cost model for the tests which can be performed. Fault tree generation can be performed automatically or guided by service know-how, i.e. knowledge concerning preferable fault tree topologies.

In order to realize these concepts, we prototypically implemented the MAD system (Modeling, Analyzing, and Diagnosing) which is specified in this report. In our application, mainly, electrical circuits of forklifts have to be diagnosed. One of these systems, the accelerator pedal circuit, is presented in the following. Faults handled by the current STILL computer diagnosis system are described. Furthermore, we demonstrate how fault

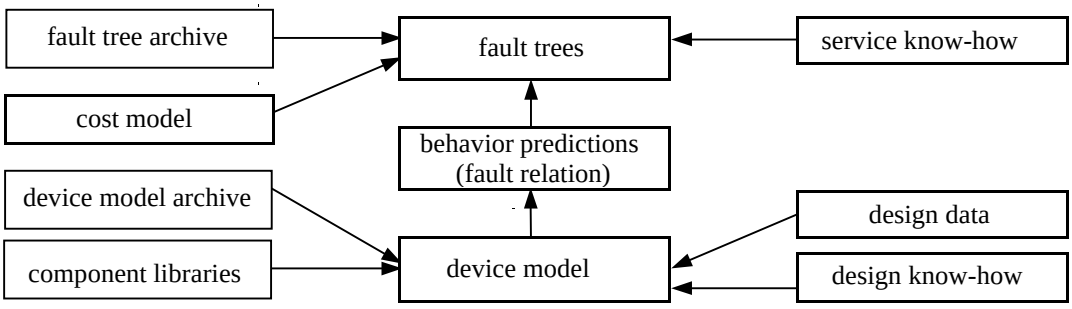


Figure 2-1: Basic concepts of model-based fault tree generation

identification is currently performed. Although considering a certain forklift circuit, faults, symptoms, and diagnosis strategies seem to be representative for a wide range of applications.

Figure 2-2 shows the wiring diagram of the accelerator pedal circuit which enables the electronic control unit (ECU) to measure the accelerator pedal position. The pedal determines a potentiometer position which controls the value of voltage UFG. In addition, the pedal is connected to a switch controlling voltage UFGS. The ECU provides a supply voltage VCC and it measures UFG and UFGS. Thus, the pedal position is supplied redundantly to secure reliability of the measurements.

Fault trees of the current STILL diagnosis system allow to identify the following faults: Wires located between the ECU and connector X16 may break due to mechanical stress. The mechanical connection between the accelerator pedal and the switch 1S16 may also break. In this case, the switch is independent from the actual pedal position and it connects either wire 1 and wire 2 or wire 2 and wire 5. The voltage VCC may be supplied incorrectly, i.e. the voltage is not between 9.8V and 10.1V. Additionally, the mechanical connection between accelerator pedal and potentiometer 1B1, may not be adjusted correctly or may even be broken. For fault identification, the current STILL diagnosis system utilizes measurements and direct component fault identifications. Additionally, certain operating modes are effectuated to simplify fault identification.

- **Measurements.** The ECU automatically measures UFG and UFGS. Beside the quantitative value of UFG, the ECU provides two error flags UFG_HIGH and UFG_LOW with values OK and notOK which are also utilized for fault identification. These flags indicate that UFG exceeds or falls below certain threshold values. The quantitative value of UFGS is also automatically mapped to values OPEN and CLOSED. In addition to these automated measurements, service technicians manually measure the voltage drop from wire 8 at connector X16 to ground.
- **Fault identification.** The breaking of some wires of the accelerator pedal circuit is directly verified by attaching these wires to a test device which measures the conductance of these cables.
- **Operating modes.** The accelerator pedal can be kicked down and released again which defines two different operating modes. In some cases, after opening connector X16, service technicians insert a short circuit (a bridge) between wire 1 and wire 2. This bridge is a substitute for switch 1S16 in the state shown in Figure 2-2. The current diagnosis system explicitly exploits these and other operating modes for diagnosis.

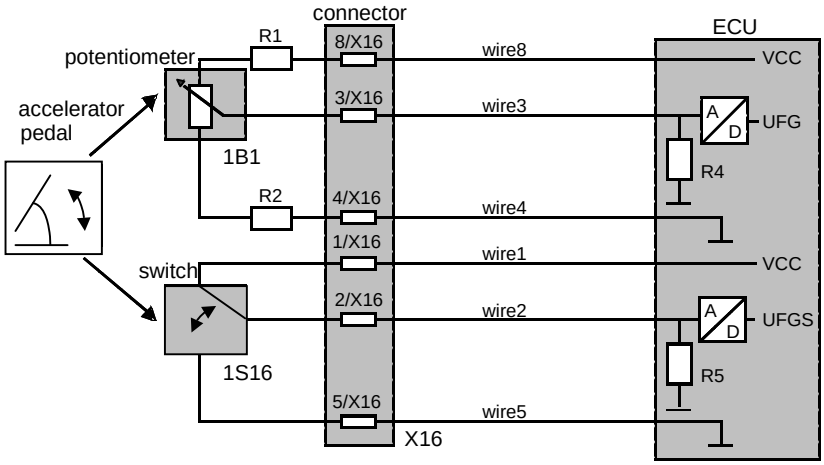


Figure 2-2: Wiring diagram of a forklift accelerator pedal circuit.

MAD allows qualitative electrical device behavior prediction, including faulty behavior. Since in our application, fault identification is performed in steady states of devices only, explicit representation of temporal dependencies is not necessary. Behavior predictions concern steady state behavior only. Component memory cannot be modeled so far. MAD has been developed to deal with typical circuits of the automotive domain. These systems usually consist of actor and sensor circuits and ECU. ECU usually read out sensor data from sensor circuits and they control actor circuits. Due to analogies between electrics, mechanics, and pneumatics, in principle, dealing with components of other technical domains is possible.

For each component, direct testing whether the component behaves correct or faulty can be modeled (direct fault identification). Component models can represent that components can be replaced assuming that their substitutes behave correctly (component replaceability). Measurements, error flags, and observations can be specified. Multiple faults, short circuits, and device operating modes can be individually defined.

MAD allows automatic generation as well as manual modification of fault trees. All faults and fault combinations considered in the device model occur in the corresponding fault tree, and tests are selected correctly to discriminate fault sets. This holds even when fault trees are modified manually. Furthermore, average diagnosis cost is minimal within the constraints imposed by a prespecified fault tree structure. MAD fault trees have been successfully integrated into existing STILL diagnosis systems.

2.2 Goals of the modeling module development

The modeling module of MAD allows model-based behavior predictions for electrical circuits of the automotive domain. These circuits usually consist of components that show a variety of different behavior types, such as analog, digital, static, dynamic, linear, nonlinear and software-controlled behavior. Considering model-based fault tree generation for these circuits, we could identify the following goals for the development of MAD's modeling module.

- **Representing current STILL fault identification is essential.** To secure acceptance among service technicians, MAD's fault trees should represent established STILL fault identification strategies. That is, fault identification guided by automatically generated fault trees should be similar to current STILL fault identification. Thus, adequately modeling circuit faults and representing current STILL fault identifying actions is essential as well as accurate symptom prediction.
- **Minimization of device modeling complexity is essential.** To secure acceptable cost of fault tree generation and maintenance the complexity of the device modeling process has to be minimized. Thus, massive utilization of model libraries and extensive reuse of existing diagnosis system components and integration of available resources such as design data, service expertise, and expert knowledge from the design process is essential.
- **Qualitative modeling is essential.** A fault model describes a certain component fault or a fault class. For fault tree generation, behavior predictions are computed for all fault models of the device model. All faults and fault classes represented by fault models occur in generated fault trees. In heterogeneous circuits, in case of malfunctions, analog parameters such as resistances may have any value different from the expected value. If fault models represent component faults by exact numbers, for systematic faulty behavior analysis, the device model would show an extensive number of fault models. Thus, behavior predictions would be hardly tractable and fault trees would be complex and hard to handle. Hence, describing faults by exact numbers would be highly inappropriate. For model-based fault tree generation, component models including fault models can be grounded on interval-based parameter descriptions with interval boundaries given by sharp quantitative parameter values (see Chapter III-3) [Faure et al. 99]. In this case, for behavior prediction, complex global optimization problems have to be solved. As another disadvantage of quantitative interval-based models, the modeling process is difficult because, frequently, specifying sharp interval boundary values is hardly possible. For example, for designers of electrical circuits, specifying an interval for the resistance of a certain electrical wire is hardly possible because manufacturers of simple wires frequently do not provide data sheets in which this parameter is specified. In STILL fault trees, faults and symptoms are described qualitatively. Thus, in principle, qualitative device modeling is adequate. As described in Section 2-3**,** qualitative models show a number of advantages. In particular, dealing with product variants is facilitated. As another point, qualitative models (as well as interval-based quantitative models) allow tractable systematic analysis of faulty device behavior because a single fault model can represent a fault class covering a wide range of faulty component behavior. As a disadvantage, due to the abstract character of qualitative models, sometimes correct and faulty device behavior may not be distinguishable. However, in our application the advantages of qualitative models prevail.
- **Steady state behavior prediction suffices.** If STILL service workshops apply fault-tree-based diagnosis equipment, only steady state diagnosis is performed. Therefore, only steady state behavior of physical components has to be represented in component models. In particular, an explicit representation of temporal dependencies is not necessary.
- **Integration of expert knowledge is essential.** Diagnosis models only need to represent components which can be replaced in case of malfunctions. Thus, modeling elementary components is sometimes unnecessary and, in principle, diagnosis models can show a high degree of abstraction. Abstract and simple circuit models lead to uncomplex computations of behavior predictions even if circuits consists of a large number of basic components. Thus, modeling circuits on a high degree of abstraction is essential. To assure accurate abstract device modeling, expert knowledge concerning intended device behavior and component replaceability as well as know-how related to negligible physical effects should guide the modeling process. This reflects the insight that the design of technical systems and of appropriate innovative diagnosis systems is inseparable.
- **Dealing with a wide range of different fault and symptom classes is essential.** In heterogeneous electrical circuits, faults may slightly modify component parameter values or may even change device structures. Hence, a wide range of different symptoms, such as slight deviations of parameter values and total loss of functionality may occur.
- **Faulty behavior predictions have to be avoided.** As a disadvantage of qualitative approaches, frequently, incorrect behavior predictions (spurious solutions) [Struss 1990] occur. If a fault tree is based on spurious behavior predictions, fault identification is not reliable. Hence, avoiding spurious behavior predictions is essential.

Although these requirements arise from our specific task in the forklift application scenario, they seem to be relevant for a variety of different diagnostic tasks such as failure mode and effects analysis (FMEA) for instance.

Qualitative reasoning about fault effects in electrical circuits has reached a level of achievement which allows it to be used for real world industrial applications. For instance, the FLAME system [Pugh et al. 1996] performs FMEA as well as sneak circuit analysis in the automotive domain. The connectivity method [Struss et al. 1995] is employed for automated FMEA and diagnosis guidelines generation for mechatronic car subsystems. The qualitative SPS method [Mauss 1998] is the basis for diagnostic fault tree generation. These approaches are promising but they do not fulfill all of the requirements enumerated above. In particular, these methods cannot deal with slight parameter deviations because their qualitative parameter representations are too simple. That is, in principle, qualitative values describe parameters in terms such as positive, zero, and negative and only actual values are represented. Parameter deviations from reference values are not explicitly described. Furthermore, some of these methods produce spurious behavior predictions. Since reasoning about parameter deviations is fundamental, in [Milde et al. 1997] we introduced the SDSP method in order to demonstrate that, in principle, qualitative reasoning about deviations in electrical circuits is possible. In Section 2.3, we present QNA, MAD's method for qualitative electrical circuit analysis.

2.3 QNA: Qualitative network analysis

Dealing with electrical circuits of forklifts, a major focus of our work is model-based prediction of correct and faulty device behavior. According to goals enumerated in the previous section, we developed a new method for Qualitative electrical Network Analysis (QNA) which is the subject of this section. QNA's qualitative calculus allows reasoning about actual parameter values and deviations from reference values. The calculus is specifically designed to avoid spurious solutions. Main ideas of this part of the report are published in [Milde et al. 1999]. As known from electrical engineering, QNA represents electrical circuits by equivalent networks. These networks consist of standard component models which show no internal structure but they show well-defined and idealized behavior. Controlled versions of standard component models are provided.

2.3.1 Standard components

MAD only provides two different types of standard component models, i.e. passive and active models showing passive and active behavior modes, respectively. Passive behavior modes are "consumer", "insulator", and "conductor". Active component models qualitatively represent idealized voltage sources providing different voltage levels. The behavior of idealized voltage sources is well-known from electrical engineering. Consumers are passive and their current/voltage characteristic is monotonous, i.e. they show positive resistances. Idealized conductors do not allow any voltage drop. Thus, they do not show any resistance at all. Idealized barriers do not allow any current, that is, their resistance is infinite. Standard components can be connected in combinations of series, parallel, star and delta groupings. This simple internal representation of electrical circuits is sufficient for the following reasons.

- A small number of qualitative standard components suffices, because, often, different physical components show similar electrical behavior, i.e. their current/voltage characteristics differ only slightly. Qualitative versions of these current/voltage characteristics are frequently identical.
- QNA's standard components are deliberately selected so that important behavior classes of the application domain can be represented adequately.
- An explicit representation of temporal dependencies is not necessary because we focus on steady state behavior analysis.

Due to analogies between electrics, mechanics and hydraulics, the internal QNA representation is, in principle, also adequate for other technical domains.

2.3.2 Qualitative representation of physical parameters and variables

Due to the standard components described in Section 2.3.1, in QNA, only three different parameter types have to be represented, i.e. current, voltage, and resistance. For each parameter type, actual values and deviations are explicitly represented because, as stated before, reasoning about these values is essential.

In [Malik et al. 1996], it is demonstrated that, in principle, reasoning about actual values and deviations is possible without considering reference values. Thus, at first sight, representing reference values does not seem to be necessary. Furthermore, if quantitative parameter values were used, reference values would be redundant *(reference value = actual value - deviation).*

Nevertheless, in QNA, reference values are explicitly represented because, if qualitative values are considered, reference values are not redundant. This can be demonstrated by considering a certain parameter, assuming that its actual, reference, and deviation value are all known as positive. Representing actual and deviation values only, the corresponding reference value can be computed as *positive - positive = (negative or zero or positive).* Hence, if qualitative values are considered, explicit representation of knowledge concerning actual values, deviations, and reference values is more precise than representing actual values and deviations only. Moreover, in Section 2.3.5, we demonstrate that QNA's threefold parameter representation is essential for accurate behavior predictions. In the following, QNA's qualitative parameter representation is described in detail.

For each parameter type, QNA's qualitative representation consists of three attributes, i.e. actual value, reference value and deviation value. For each of these attributes, QNA provides a specific set of qualitative interval-based values. Table 2-1, 2-2, and 2-3 show attributes and corresponding qualitative value sets of resistances, currents, and voltages (abbreviations in brackets). The semantics of the qualitative values should be obvious.

attributes	qualitative values
actual value (act)	zero (0), positive (pos), positive-infinite (pinf)
reference value (ref)	zero (0), positive (pos), positive-infinite (pinf)
deviation value (dev)	negative (neg), zero (0), positive (pos)

Table 2-1: Qualitative representation of resistance

attributes	qualitative values
actual value (act)	negative (neg), zero (0), positive (pos)
reference value (ref)	negative (neg), zero (0), positive (pos)
deviation value (dev)	negative (neg), zero (0), positive (pos)

Table 2-2: Qualitative representation of current

attributes	qualitative values
actual value (act)	negative-impossible, negative-maximum, negative-between, zero, positive-between, positive-maximum, positive-impossible
reference value (ref)	negative-impossible, negative-maximum, negative-between, zero, positive-between, positive-maximum, positive-impossible
deviation value (dev)	negative, zero, positive

Table 2-3: Qualitative representation of voltage

QNA's set of standard components does not include idealized current sources. Thus, in QNA's internal device models, voltages show certain limits and voltage values beyond these limits can be considered as impossible values (see Table 2-3). Due to QNA's explicit representation of voltage limits, in principle, dealing with logical circuits is possible. For instance, logical values *(true, false)* can be mapped to QNA's voltage values zero and *positive-maximum* (see Section 2.4.3). Furthermore, QNA's qualitative voltage representation allows to handle electrical devices showing more than only one source. In particular, the representation of impossible voltage values paves the way to define a qualitative version of the superposition principle well-known from electrical engineering. Dealing with logical values as well as handling multiple sources is the basis for dealing with hybrid systems consisting of both analog and digital subsystems.

2.3.3 Automated computation of qualitative values

In this section, QNA's computation of qualitative values of parameter attributes is described. In order to compute qualitative values, local propagation methods have been investigated [Struss 1990]. Since detailed

studies proved that local propagation in electrical networks is not successful, we follow [Mauss et al. 1996]. That is, networks are transformed into trees which explicitly represent the network structure. Exploiting these structure trees, qualitative device behavior can in fact be computed by local propagation.

QNA's structure trees explicitly represent four different elementary network topologies, i.e. series, parallel, star and delta groupings. Structure trees show two different types of nodes, i.e. resistance nodes and equation nodes. Resistance nodes represent (compensation) resistances and the corresponding currents and voltages. Equation nodes represent behavior of elementary network topologies. That is, these nodes represent equations that hold between parameters of adjacent resistance nodes (see Figure 2-3). These equations, in principle, allow local propagation. The following simple example outlines the concepts of network transformation and local propagation in structure trees.

Figure 2-3 shows a subnetwork consisting of two resistances *R1* and *R2*. The network is transformed into a structure tree which explicitly represents that *R1* and *R2* are grouped in parallel. That is, both resistances, their compensation resistance *Rp*, and the corresponding currents and voltages are represented in resistance nodes. In the equation node, there are physical dependencies that hold in this particular parallel grouping. In Figure 2-3, we do not present all parallel grouping equations utilized by QNA.

The physical dependencies represented in the equation node in Figure 2-3 allow local propagation in the structure tree. First, the compensation resistance *Rp* is determined by bottom-up propagation. That is, given *R1* and *R2*, *Rp* can be computed by applying the equation *Rp = (R1 * R2) / (R1 + R2).* Second, current and voltage values are determined by top-down propagation. For instance, given Ip, I1 can be computed by applying the well-known current divider rule *I1 = R2 / (R1 + R2) * Ip*. At first sight, it might be surprising that, in Figure 2-3, *I1* can also be computed by applying *I1 = Up / R1*. In Section 2.3.6, we outline QNA's alternative computations of qualitative attribute values.

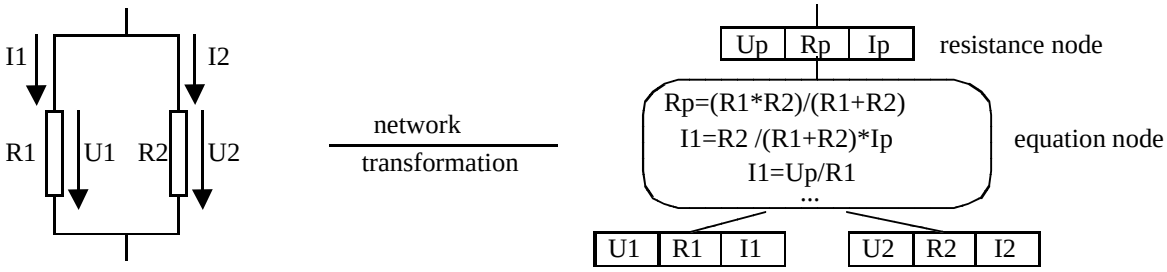


Figure 2-3: Parallel grouping network and corresponding structure tree

Unlike other approaches such as the FLAMES system, the qualitative SPS method, and the Connectivity method, QNA offers certain features to improve the accuracy of qualitative behavior predictions. In the following, these features are summarized. In Section 2.3.8, we enumerate preconditions which secure that QNA's qualitative calculus is sound and complete. Additionally, the proof of these properties is sketched. Note that definitions of soundness and completeness are taken from [Struss 1990].

2.3.4 Complex qualitative operators

Rather than relying on qualitative versions of basic arithmetic operators (+, -, *, /), QNA computes qualitative attribute values by a set of operators which are qualitative versions of quantitative equations represented in equation nodes of structure trees. QNA's qualitative calculus is based on about 100 qualitative operators represented by a set of tables. A limited number of operators suffices because QNA's internal representation of electrical circuits offers a limited number of standard components, behavior modes, and elementary network structures. In the following, we outline how QNA's qualitative operators are defined by exemplarily considering the parallel grouping example (see Figure 2-3). We demonstrate that QNA's utilization of qualitative versions of equations is fundamental for accurate device behavior prediction.

As noted above, the compensation resistance of a parallel grouping of two resistances *R1* and *R2* can be computed by applying the equation *Rp = (R1 * R2) / (R1 + R2*). In QNA, the qualitative attribute values of compensation resistance *Rp* are calculated by applying four qualitative operators, i.e. *QRp_act, QRp_ref, QRp_dev1*, and *QRp_dev2*. In this section, we elaborate on *QRp_act* and *QRp_ref. QRp_dev1* and *QRp_dev2* are described in Section 2.3.5.

QRp_act is a qualitative versions of *Rp = (R1 * R2) / (R1 + R2)*. This operator computes the actual values of the compensation resistance *Rp* from actual values of *R1* and *R2*. *QRp_act* is defined by applying the corresponding quantitative equation to the intervals represented by the qualitative actual values of *R1* and *R2*. That is, for the definition of *QRp_act*, interval arithmetics is performed and certain limits are calculated. Table 2-4 presents the definition of *QRp_act*. *QRp_ref* is also a qualitative versions of *Rp = (R1 * R2) / (R1 + R2)*. This operator computes the reference values of the compensation resistance *Rp* from reference values of *R1* and *R2*. Since actual values and reference values of resistances are described by the same set of qualitative values (see Table 2-1), Table 2-4 also presents the definition of *QRp_ref*.

R1_act | R2_act R1_ref | R2_ref	0	pos	pinf
0	0	0	0
pos	0	pos	pos
pinf	0	pos	pinf

Table 2-4: *QRp_act* and *QRp_ref*, computation of actual values and reference values of compensation resistance *Rp*

Note that, qualitative actual and reference values of *Rp* cannot be derived by applying qualitative basic arithmetics. In particular, evaluation of *Rp = (R1 * R2) / (R1 + R2)* by stepwise applying qualitative basic arithmetic operators is impossible because qualitative multiplication is undefined if *R1* and *R2* show the qualitative actual value zero and positive-infinite, respectively (see shaded cells in Table 2-4). Therefore, QNA's definitions of qualitative operators are fundamental for accurate computation of qualitative values.

2.3.5 Exploitation of threefold parameter representation

In this subsection, we introduce the qualitative operators *QRp_dev1* and *QRp_dev2* that allow computation of *Rp*'s deviation values. We demonstrate that QNA's threefold parameter representation is the basis for the definitions of these operators which are both fundamental to assure accurate computation of Rp's qualitative deviation values.

Rp_ref | Rp_act	0	pos	pinf
0	0	neg	neg
pos	pos	neg / 0 / pos	neg
pinf	pos	pos	0

Table 2-5: *QRp_dev1*, computation of deviation values of compensation resistance *Rp*

QRp_dev1 is a qualitative version of the equation *deviation = actual value - reference value* which holds for each parameter type. *QRp_dev1* computes *Rp*'s qualitative deviation values from Rp's qualitative actual and reference values. Thus, unlike deviation computation in [Malik and Struss 1996], QNA's computation of deviations is inseparable from computation of qualitative reference values. The definition of *QRp_dev1* is presented in Table 2-5. If actual and reference value are both pinf the *equation deviation = actual value - reference value* cannot be applied because *infinite - infinite* is undefined. In this case the deviation value zero is reasonable because, any other value indicates that actual behavior is different from reference behavior. Considering *0 < pos < pinf*, Table 5 should be obvious ("/" means logical "or").

QP_dev2's definition is based on the assumption that none of the resistances *R1* and *R2* is zero. In this case, *Rp* is a monotonously increasing function of *R1* and *R2* because *dRp/dR1 > 0* and *dRp/dR2 > 0* hold. That is, an increasing (decreasing) value of *R1* or *R2* leads to an increasing (decreasing) value of *Rp*.

QRp_dev2 qualitatively represents that *Rp* is a monotonously increasing function of *R1* and *R*2. Applying *QRp_dev2*, qualitative deviations of *Rp* are computed from deviations of *R1* and *R2*. The definition of *QRp_dev2* is presented in Table 2-6. Note that Table 2-6 cannot be derived by stepwise evaluation of

*Rp = (R1 * R2) / (R1 + R2).* Again, QNA's exploitation of complete equations is the basis for accurate behavior prediction.

R2_dev | R1_dev	neg	0	pos
neg	neg	neg	neg / 0 / pos
0	neg	0	pos
pos	neg / 0 / pos	pos	pos

Table 2-6: *QRp_dev2*, computation of deviation values of compensation resistance *Rp*

As described above, *QRp_dev2* can only be applied if *0 < R1* and *0 < R2* hold. This applicability condition has to be fulfilled by actual and reference values. Hence, the applicability condition implies that in QNA, computation of qualitative deviations is inseparable from computation of actual and reference values. As another point, due to *QRp_dev2*'s applicability condition, *QRp_dev1* is fundamental for the completeness of QNA's computation of Rp's deviation values. The following simple example demonstrates that *QRp_dev2* is essential to secure soundness of QNA's computation of Rp's deviation values.

QNA's computation of qualitative attribute values of the compensation resistance *Rp* of a parallel grouping of two resistances *R1* and *R2* is investigated (see Figure 2-4). *R1* and *R2* are represented *by R1_[act_pos, ref_pos, dev_0] and R2_[act_pos, ref_pos, dev_pos]*. First, applying *QRp_act* and *QRp_ref* (see Table 2-4), Rp's qualitative actual and reference values are computed as *act_pos and ref_pos*. Second, applying *QRp_dev1* (see Table 2-5), qualitative deviation values of Rp are inferred as *dev_(neg / 0 / pos)*. Third, utilization of *QRp_dev2* (see Table 2-6) leads to dev_pos. Note that the applicability condition of *QRp_dev2* is obviously fulfilled.

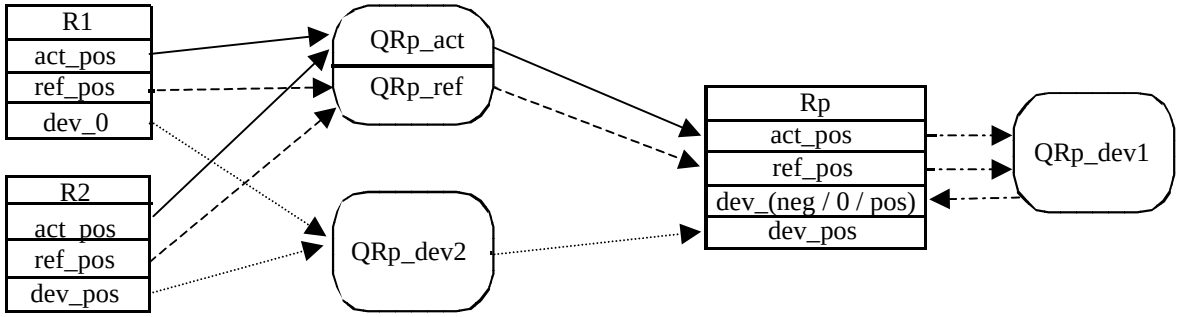


Figure 2-4: Computation of qualitative attribute values of parallel compensation resistance *Rp*

In this example, applying *QRp_dev1* in fact generates spurious solutions that can be avoided by utilization of *QRp_dev2*. If Rp's actual and deviation values were computed such as in [Malik et al. 1996], in this example, unsound results would be obtained. In this subsection, we have demonstrated that *QRp_dev1* and *QRp_dev2* are both necessary to assure soundness and completeness of QNA's computation of Rp's qualitative deviation values. Hence, QNA's threefold qualitative parameter representation is essential for accurate behavior predictions.

2.3.6 Twofold computation of current and voltage values

In this subsection, the computation of qualitative attribute values of currents and voltages is considered. In QNA, each current and voltage represented in the structure tree is computed twice. First, it is derived from the current of the corresponding father resistance node. Second, it is computed from the voltage of the father node.

For example, considering the parallel grouping shown in Figure 2-3, the current *I1* through resistance *R1* is computed from *Ip*, the current of the father node by applying the well-known current divider rule *I1 = R2 / (R1 + R2) * Ip*. Additionally, *I1* is calculated from *Up*, the voltage of the father node by applying *I1 = Up / R1*.

If quantitative parameter values were considered, the alternative computations of I1 would produce the same result because they are based on a redundant set of physical dependencies between parameters of adjacent resis-

tance nodes. Considering qualitative values, it can be demonstrated that QNA's twofold computation of current and voltage values is essential to avoid spurious behavior predictions.

2.3.7 Global computation of qualitative attribute values

In addition to local propagation of qualitative values, QNA globally analyses network structures and structure trees in order to eliminate spurious predictions. In particular, a global analysis of the network structure allows to determine current directions. Knowledge about current directions can be used to eliminate some spurious predictions concerning actual and reference values of currents.

Subnetworks that behave like passive electrical double-poles can easily be identified by a global analysis of the structure tree. Due to Kirchhoff's laws, voltage drops across components and groupings of components inside a certain passive double-pole network are equal or of lower amount than the voltage drop across the two poles of the double-pole. This inequality can be exploited to eliminate some spurious predictions concerning actual and reference values of voltages. However, a detailed analysis of the soundness of QNA's calculation of qualitative attribute values still has to be performed.

2.3.8 Properties of the qualitative calculus

The qualitative calculus described so far is sound and complete if the network shows the following characteristics. First, there is only one source in the circuit. Second, the single fault assumption holds. Third, the network can be structured in combinations of series and parallel groupings of standard components. Fourth, components do not show internal dependencies, i.e. their behavior does not depend on certain current or voltage values. If these conditions are fulfilled, the proof of soundness and completeness can be sketched as follows.

Structure trees consist of combinations of subtrees such as exemplarily shown in Figure 2-3. These subtrees consist of two child resistance nodes and one father resistance node. Equation nodes are not significant for the explanations in this subsection. The father node is either the root node of the structure tree or it represents a series or parallel grouping such as in Figure 2-3. Note that, propagation at the root node is different from propagation described in this paper. Thus, there are three different subtree types to be investigated, i.e. root node, series grouping, and parallel grouping.

It can be shown that for all subtree types and all possible combinations of qualitative attribute values of the source and of resistances, QNA's qualitative calculus computes exactly one qualitative value for attributes of currents, voltages and compensation resistances. That is, there are no disjunctions of qualitative values computed at all. This suffices to prove soundness and completeness because, in [Struss 1990] it is shown that local propagation methods grounded on interval-based qualitative values and interval arithmetics are complete but may be unsound. Since these methods are complete, unsound results may only occur if disjunctions of qualitative values are computed.

Note that the conditions enumerated at the beginning of this subsection, limit the number of possible attribute values. For instance, the single fault assumption secures that Table 2-6 is never evaluated with *R1_[dev_pos]* and *R2_[dev_neg]*. Due to some extensions of the calculus not summarized here, correct results are computed even when multiple sources or multiple faults occur. This secures correct results even if components show internal dependencies. Anyhow, if nested star and delta groupings of standard components occur, the calculus is unsound.

2.3.9 Dealing with variable component behavior

Some electrical or electronic components show variable behavior depending on certain parameter values. That is, they show more than one ok behavior and certain physical parameter values determine the actual ok behavior. For example, the current through the coil of a relay controls whether the relay is open or closed. QNA represents variable components by controlled versions of the standard component models described in Section 2.3.1. These controlled standard models allow the representation of causal parameter dependencies in so-called QNA behavior tables. In these tables, for all combination of qualitative input values, qualitative output values are given. Table 2-7 shows some rows of the QNA behavior table of a relay which is closed if the coil current is positive. Thus, the coil current is the input and the resistance of the switch is the output. Component models showing QNA behavior tables are called "table component models". In the following, we describe the integration of QNA behavior tables into MAD's computation of qualitative values.

input: coil current			output: switch resistance		
act	ref	dev	act	ref	dev
0	0	0	pinf	pinf	0
pos	0	pos	0	pinf	neg
pos	pos	0	0	0	0
pos	pos	pos	0	0	0
0	pos	neg	pinf	0	pos

Table 2-7: Extract of QNA relay behavior table

To simplify explanations, we regard the relay circuit example presented in Figure 2-5. When the switch is closed, there is electric current Ic through the relay coil and, thus, the relay switch is also closed. In this case, there is current through the light bulb and it is shining. When the switch is open the light bulb does not shine. QNA's circuit model consists of standard component models building a QNA network and the QNA relay behavior table presented in Table 2-7. QNA analyzes this model in a two step process, i.e. network analysis and overall behavior computation.

1. **Network analysis.** In this step, causal parameter dependencies represented in the QNA relay behavior table

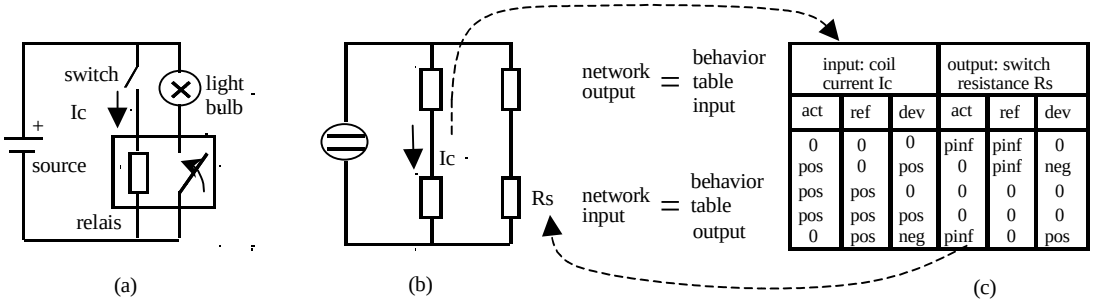


Figure 2-5: Simple relay circuit (a), corresponding QNA network (b), and QNA relay behavior table (c)

are omitted. For each behavior table output (= network input), the QNA network is analyzed as described in Sections 2.3.3 to 2.3.8. The result is a network behavior relation consisting of qualitative values of network parameters. Each tuple of this relation is a network behavior prediction for a certain network input (= behavior table output).

2. **Overall behavior computation.** The QNA relay behavior table is internally represented as a relation. To compute overall behavior from the QNA relay circuit model (i.e. QNA network + QNA relay behavior table), the network behavior relation obtained in the previous step and the relay behavior relation are combined using the natural join operator well-known from relational algebra. Each tupel of the resulting relation describes a possible behavior of the relay circuit.

This method can be easily extended to handle circuit models showing more than one table component model. During "network analysis", all causal parameter dependencies are omitted and the QNA network is analyzed for all combinations of network input. "Overall behavior computation" combines all QNA behavior tables and the network behavior relation by multiple appliance of the natural join operator.

2.4 COMEDI

MAD's internal electrical circuit representation described in Section 2.3 is a convenient basis for automated fault tree generation because it allows adequate fault symptom prediction provided that the underlying device model is appropriately generated. However, adequate electrical device modeling based on QNA's internal models is a challenging task consisting of the following steps. First, a set of instances of QNA's internal standard component models has to be selected. Second, these standard component models have to be assembled to a device model. Third, extensive QNA behavior tables must be generated if causal parameter dependencies have to be represented.

In our application, we deal with heterogeneous electrical circuits consisting of components showing a variety of different behavior types frequently depending on the electrical context. Considering these circuits, automatically generating adequate diagnosis models from electronic product data does not seem to be feasible because, mostly, product data do not comprise sufficient information. Manually generating diagnosis models bears the possibility to integrate expert knowledge into the modeling process. As described in Section 2.2, in our application, know-how acquired during circuit design and forklift service can be very helpful for diagnosis circuit modeling. Therefore, we designed MAD such that engineers from the electronic design division can generate diagnosis device models bringing in their expert know-how. Nevertheless, manually performing the above-described modeling steps has to be supported and facilitated for the following reasons.

- We developed MAD's electrical circuit representation to assure adequate automated fault symptom prediction. We did not focus on representing electrical circuits from a design engineers point of view. Thus, for design engineers, adequately dealing with MAD's internal standard component models and threefold parameter representation may be difficult.
- MAD's internal models of electrical and electronic standard components can be very complex. For instance, the behavior of a logical AND circuit is usually described by the well-known truth table. This table has exactly four rows because there are two input voltages and, in the logical circuit domain, there are only two distinct voltage values, i.e. "true" and "false". Since QNA provides 57 different qualitative voltage values (see Section 2.3.2), the corresponding QNA behavior table consists of 57 * 57 = 3249 rows. Obviously, manually generating this behavior table is costly and may even be faulty.

In order to facilitate electrical device modeling, MAD provides a user interface COMEDI (COmponent Modeling EDItor) which is described in the following subsections. The basic concepts of COMEDI are also presented in [Milde et al. 2000]. COMEDI is similar to a CAD tool which hopefully assures a high degree of acceptance among design engineers (see Figure 2-14). MAD's internal electrical circuit representation is completely hidden from users. Electrical device models can be easily assembled, utilizing predefined models from three different libraries (see Figure 2-6).

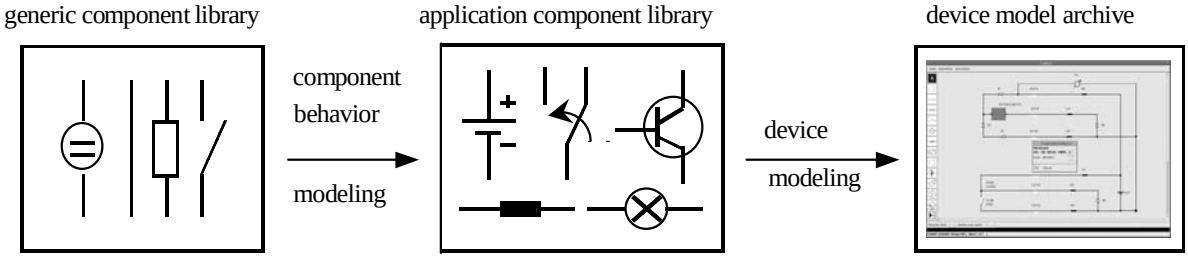


Figure 2-6: COMEDI libraries and the device modeling process

- The generic component library contains all elementary component class models that are provided by MAD's internal representation of electrical circuits. Thus, the generic component library cannot be extended. It consists of 728 different component class models.
- The application component library contains models of component classes that are explicitly designed for a certain application. These component class models are interactively generated, based on generic component class models. Thus, the application component library can be extended.
- The device model archive allows systematic reuse and modification of device models that were created during former modeling sessions. These models are assemblies of application component models.

Providing library models is fundamental because utilization of libraries massively reduces the complexity of the modeling process which is essential for the acceptance of MAD in our application. As shown in Figure 2-6, COMEDI device models are assemblies of application component class models which are based on generic component class models. Hence, to prepare the description of circuit modeling, in the following, we outline COMEDI's generic component class models and the hierarchical structure of the generic component library which is shown in Figure 2-7.

2.4.1 Generic component library

Each of COMEDI's generic component class models shows a set of basic behavior modes which are presented in Section 2.3.1. These behavior modes are possible actual behaviors, i.e. each of these behavior modes can be chosen to model actual component behavior (correct and faulty). Additionally, each generic component class model shows one behavior mode describing component reference behavior. Thus, a generic component class model comprises a selection of actual behavior modes and exactly one reference behavior mode. Modeling a certain actual behavior means selecting one behavior mode from the set of possible actual behaviors. When the reference and the actual behavior mode are identical, usually, correct component behavior is modeled.

The internal representation of the generic component library is hierarchical such that the most general model is the root of a tree and the most specific models can be found in the leaves (see Figure 2-7). The generic component model (the root) consists of two sets of behavior modes, i.e. possible and reference behavior modes. Both sets contain all basic behavior modes described in Section 2.3.1, both active and passive behavior modes. In the next lower level in Figure 2-7, passive and active component models also show sets of possible and reference behavior modes but these sets contain only active and passive behavior modes, respectively. For each model in the next lower level in Figure 2-7, the set of reference behavior modes is restricted to exactly one reference (ref) behavior mode. Models found in the leaves additionally restrict the number of possible (pos) behavior modes. Since generic component class models show exactly one reference behavior mode, for application component class modeling, only generic component models from the two lowest levels in Figure 2-7 can be utilized. Using COMEDI, modeling electrical circuits consists of component behavior modeling and device modeling (see Figure 2-6). Both tasks are described in the following two subsections.

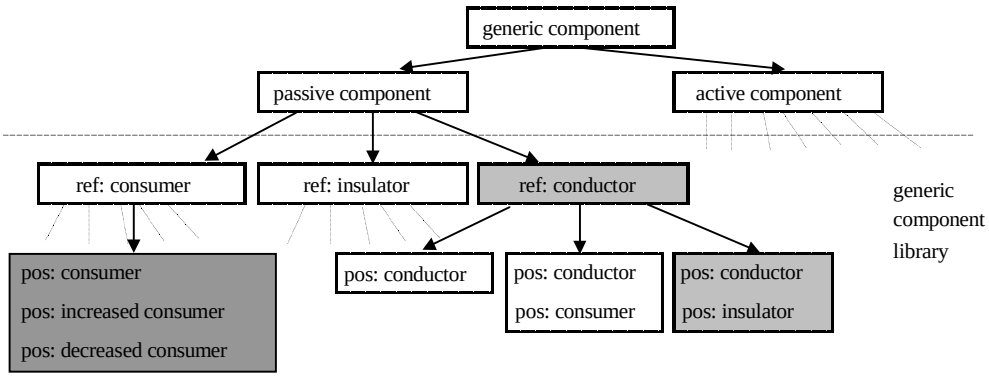


Figure 2-7: Selected parts of the hierachical structure of COMEDI's generic component library

2.4.2 Application component modeling

In this subsection, the interactive generation of models of application component classes is described. These models are based on generic component class models. Resistors, wires, switches, batteries, sensors, relays, diodes, logical circuits, etc. are typical application component classes. We consider wire 4 of the accelerator pedal circuit as an illustrating example.

1. **Operating mode modeling.** In this step, the electrical behavior of an application component class is modeled. Components may show different modes of operation. Switches, for example, can be open or closed. Each operating mode of a component class is explicitly represented by a generic component class model. Thus, if a component class shows different operation modes, its application component class model consists of more than one generic component class models. In order to model a certain operating mode, first, reference behavior is modeled, second, possible actual behavior is determined.

   Reference behavior modeling. According to Figure 2-7, first of all, it is decided whether the model is active or passive. Then an adequate reference behavior mode is selected and a name is given to this behavior mode. Usually, this step is performed to determine correct component behavior. Obviously, wire 4 is a passive component. Its internal resistance is very small and negligible in the context of accelerator pedal circuit analysis. Hence, we determine the correct behavior of wire 4 by selecting the reference behavior mode "ref: conductor" which is shaded pale in Figure 2-7. An adequate name for this behavior mode is "conducting".

Possible actual behavior modeling. In this step, all possible actual component behavior is modeled which is relevant for automated fault tree generation. That is, correct and faulty behavior has to be represented by choosing an available set of possible behavior modes. Available sets of possible behavior modes are represented by the node selected in the previous modeling step and its successors (see Figure 2-7). Correct behavior of wire 4 is already discussed above. If the wire is broken there is no current and the wire works like an insulator. Hence, for possible behavior modeling, we choose the "pos: conductor, pos: insulator" model from the generic component library (see Figure 2-7). An appropriate name for "pos: insulator" is "broken". Figure 2-8 (a) shows the application component class model generated so far. Additionally, in Figure 2-8 (b), there is an application component class model of a switch in order to present a model of a component showing two operating modes.

Note that, for some reference behavior modes, there are corresponding deviative behavior modes offered for possible actual behavior modeling. For instance, if the reference behavior is "ref: consumer" then automatically "pos: increased consumer" and "pos: decreased consumer" are offered which is exemplarily shown in the dark shaded box in Figure 2-7. Internally, these behavior modes can be realized due to qualitative deviation values.

2. **Measurement modeling.** After the electrical behavior of a certain application component class is modeled, measurements can be defined. Both, the voltage drop across the generic component class model and the current through the component model can be subject of a measurement. As a consequence of modeling a certain measurement, there is a column for the parameter in the fault relation showing corresponding qualitative parameter values.
3. **Observation modeling.** Similar to functional labeling [Price et al. 1996a], user-defined strings such as "light bulb shining" can be attached to qualitative current and voltage values to model typical observations used for diagnosis. For each observation in a device model, there is a column in the fault relation. Note that, observations can be utilized to model the error flags in our application.
4. **Replaceability modeling.** As noted above, during diagnosis sessions, components may be replaced by correct-working substitutes to simplify fault identification. If replaceability of an application component class is modeled, then, first, each generic component class model describing a certain operating mode is duplicated. In a second step, for each duplicate, all possible behavior modes different from the reference

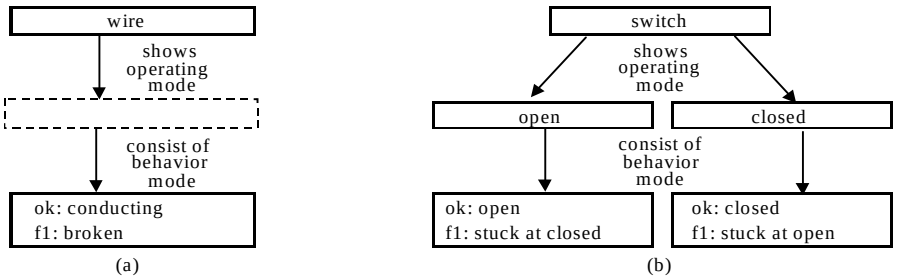


Figure 2-8: Application component class models of wire (a) and switch (b)

behavior mode are removed. Third, the resulting generic component class models are added to the application component class model. By this means, for each operating mode, there is a corresponding operating mode automatically modeled showing no faulty behavior.

5. **Fault identification modeling.** For each application component class model, fault identification can be interactively declared to be possible. For each fault identification in the device model, there is a column in the fault relation directly indicating whether a fault occurred.

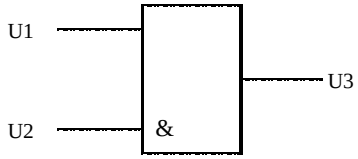
If complex components such as amplifying circuits and logical circuits have to be modeled, frequently, i/o behavior models may be required neglecting unnecessary structural and behavioral details. In the next section, generating i/o component models is described.

2.4.3 Building i/o application component class models

Application component class models described in Section 2.4.2 are assemblies of generic component class models which show predefined behavior, i.e. their current / voltage characteristics are predefined. Thus, in an application component class model, parameter values of voltage sources and resistances are predefined and values of current and voltage are implicitly determined by generic component behavior, the internal structure of the application component model, and its electrical context. There are no behavior descriptions explicitly relating parameters which belong to different generic component models. Thus, the expressiveness of application component class models described in Section 2.4.2 is limited. In this section, i/o application component class models are introduced which extend MAD's modeling ability described so far. In i/o application component models, parameter dependencies can be defined even if concerned parameters belong to different generic component models. Definitions of these parameter dependencies are grounded on controlled versions of generic component models.

A complex component consists of a large number of simpler components which build the internal structure of the complex component with corresponding internal behavior. For example, an amplifying circuit may consists of some hundred basic components. Usually, complex components and subcircuits are designed such that they provide a certain overall behavior. The overall behavior of an amplifying circuit is its i/o behavior, i.e. input * gain = output. As known from electrical engineering, modeling circuit behavior without rebuilding the internal circuit structure in the model is helpful to reduce the device model complexity. In this case, mainly, circuit i/o behavior is represented, e.g. input * gain = output. In i/o models, i/o parameter dependencies are explicitly specified by equations or relations whereas the hardware realization of these dependencies is omitted.

Considering modeling for diagnosis, modeling i/o component behavior is reasonable because, often, fault identification means assigning faults to complex components which can be replaced. Identifying basic component faults is not necessary and, thus, explicit representation of basic components is not adequate. In our application, additionally, i/o behavior modeling is essential for adequate fault symptom prediction and the representation of established STILL diagnosis strategies. Thus, we designed COMEDI such that the interactive generation of i/o application component class models is supported. Users can define qualitative values for controlled (output) and controlling (input) parameters. Causal dependencies between these parameters are represented in so-called COMEDI i/o behavior tables. External non-electric controlling (input) parameters can be modeled. i/o application component models can also be stored in the application component library. Engineers can easily build these models exploiting their qualitative understanding of how certain complex components or subsystems work.



U1	U2	U3
F	F	F
F	T	F
T	F	F
T	T	T

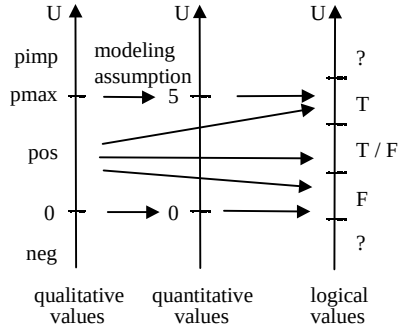
behavior

Figure 2-9: Logical model of AND gate

Rather than completely presenting COMEDI's features for i/o application component modeling, in this report, we demonstrate how COMEDI users can model a logical AND gate which shows well-known i/o behavior presented in Figure 2-9. This simple example demonstrates that, using MAD, modeling combinatorial circuits (i.e. logical circuits without memory) is uncomplex. We model correct AND gate behavior only, i.e. there are no fault models.

- **Input modeling.** Input pins of the AND gate are passive and input currents are negligible. Thus, insulators are adequate models for these pins (see Figure 2-12). The AND gate interprets voltage levels at input pins as logical input values "true" (T) and "false" (F) as shown in Figure 2-10. Hence, for input modeling, QNA's qualitative voltage values can be mapped to logical values (see Figure 2-10). The "?" indicates unknown component behavior. In our case, unexpectedly high or low voltage levels may destroy the AND gate which is indicated by "?". The "?" is specially treated during fault symptom prediction. Note that, we explicitly assume that, at input pins, QNA's qualitative value *pmax* stands for a quantitative voltage level which is interpreted as logical "T". Different definitions of value mappings can be reasonable.

- **Output modeling.** At its output pin, the AND gate provides a certain voltage level. Thus, for output modeling a controlled voltage source is adequate (see Figure 2-12). The actual amount of the output voltage is determined by the actual logical output value as shown in Figure 2-11. Thus, for output modeling, logical values can be mapped to QNA's qualitative voltage values (see Figure 2-11).

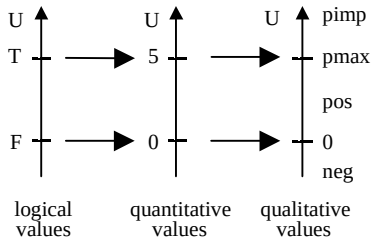


qualitative value	logical value
pimp	T / ?
pmax	T
pos	T / F
0	F
neg	F / ?
nmax	?
nimp	?

value mapping qual_to_log

Figure 2-10: Mapping of QNA's qualitative values to logical values

- **i/o behavior modeling.** In COMEDI, i/o parameter dependencies are represented in COMEDI behavior tables which are based on user defined qualitative values of controlled (output) and controlling (input) parameters. In the *AND* circuit example, the output voltage *U3* causally depends on both input voltages *U1* and *U2*. The corresponding COMEDI i/o behavior table is the logical AND table presented in Figure 2-9.



logical value	qualitative value
T	pmax
F	0

value mapping log_to_qual

Figure 2-11: Mapping of logical values to QNA's qualitative values

Generation of i/o application component class models described so far can be summarized as follows. First, the model structure is build which consists of QNA's standard components with corresponding basic behavior modes. Second, controlled (output) and controlling (input) parameters are determined. Third, value mappings are interactively defined for controlled and controlling parameters such that modeling causal i/o dependencies is simplified. Fourth, causal i/o parameter dependencies are represented by COMEDI i/o behavior tables based on user defined qualitative values.

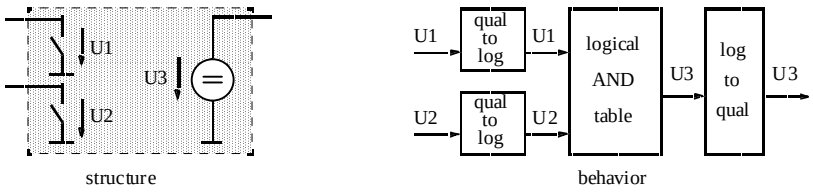


Figure 2-12: Abstract application component class model of AND gate

Note that, as described in Section 2.3.9, QNA represents causal parameter dependencies in complex QNA behavior tables which show actual, reference, and deviation values for all parameters (see Table 2-7). These complex QNA behavior tables are automatically computed from definitions of value mappings and COMEDI i/o behavior tables. Thus, for i/o application component modeling, COMEDI users do not have to be familiar with QNA's threefold parameter representation.

Considering the accelerator pedal example, the application component class model of the potentiometer 1B1 is an i/o model showing a controlled (output) voltage source to model the voltage at the middle of the potentiometer. This voltage causally depends on the (input) voltage drop across the potentiometer and the (input) position of the accelerator pedal. The pedal position is modeled as an external non-electric controlling (input) parameter.

2.4.4 Application component library

The application component library consists of several abstract model classes, representing typical application component classes such as resistors, wires, switches, batteries, sensors, relays, diodes, logical circuits, etc. Each abstract model class shows one or more concrete subclasses (see Figure 2-13 (a)). Different concrete model classes represent distinct physical characteristics of correct or faulty component behavior. Regarding a wire again, in a concrete model class, the wire resistance can be represented or it can be neglected. Faulty behavior such as “wire broken” and “crimped wire” may be considered. Thus, several different concrete subclasses of the abstract wire model class are reasonable (see Figure 2-13 (b)). Each concrete model subclass is defined by an application component class model.

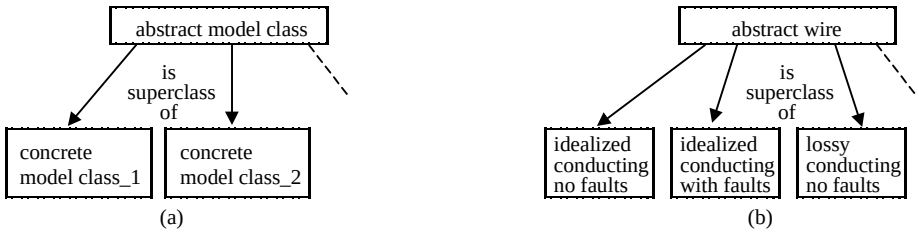


Figure 2-13. Abstract classes and concrete subclasses in the application component library

After interactively generating an application component class model it is stored in the application component library. Either the new model defines a new abstract model class with a corresponding concrete subclass or it extends an existing abstract class defining a new concrete subclass. Instances of concrete subclasses are utilized for device modeling.

A component model showing two or more distinct ok behavior (operating modes) is called a variable component model. A switch model for instance, is a variable component model because there are two operating modes, i.e. “open” and “closed”. There are three different types of variable component models, i.e. context component models, input component models, and operating mode models.

- **Context component models** show internal controlling parameters determining the valid operating mode. Internal controlling parameters are current or voltage occurring inside the modeled system. Their values are determined by device structure and component behavior represented in the device model. For instance, the current through the relay coil can be the internal controlling parameter of a COMEDI relay model. Since the behavior of variable components with internal controlling parameters massively depends on their actual electrical context we call their COMEDI models “context component models”.
- An **input component model** shows external controlling parameters determining the valid operating mode. In COMEDI, external controlling parameters are parameters whose causal origins are outside the modeled electrical system (see Section 2.4.3). Hence, these parameters are part of the input to the modeled system. For instance, a COMEDI photo cell model may show the luminous intensity as an external controlling parameter. As another example, the environment temperature can be the controlling parameter of a temperature sensor model.
- **Operating mode models** do not show any controlling parameter. For instance, models of manually operated switches may show no controlling parameters.

2.5 Device modeling and fault symptom prediction using COMEDI

MAD allows qualitative fault symptom prediction with respect to a certain reference device behavior. Thus, before fault analysis can take place, the corresponding reference device behavior has to be determined. Since in

COMEDI, modeling electrical devices and predicting device behavior is not separated, both tasks are described in this section.

2.5.1 Modeling device reference behavior

The following enumeration of operations describes how device reference behavior can be determined.

1. **Device structure modeling.** Similar to a CAD tool, icons representing abstract classes of application component models can be assembled on the screen to determine the device model structure. Device models from the device model library can be reused, modified and integrated. Note that, measuring points and considerable short circuits are also modeled in this operation. In COMEDI, there are unique icons to represent these objects although they are no physical components.
2. **Device behavior modeling.** For each component represented by an icon in the device model, a concrete model class (an application component class model) can be interactively selected from a menu. This operation determines the physical phenomena and all possible behavior (correct and faulty) represented in the device model. Since every abstract model class has a default concrete model class, device behavior modeling is optional.
3. **Input modeling.** For all input component models, values of external controlling parameters can be determined by, choosing parameter values from a menu. Since external controlling parameters have default values, this step is optional.
4. **Operating mode modeling.** For each operating mode model, one of its operating modes can be selected by, selecting operating modes from a menu. Since operating mode models have default operating modes, this step is optional.
5. **Context modeling.** The modeling steps described above non-ambiguously identify the reference behavior of all component models except context component models such as relay models. For these models, selecting COMEDI's menu item "simulation / reference simulation" automatically computes validity of alternative operating modes. "reference simulation" can have three different results.

   (I) consistent and non-ambiguous. For each context component model, exactly one operating mode is valid. That is, the device model describes exactly one reference device behavior. In this case, reference behavior is determined and fault symptom analysis can be started.

   (II) consistent but ambiguous. For each context component model, at least one operating mode is valid, and, there is one (or more) context component models for which more than one operating mode is valid. That is, the device model describes more than one reference device behavior. In this case, the behavior of a certain context component model can be individually determined by choosing a certain operating mode from a menu. By this means, the electrical context of the context component model is implicitly modeled. After manually selecting operating modes of context component models, "simulation / reference simulation" has to be restarted. Immediately performing fault symptom prediction is not possible.

   (III) inconsistent. There are one or more context component models which do not show any valid operating mode. That is, according to the device model, no steady state reference device behavior is possible. Either the device model is faulty or the modeled electrical system does not show steady state behavior.

   When COMEDI's "simulation / reference simulation" results in consistent and non-ambiguous, context modeling has succeeded. That is, a reference operating mode of the device is determined.

In COMEDI, a reference operating mode is a consistent combination of external parameter values and operating modes of operating mode models and context component models. Device reference operating modes can be saved to be available for later fault symptom analysis. Device models usually show several different reference operating modes.

2.5.2 Analysis of faulty device behavior

COMEDI offers two different types of faulty device behavior analysis. First, COMEDI allows symptom prediction for a certain component fault (combination) with respect to a certain reference operating mode. Second, extensive faulty device behavior analysis can be performed which is the basis for automated fault tree generation.

- **Fault modeling and symptom prediction.** To model a certain component fault, with respect to a certain reference operating mode, first, the device operating mode is chosen from a menu. Second, a component icon is selected and the corresponding faulty behavior mode is chosen from a menu. Modeling multiple faults is possible. After the reference operating mode and the component fault are determined, selecting COMEDI's menu item "simulation / single fault simulation" starts fault symptom prediction. As a result, for all measurements and observations, qualitative values and strings are presented on the screen, respectively.
- **Extensive faulty device behavior prediction.** Selecting COMEDI's menu item "simulation / simulation of all faults" activates extensive faulty device behavior prediction. For all predefined reference operating modes, and for all component faults considered in component models, and for all short circuits considered in the device model, fault symptoms are automatically predicted. As a result, for all measurements and observations, qualitative values and strings are computed, respectively. COMEDI users can define multiple faults to be considered during the symptom prediction process.

2.5.3 Modeling and analyzing the accelerator pedal circuit

In Figure 2-14, the COMEDI device model of the accelerator pedal circuit is presented. As described in Section 2.4.2, all wires are modeled as idealized conductors that may break. Correct and faulty behavior of wire 4 is shown in the small window in the center of Figure 2-14. Fault identifications exists for these wires which can be verified in the lowest section of the fault relation shown in Figure 2-15. Note that, results of fault identifications are independent from device operating modes.

Both resistors R4 and R5 show "insulator" behavior, no fault modes, and labeling to model the measurements UFG_LOW, UFG_HIGH, and UFGS, respectively. Additionally, R4 shows a voltage measurement to model UFG (see fault relation in Figure 2-15). The manual voltage measurement UG described in Section 2.2 is modeled by a special multimeter component model (see Figure 2-14). Note that, in MAD's fault relations, measurements, observations, and fault identifications are simply called "tests" because they are identically utilized for fault tree generation.

There are two device operating modes defined for the accelerator circuit model, i.e. "STANDARD" and "KICKDOWN". "STANDARD" indicates that the accelerator pedal is not kicked down. Hence, switch 1S16 is in the operating mode shown in Figure 2-2 and the potentiometer is in a middle position. If the pedal is kicked down, operating mode "KICKDOWN" is reached. In this case, the switch has changed its operating mode and the potentiometer has reached an upper position. Automated behavior predictions are performed for all possible component behavior (correct and faulty) and all device operating modes considered in the accelerator pedal circuit device model. That is, for all device operating modes, the impact of faults on measurements and observations is computed. The results are stored in the fault relation shown in Figure 2-15 with qualitative values and in Figure 2-16 with quantitative values.

2.6 Characteristics and advantages of MAD's modeling approach

MAD's internal representation of electrical circuits is based on simple qualitative parameter values and a small set of standard components and basic component topologies. This simple representation suffices for the following reasons.

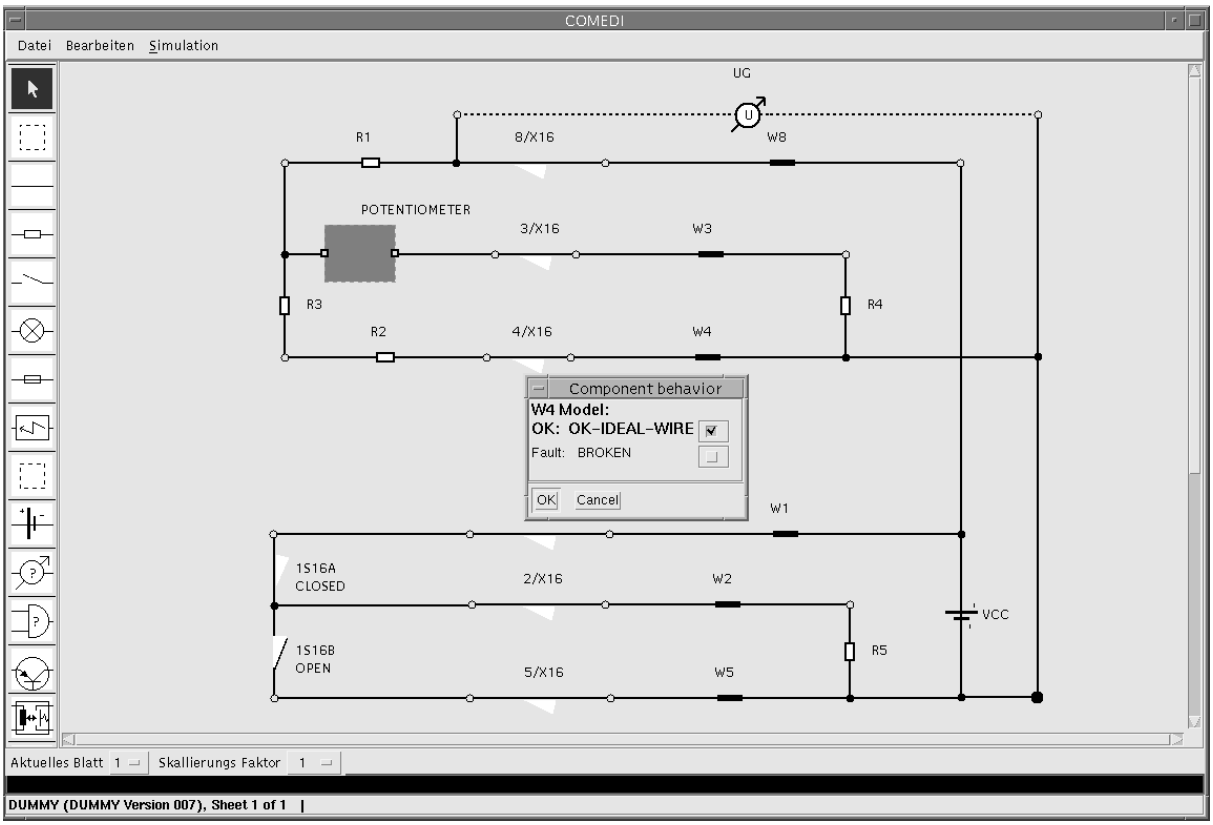


Figure 2-14: COMEDI model of accelerator pedal circuit

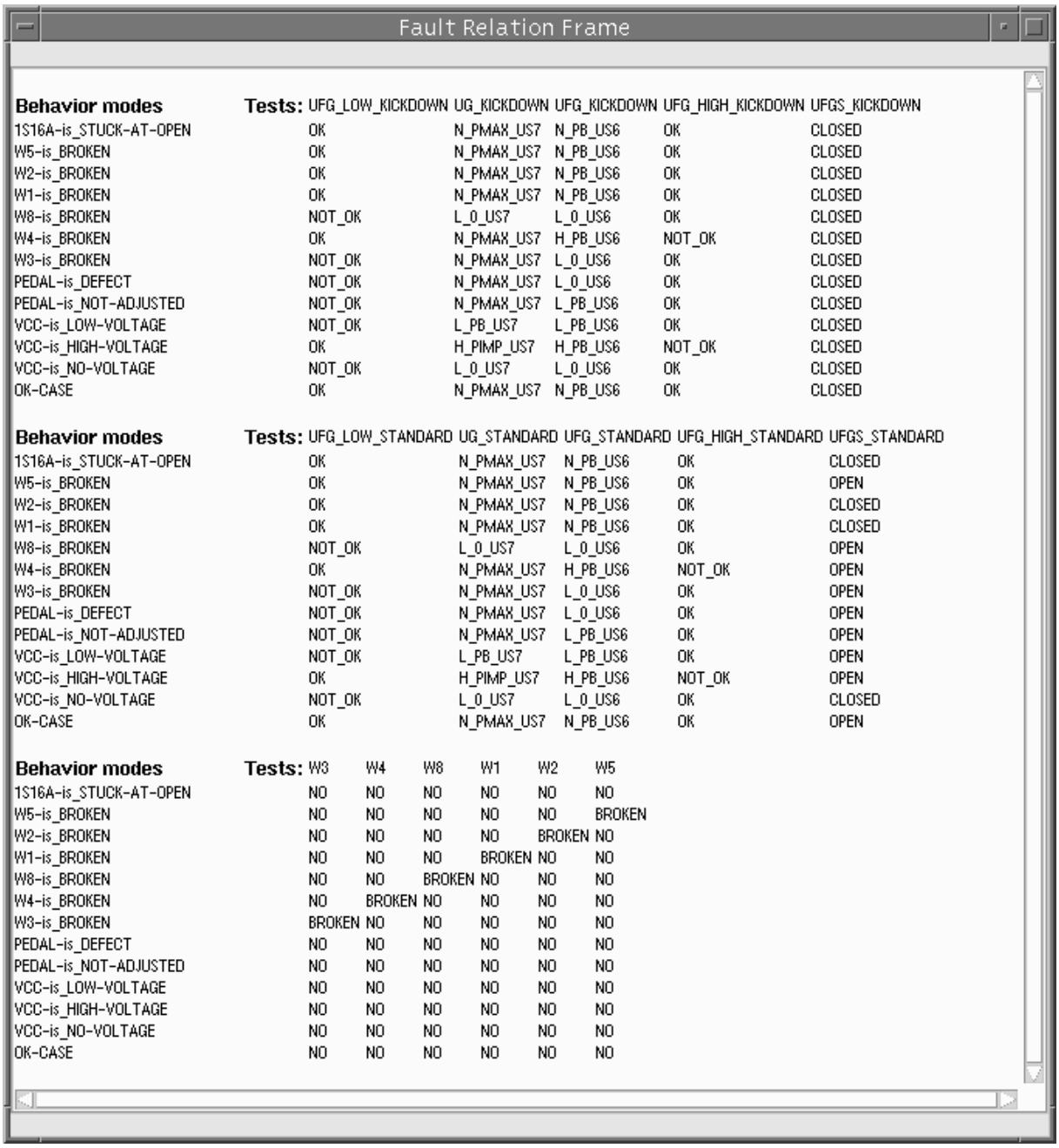
- A small number of qualitative standard components is sufficient, because, often, different physical components show similar electrical behavior, i.e. their current/voltage characteristics differ only slightly. Qualitative versions of these current/voltage characteristics are frequently identical.
- QNA's standard components are deliberately selected so that important behavior classes of the application domain can be represented adequately.
- An explicit representation of temporal dependencies is not necessary because we focus on steady state behavior analysis.

MAD's simple internal representation of electrical circuits shows the following advantages:

- Due to the simplicity of MAD's internal models, a limited number of well-known equations suffices for circuit analysis. Thus, it is possible to manually generate optimized qualitative versions of these equations such that (some) spurious symptom predictions can be avoided.
- Since qualitative parameter values represent intervals, they can be utilized to represent a wide range of different component faults which is the basis for tractable, systematic, and complete analysis of faulty device behavior. Utilizing quantitative parameter values, systematic analysis of faulty device behavior cannot be both complete and tractable.
- Due to MAD's threefold representation of parameter values, adequate description of deviative faults and symptoms is possible as well as modeling of structural faults and total losses.

COMEDI completely hides MAD's internal electrical models from users such that, for device modeling, knowledge concerning MAD's qualitative values and calculus is not required at all. Device models are assemblies of component library models. Libraries can be easily extended such that reusability of component and device models is secured. We developed COMEDI to support design engineers at their modeling task. COMEDI enables these experts to integrate their know-how at several different stages.

- Creating application component models, COMEDI users make several abstractions, i.e. they integrate know-how concerning intended and faulty component behavior, ignorable physical phenomena, and diagnosis granularity.



Behavior modes	Tests:	UFG_LOW_KICKDOWN	UG_KICKDOWN	UFG_KICKDOWN	UFG_HIGH_KICKDOWN	UFGS_KICKDOWN
1S16A-is_STUCK-AT-OPEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W5-is_BROKEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W2-is_BROKEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W1-is_BROKEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W8-is_BROKEN		NOT_OK	L_0_US7	L_0_US6	OK	CLOSED
W4-is_BROKEN		OK	N_PMAX_US7	H_PB_US6	NOT_OK	CLOSED
W3-is_BROKEN		NOT_OK	N_PMAX_US7	L_0_US6	OK	CLOSED
PEDAL-is_DEFECT		NOT_OK	N_PMAX_US7	L_0_US6	OK	CLOSED
PEDAL-is_NOT-ADJUSTED		NOT_OK	N_PMAX_US7	L_PB_US6	OK	CLOSED
VCC-is_LOW-VOLTAGE		NOT_OK	L_PB_US7	L_PB_US6	OK	CLOSED
VCC-is_HIGH-VOLTAGE		OK	H_PIMP_US7	H_PB_US6	NOT_OK	CLOSED
VCC-is_NO-VOLTAGE		NOT_OK	L_0_US7	L_0_US6	OK	CLOSED
OK-CASE		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED

Behavior modes	Tests:	UFG_LOW_STANDARD	UG_STANDARD	UFG_STANDARD	UFG_HIGH_STANDARD	UFGS_STANDARD
1S16A-is_STUCK-AT-OPEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W5-is_BROKEN		OK	N_PMAX_US7	N_PB_US6	OK	OPEN
W2-is_BROKEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W1-is_BROKEN		OK	N_PMAX_US7	N_PB_US6	OK	CLOSED
W8-is_BROKEN		NOT_OK	L_0_US7	L_0_US6	OK	OPEN
W4-is_BROKEN		OK	N_PMAX_US7	H_PB_US6	NOT_OK	OPEN
W3-is_BROKEN		NOT_OK	N_PMAX_US7	L_0_US6	OK	OPEN
PEDAL-is_DEFECT		NOT_OK	N_PMAX_US7	L_0_US6	OK	OPEN
PEDAL-is_NOT-ADJUSTED		NOT_OK	N_PMAX_US7	L_PB_US6	OK	OPEN
VCC-is_LOW-VOLTAGE		NOT_OK	L_PB_US7	L_PB_US6	OK	OPEN
VCC-is_HIGH-VOLTAGE		OK	H_PIMP_US7	H_PB_US6	NOT_OK	OPEN
VCC-is_NO-VOLTAGE		NOT_OK	L_0_US7	L_0_US6	OK	CLOSED
OK-CASE		OK	N_PMAX_US7	N_PB_US6	OK	OPEN

Behavior modes	Tests:	W3	W4	W8	W1	W2	W5
1S16A-is_STUCK-AT-OPEN		NO	NO	NO	NO	NO	NO
W5-is_BROKEN		NO	NO	NO	NO	NO	BROKEN
W2-is_BROKEN		NO	NO	NO	NO	BROKEN	NO
W1-is_BROKEN		NO	NO	NO	BROKEN	NO	NO
W8-is_BROKEN		NO	NO	BROKEN	NO	NO	NO
W4-is_BROKEN		NO	BROKEN	NO	NO	NO	NO
W3-is_BROKEN		BROKEN	NO	NO	NO	NO	NO
PEDAL-is_DEFECT		NO	NO	NO	NO	NO	NO
PEDAL-is_NOT-ADJUSTED		NO	NO	NO	NO	NO	NO
VCC-is_LOW-VOLTAGE		NO	NO	NO	NO	NO	NO
VCC-is_HIGH-VOLTAGE		NO	NO	NO	NO	NO	NO
VCC-is_NO-VOLTAGE		NO	NO	NO	NO	NO	NO
OK-CASE		NO	NO	NO	NO	NO	NO

Figure 2-15: Qualitative fault relation of the accelerator pedal circuit

- Building a device model, for basic electrical and electronic components, COMEDI users select the corresponding application component class models from the library. This operation requires knowledge concerning the electrical context the component is embedded in, i.e. know-how from the design process.
- Building a device model, COMEDI users explicitly model considerable operating modes, short circuits, and multiple faults, bringing in service experience.
- Modeling device reference operating modes, COMEDI users explicitly determine reference behavior of context component models. This operation requires expert knowledge concerning the intended device behavior.

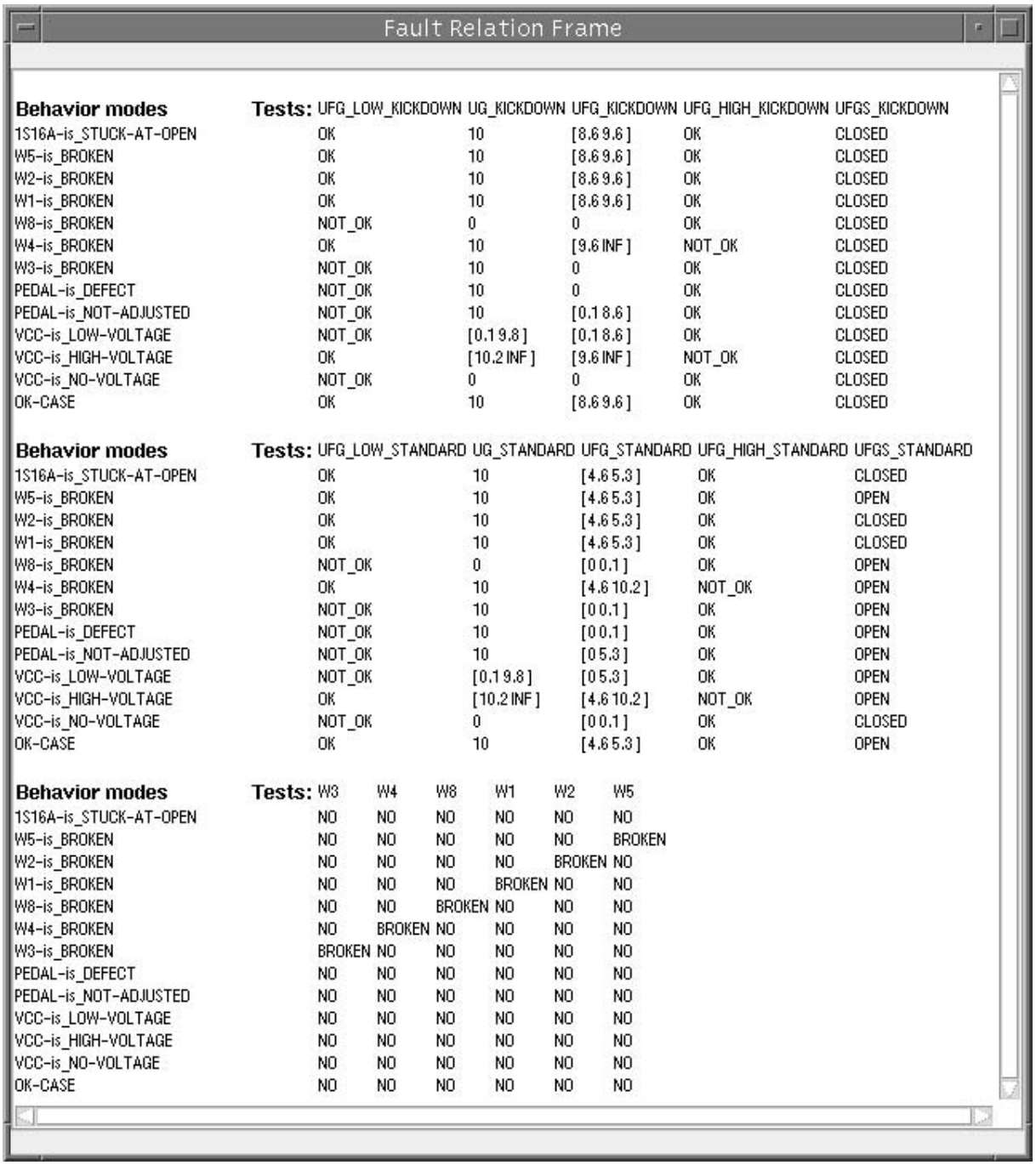


Figure 2-16: Quantitative fault relation of the accelerator pedal circuit

2.7 Generating Structured Fault Sets

An intelligent diagnostic system gives instructions concerning measurements and finally yields the fault causing the malfunctioning of a technical device. To be efficient in its search for the fault, the system has to instruct the maintenance engineer to perform a sequence of tests with the following properties: a sequence should only consist of tests really necessary to identify the fault and detecting a fault should be as cost-effective as possible. Unfortunately, determining such sequences of tests is a difficult and time consuming task if it has to be done manually.

Sequences of tests can be given by a decision tree. A widely used approach to determine a decision tree is to apply the entropy criteria ([Quinlan 1986], [Struss 1994], [Mauss et al. 1996], [Price et al. 1996b]). Since detecting a fault should be as cost-effective as possible, considering entropy is not sufficient. In addition, costs

of tests and dependencies between them have to be taken into account. To be able to adjust a diagnostic system to slight variations and progressive development of the technical device, methods to interactively specify parts of a decision tree should be placed at the user´s disposal.

In the system MAD, we focus on determining that most cost-effective sequence of tests. MAD fulfills this task by developing structured fault sets (in the following abbreviated SFS) based on information derived from a model of the device to be diagnosed (see Section 2.5.3). Deriving the cost-effectivest sequence of tests is achieved by taking into account information about cost and effort. Additionally, MAD offers operations for manually adjusting automatically developed SFS. A runtime version of the diagnostic system is then derived by compiling the SFS. A compiled SFS can be integrated in diagnostic systems, which process the SFS and presenting tests to be performed to the service technicians (compare diagnostic system DiaMon, as described in chapter II-3). Normally this is done by starting at the root node and successively checking the test clauses.

In the following, we describe how SFS can be generated from system models previously defined and analyzed (as described in the previous sections), how algorithms of diverse areas like machine learning, search, and algebra can be applied for generating SFS taking into account costs of tests, and how these algorithms can be integrated into an SFS-editor to support the interactive SFS generation by automatic methods and guarantee a sound and complete SFS based on the system model.

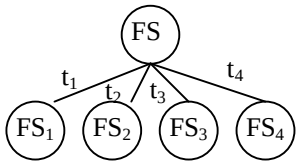


Figure 2-17: Sets, subsets, and test clauses. Test clause t_i is true for all faults of FS_i.

2.7.1 Some definitions

The basis for the generation of SFS is the fault relation (see Section 2.5.3). The SFS generation starts with combining all faults in the fault relation to one fault set called root set. This fault set FS_{all} is split into n disjoint fault sets FS_i being a full partition of FS_{all} (see Figure 2-17). In the following, FS_i are called adjacent fault sets. This splitting is repeated for every fault set as long as there are faults in the fault set that can be discriminated by some difference between their test values.

When generating SFS, we distinguish between “fault structure“ and “test clauses“. With “fault structure“ we mean the result of arranging faults to form fault sets and of arranging fault sets according to the subset relation. A “test clause” for FS_i declares criteria to discriminate the faults in FS_i from the faults in the sets adjacent to FS_i. This is achieved by specifying properties that hold for the faults in FS_i and do not hold for the faults in its adjacent sets, i.e. the fault structure and the test clauses have to “match”. Test clauses are incorporated as edge information into the SFS.

For the automatic algorithms present in Section 2.7.3, a test clause t_i for a fault set FS_i is a simple equation of the form $T = v$ with T being one of the tests of the fault relation (i.e. measurements, observations, and fault identification defined in the model, see Section 2.5.2) and v being a value of the domain of T (´univariate´ test clauses [Murthy 1997]). For all adjacent sets FS_i, the test clauses t_i contain the same T. In our case, v is a qualitative value of a finite set qualitatively describing the result of a test. Each qualitative value can be mapped to a quantitative interval. In the best case, all leaves of the structure only contain a single fault, i.e., all faults can be discriminated by tests given in the fault relation. For computing a SFS, there are, in principle, three possibilities:

- Neither fault structure nor test clauses are given but both are determined automatically.
- The fault structure is given. Test clauses matching the structure are determined automatically.
- Test clauses are given. The structure matching the test clauses is determined automatically.

The third possibility is the trivial case where the instructions for the diagnosis process are already given, namely which tests have to be performed to identify the faults. In this case, the fault sets are created according to the test clauses by simply considering the fault relation. Before elaborating on the first two possibilities in Section 2.7.3 and 2.7.4, respectively, we first define a cost model as a basis for considering costs when generating SFS.

2.7.2 Cost model and cost computation

To be able to take into account costs of test clauses when a sequence of test clauses for identifying a fault is computed, a cost model is defined by the user. As a basis of that cost model, an ordered sequence of basic working tasks can be introduced for each test occuring in the fault relation. These tasks have to be performed to obtain a value for a specific measurement, observation or fault identificaion of the technical system. For example, such tasks have to be performed to obtain a value of voltage *UG* of the accelerator pedal example. Examples for basic working tasks are “open the front bonnet”, “seperate the battery”, “pull plug *P1*”. Cost of a basic working task are represented by integers describing the effort necessary to perform the task. These cost model is added to the device model (see Figure 2-1).

Sequences of basic working tasks implicitly define dependencies between tests. These dependencies are considered, when the cost of a test clause *tc* are computed. This is done as follows: The cost *C(tc)* of a test clause *tc* is defined as the sum of costs of each test *T* occuring in *tc*, for univariate test clauses only one test occurs (see Section 2.7.4 for other tests). Thus, in this case *C(tc*) = *C(T)*, where *C(T)* is the cost of a test *T*. The cost of a test *T* occuring in a test clause *tc* is defined as the costs of the basic working tasks to be performed for *T*. The basic working tasks to be performed for *T* depend on the order of tests to be performed. Let tc_i and tc_{i+1} follow each other in the SFS, and each has only one test clause (i.e. both are univariate test clauses) with tests T_i and T_{i+1}; these tests may have sequences of basic working tasks s_i and s_{i+1}. Let *c-subseq* be a function, that computes a common subsequence of two sequences, starting from the first position of the sequences. The basic working tasks to be performed for T_{i+1} are those tasks that do not occur in *c-subseq*(T_i, T_{i+1}), i.e. those tasks that are not performed for T_i. Thus, the cost for test T_{i+1} is the sum of costs of the basic working tasks that are new in respect to test T_i. When the fault structure and tests are computed automatically (Section 2.7.3) or only the tests are computed (Section 2.7.4) the costs of the generated test clauses are computed dynamically taking the order of test clauses into account. Thus, dependencies between tests are considered.

2.7.3 Determining fault structure and tests

We present three algorithms to automatically compute a SFS. The first algorithm ($GSFS^{ID3}$) uses only information theoretic features like entropy but no test costs. This algorithm is based on ID3 [Quinlan 1986]. The second one ($GSFS^{ID3C}$) additionally considers costs of tests and follows [Norton 1989]. Both are heuristic algorithms that do not compute the optimal SFS. In an optimal SFS, the average cost to identify a fault are minimal given a probability distribution with $\sum p(f_i) = 1$, $f_i \in F$ and *F* being the set of all faults. This optimal solution can be computed by the algorithm $GSFS^{A^*}$ being an application of the well-known search algorithm A* (see e.g. [Russel et al. 1995]).

State Space. All three algorithms perform a search in a state space. However, each algorithm uses different cost criteria and management of alternative states. At the beginning, a set of faults is given by the user[1]. This set constitutes the start state. For its faults, a SFS is created. A state contains a set of fault sets being the current leaves of a growing SFS. Two states are equal if these leaves are equal. A successor of a state is generated by selecting a test *T* capable of partitioning one leaf fault set into at least two subsets. For all faults in a subset FS_i, the same value *v* is obtained for *T*. For each subset FS_i, a different value *v* is obtained for *T*. Thus, *T* = *v* is the test clause for FS_i. All successors of a state are generated by selecting each of those tests for each leaf. Each path in the state space represents a possible SFS (see Figure 2-18).

Heuristic search without considering costs. In $GSFS^{ID3}$, each state is evaluated by the entropy criteria, which takes into account the information gain of a selected measurement and the probability distribution of the faults. The evaluation formula for faults with equal probability is shown in Figure 2-19. For other probability distributions, see [Quinlan 1986] or [Russel et al. 1995].

The information gain of a test *T* (i.e., for one state) is defined as the difference between the information requirement before and after partitioning a fault set according to *T*. In $GSFS^{ID3}$, each successor state is evaluated with the formula given in Figure 2-19.

[1] This can be done by selecting *one* fault set, that will be the root of the SFS to be created, or a number of fault sets, that will be the leaves of SFS to be created. For simplicity, we only focus on the first case.

The successor with highest information gain is selected (i.e., the information gain is maximized) as the *only* successor state. Thus, all alternatives are discarded.

$GSFS^{ID3}$ is based on ID3. ID3 can be applied because there are only finite tests, i.e. the domains tests used here are finite, discrete sets of qualitative values. This algorithm computes the structure with the lowest depth in average [Quinlan 1986], yet distinct costs of tests are not handled by $GSFS^{ID3}$. However, probability distributions of faults can be considered. Moreover, because only one successor is selected, efficient generation of SFS is guaranteed.

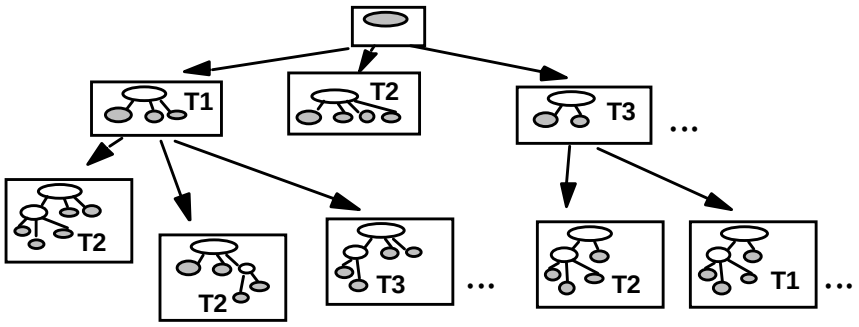


Figure 2-18: Searching a SFS in a state space. Circles denote fault sets, rectangles search states, T_j last selected tests to generate that state. For clearness, not only the leaves (which are gray) of the current SFS are shown but the whole SFS. In each path only distinct tests occur, in different paths same tests may occur.

Another property of ID3 is its generalization capability. This property of ID3 is not of interest in our case, as a decision tree is created by using the fault relation which contains all sets of test values that can occur in the technical system according to the model.

$$IG(FS_i, T) = \log|FS_i| - \left(\frac{1}{|FS_i|} \cdot \sum_{j=1}^{k} |FS_i| \cdot \log |FS_{ij}| \right)$$

Figure 2-19: Computing the information gain (IG) with equally distributed faults. FS_i denotes a fault set which is partitioned by T into k subsets FS_{ij}, with k being the domain size of T.

Heuristic search considering costs. In $GSFS^{ID3C}$, each state is evaluated by weighting the information gain of a test by its cost (see Figure 2-20). Thus, the information gain per cost unit is considered.

This algorithm has the same properties as $GSFS^{ID3}$ (being effective, using probability distributions, discarding alternative successors). Thus, the algorithm selects the most cost-effective test at each local point in the state space. Experiments showed that in most cases $GSFS^{ID3C}$ yields sufficiently good results.

$$IGC(FS_i, T) = \frac{IG(FS_i, T)}{C(T)}$$

Figure 2-20: Weighting the information gain of test T with costs of T.

Optimal search. $GSFS^{A^*}$, being an application of the well-known search algorithm A*, computes a SFS with minimal average cost for identifying faults. To apply A*, a search model must be given by specifing the definition of a state, a start state, a goal state, a successor function, a cost function *g* to evaluate a state, and a heuristic function *h* estimating the cost between a state and the goal state. The sum of *g* and *h*, must increase monotonically for successive states. A* stores all states according to that sum in a so called OPEN list. The state with the lowest value the sum of *g* and *h*, is expanded, i.e., its successors are created by the successor function. Thus, A* computes the optimal solution. Furthermore, if *h* underestimates the real costs, A* guarantees to expand only necessary states.

In $GSFS^{A^*}$, a state is given by a set of fault sets representing the current leaves. Successor states are created by applying tests to these leaves, as described in Figure 2-18. The goal state consists of fault sets containing faults with equal fault definitions, i.e. those faults cannot be discriminated. The start set is defined by the union of all

faults. Each state is evaluated by the function *g* defined as the sum of the diagnostic effort for each fault *f*. The diagnostic effort of a fault *f* is the sum of all test costs (see Section 2.7.2) on the path between the current leaf fault set containing *f* and the root fault set. To guide the search, the heuristic function *h* is used.[2] To demonstrate that *h* never overestimates real cost for reaching the goal, a certain leaf *b* is considered. There is a set of available tests not yet used on the path between *b* and the root. These tests show corresponding costs and domains. *kmax* is the maximum size of these domains. Available tests can be utilized to recursively split *b* into subsets of faults until all subsets contain faults that cannot be discriminated. Hence, these tests allow the generation of a subtree below *b*. To underestimate fault identification cost in a cost-optimized subtree, we consider an impossible subtree showing the following properties:

1. All available tests split fault sets into *kmax* subsets containing the same number of faults, i.e. we consider a balanced subtree.
2. The depth of the subtree is $\lfloor \log_{kmax}(|b|) \rfloor$, $\lfloor \ \rfloor$ (where is the floor operation).
3. The test first performed for fault identification, is the cheapest available test. The second test performed is the is the second cheapest test, and so on.

For each fault in *b*, cost of fault identification in the impossible subtree is:

$$\sum_{i=1}^{\lfloor \log_{kmax}(|b|) \rfloor} C(T_i) \quad \textit{with } T_i \textit{ is i-th cheapest unused } T$$

In Figure 2-21, the definitions of *g* and *h* are given. *g* monotonically increases for successive states because an additional test is selected to determine a successor state. *h* ensures that the real costs necessary for reaching the goal are never overestimated. Besides this evaluation of states, GSFS$^{A^*}$ can reactivate the most cost-effecive alternative state *s* if a selected path yields to states being more expensive than *s*. This yields to an optimal fault tree.

$$g(state) = \sum_{i=1}^{n} de(f_i), \quad with \quad de(f_i) = p(f_i) \cdot \sum_{T \in path(f_i)} C(T)$$

$$h(state) = \sum_{b \in leaves(state)} h(b), with \ h(b) = |b| \cdot \sum_{i=1}^{\lfloor \log_{kmax}(|b|) \rfloor} C(T_i)$$

Figure 2-21: Cost function *g*() and heuristic function *h*() for computing an optimal SFS with *n* faults f_i and probability $p(f_i)$. *de* denotes the diagnostic effort. T_i is the i-th cheapest unused T.

Example. In Figure 2-22, SFS based on the same fault relation but generated by the three algorithms GSFS^{ID3}, GSFS^{ID3C}, and GSFS$^{A^*}$ are presented. GSFS^{ID3} uses test *T4* with the best information gain (all faults can be diagnosed in one step) but ignores high test cost. Because GSFS^{ID3C} takes into account costs, it selects *T5* being a cost-effective test with good information theoretical properties. Since no alternatives are stored, two subtrees are necessary for discriminating all faults. To reduce test costs to a minimum, alternative SFS are considered in GSFS$^{A^*}$. Thus, this algorithm finds the SFS with the lowest test cost in average.

2.7.4 Determining test clauses for given fault structures

We now elaborate on the case that only a fault structure is specified where as the test clauses matching that fault structure have to be determined automatically. The fault structure can be given by the user or can be given by the aggregate structure of the device. In the last case the fault structure is generated such that subsets of faults correspond to faults of the same physical component.

Let us first repeat the requirements a test matching a given fault structure has to meet. A test clause belonging to a fault set FS has to cover every fault in FS, i.e., it has to be complete. Furthermore, it must not cover any of the faults in the sets adjacent to FS, i.e., it has to be sound. In the following, a test clause meeting these requirements is called discriminative test clause.

---

[2] In the following, the heuristic function for faults with equal probability is discussed. For other probability distributions, see [Faure et al. 1999].

The simplest way to determine a discriminative test clause is to take the definitions of all the faults in FS from the fault relation and combine them disjunctively. However, such a test clause considers every possible test whereas only some of them are really necessary to discriminate FS from all of its adjacent fault sets. As we claim to take into account costs, we have to find a more cost-effective discriminative test clause - or even the most cost-effective one at all.

We develop an algorithm that guarantees to find the most cost-effective discriminative test clause. Before we present the algorithm, we first have to extend the definition of costs for test clauses. Up to now, we only determined test clauses using only one test. However, with an arbitrary structure given, measuring only one test might no longer be sufficient to discriminate adjacent fault sets. Hence, we have to define costs for test clauses using any number of tests.

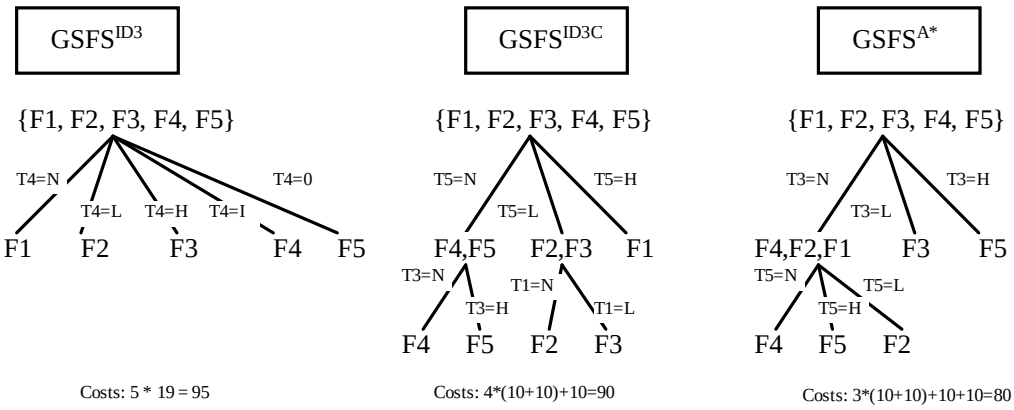


Figure 2-22: Three different SFS (the underlying fault relation can be derived from the SFS) and a cost model of: $T_1 = T_3 = T_5 = 10$, $T_4 = 19$.

Computation of costs for tests. The cost of a discriminative test clause for a fault set *FS* only depend on the tests used in the test clause and in the test clauses that have been checked before during the diagnosis process.

The total costs *C(tc)* of a discriminating test clause *tc* are defined as the sum of $C(T_i)$ where T_i being a test in the set of tests to be performed according to the sequence of tests that have to be checked during the diagnosis process until FS is reached (including the test for FS itself), and $C(T_i)$ being the cost for test T_i (see Section 2.7.2). Note that

- every test has to be performed only once, although it might appear more than once in that sequence of tests;
- costs do not depend on the number of values assigned to tests;
- the definition of cost of a test given in Section 2.7.2 still holds, i.e. cost of a test are computed considering basic working tasks.

Determining the shortest most cost-effective discriminative test. Our algorithm for determining the most cost-effective discriminative test for a fault set consists of three steps:

1. According to the definition given above, we determine the total cost for every combination of tests to be used in a test clause. The combinations are arranged by increasing costs.
2. Starting with the most cost-effective combination, we check these combinations for the existence of a discriminative test clause until such a combination is found. All discriminative test clauses that can be determined with this combination are guaranteed to be the most cost-effective.
3. For this combination, we determine the shortest of the most cost-effective discriminative test clauses.

In the following, we elaborate on steps 2 (existence check, see Figure 2-23) and 3 (determining the shortest discriminative test clause, see Figure 2-24).

Let
- $T_{i_1} \ldots T_{i_n}$ be the combination of test clauses to be checked,
- v_{ri} be the value for fault f_r at test clause T_i,
- $v_{ri_1} \ldots v_{ri_n}$ be a partial fault definition for f_r only consisting of the values at test clauses $T_{i_1} \ldots T_{i_n}$,
- FS be the fault set that the test clause is to be determined for,
- AFS be the union of all the adjacent fault sets of FS.

A discriminative test clause for $T_{i_1} \ldots T_{i_n}$ exists iff
$\{(v_{ri_1} \ldots v_{ri_n}) \mid f_{r \in FS}\} \cap \{(v_{r'i_1} \ldots v_{r'i_n}) \mid f_{r' \in AFS}\} = \{\}$.
If this condition holds, there is at least one discriminative test clause:
the disjunction of all elements in $\{(v_{ri_1} \ldots v_{ri_n}) \mid f_{r \in FS}\}$.

Figure 2-23: Existence check algorithm

As we want to find a shorter discriminative test clause, we perform step 3 in addition. To achieve this, we search for partial test clauses (denoted p_m in the algorithm below), i.e., test clauses only containing a subset of k tests out of $T_{i_1} \ldots T_{i_n}$ and being sound and complete for a subset of the faults in FS. As we are looking for a test clause that is as short as possible, we have to find partial test clauses containing the least possible tests. Therefore, we check the smallest subsets first, starting with $k = 1$. Moreover, a partial test clause has to cover as many faults as possible. Hence, for every k, we determine a covering function specifying which faults out of the subset of FS can be covered by which partial test clauses. By determining the prime implicants of the covering function, we obtain minimal sets of partial test clauses with each set covering the whole subset. The shortest of these prime implicants (the one consisting of the fewest partial test clauses) is selected. The search for partial test clauses stops when all of the faults of FS are covered. Disjunctive combination of the selected prime implicants then yields the shortest discriminative test clause. The algorithm for step 3 is as follows:

Set faults_to_cover to FS.
Initialize *resulting_test_clause*.
Set k to 1.
Repeat
  For $m = 1 \ldots \binom{n}{k}$ [3]:

    Select k tests out of $T_{i_1} \ldots T_{i_n}$
    and let $T_{i_1} \ldots T_{i_k}$ be the selected tests.
    Set p_m to the disjunction of all conjunctions
    $(T_{j_1} = v_{rj_1} \wedge \ldots \wedge T_{j_k} = v_{rj_k})$
    with $f_r \in FS$
    and $(v_{rj_1} \ldots v_{rj_k}) \notin \{(v_{r'j_1} \ldots v_{r'j_k}) \mid f_{r' \in AFS}\}$.
  Determine the covering function (see below).
  Determine the disjunction of prime implicants of the covering function (see below).
  Select the shortest prime implicant (see below) and add it disjunctively to
  resulting_test_clause.
  Delete the faults covered by the prime implicant from *faults_to_cover*.
  Set k to k+1.
until *faults_to_cover* = {}.

Figure 2-24: Determining the shortest discriminative test algorithm

- **Determination of the covering function.** For every fault in the subset of FS covered by the disjunction of p_m for $m = 1..., \binom{n}{k}$ the covering function denotes which p_m can be used to cover the fault. The function is a conjunction of disjunctions with every disjunction consisting of at least one p_m.

  Example. Let $\{ f_1, f_2, f_3 \}$ be the subset of FS covered by the disjunction of p_m for m=1... $\binom{n}{k}$.

[3] number of possible selections of k elements out of n elements

Then, the covering function $p_4 \wedge (p_1 \vee p_2) \wedge (p_1 \vee p_2 \vee p_3)$ can be read as follows: to cover f_1, *p4* is needed. To cover f_2, *p1* or *p2* can be used. To cover f_3, *p1*, *p2*, or *p3* can be used.

- **Determination of the prime implicants of the covering function.** Since the covering function only contains variables without negations, the prime implicants are simply delivered by using

  $p_i \wedge (p_j \vee p_k) = p_i \wedge p_j \vee p_i \wedge p_k$ and $p_i \vee p_i \wedge p_j = p_i$.

  Determining the disjunction of the prime implicants of the covering function in the example above yields

  $p_4 \wedge p_1 \vee p_4 \wedge p_2$.

  Either of the prime implicants $p_4 \wedge p_1$ and $p_4 \wedge p_2$ covers the whole subset of *FS*.

  The meaning of prime implicant $p_4 \wedge p_1$ is as follows: to cover all of the faults of the subset, both p_4 and p_1 have to be selected. However, in the *resulting_test_clause*, the partial tests p_4 and p_1 have to be combined disjunctively, because the meaning of a test for a fault set *FS* is: one of the faults of *FS* is present iff one of the partial tests is true.

- **Selection of the shortest prime implicant**. The prime implicant minimizing $\sum_{p_m \in \text{prime implicants}} \text{length}(p_m)$ with *length*(p_m) := number of disjuncts in p_m is selected.

To clarify step 3 of the algorithm (see Figure 2-25), we make use of a small example.

Example. Let { f_1, f_2, f_3, f_4, f_5 } be the fault set *FS* for which the discriminative test clause is to be determined. Let { f_6, f_7, f_8, f_9 } be the union of the faults in the sets adjacent to *FS*.

Assume that we have already performed steps 1 and 2. We have found the combination of tests $M_1\ M_3\ M_4$ to be the most cost-effective one, as it is the first combination in the arrangement of combinations obtained by step 1 with a positive existence check.

The fault definitions reduced to the relevant tests are given Table 2-8 and a stepwise processing of step 3 is given in Figure 2-25.

	M1	M3	M4
f_1	a	d	f
f_2	b	d	e
f_3	a	c	f
f_4	b	d	d
f_5	b	c	e
f_6	b	c	b
f_7	b	b	e
f_8	d	d	e
f_9	b	d	b

Table 2-8: The fault relation related to example in Figure 2-25.

```
Step 3 yields:
faults_to_cover := { f1, f2, f3, f4, f5 }
resulting_test_clause := empty disjunction
k := 1
        selected test: M1
        p1:= M1 = a
        selected test: M3
        p2:= {}
        selected test: M4
        p3:= (M4 = f) ∨ (M4 = d)
        subset of FS covered by p1 ∨ p2 ∨ p3: { f1, f3, f4 }
        covering function for the subset: (p1 ∨ p3) ∧ (p1 ∨ p3) ∧ p3
        disjunction of prime implicants of the covering function: p3
        shortest prime implicant: p3
        resulting_test_clause := (M4 = f) ∨ (M4 = d)
        faults_to_cover := { f2, f5 }
k := 2
        selected combination of tests: M1 M3
        p1:= {}
        selected combination of tests: M1 M4
        p2:= {}
        selected combination of tests: M3 M4
        p3:= (M3 = c  ∧  M4 = e)
        subset of FS covered by p1 ∨ p2 ∨ p3: { f5 }
        covering function for the subset: p3
        disjunction of prime implicants of the covering function: p3
        shortest prime implicant: p3
        resulting_test_clause := (M4 = f) ∨ (M4 = d) ∨ (M3 = c ∧ M4 = e)
        faults_to_cover := { f2 }
k := 3
        selected combination of tests: M1 M3 M4
        p1:= (M1 = b ∧ M4 = d ∧ M4 = e)
        subset of FS covered by: { f2 }
        covering function for the subset: p3
        disjunction of prime implicants of the covering function: p1
        shortest prime implicant: p1
        resulting_test_clause :=
        (M4 = f) ∨ (M4 = d) ∨ (M3 = c ∧ M4 = e) ∨ (M1 = b ∧ M3 = d ∧ M4 = e)
        faults_to_cover := { }
```

Figure 2-25: Determining the shortest discriminative test clause (example of step 3 of the algorithm).

2.7.5 The SFS-Editor

In Sections 2.7.3 and 2.7.4, we presented algorithms for automatically determining fault structure as well as tests clauses. These algorithms are implemented in an SFS-editor, being a part of the system MAD. The SFS-editor additionally places at the user‘s disposal methods to manually adjust the fault structure of SFS that have been generated automatically. A combination of automatic and manual methods can be applied in an interactive process to generate SFS.

The reader may wonder why those automatical methods are not sufficient. There are, of course, situations in which an adaption of SFS is achieved simply by adapting the COMEDI model resulting in recomputing the fault relation and automatic regeneration of the respective SFS. Examples are changes in device design or metering point positioning. On the other hand, several reasons can be stated for the necessitiy of manually changing SFS. First, what can be modeled with COMEDI up to now is not entirely sufficient. We only cover the majority of electrical and electronic circuits. Mechanics and hydraulics are totally omitted in the present system, but can cause malfunctioning, too. Second, maintenance engineers do have useful experience that should be considered in a diagnosis system. For example, inserting and pulling a plug several times may clear malfunctioning by

removing corrosion. Moreover, maintenance engineers often have knowledge about the frequency of specific faults. This kind of information can in fact be considered when building a model using COMEDI, but it is presumed that knowledge about the frequency of faults is available to the user of the SFS-editor, too. The third reason is that aspects of the surrounding have to be considered. This refers to where the technical device has been employed as well as where maintenance takes place. For example, a moist surrounding will probably cause faults different from those caused by a dry and dusty one. Finally, having applied one of the automatic methods that does not guarantee optimization concerning costs, is another reason. In this case, the automatic method has done the major work, and then it is up to the user to realize some improvement. For all the reasons mentioned above, it is useful or even necessary to adapt the structure of the SFS that have been developed automatically.

The interactive process of generating SFS is organized as follows. For each fault set, the user can choose among automatic and manual operations to split that set. When applying one of the automatic methods, the user can specify the depth of the SFS to be generated, i.e., the whole SFS can be generated automatically as well as only partial structures. After having changed the arrangement, the user can invoke the operation of automatically adapting the test clauses (Section 2.7.4) to see how his structural changes have affected the test clauses.

The manual operations offered by the SFS-editor are

- inserting a fault into an arbitrary fault set (i.e., the user can place the fault into a set regardless of the test clauses)
- classifying a (possibly new) fault (i.e., the fault is automatically placed into the fault set it belongs to according to the test clauses)
- deleting a fault from a fault set
- transporting a fault from a fault set into a different one (i.e., deleting and inserting performed in one step)
- merging two fault sets FS1 and FS2 (i.e., creating a new fault set FS3 being the union of FS1 and FS2)
- dividing a fault set (i.e., the fault set is partitioned into disjoint subsets according to the user‘s default)
- deleting a fault set

2.7.6 Properties of the algorithms

To evaluate the presented algorithms, we applied them to diverse circuits being part of the electrics of fork lift trucks. A circuit has between 10 and 50 faults and up to 30 tests in distinct operation states. The runtime was as expected: $GSFS^{A^*}$ took multiple times of the runtime of the other algorithms (e.g., up to 400 times higher than $GSFS^{ID3}$). This difference depends on the number of alternatives to be considered. This number internally depends on the domains of the tests and the discrimination capability of tests. Thus, it is highly domain specific. Since we want to generate the most cost-effective SFS, we integrate $GSFS^{A^*}$ into our system although it is very time consuming. By the way, high construction times are acceptable because the generation of SFS is done before the diagnosis process. We decided to integrate $GSFS^{ID3C}$ in addition because this non-optimal but more efficient algorithm yields sufficiently good results. If all tests are equally expensive, $GSFS^{ID3}$, being the most efficient algorithm, should be used. Therefore, $GSFS^{ID3}$ is integrated into our system, too. Thus, the user can decide whether a quick but non-optimal or an optimal but time-consuming result should be created.

Because the SFS we constructed is based on a complete fault relation where all faults with all their effects are included (due to the model based approach described in Section 2.3), we are not interested in the generalization property of the SFS. Thus, our optimality criteria does not depend on the generalization accuracy but on the average cost necessary to identify a fault.

The automatic determination of tests and structure is guaranteed to be consistent with the model because it is grounded on the fault relation. This property even holds after manual changes of the structure, because the tests for a given structure are determined based on the fault relation.

The manufacturer STILL develops its SFS with a simple editor where the arrangements of faults as well as all tests have to be typed in manually. Thus, either incorrect test and inefficient test sequences can be defined. In contrast to this, our algorithms guarantee soundness and efficiency according to test cost. Moreover, manual definition of a SFS is a very time consuming task. For example, this takes several months for the electrics of a fork lift truck. In contrast to this, by our modelling approach and the developed algorithms a SFS is obtained in substantially less time (some weeks).

2.7.7 Generated structured fault sets for the accelerator pedal circuit

To compute a cost optimal SFS by using a given fault relation, first a cost model must be given. In Figure 2-26 for each test of the accelerator pedal circuit for simplicity a number describing the cost is given. With these cost model and the fault relation presented in Figure 2-15 the SFS in Figure 2-27, 2-28, and 2-29 are automatically computed. In Figure 2-27 the result is computed with $GSFS^{ID3}$. In Figure 2-28 the result is computed with $GSFS^{ID3C}$ and in Figure 2-29 the optimal SFS is computed with $GSFS^{A^*}$. For each fault set the fault names and in the last row the test clause are shown. For each fault in the fault set these test clauses are satisfied.

test / operating mode	cost	test / operating mode	cost
UFG_LOW / KICKDOWN	6	UFG_HIGH / STANDARD	1
UG / KICKDOWN	8	UFGS / STANDARD	1
UFG / KICKDOWN	7	W3	10
UFG_HIGH / KICKDOWN	1	W4	10
UFGS / KICKDOWN	1	W8	10
UFG_LOW / STANDARD	5	W1	10
UG / STANDARD	6	W2	10
UFG / STANDARD	9	W5	10

Figure 2-26: A cost model for each test in each operating mode of the accelerator pedal circuit

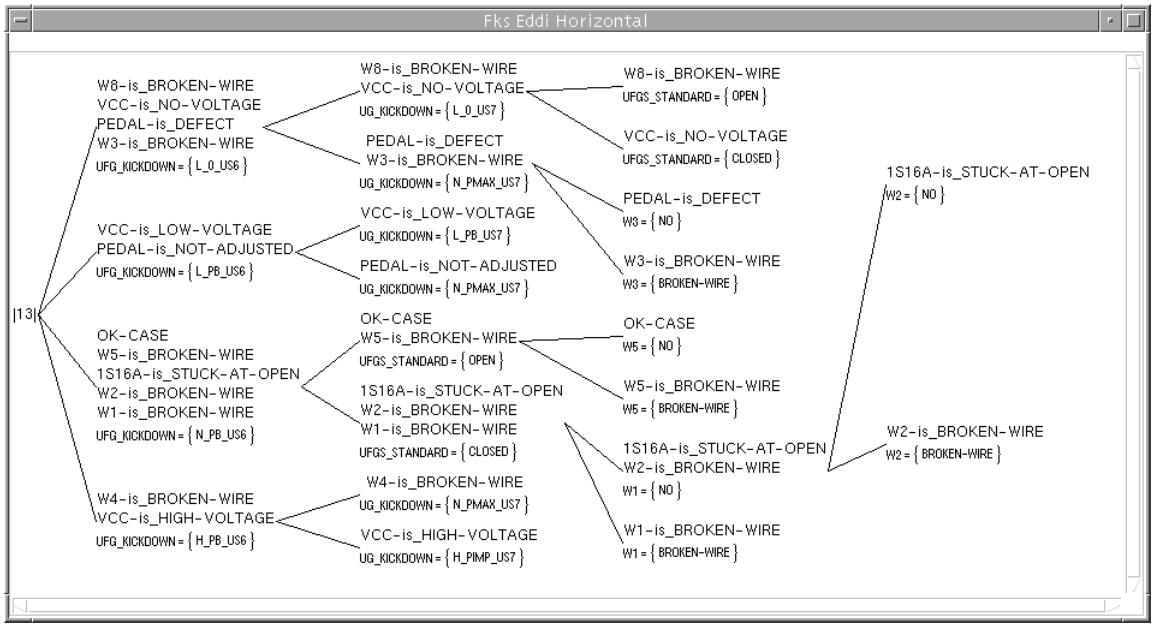


Figure 2-27: SFS of accelerator pedal circuit computed by $GSFS^{ID3}$ the average fault identification cost is 19.38.

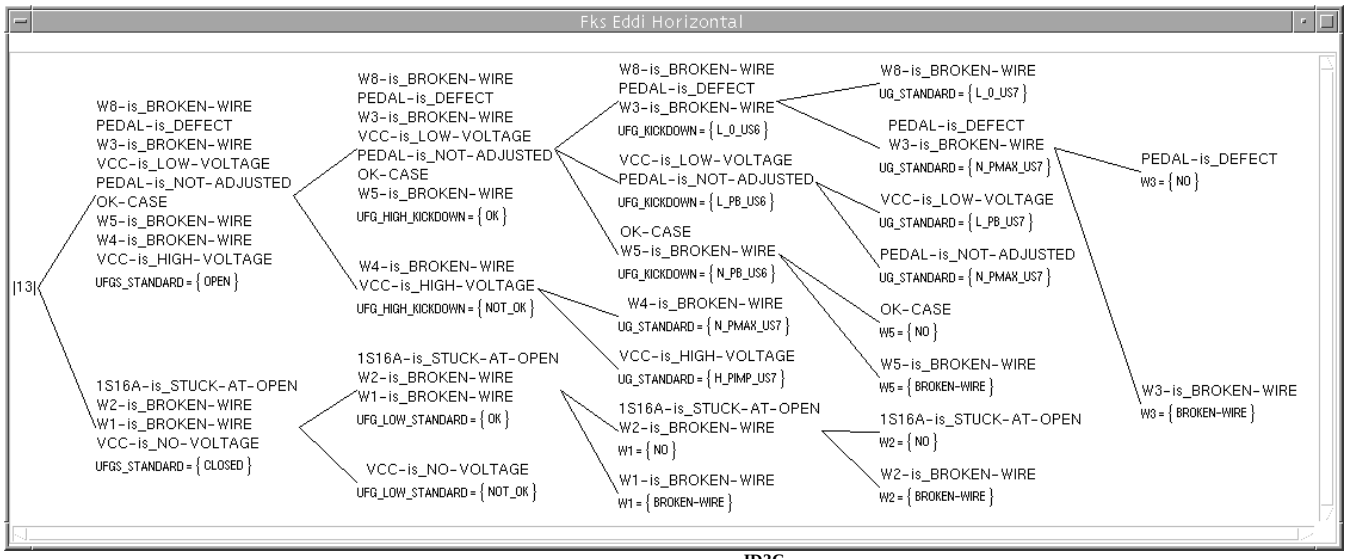


Figure 2-28: SFS of accelerator pedal circuit computed by GSFS^{ID3C} the average fault identification cost is 17.15.

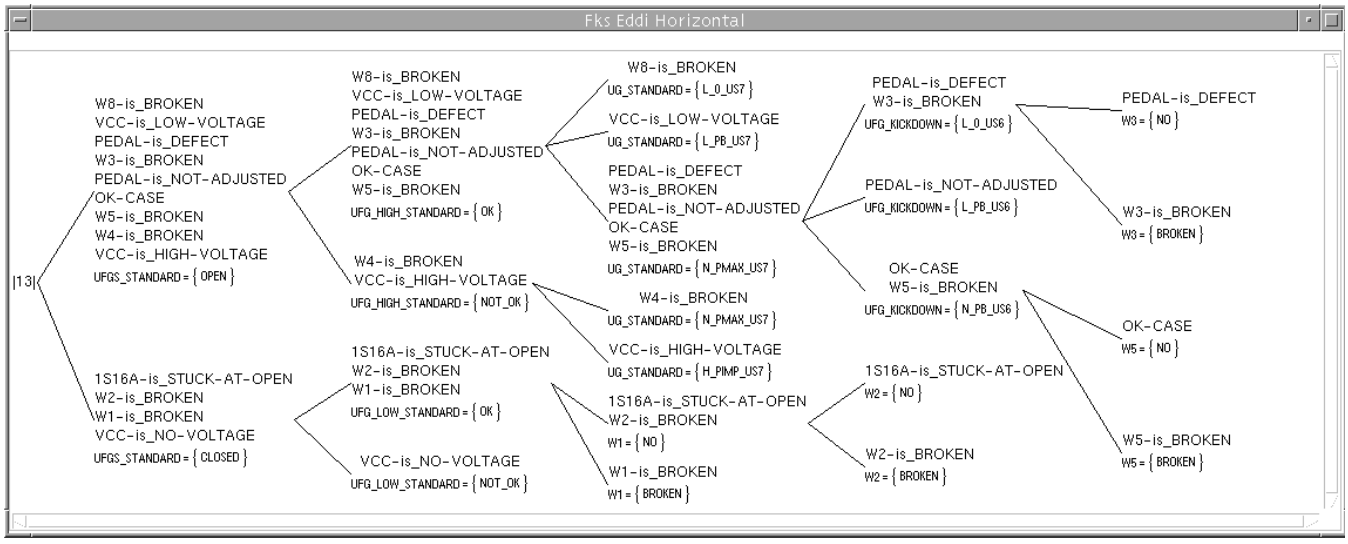


Figure 2-29: SFS of accelerator pedal circuit computed by GSFS^{A*} the average fault identification cost is 17.00.

3 Modellbasierte Diagnose mit DiaMon

Roman Cunis – ServiceXpert GmbH Hamburg

3.1 Hintergrund

Das Diagnosesystem DiaMon wurde von der Mikroelektronik Anwendungszentrum Hamburg GmbH (kurz: MAZ) in den Jahren 1993–1996 zusammen mit der Firma STILL entwickelt. Ausgehend von einem Maschinenüberwachungssystem, das im MAZ bereits in der Entwicklung war, wurde ein interaktives System zur Diagnose von Maschinen beliebiger Art (speziell: Gabelstapler) geschaffen. Es führt einen Servicetechniker durch eine Diagnosesitzung und berücksichtigt dabei aktuelle Daten, die aus dem Gerät ausgelesen werden. Auf diese Weise wurde es möglich, kompetente, zielgerichtete und weitgehend fehlerfreie Diagnose an Geräten zu ermöglichen, die durch einen hohen Steuerungsanteil für einen Servicetechniker mit herkömmlichen Methoden zunehmend unüberschaubarer wurden.

Das MAZ ist eine Firma im Besitz der Stadt Hamburg, deren Hauptzweck es ist, neue Technologien in enger Kooperation mit den Hamburger Universitäten und interessierten Anwendungspartnern zu neuen Produkten zu entwickeln. Erweisen sich diese Produkte als vielversprechend für einen erfolgreichen Markteintritt, wird die Arbeitsgruppe, in der das Produkt entstanden ist, ausgegründet. Es entsteht eine neue Firma, die an einen interessierten Investor verkauft wird.

Ausgehend von dem Erfolg, den das System bei der Firma STILL hatte, wurde am MAZ 1996 beschlossen, das Diagnosesystem zu einem allgemeinen Produkt weiterzuentwickeln. Die Ausgründung erfolgte im Jahre 1998: Es entstand die ServiceXpert GmbH, die das Produkt DiaMon in ein umfassendes Portfolio zur Unterstützung des technischen Kundendiensts einbettete.

Das Diagnosesystem DiaMon unterstützt einen Servicetechniker bei der Diagnose auf der Basis von Wissen über das Gerät, dessen mögliche Fehler und dazu anwendbare Service- und Reparaturschritte. Dieses Wissen wird in einer sogenannten Diagnose-Datenbank in Form von erweiterten Fehlerbäumen abgelegt. Die Erstellung der Diagnose-Datenbank erfolgt manuell von erfahrenen Servicemitarbeitern und technischen Redakteuren.

Der Anwender STILL war daran interessiert, die Erstellung der Fehlerbäume, die immer auch ein hohes Maß von Strukturwissen über das Gerät verkörpern, zu erleichtern und bis zu einem gewissen Grad zu automatisieren, in dem in der technischen Entwicklung vorhandenes Struktur- und Modellwissen automatisch zu Fehlerbäumen aufbereitet wird. Mit dieser Zielsetzung nahmen STILL, MAZ und die Universität Hamburg am Projekt INDIA teil. Die im MAZ begonnenen Arbeiten wurden nach der Ausgründung der Arbeitsgruppe von der ServiceXpert GmbH fortgeführt.

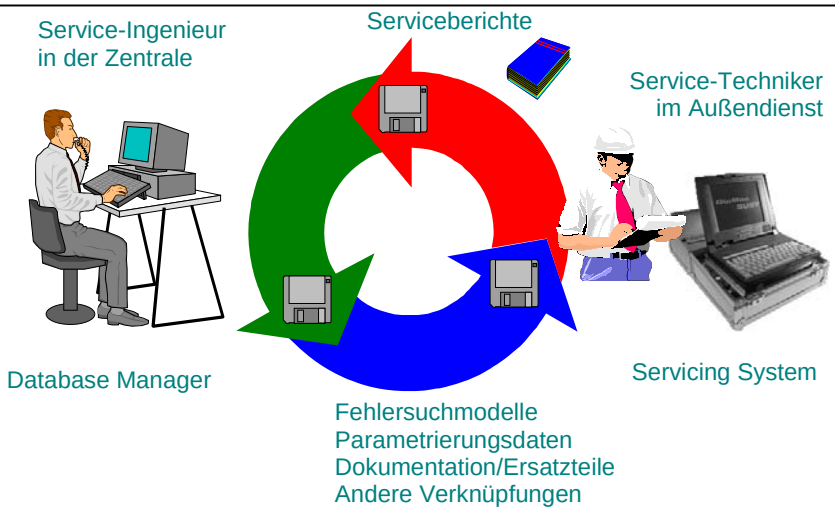


Abbildung 3-1: DiaMon-Einsatzszenario

MAZ bzw. ServiceXpert GmbH verfolgten im Rahmen des Projekts die folgenden Ziele:

1. Qualitative Modellbildung und Analyse: Für die modellbasierte Analyse von Schaltungen im speziellen und vergleichbar strukturierten Gerätekomponenten im allgemeinen wurde am MAZ eine Modellierungssprache samt zugehörigem Analysewerkzeug konzipiert und prototypisch realisiert (erster Anwendungsprototyp). Ziel dabei war es, aus Sicht des Diagnosesystems und des Anwenders in dieser Projektphase Anforderungen und Lösungsmöglichkeiten herauszuarbeiten, die geeignet sind, basierend auf Struktur- und Verhaltensmodellen letztlich Fehlerbäume zu erzeugen. Diese Arbeiten wurden erfolgreich durchgeführt, allerdings zeigten sich in der gewählten Modellierungsform auch einige Schwächen. Diese Ergebnisse, die positiven Ansätze wie die zu vermeidenden Schwächen, dienten als Input für die Arbeiten des LKI (siehe Kapitel II-2).
2. Übernahme von Fehlerklassenstrukturen in DiaMon: Der zentrale Teil der Entwicklungsarbeiten hatte zum Ziel, die vom modellbasierten Analysesystem des LKI erzeugten Fehlerklassenstrukturen in Fehlerbäume umzuwandeln und so für das Diagnosesystem verwendbar zu machen. Die ursprünglich angestrebte „tiefe“ Integration des Modelleditors in das DiaMon-System wurde aus produktplanerischen Gründen zugunsten einer reinen Importschnittstelle zum LKI-seitig geschaffenen Modelleditor aufgegeben. Details des Diagnosesystems und des Übernahmevorgangs werden in den Abschnitten 3.2 und 3.1 dargestellt.
3. Integration des Diagnosesystems in den Produktionsablauf bzw. mit Produktdaten: Die Integrationsarbeiten umfaßten Untersuchungen zu verschiedenen Schnittstellen des modellbasierten und fehlerbaumbasierten Diagnosesystems in den Produktionsablauf beim Anwender. Dabei ging es zum einen um eine softwaretechnische Verknüpfung der Software mit anderen serviceunterstützenden Programmen beim Anwender (wie z.B. Kundendienstinformations- und –steuerungssysteme (KISS) oder Online-Servicedokumentation und –Ersatzteilkataloge). Zum anderen wurden Möglichkeiten untersucht, vorhandene Produktdaten zu importieren oder aber zumindest das entstehende Diagnosewissen mit vorhandenen Dokumentationen zu verknüpfen. Details zu diesem Arbeitspaket finden sich in Abschnitt 3.4.

3.2 Das Diagnosesystem DiaMon

Das Diagnosesystem DiaMon umfaßt drei Hauptmodule:

- Die Diagnose dient der interaktiven geführten Fehlersuche am Gerät. Dies beinhaltet die Aufnahme von Zustandsdaten über das Gerät entweder automatisch durch Zugriff auf dessen elektronische Steuerung oder manuell durch Benutzereingabe, die Auswertung dieser Daten im Fehlerbaum und schließlich die Anweisung von zur Reparatur eines erkannten Fehlers notwendigen Arbeitsschritten.

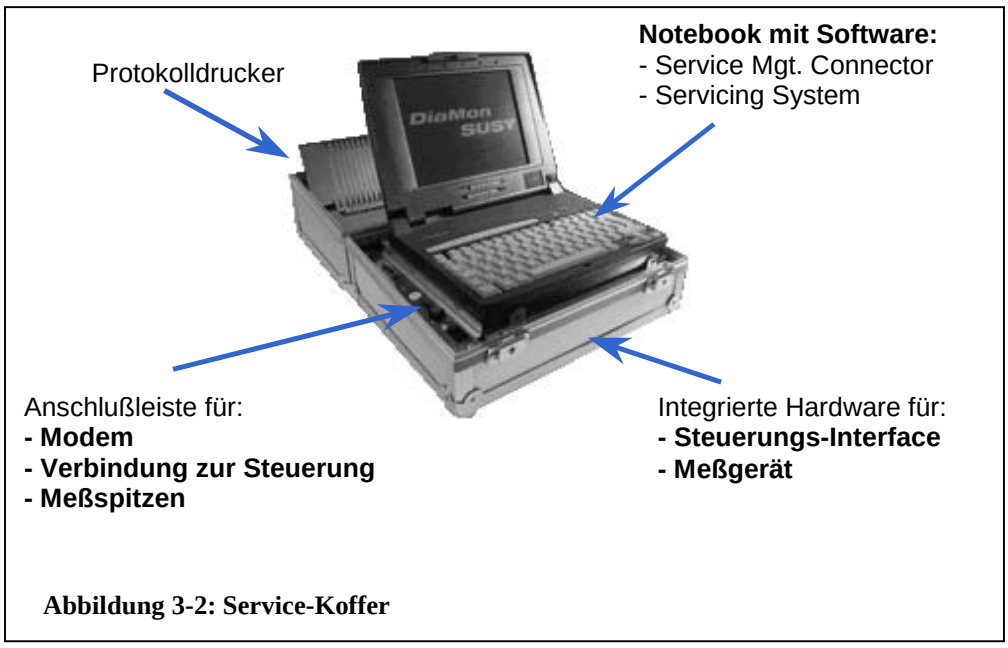


Abbildung 3-2: Service-Koffer

- Das Device Setup dient der Anzeige und Einstellung von Geräteparametern. Diese werden automatisch aus der Steuerung des Geräts ausgelesen und in geeigneter Form zur Anzeige gebracht. Der Servicetechniker kann Parameter ändern, Parametersätze auf Konsistenz prüfen und schließlich Änderungen wieder in die Steuerung des Geräts zurückschreiben.
- Das Device Monitoring dient der Anzeige von Meßwertverläufen in geeigneter grafischer Form.

Das Diagnosesystem geht von einem Einsatzszenario aus (vgl. Abbilung 3-1), das sich wie folgt darstellen läßt:

4. Ein Serviceingenieur in der Zentrale des Geräte- oder Maschinenherstellers erstellt das Diagnosewissen mit der DiaMon-Komponente Database Manager. Das Diagnosewissen umfaßt die eigentlichen Fehlersuchmodelle in Form von Fehlerbäumen, Daten für die interaktive Parametrierung des Geräts (Device Setup), Dokumentation zu durchzuführenden Reparaturschritten und dabei zu verwendenden Ersatzteilen, sowie schließlich beliebige Verknüpfungen zu anderen Programmen und Dokumenten, die dem Servicetechniker auf seinem Computer zur Verfügung stehen. Dieses Diagnosewissen wird dem Techniker komplett bereitgestellt. Als Übermittlungsmedien dienen CD-ROMs oder aber auch beliebige Fernübertragungstechniken.
5. Mit der Einsatzkomponente von DiaMon, dem sogenannten Servicing System, führt der Servicetechniker im Außendienst dann Diagnose, Device Setup oder Device Monitoring für ein betroffenes Gerät durch. Ein typisches Techniker-Notebook enthält alle benötigten Daten und Programme, sowie Hardware, die für den (kunden- und gerätespezifischen) Austausch von Daten zwischen dem PC und der Steuerung des untersuchten Geräts notwendig ist. Abbildung 3-2 zeigt ein solches Notebook, das hier zusätzlich mit einem elektronischen Meßgerät für die automatisierte Datenerfassung und einem Protokolldrucker ausgestattet ist.
6. Die Einsatzkomponente von DiaMon protokolliert alle durchgeführten Aktionen des Technikers. Der so entstandene Servicebericht kann abschließend wieder an die Servicezentrale übermittelt werden. Zusätzlich hat der Techniker die Möglichkeit, jederzeit in das Protokoll eigene Bemerkungen einzufügen, z.B. Hinweise auf Unzulänglichkeiten in der Modellierung. Diese Notizen wertet der Serviceingenieur wiederum aus, um das Diagnosewissen zu aktualisieren und zu vervollständigen.

Für die Arbeiten im Rahmen von INDIA ist nur das Diagnosemodul von DiaMon relevant. Dieses wird im folgenden näher erläutert.

3.2.1 Fehlerbäume: Strukturiertes Diagnosewissen

Fehlerbäume enthalten das strukturierte Diagnosewissen über das betroffene Gerät. Sie enthalten die folgenden Objekte (vgl. Abbildung 3-3):

- **Fehler** repräsentieren mögliches Fehlverhalten des Geräts bzw. am Gerät. Fehler entsprechen Defekten, Störungen oder Ausfällen. Ein Fehler kann sowohl an einem einzelnen Geräteteil als auch an einer Gerätekomponente, einer Gerätefunktion oder dem gesamten Gerät auftreten. Fehler am gesamten Gerät sind letztlich auf Fehler an einzelnen Teilen zurückzuführen. Dieses Ursache-Wirkungs-Verhältnis wird vom Fehlerbaum ausgedrückt: die „Blattfehler“ entsprechen den speziellsten, beobachtbaren bzw. feststellbaren Fehlern an Einzelteilen des Geräts, der „Wurzelfehler“ entspricht im Prinzip der Feststellung „Gerät defekt“. Blattfehler entsprechen nicht unbedingt den kleinsten tauschbaren Teilen am Gerät: ein tauschbares Teil kann durchaus aufgrund verschiedener Ursachen ausgetauscht werden müssen, und es macht häufig Sinn, die genaue Ursache auch dann festzustellen (sprich: den Blattfehler), wenn anschließend unabhängig vom Ergebnis das tauschbare Teil komplett ausgewechselt wird.
- **Fehlermodule** dienen der Modularisierung des Fehlerbaums entlang der physikalischen oder funktionalen Struktur des Geräts. Jedes Fehlermodul ist ein in sich abgeschlossener Fehlerbaum. Allerdings kann ein komplettes Fehlermodul als Ursache in ein anderes, übergeordnetes Fehlermodul eingehen. Der Wurzelfehler eines Fehlermoduls repräsentiert die Tatsache, daß das entsprechende Gerätemodul oder die Gerätefunktionalität ausgefallen oder beeinträchtigt ist. Existieren mehrere Fehlermodule repräsentieren sie in ihrer Gesamtheit den Fehler „Gerät defekt“.
- **Meßgrößen** nehmen aktuelle Zustandsdaten über das Gerät auf. Meßgrößen können entweder numerische oder symbolische (z.B: „auf“, „zu“) Werte annehmen. Fehlern, die aufgrund von Meßgrößen festgestellt werden können, sind sogenannte Eintrittsbedingungen zugeordnet. In diesen werden Meßgrößen miteinander oder mit konstanten Werten verglichen, um einen Fehler als eingetreten oder auch eindeutig als ausgeschlossen zu kennzeichnen. (Im letzten Fall wird von Ausschlußbedingungen gesprochen.) Ist keins von beiden möglich, wird ein Fehler (bzw. sein Zustand) als „unbekannt“ betrachtet.

- **Messungen** beschreiben, wie eine oder mehrere Meßgrößen tatsächlich bestimmt werden. Die eigentliche Meßaktion ist dabei entweder ein lesender Zugriff auf die Steuerung des Geräts (**Meßfunktion**) oder eine Frage an den Bediener (**Benutzerfrage**). Unter Umständen ist es notwendig, zur Durchführung der Meßaktion erst einige vorbereitende Arbeitsschritte durchzuführen. Dies können Einstellungen in der Steuerung sein (**Stellprozeduren**, z.B. Aktivieren eines Fehlersuchmodus) oder Anweisungen an den Bediener (**Anweisungen**, z.B. „Schraube den Deckel X ab...“). Neben diesen sogenannten Voreinstellungen für eine Messung ist es konsequenterweise auch möglich, Nacheinstellungen anzugeben, mit denen (in den meisten Fällen) die vorbereitenden Schritte wieder rückgängig gemacht werden. DiaMon enthält einen Mechanismus zur Verwaltung von Einstellungsschritten und den dadurch herbeigeführten Gerätezuständen, so daß sichergestellt ist, daß Einstellungen, die in einer Folge von Messungen mehrfach benötigt werden, nur einmal angewiesen werden.
- Fehlern, die repariert werden können — das sind typischerweise solche Fehler, die „Fehlverhalten an tauschbarem Teil“ ausdrücken — werden Reparaturanweisungen zugeordnet. Dieses sind wieder entweder manuelle **Anweisungen** an den Bediener oder **Stellprozeduren** zum Einstellen von Geräteparametern.
- Zur Verdeutlichung von **Anweisungen** und **Benutzerfragen** können diesen darüber hinaus Bilder und Ersatzteilinformationen zugeordnet werden.
- **Komponenten** nehmen technische Daten für Komponenten des Geräts auf. Sie sind zu Informationszwecken den Fehlermodulen zugeordnet. Komponenten dienen auch der Verwaltung von gerätespezifischen und ggf. variantenabhängigen Vergleichswerten, die in Eintritts- bzw. Ausschlußbedingungen verwendet werden können (z.B. „max. zulässige Temperatur“).

3.2.2 Diagnoseablauf

Zu Beginn einer Diagnosesitzung werden sogenannte Einstiegsfehler als Untersuchungskandidaten ausgewählt: dabei handelt es sich typischerweise um die Wurzelfehler aller Fehlermodule. Untersuchungskandidaten haben immer eine Eintritts- oder Ausschlußbedingung. Dann werden alle Meßgrößen, die für deren Auswertung benötigt werden, festgestellt. Wurden diese Meßgrößen noch nicht bestimmt, werden deren Messungen ausgeführt. Die anschließende Auswertung der Bedingungen kennzeichnet die Einstiegsfehler als „eingetreten“, „ausgeschlossen“ oder „unbekannt“. Dieses Ergebnis wird durch den Fehlerbaum von der Wurzel zu den Blattfehlern hin („top-down“) propagiert. Dabei werden z.B. Ursachenfehler für eingetretene Fehler als verdächtig markiert, Ursachenfehler für ausgeschlossene Fehler ebenfalls als ausgeschlossen.

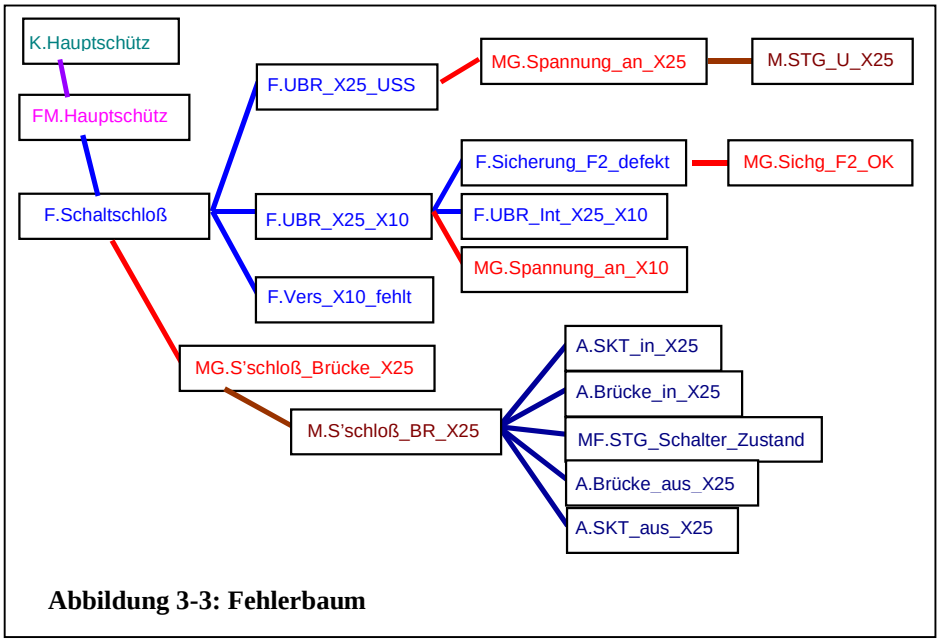


Abbildung 3-3: Fehlerbaum

Die verdächtigen Fehler werden jetzt zu neuen Untersuchungskandidaten. Das Verfahren wird entsprechend fortgesetzt, bis zu jedem eingetretenen Wurzelfehler ein eingetretener Ursachenblattfehler oder zumindest ein eingetretener Ursachenfehler mit Reparatur festgestellt wurde.

Meßgrößen und Ursachenfehler können an mehr als einem Fehler verwendet werden. Der Fehlerbaum ist also im streng mathematischen Sinne ein gerichteter, azyklischer Fehlergraph. Dies hat zur Folge, daß die Bestimmung von Meßgrößen, die für einen Fehler ausgelöst wurde, durchaus sofort zur Bestimmung von anderen Fehlern führen kann. Nach jeder Messung wird daher ebenfalls eine Propagation im Fehlerbaum durchgeführt, diesmal jedoch von den Blättern zur Wurzel („bottom-up“).

Reparaturen an eingetretenen Fehlern werden schließlich dem Bediener zur Durchführung angewiesen. Anschließend wird der betroffene Teilbaum des Fehlerbaums erneut ausgewertet, um den Erfolg der Reparatur zu überprüfen. Die Diagnose endet, wenn keine eingetretenen Fehler mehr vorhanden sind.

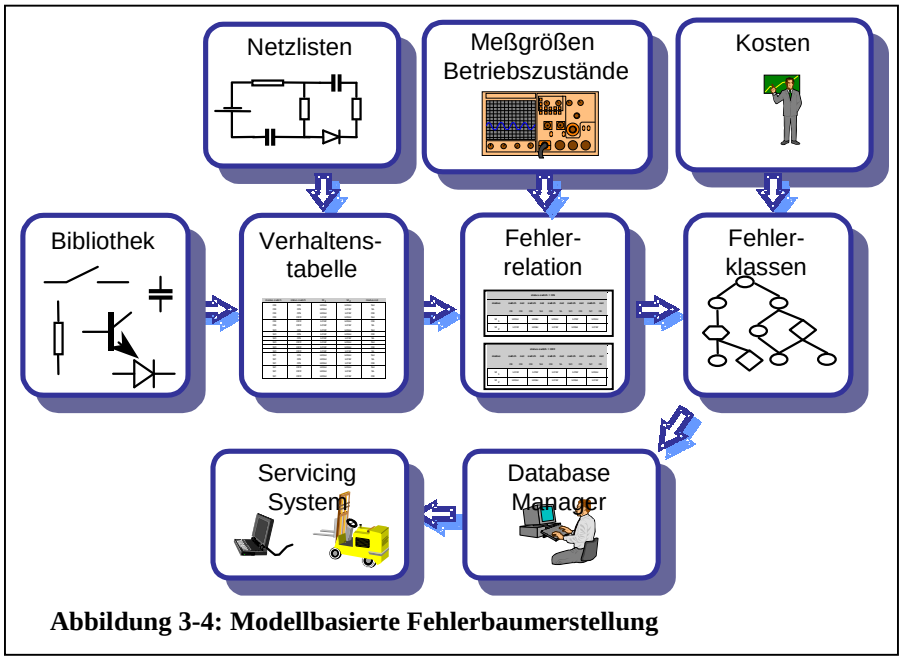


Abbildung 3-4: Modellbasierte Fehlerbaumerstellung

3.2.3 Erfassung von Fehlerbäumen

Für die Erfassung des Diagnosewissens im allgemeinen und der Fehlerbäume im speziellen stellt DiaMon den sogenannten Database Manager bereit. Dieser bietet eine komfortable Oberfläche an, in der Diagnoseobjekte erzeugt werden, ihre Attribute, Einstellungen und Verknüpfungen bearbeitet werden und diese Verknüpfungen graphisch dargestellt werden können.

3.3 Modellbasierte Diagnose und DiaMon

3.3.1 Gesamtszenario

Das Gesamtszenario für die modellbasierte Diagnose mit DiaMon basiert auf den Ergebnissen aus der vorgeschalteten modellbasierten Analyse im System MAD (vgl. Abbildung 3-4 und Kapitel II-2).

MAD kombiniert eine konkret vorliegende Schaltungsbeschreibung (oder ggf. auch eine Ersatzschaltung für nicht-elektronische Zusammenhänge) mit den Verhaltens- und Strukturbeschreibungen für deren Elemente, die in einer geeigneten Bibliothek vorliegen. Aus diesen Daten wird —vereinfacht dargestellt— eine Verhaltenstabelle berechnet, zu der Informationen über Meßpunkte in der Schaltung (repräsentiert durch Meßgrößen) und Betriebszustände, in denen die gemessenen Werte relevant sind, hinzugefügt werden: Es ergibt sich die zugehörige Fehlerrelation (vgl. II-2.2). Diese ordnet bestimmten Kombinationen von Werten von Meßgrößen (jeweils bezogen auf einen Betriebszustand, kurz: MG_{BZ}) einen oder mehrere Fehlerzustände der Schaltung zu (vgl. Abbildung 3-5).

Betr.-Zstd	STANDARD					GETRETEN			
Meß-Größe	UFG_KLEIN	UG	UFG	UFG_GROSS	UFGS	UFG_KLEIN	UG	...	Fehler
	OK	10	[4.6 5.3]	OK	ZU	OK	10	...	1S16A_ist_verklemt
Werte-	OK	10	[4.6 5.3]	OK	AUF	OK	10	...	W5_IST_LEITUNGSBRUCH
	OK	10	[4.6 5.3]	OK	ZU	OK	10	...	W2_IST_LEITUNGSBRUCH
kombi-	OK	10	[4.6 5.3]	OK	ZU	OK	10	...	W1_IST_LEITUNGSBRUCH
	NICHT_OK	0	0	OK	AUF	NICHT_OK	0	...	W8_IST_LEITUNGSBRUCH
natio-	OK	10	[5.3 inf]	NICHT_OK	AUF	OK	10	...	W4_IST_LEITUNGSBRUCH
	NICHT_OK	10	0	OK	AUF	NICHT_OK	10	...	W3_IST_LEITUNGSBRUCH
nen	NICHT_OK	10	0	OK	AUF	NICHT_OK	10	...	FAHRGEBER_IST_DEFEKT
	NICHT_OK	10	[0.1 4.6]	OK	AUF	NICHT_OK	10	...	FAHRG._IST_DEJUSTIERT
	NICHT_OK	[0.1 9.8]	[0.1 4.6]	OK	AUF	NICHT_OK	[0.1 9.8]	...	VCC_IST_ZU_NIEDRI._SPG
	OK	[10.2 inf]	[5.3 inf]	NICHT_OK	AUF	OK	[10.2 inf]	...	VCC_IST_ZU_HOHE-SPG
	NICHT_OK	0	0	OK	ZU	NICHT_OK	0	...	VCC_IST_KEINE_SPG
	OK	*10*	*[4.6 5.3]*	*OK*	*AUF*	*OK*	*10*	...	*OK_FALL*

Abbildung 3-5: Ausschnitt aus der Fehlerrelation für das Beispiel „Fahrgeber“

Mit einem geeigneten Klassifikationsalgorithmus (wie z.B. ID-3, A*) wird aus der Fehlerrelation unter Hinzunahme von Optimierungskriterien (wie z.B. Kosten der Durchführung einer Messung) eine Fehlerklassenstruktur (FKS) erzeugt (vgl. Abbildung 3-6). Dieser Schritt kann ebenfalls in MAD durchgeführt werden (vgl. II-2.4). Für die Verwendung mit DiaMon jedoch wird die Fehlerrelation aus MAD in geeigneter textueller Form exportiert und in eine INDIA-Version des Database Manager importiert. Dieser ist selbst in der Lage, die Klassifikation vorzunehmen.

Eine Fehlerklassenstruktur stellt einen Entscheidungsbaum dar, an dessen Knoten Vergleiche von MG_{BZ} mit Referenzwerten stehen, und an dessen Blättern Fehlerzustände annotiert sind. Treten in einem Blatt mehr als ein Fehlerzustand auf, heißt das, daß diese Fehler mit den gegebenen Modelldaten nicht differenziert werden können. (Im folgenden Beispiel sind aus Platzgründen nicht immer alle Fehlerzustände eines Blatts aufgelistet.)

Der Database Manager wandelt diese Fehlerklassenstruktur anschließend in einen DiaMon-Fehlerbaum um. Dabei werden folgende Objekte erzeugt:

- Eine Fehlerklassenstruktur wird genau durch ein **Fehlermodul** repräsentiert (z.B. „FM.Schaltung“). Der unbeschriftete Wurzelknoten der FKS wird zu einem entsprechenden Wurzelfehler im Baum („F.Fehler_in_Schaltung“), für den keine Eintrittsbedingung angegeben ist.
- Für jeden Vergleichsknoten in der Fehlerklassenstruktur wird ein **Fehler** im Fehlerbaum erzeugt, der als Eintrittsbedingung genau den entsprechenden Vergleich erhält. Mangels näherer Informationen über diesen Fehler erhält das erzeugte Fehlerobjekt einen intern generierten Namen (etwa „F.F123“).
- Für die Fehlerzustände an den Blättern der FKS werden ebenfalls **Fehler** angelegt, die genau diesen Fehlerzustand darstellen, jedoch ohne Eintrittsbedingung (also z.B. „F.W1_ist_Ltgsbruch“). Treten an einem Blatt mehrere nicht differenzierbare Fehlerzustände auf, wird für jeden ein entsprechender **Fehler** angelegt (vgl. Beispiel in Abbildung 3-10).
- Für jede verwendete MG_{BZ} wird eine **Meßgröße** sowie die zugehörige **Messung** angelegt (für $UG_{Standard}$ z.B. „MG.UG_STANDARD“). Die Messung erhält als standardisierte Meßaktion eine geeignete **Benutzerfrage**. (Wird die Modellierung von Meßgrößen im MAD um geeignete Zusatzinformationen erweitert, kann auch direkt eine passende **Meßfunktion** eingesetzt werden.)

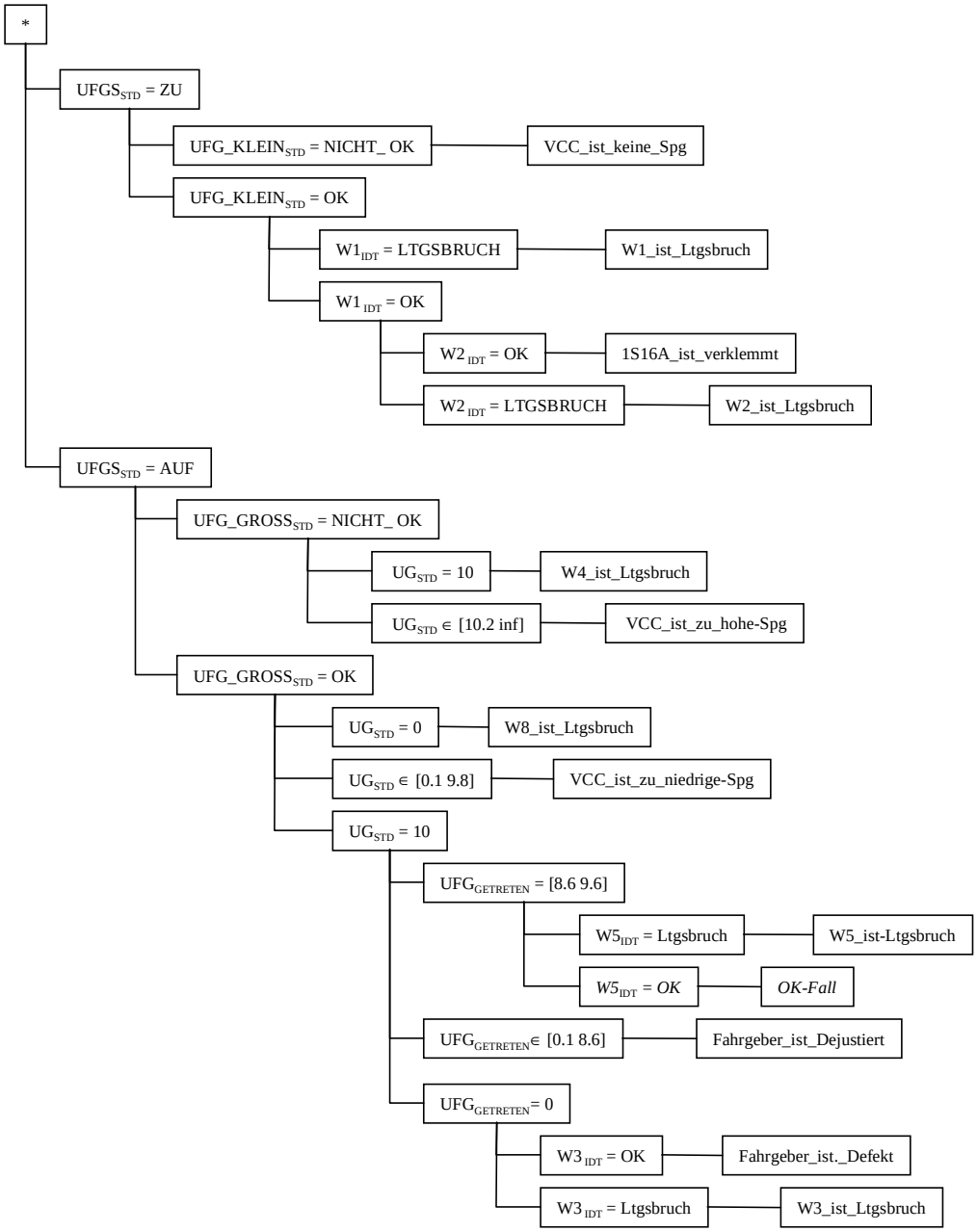


Abbildung 3-6: Fehlerklassenstruktur für das Beispiel „Fahrgeber“.
(Abkürzungen: Std – Standard, Ltg – Leitung, Spg – Spannung)

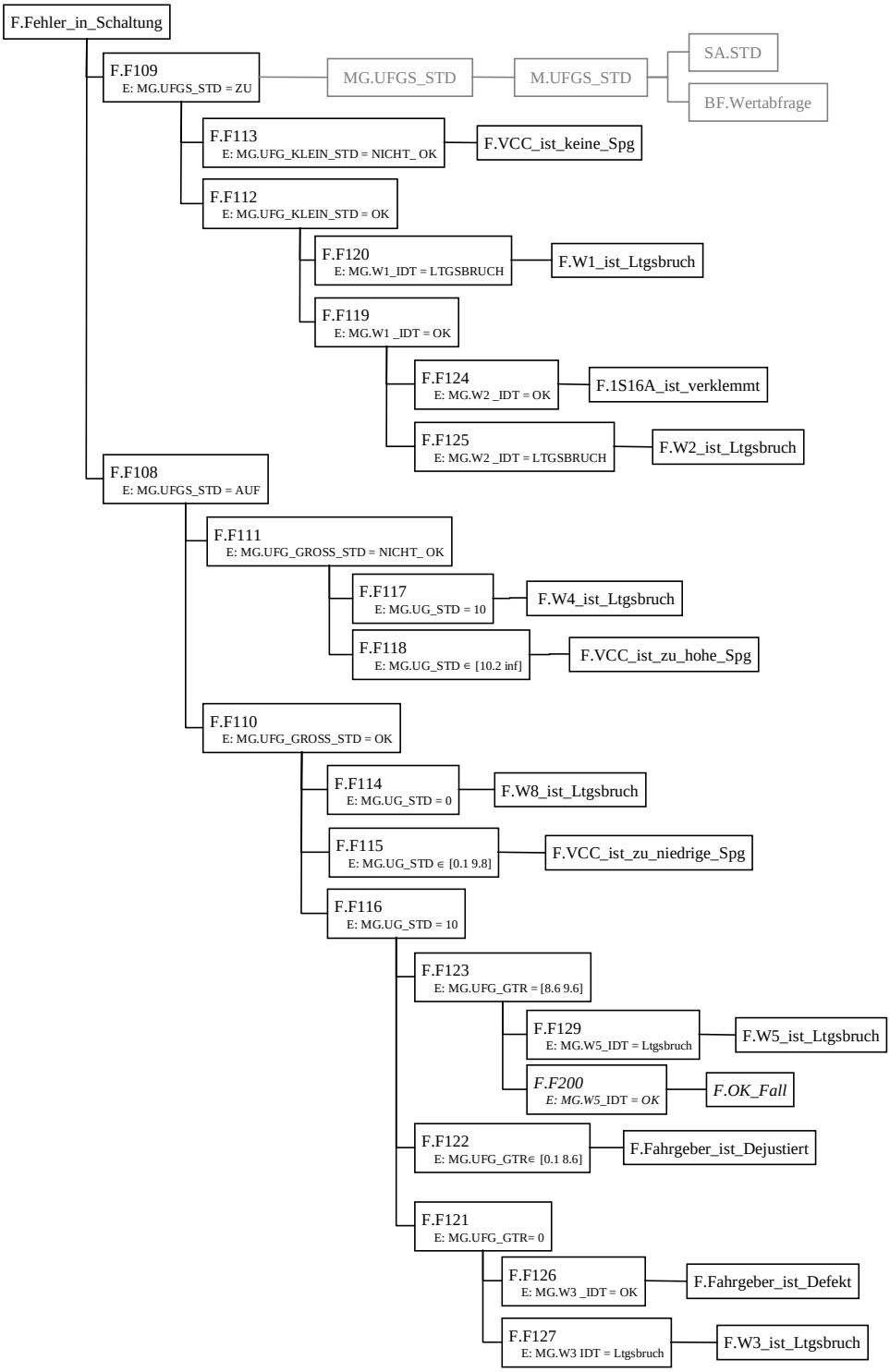


Abbildung 3-7: Generierter Fehlerbaum für das Beispiel „Fahrgeber“.
(Abkürzungen: Std – Standard, Ltg – Leitung, Spg – Spannung)

- Für jeden zu betrachtenden Betriebszustand wird eine **Stellanweisung** erzeugt (z.B. „SA.STD“), die den Servicetechniker anweist, den Betriebszustand herbeizuführen. Diese Stellanweisung wird in den Messungen für die MG_{BZ} als Voreinstellung eingefügt.
- Für die Referenzwerte in den Vergleichsausdrücken liegen bereits im MAD-Modell neben ihrem eigentlichen numerischen Wert symbolische Begriffe vor (wie z.B. „Max-Temp“, „Min-Temp“). Für die importierte FKS wird eine **Komponente** angelegt, an der diese symbolischen Begriffe als **Standardwerte** mit den entsprechenden Belegungen definiert werden. In den Fehlereintrittsbedingungen werden konsequenterweise diese Standardwerte statt der numerischen Referenzwerte verwendet (im Beispiel aus Platzgründen nicht gezeigt).

Für das gezeigte Beispiel sähe der erzeugte Fehlerbaum dann wie in Abbildung 3-7 aus (Meßgröße und Messung sind nur für den ersten **Fehler** gezeigt).

Der Serviceingenieur muß nun mit Hilfe des Database Manager den aus dem Modell generierten Fehlerbaum nachbearbeiten. Nachbearbeitungsschritte sind unter anderem:

- Einsetzen von Stellprozeduren und Meßfunktionen für die generierten Stellanweisungen und Benutzerfragen, wo immer möglich und sinnvoll.
- Ausformulieren von Texten an Anweisungen und Benutzerfragen, die nicht direkt an das Gerät gerichtet werden können. Gegebenenfalls auch Hinzufügen von Bildern und Ersatzteilen zur näheren Illustration.
- Umbenennen von internen Fehlerkürzeln in sinnvolle Fehlernamen, wo immer die automatisch erzeugten Fehler einem identifizierbaren Fehlverhalten des Geräts entsprechen.
- Wo notwendig, Ausdifferenzieren von Blattfehlern ohne Eintrittsbedingungen, für die das Modell keine genauere Angaben enthielt.
- Wo notwendig, Einbetten des erzeugten Fehlerbaums in ein Gesamtfehlermodell des Diagnosesystems, möglicherweise auch Hinzufügen weiterer Fehlerknoten, die nicht aus dem Modell hervorgehen.

Der so erzeugte Diagnose-Fehlerbaum kann abschließend im DiaMon Servicing System eingesetzt werden.

3.3.2 Fehlerklassenstruktur vs. Fehlerbäume

Auch wenn FKS und Fehlerbaum strukturell praktisch gleich aussehen und das voranstehende Umwandlungsverfahren einfach und intuitiv einwandfrei erscheint, sind doch weitere Umwandlungsschritte notwendig, da die Semantiken der FKS und der Fehlerbäume sich in einigen Punkten unterscheiden.

- Eine FKS ist ein Entscheidungsbaum: Die Vergleiche an den einzelnen Knoten sagen nur aus, in welchem Zweig des Baums die Untersuchung fortgesetzt werden soll, sie repräsentieren aber in den seltensten Fällen tatsächliche Fehlzustände im Gerät. Die Blätter der FKS stellen die möglichen Ergebnisse der Suche dar, d.h. die möglichen Fehlerzustände *aber auch* den Zustand, daß überhaupt kein Fehler vorliegt. Wenn die Bedingungen an der Wurzel „MG.A > 0“ bzw. „MG.A <= 0“ lauten, heißt das nur, daß dies ein wichtiges Kriterium ist, um die nachfolgende Suche so geschickt wie möglich einzuschränken. Es heißt nicht, daß „MG.A > 0“ (oder das Gegenteil) auch immer ein Indiz für einen Fehler in einer Komponente oder Funktionalität ist.
- Der Fehlerbaum repräsentiert Fehler in einer Ursache-Wirkung-Beziehung. Für einige (aber nicht notwendigerweise alle) Fehler können Eintrittsbedingungen angegeben werden, um den Fehler näher zu bestimmen. Gilt der Wurzelfehler aber als eingetreten, ist das bereits der Beweis, daß mindestens ein Blattfehler im Baum ebenfalls eingetreten sein muß. Im Fehlerbaum kann demzufolge kein Blattfehler auftreten, der „Kein Fehler“ heißt. (Für den Servicetechniker wäre das auch ziemlich unbefriedigend: nach einer längeren Suche erfährt er vom System, das er den Fehler gefunden hat: „Es liegt keiner vor“.) Die Eintrittsbedingung am Wurzelfehler prüft daher immer das Vorkommen eines Fehlersymptoms, das zumindest aussagt, daß wirklich ein Fehler vorliegt, ggf. auch Hinweise darauf gibt, in welcher Komponente oder Funktionalität. Mehr noch, DiaMon nutzt die Ursache-Wirkungs-Zusammenhänge auch aktiv aus, um Fehler einzukreisen ohne überflüssige Messungen durchzuführen: Hat ein als eingetreten erkannter Fehler mehrere Ursachen und wurden von diesen alle bis auf eine bereits als „nicht eingetreten“ verworfen, schließt DiaMon automatisch darauf, daß der verbleibende Fehler als eingetreten zu gelten hat, *ohne* dessen Eintrittsbedingung noch zu prüfen.

Es sind demzufolge weitere Umwandlungsschritte notwendig, um nach der oben beschriebenen „naiven“ Umwandlung der FKS zu einem Fehlerbaum zu kommen, der für den Einsatz im Diagnosesystem tatsächlich geeignet ist:

1. „‚Kein Fehler' ist kein Fehler": Tritt in einem Teilbaum der FKS der Blattzustand „Kein Fehler" auf, so muß der zugehörige Fehler unter geeigneter Umstrukturierung aus dem Fehlerbaum entfernt werden.
2. „Mehrfachtest für Meßgrößen": Wird dieselbe MG_{BZ} an verschiedenen Stellen der FKS zur Entscheidung herangezogen, so gilt der spätere Test immer nur unter der Annahme, daß alle vorangegangenen bereits durchgeführt wurden. Dies muß im Fehlerbaum explizit gemacht werden.
3. „Mehrfachklassifikation von Fehlern": Wird derselbe Fehlerzustand in der FKS in verschiedenen Teilbäumen erreicht, darf dafür im Fehlerbaum nicht nur ein entsprechend vernetzter Fehlerknoten erzeugt werden.

Diese Probleme und ihre Behebung werden im folgenden detailliert diskutiert.

3.3.2.1 Problem 1: ‚Kein Fehler' ist kein Fehler

Tritt in einem Teilbaum der FKS der Blattzustand „Kein Fehler" auf, so muß der zugehörige Fehler unter geeigneter Umstrukturierung aus dem Fehlerbaum entfernt werden.

Dazu wird der erzeugte Fehlerbaum von den Blättern her zur Wurzel hin umgeformt: ein Fehlerknoten mit Eintrittsbedingung, der als einzige Ursache den Fehlerzustand „Kein Fehler" enthält, wird ersatzlos entfernt. Für alle darüberliegenden Fehlerknoten in dem betroffenen Teilbaum gilt jetzt, daß ihre Eintrittsbedingung *nicht mehr* aussagt, daß in diesem Fehlerteilbaum tatsächlich ein Fehler ist, sondern *nur*, daß dort ein Fehler sein *könnte*. Es kann also nur geschlossen werden, daß bei Nichterfüllung der Eintrittsbedingung der Teilbaum nicht weiter untersucht werden muß.

DiaMon bietet die Möglichkeit, daß ein Fehler statt einer Eintrittsbedingung eine Ausschlußbedingung erhält, bei deren Erfüllung der Fehler als „nicht eingetreten" markiert wird, andernfalls jedoch der Fehler nur als „unbekannt" bestimmt wird, so daß die Untersuchung unterhalb des Fehlers mit weiteren Verdächtigen fortgesetzt wird.

Dies wird bei der Umwandlung des Fehlerbaums genutzt: Jede Eintrittsbedingung, in deren Teilfehlerbaum „Kein Fehler" aufgetreten ist, wird durch Negation in eine solche Ausschlußbedingung umgewandelt. Aus einer Eintrittsbedingung, die besagt, „Wenn MG.A > 0, dann ist ein Fehler in diesem Teilbaum", wird die Aussage: „Wenn nicht MG.A > 0, dann ist garantiert kein Fehler in diesem Teilbaum".

In der Abarbeitung des so umgewandelten Fehlerbaums durch DiaMon entsteht jetzt das gewünschte Verhalten: Gegebenenfalls wird der komplette Teilbaum untersucht und schließlich keiner der darin enthaltenen Fehler als eingetreten festgestellt (der Fehler „Kein Fehler" ist ja entfernt worden): das Gerät ist (was diesen Teilbaum anbetrifft) in Ordnung. Im Unterschied zu vorher wird bei gleicher Vergleichsanzahl der Baum als „in Ordnung" gekennzeichnet anstatt ihn als fehlerhaft mit dem Fehler „Kein Fehler" zu markieren.

Im behandelten Schaltungsbeispiel tritt ein solcher Fall nicht auf. Der Umwandlungsalgorithmus muß aber grundsätzlich in der Lage sein, mit „Kein Fehler"-Klassifikationen umgehen zu können. Zu Demonstrationszwecken wurde in die Fehlerrelation in Abbildung 3-5 eine Wertekombination mit dem Fehlerzustand „OK_Fall" aufgenommen. (Dort und in den nachfolgenden Darstellungen der FKS und des Fehlerbaums sind die entsprechenden Einträge *kursiv* dargestellt.) Nachdem für die Umwandlung angegeben wurde, daß „OK_Fall" kein Fehler ist, sieht der untere Fehlerteilbaum dann wie in Abbildung 3-8 gezeigt aus (der andere Teilbaum bleibt unverändert, da dort nur wahre Fehler enthalten sind).

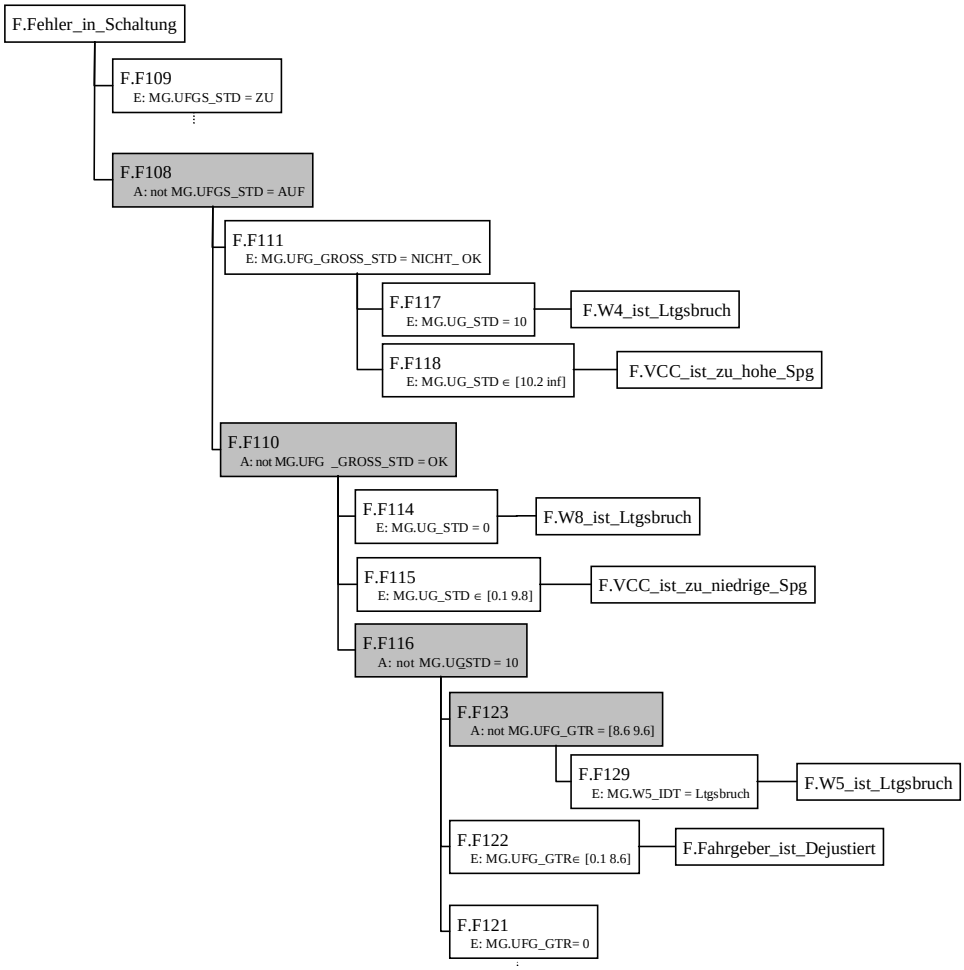


Abbildung 3-8: Fehlerbaum nach Entfernen von „Kein Fehler“-Fehlern

3.3.2.2 Problem 2: Mehrfachtest für Meßgrößen

Wird dieselbe MG_{BZ} an verschiedenen Stellen der FKS zur Entscheidung herangezogen, so gilt der spätere Test immer nur unter der Annahme, daß alle vorangegangenen bereits durchgeführt wurden. Folgt in einem Teilbaum unterhalb der Bedingung „MG.A > 0“ später die Abfrage „MG.A < 10“ wird genaugenommen der Fall geprüft, daß MG.A zwischen 0 und 10 liegt.

DiaMon hingegen nimmt an, daß der Fehlerbaum und seine Bedingungen jederzeit sowohl „top-down“ als auch „bottom-up“ propagiert werden können, die einzelnen Bedingungen werden als für sich alleine aussagekräftig angesehen. Wird also MG.A zu „<= 0“ bestimmt, wäre die Eintrittsbedingung in den Teilbaum nicht erfüllt, sein Wurzelfehler also „nicht eingetreten“, demzufolge gälten auch alle weiteren Fehler in diesem Teilbaum als „nicht eingetreten“. Die darin vorkommende Bedingung „MG.A < 10“ würde jedoch als eingetreten erkannt, da „MG.A <= 0“ ist: DiaMon würde einen Modellierungswiderspruch erkennen, da ein eingetretener Fehler in einem als nicht eingetreten erkannten Teilbaum auftreten würde, und diesen dem Benutzer entsprechend melden.

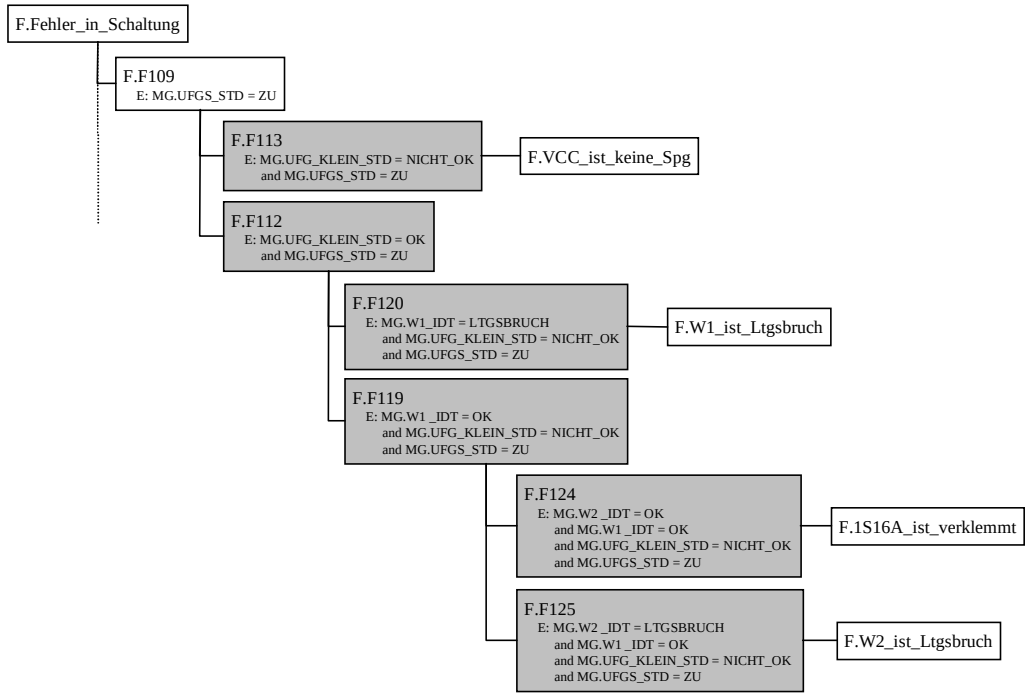


Abbildung 3-9: Fehlerbaum nach Berücksichtigung von Mehrfachtests für Meßgrößen

Dies gilt natürlich auch dann, wenn andere Meßgrößen involviert sind: Tritt sowohl in einem mit „MG.B = 0“ als auch in einem mit „MG.B ≠ 0“ beginnenden Teilbaum „MG.A > 0“ auf, so ist der Test für MG.A immer unter der Annahme zu verstehen, daß der Test für MG.B entsprechend ausgefallen ist. Wiederum würde DiaMon einen falschen Schluß ziehen: Wird MG.A zu „> 0“ bestimmt, wären beide zugehörigen Fehler „eingetreten“, wegen der sich gegenseitig ausschließenden Bedingungen an MG.B aber nur einer der beiden Teilbäume tatsächlich verdächtig.

Um dies zu vermeiden, muß bei der Umwandlung der FKS in den Fehlerbaum die implizite Konjunktion der Bedingungen, die in der FKS enthalten ist, im Fehlerbaum explizit gemacht werden. Für jede Meßgröße, die mehrfach in Bedingungen herangezogen wird, müssen die „von oben“ ererbten Bedingungen mit „und“ verknüpft hinzugefügt werden: Aus „.MG.A < 10“ wird z.B. „MG.A > 0 UND MG.A < 10“. Abbildung 3-9 zeigt einen entsprechend umgewandelten Ausschnitt des Beispielbaums.

3.3.2.3 Problem 3: Mehrfachklassifikation von Fehlern

Wird derselbe Fehlerzustand in der FKS in verschiedenen Teilbäumen erreicht, so deutet das daraufhin, daß es sich um verschiedene Ausprägungen desselben, etwas allgemeineren Fehlerzustands handeln muß, da die bestimmenden Daten (in den Meßgrößen) offensichtlich widersprüchlich sind. Dies kommt durchaus häufig vor, wenn die Fehlerklassifikation im zugrundeliegenden Modell nicht feinkörnig genug durchgeführt wurde. Es könnte z.B. sein, daß sowohl „MG.A > 0“ als auch „MG.A <= 0 UND MG.B > 0“ beide als (der Einfachheit halber) „Totalausfall“ klassifiziert werden, für die auch nur eine Reparatur angegeben wird. Da MG.A aber nicht beide Bedingungen gleichzeitig erfüllen kann, handelt es sich hinsichtlich MG.A eindeutig um verschiedene Ursachen für den „Totalausfall“.

Demzufolge darf im Fehlerbaum nicht nur ein entsprechend vernetzter Fehlerknoten erzeugt werden. Denn wird dieser auf dem einen Weg als „eingetreten“ erkannt, hieße das, das auf allen Wegen die Fehler „eingetreten“ sein müßten, was an der Bedingung um MG.A aber nicht der Fall ist. Wiederum würde DiaMon auf einen Modellierungsfehler erkennen und eine erfolgreiche Diagnose verweigern.

Das Problem ließe sich einfach vermeiden, wenn für jeden gleich klassifizierten Blattknoten verschiedene Fehler entsprechend der zugrundeliegenden verschiedenen Ursachen erzeugt würden („F.Totalausfall_1“ usw.). Allerdings gäbe es dann den Fehler „F.Totalausfall“ mit zugehöriger Reparatur nicht, obwohl dies für den Benutzer offensichtlich eine sinnvolle Fehlerfeststellung ist.

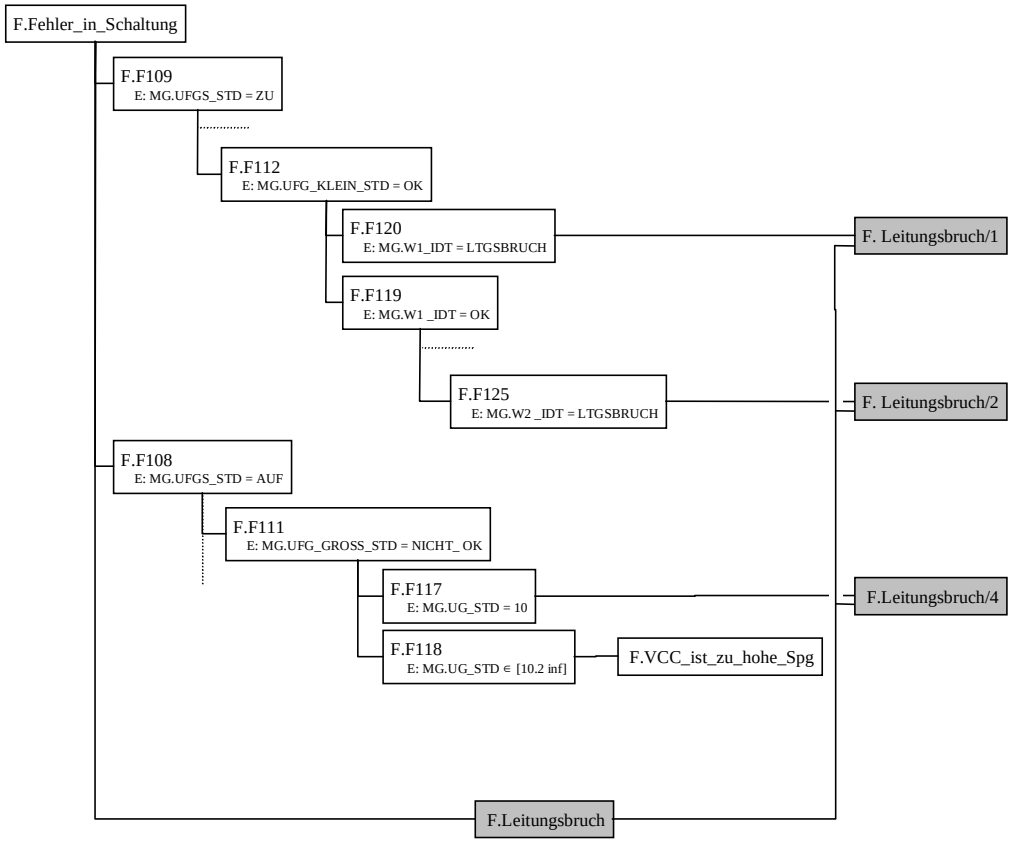


Abbildung 3-10: Fehlerbaum mit Teilbaum für mehrfach klassifizierte Fehler

„F.Totalausfall“ muß also ebenfalls erzeugt werden, aber nicht als Ursachenfehler für die betroffenen Teilbäume, sondern als zusätzlichen Wirkungsfehler in einem eigenen Teilbaum, der alle vorkommenden Ausprägungen als Ursachenfehler enthält, für den selbst aber keine Eintrittsbedingung angegeben ist. Wird jetzt im Zuge der Diagnose einer der Ursachenfehler als „eingetreten“ festgestellt, würde infolge der Propagation „bottom-up“ durch den Fehlerbaum auch „F.Totalausfall“ als eingetreten erkannt und die zugehörige Reparaturanweisung angewiesen.

Würden in dem diskutierten Beispiel die Leitungsbruch-Fehler alle gleichermaßen zu „F.Leitungsbruch“ klassifiziert, ergäbe sich dort die beschriebene Konstellation wie in Abbildung 3-10 gezeigt.

3.4 Integration mit Produktionsabläufen

Hauptziel des Anwenders im Rahmen des INDIA-Projekts war eine merkliche Beschleunigung der Erstellung von Fehlerbäumen. Unter diesem Aspekt war nicht nur die Einführung der modellbasierten Diagnose relevant, sondern auch die Verbesserung der Akquisition von Modelldaten durch Import von beim Anwender bereits existierenden Modellbeschreibungen.

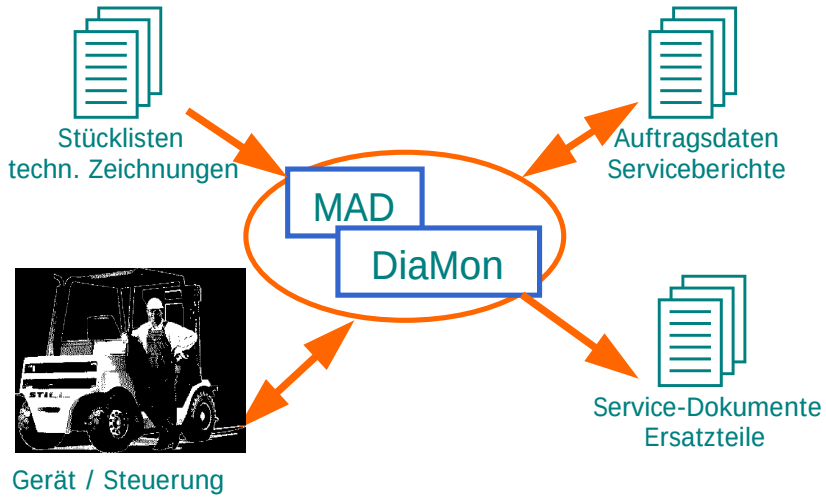


Abbildung 3-11: Integration in die Produktionsumgebung

Im zweiten Projektschwerpunkt wurden daher Integrationsmöglichkeiten des Diagnosesystems in den Produktionsdatenfluß bzw. den Produktionsablauf untersucht. Bei Projektbeginn stand der Import von technischen Zeichnungen und Stücklisten im Vordergrund: Für den ersten Anwendungsprototyp wurde eine Importfunktion konzipiert und implementiert, mit der Netzlisten als Grundlage für Schaltungsmodelle gelesen werden konnten. Im weiteren Verlauf des Projekts verschob sich der Fokus im Wesentlichen aus zwei Gründen:

- Erfahrungen der ServiceXpert GmbH zeigten, daß die Integration mit anderen Aspekten der Produktion und des Service-Alltags mindestens genauso wichtig sind für einen erfolgreichen Einsatz eines Diagnosesystems in der Anwendung (sei es nun modellbasiert oder nicht).
- Ergebnisse der ursprünglichen Aufgabenstellung zeigten, daß
  - für diesen Bereich ein Austauschformat existiert: EDIF („Electronic Document Interchange Format“),
  - beim Anwender STILL zuwenig geeignete Beispieldaten vorliegen, um den praktischen Einsatz eines Imports testen zu können.

Daraufhin wurden mit Zustimmung aller Projektpartner auch andere, vielversprechendere Aspekte der Integrationsproblematik untersucht (vgl. Abbildung 3-11):

- Austausch von Auftragsdaten und Serviceberichten mit einem KISS-System („Kundendienstinformations- und –steuerungssystem“) oder vergleichbaren serviceunterstützenden Applikationen (s. 3.4.1).
- Kommunikation und Datenaustausch mit der Steuerung zu untersuchender Geräte:

  In diesem Zusammenhang mußte festgestellt werden, daß die Kommunikationsschnittstellen von Gerätesteuerungen weit davon entfernt sind, standardisiert oder auch nur ähnlich zu sein. In der Automobilindustrie wurde in den vergangenen Jahren das KEYWORD 2000-Protokoll entwickelt, das zumindest für Steuerungskomponenten verschiedener Hersteller in Autos verschiedener Hersteller für eine einheitliches Kommunikationsprotokoll sorgen soll. Es mehrten sich Anzeichen, daß auch Hersteller von Steuerungen für andere Geräte dieses Protokoll einsetzen würden: Leider ist daraus (bisher?) nichts geworden. Zur Zeit verfügt das DiaMon-Diagnosesystem über eine proprietäre und unabhängig von INDIA entwickelte, anpaßbare und DDE-basierte Kommunikationsschicht für den lesenden und schreibenden Zugriff auf Gerätedaten, auf die in diesem Rahmen nicht weiter eingegangen wird.
- Verknüpfung des Diagnosewissens mit anderen Formen des Servicewissens (Service-Dokumentation, Ersatzteilkataloge), sowohl „passiv“ (Verweise) als auch softwaretechnisch „aktiv“ (Hotlinks) (s. 3.4.2).

3.4.1 Integration mit Applikationen

Eine offene Integration mit anderen serviceunterstützenden Softwareapplikationen erwies sich als praktisch unmöglich, weil in diesem Bereich bisher keinerlei Standards vorhanden sind.

Eine Art Quasi-Standard aufgrund hoher Verbreitung beim Anwender stellt SAP bzw. deren Komponenten für das Servicemanagement (SM) dar. Eine generelle Integration mit SAP-Applikationen wurde jedoch nicht weiter verfolgt, und zwar aus den folgenden Gründen:

- Der Aufwand hätte den Projektrahmen von INDIA überstiegen.
- Wegen der Marktdurchdringung von SAP sind bereits einige Wettbewerber damit beschäftigt, kommerzielle Integrationen mit SAP-Servicemanagement zu entwickeln, so daß sich für die ServiceXpert eine Neuentwicklung nicht gelohnt hätte.

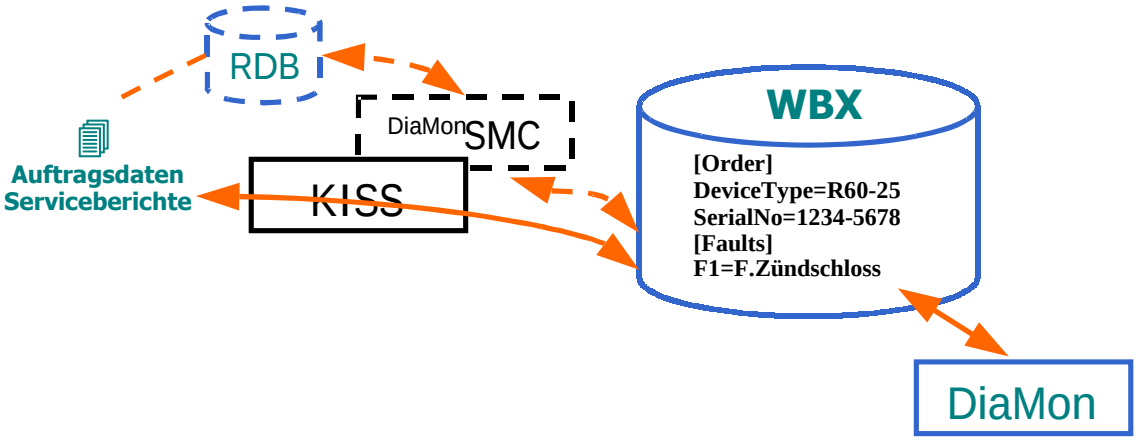


Abbildung 3-12: WBX — Workbench eXchange

Daraufhin wurde ein allgemeiner, softwaretechnischer Ansatz verfolgt, der den Austausch beliebiger Daten zwischen dem Diagnosesystem und anderen Applikationen unterstützen sollte. Dieser sollte die folgenden Randbedingungen erfüllen:

- Weitgehende Unabhängigkeit von speziellen Datenformaten oder applikationsspezifischen Datenumfängen, d.h. Einsatz eines möglichst universellen, offenen Austauschformats.
- Lesende und schreibende Zugriffe auf die Austauschdaten mit einem Standardprotokoll des zugrundeliegenden Betriebssystems (hier: Windows), so daß andere Applikationen leicht darauf zugreifen können bzw. so daß Anpassungskomponenten zwischen dem Austauschformat und einer Fremdapplikation leicht realisiert werden können.
- Einfache datenbankbasierte Integration mit Applikationen, die ihre Daten in einer relationalen Datenbank mit universellen Zugriffsmöglichkeiten (SQL, ODBC) ablegen.

Dies resultierte in dem Konzept des WBX-Austauschobjekts (vgl. Abbildung 3-12). Dabei steht WBX für „WorkBench eXchange“: als „Workbench“ wird die Gesamtheit aller dem Servicetechniker für seine Arbeit zur Verfügung stehenden Applikationen betrachtet, sozusagen seine computerbasierte „Werkbank“.

Das WBX-Objekt ist als individuelle Softwarekomponente realisiert, die auf dem Computer permanent allen Anwendungen zur Verfügung steht. Als Technologie wird die unter Windows standardisierte COM-Technik verwendet. Das WBX-Objekt hält alle Austauschdaten in einem offenen Textformat bestehend aus Sektionen, Einträgen und Werten. Dieses Format wurde von Windows für Initialisierungsdateien und die sogenannte Registry eingeführt. Es hat gegenüber starren Formaten mit endlich vielen vordefinierten Feldern den großen Vorteil, daß jede Applikation nach eigenem Gutdünken beliebig viele Daten hineinschreiben kann, ohne daß dadurch die Fähigkeit anderer Applikationen beeinträchtigt wird, genau nur die Daten auszulesen, die für sie interessant sind.

Für einen erfolgreichen Austausch müssen natürlich zwischen je zwei Applikationen Mindestvereinbarungen über den Datenumfang getroffen werden. Im Rahmen des WBX-Konzepts von DiaMon sind diejenigen Einträge über Serviceaufträge (u.a. Seriennummer und Typ des zu betreuenden Geräts) und Serviceberichte (u.a. erkannte Fehler, verwendete Ersatzteile, Kommentare des Servicetechnikers) vordefiniert, die vom Diagnosesystem aus benötigt bzw. bereitgestellt werden.

Für den Austausch mit Applikationen über eine relationale Datenbank (RDB) wird eine weitere Komponente bereitgestellt, der sogenannte DiaMon Service Management Connector (SMC). Dabei handelt es sich um eine

frei konfigurierbare Komponente, die auf der einen Seite auf das WBX-Objekt zugreift und auf der anderen Seite via SQL und ODBC mit jeder beliebigen relationalen Datenbank kommunizieren kann. Im Rahmen der Konfiguration wird festgelegt, welche Datenbankfelder welchen WBX-Einträgen entsprechen. Damit steht eine einfache Anbindung von KISS-Programmen, die ihre Daten in einer relationalen, von außen zugreifbaren Datenbank ablegen, bereit. Eine Modifikation des KISS-Programms oder die Entwicklung einer speziellen Anpassungskomponente entfällt.

3.4.2 Integration mit Dokumenten

Für die Integration von Dokumenten wurden die Möglichkeiten untersucht, das Diagnosewissen *als Dokument* in den Service- und Produktionsablauf des Anwenders zu integrieren.

Integration in den Serviceablauf bedeutet, daß es dem Servicetechniker jederzeit möglich sein sollte, zwischen der Diagnose bzw. einer angezeigten Serviceanweisung und anderen Formen von Service-Dokumentation, wie z.B. Online-Handbücher, -Ersatzteilkataloge u.ä., zu wechseln. Dazu müssen zum einen aktivierbare Verweise in die Diagnoseanleitungen eingebettet werden (*Hotlinks*), zum anderen sollte es auch möglich sein, umgekehrt Verweise auf das Diagnosewissen in beliebige Fremddokumente einzufügen. Dazu ist es notwendig, für das Diagnosewissen ein Dokumentenformat zu wählen, das auf standardisierte Weise referenzierbar ist.

Integration in den Produktionsablauf in Hinsicht auf Dokumente bedeutet, daß der Anwender die Möglichkeit haben muß, das Dokument „Diagnosewissen“ zusammen mit seinen anderen Dokumenten auf möglichst einheitliche Weise zu verwalten. Verweise zwischen den verschiedenen Dokumenten sollten von zentraler Stelle aus gepflegt und auf Konsistenz geprüft werden können. Das Zusammenstellen von zusammengehörigen Dokumenten, z.B. für eine bestimmte Gerätegeneration oder -variante, sollte ebenfalls von zentraler Stelle aus möglich werden.

Herkömmliche Textverarbeitungs- und Dokumentenmanagementsysteme sind durchaus in der Lage, all diese Anforderungen in Hinsicht auf Querverweise und Variantenverwaltung zu erfüllen, aber herkömmliche Dokumentenformate sind ungeeignet, das Diagnosewissen, das in seiner Struktur am ehesten einer vernetzten Datenbank entspricht, zu repräsentieren.

Mit XML gibt es seit kurzem erstmalig die Möglichkeit, Daten und Dokumente in einer einheitlichen Form auszudrücken und zu verwalten. Im folgenden wird ein Konzept vorgestellt, wie XML für die Integration von Servicedokumenten genutzt werden kann.

```
<?xml version="1.0" ?>
<Book>
     <Titel>Service-Dokumentation für Gerät X</Titel>
     <Author>Hans Müller</Author>
     <Company>Anwender GmbH</Company>
     <Chapter No="1">
          <Titel>Überblick über das Gerät X</Titel>
          <Paragraph>
               Eine ganze Menge Text...
          </Paragraph>
     </Chapter>
</Book>
```

```
<?xml version="1.0" ?>
<SpareParts>
     <Company>Anwender GmbH</Company>
     <Part SerialNo="C-1234">
          <Name>Steuerung C004</Name>
          <Description>
               Eine ganze Menge Text...
          </Description>
          <ValidFor>
               <Device>Gerät X</Device>
               <Device>Gerät X.1</Device>
          </ValidFor>
     </Part>
</SpareParts>
```

Abbildung 3-13: Beispiele für XML-Dokumente

3.4.2.1 XML

Mit XML (eXtendable Markup Language) ist seit 1998 ein neuer Standard für die Verwaltung und Repräsentation elektronischer Dokumente aufgetaucht, der zur Zeit eine rasante Entwicklung hinsichtlich Verbreitung und Verfügbarkeit unterstützender Software durchmacht. Mit XML wird in Dokumenten aller Art konsequent Inhalt von Form getrennt — zum Vergleich: ein Textverarbeitungsprogramm erfaßt Dokumenteninhalte immer eng verbunden mit der Darstellungsform.

XML ist eine Teilmenge von SGML, das bereits seit 10 Jahren in einigen Branchen für die Darstellung technischer Inhalte Fuß gefaßt hat, von einigen Publikationssystemen unterstützt wird, aber wegen seiner hohen Komplexität große Akzeptanzprobleme hatte.

XML ist verwandt mit HTML, der Sprache des Internets: das Format ist das gleiche und wie HTML kann XML Verknüpfungen innerhalb und zwischen beliebigen Dokumenten ausdrücken. Im Gegensatz zu HTML, das wiederum Inhalt und Form eines Dokuments vermengt, enthält XML nur den reinen Inhalt. Dabei ist es unerheblich, ob dieser Inhalt bereits „dokumentennah“ strukturiert ist (also z.B. in Titel, Kapitel, Absätze usw.) oder beliebige Daten ausdrückt (Adressen, Produktdaten, Ersatzteile, usw.). Abbildung 3-13 zeigt zwei kurze Beispiele.

Sogenannte *Stylesheets* (etwa: Stilvorlagen) beschreiben die Umsetzung eines XML-Dokuments nach HTML und damit in eine lesbare Dokumentation. Zu einem Dokument können beliebig viele Stylesheets existieren, die Inhalte in unterschiedlicher Form und Umfang (Filterung, Sortierung, Indizierung usw.) zur Anzeige bringen. Damit können z.B. aus einer einzigen zentralen Repräsentation technischer Inhalte verschiedene Dokumente erzeugt werden (vgl. Abbildung 3-14): Benutzerdokumentationen, Serviceanleitungen, interne Darstellungen usw.

Ersatzteilkatalog für Anwender GmbH

Inhaltsverzeichnis

C-1234: Steuerung C004 ...

1. Seriennummern C-nnnn

1.1 Steuerung C004 [C-1234]

Geeignet für: Gerät X, Gerät X.1

Eine ganze Menge Text...

Abbildung 3-14: Aus XML-Daten generierte Dokumentation

Damit Daten und Dokumente eines Typs in einheitlicher Form erzeugt werden, von Stylesheets immer korrekt umgewandelt werden können und gegebenenfalls auch von anderen Applikationen verstanden und interpretiert werden können, ist es notwendig, die Form eines XML-Dokuments vorzuschreiben und auch zu validieren. Dazu dienen sogenannte *Document Type Definitions* (kurz: DTD) oder seit kurzem auch sogenannte *XML Schemas*. Eine DTD legt die Struktur eines Dokuments fest. Abbildung 3-15 zeigt die DTD für das obengezeigte Beispiel.

```
<!ELEMENT SpareParts (Company, Part+) >
<!ELEMENT Company (#PCDATA) >
<!ELEMENT Part (Name, Description, ValidFor, OrderInfo?) >
<!ATTLIST Part  SerialNo CDATA #REQUIRED >
<!ELEMENT ValidFor (Device+) >
<!ELEMENT Device #PCDATA >
```

Abbildung 3-15: Beispiel für eine DTD

XML ist die neue Sprache der Internet-Browser: in Zukunft wird jeder Browser XML-Dokumente mit und ohne Stylesheets anzeigen können, d.h. technische Inhalte können mit diesem Medium beliebig verbreitet und genutzt werden.

3.4.2.2 Diagnosewissen in XML

Für DiaMon wurde eine XML-basierte Repräsentation des Diagnosewissens entwickelt (vgl. Abbildung 3-16). Diese spiegelt die Struktur der Daten korrekt wieder (da XML hierarchische Schachtelung und beliebige Verknüpfungen vermittels Verweisen unterstützt), so daß die DiaMon-Software die entstehenden XML-Dokumente genausogut lesen kann, wie eine Datenbank.

Andererseits wird damit aus dem Diagnosewissen ein *Dokument*, das vermittels Stylesheets auch außerhalb der DiaMon-Umgebung als Servicedokumentation lesbar wird, und das vermittels Content Management sich nahtlos in die Dokumentenlandschaft eines Anwenders einfügen kann.

Stylesheets erlauben darüberhinaus, daß die Präsentation des Diagnosewissens und der -anleitungen auf elegante Weise an Darstellungswünsche des Anwenders angepaßt werden kann (z.B. in Hinsicht auf *Corporate Identity*).

Schließlich erlaubt die offene und standardisierte Verweisstruktur zwischen beliebigen XML-Dokumenten sowohl die gewünschte bidirektionale Querverweisbarkeit zwischen Diagnosewissen und anderen Service-Dokumenten.

```
<DIAMON>
     <DBINFO version="1.0" >
          <NAME>R60</NAME>
     </DBINFO>
     ...
     <FAULT id="F.184549393">
          <NAME>F.Fahr_Pedal_STG</NAME>
          <ISCAUSING >
               <XLINK id="F.184549394">F.Fehlercode_12</XLINK>
          </ISCAUSING>
          <CONDITION>MG.Fehlercode_lesen_12_Rep = 0</CONDITION>
          <REPAIR showatonce="yes" exetype="manually" >
               <XLINK id="A.167772170">A.Rep_STG_Fehlercode_12</XLINK>
          </REPAIR>
     </FAULT>
     ...
     <INSTRUCTION type="repair" id="A.167772226">
          <NAME>A.Rep_Neigenschalter</NAME>
          <TEXT format="HTML">
               Einstellung des Handhebelwegs:
               <OL>
                    <LI>Stoppmuttern zurückdrehen, ...</LI>
                    <LI>Handhebel bis zum Anschlag ...</LI>
                    <LI>Handhebel in Nullstellung bringen ...</LI>
                    <LI>Mutter eine halbe Umdrehung ...</LI>
               </OL>
               Für Details siehe
               <XLINK href="handbuch.xml">Handbuch</XLINK>
          </TEXT>
     </INSTRUCTION>
     ...
</DIAMON>
```

Abbildung 3-16: DiaMon-Diagnosewissen in XML (Ausschnitt)

3.4.2.3 Content Management

Ein großer Vorteil von XML liegt also darin, daß damit Inhalte in anwendungsseitig definierter Form in einer Weise ausgedrückt werden, die allgemeine, XML-fähige Programme anwendungsunabhängig bearbeiten können. Damit ist es möglich, daß ein Verwaltungsprogramm sämtliche Dokumente beispielsweise einer Firma und deren Verknüpfung untereinander darstellen, modifizieren, durchsuchen und verwalten kann, ohne daß es auf die besonderen Anforderungen dieser Firma zugeschnitten sein muß.

Derartige Programme werden als Content Management Systeme bezeichnet (vgl. Abbildung 3-17) und zur Zeit von etlichen Softwarefirmen angeboten. Content Management erlaubt es, Dokumente und Dokumententeile in verschiedenen Zusammenstellungen wiederzuverwenden und für verschiedene Zwecke auszuspielen. Dabei werden XML-Dokumente bis auf eine frei wählbare Detaillierungsstufe in ihre einzelnen Elemente zerlegt und diese einzeln verwaltet, für Nicht-XML-Dokumente ist die Verwaltung auf Dateiebene möglich.

Verweise aus einem XML-Dokument in ein Nicht-XML-Dokument können behandelt werden, umgekehrte Verweise nur dann, wenn es sich bei dem Nicht-XML-Format um ein anderes standardisiertes Format handelt (SGML, Word), das das gewählte Content Management System ebenfalls verstehen kann.

Damit ist es möglich, die Vorteile von XML und Content Management auch in Produktionsumgebungen auszunutzen, in denen nur wenige Dokumente bereits in XML-Format vorliegen.

3.5 Zusammenfassung

Im Rahmen des Projekts INDIA wurden bei der MAZ bzw. ServiceXpert GmbH die folgenden Ergebnisse erreicht:

7. Qualitative Modellbildung und Analyse: Für die modellbasierte Analyse von Schaltungen im speziellen und vergleichbar strukturierten Gerätekomponenten im allgemeinen wurde am MAZ eine Modellierungssprache konzipiert und prototypisch realisiert. Die Ergebnisse dieser Arbeiten dienten als Input für die Arbeiten des LKI (siehe Kapitel II-2).
8. Übernahme von Fehlerklassenstrukturen in DiaMon: Die vom modellbasierten Analysesystem des LKI erzeugten Fehlerklassenstrukturen können in das Diagnosesystem DiaMon importiert werden. Sie werden dort in semantisch äquivalente Fehlerbäume umgewandelt und so für das Diagnosesystem verwendbar gemacht.
9. Integration des Diagnosesystems in den Produktionsablauf bzw. mit Produktdaten: Im Rahmen der Integrationsarbeiten wurden Konzepte erarbeitet und realisiert, um eine offene Kommunikation des Diagnosesystems mit anderen Softwarekomponenten auf dem Notebook des Servicetechikers zu ermöglichen (wie z.B. Kundendienstinformations- und –steuerungssysteme (KISS) oder Online-Servicedokumentation und –Ersatzteilkataloge). Durch Umstellung der Datenrepräsentation des Diagnosewissens XML wurden erhebliche Vorteile hinsichtlich der Verknüpfbarkeit und der zentralen Verwaltbarkeit von beliebigen Produktions- und Servicedokumenten erzielt.

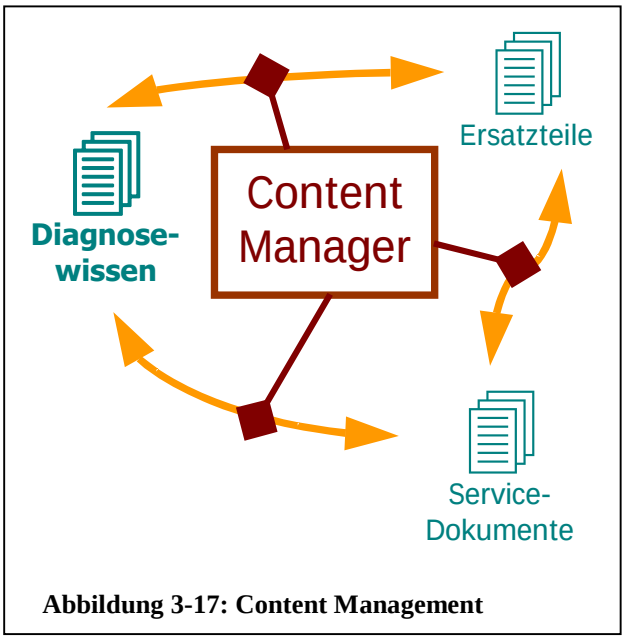


Abbildung 3-17: Content Management

Die Erfahrungen, die wir im Verlaufe dieser Arbeiten machen konnten, waren nicht nur positiv:

- Der modellbasierte Ansatz erwies sich als außerordentlich vielversprechend als *Ergänzung* bei der Erstellung von Fehlerbäumen. Für zu diagnostizierende Komponenten, für die die notwendigen Modelldaten vorliegen, können auf diese Weise korrekte Fehlerbäume erzeugt werden. Um diese Fehlerbäume jedoch tatsächlich im praktischen Einsatz verwenden zu können, ist ein nicht unerheblicher Nachbearbeitungsaufwand zu leisten. Die Vervollständigung des erzeugten Fehlerbaums vor allem um Anweisungstexte und genaue Zugriffsfunktionen zum Gerät läßt sich aus dem Modell heraus nicht vermeiden. Darüber hinaus zeigt die Erfahrung, daß auf diese Weise nur ein Teil der insgesamt für ein Gerät benötigten Fehlerbäume produziert wird: Manuelle

Erstellung von Fehlerbäumen für nicht modellierte oder nicht modellierbare Gerätekomponenten oder –funktionen muß zusätzlich geleistet werden. Die importierten Fehlerbäume müssen in das Gesamtgefüge integriert werden.

- Aus Produktsicht bleibt festzustellen, daß in den vergangenen Jahren, in denen die ServiceXpert mit ihrem Diagnosesystem am Markt tätig gewesen ist, fast keine Nachfrage nach modellbasierten Lösungen zu verzeichnen war. (Siehe auch die Erfahrungen beim Anwendungspartner STILL, Kapitel II-1.)
- Die Arbeiten an der Integration von Produktdaten lieferten wichtige Erkenntnisse für die Produktweiterentwicklung von DiaMon. Die Konzepte zum WBX-Objekt für die Interprogrammkommunikation und zur XML-basierten Datenrepräsentation für die Einbettung des Diagnosewissens in die Dokumentenwelt des Anwenders werden im Verlaufe des Jahres 2000 Eingang in die neuste Release von DiaMon finden. Allerdings litten die Integrationsarbeiten daran, daß im service- und gerätenahen Umfeld ein großer Mangel an Standards herrscht, an denen sich ein offenes Servicesystem für DiaMon orientieren könnte. Mit den Basistechnologien COM für das WBX-Objekt und XML für die Repräsentation wurden allgemeine, anwendungs- und domänenunabhängige Standards gewählt (sozusagen ein Rückzug auf den kleinsten gemeinsamen Nenner). Semantisch höherstehende, domänenspezifische Standards (z.B. für die Repräsentation von Gerätedaten, für die Gerätekommunikation oder für den Austausch von Serviceauftragsdaten) fehlen entweder vollständig, oder sind zu speziell auf eine einzelne Gerätebranche zugeschnitten. Daraus folgte auch, daß die ursprünglich im Projekt vorgesehene Zielrichtung für das Integrationsthema sich als ungeeignet erwies und im Verlaufe des Projekts eine Umorientierung notwendig wurde.
- Aus Sicht eines kleinen Produkthauses wie der ServiceXpert erwies sich die Durchführung eines Forschungs- und Entwicklungsprojekts am „lebenden Produkt“ als außerordentlich schwierig. In der Konsequenz mußten die Forschungsarbeiten immer wieder hinter aktuellen Produktanforderungen zurückstehen oder zumindest an diese angepaßt werden. Dies führte dazu, daß nicht alle ursprünglich gestellten Aufgaben in der angestrebten Tiefe bearbeitet werden konnten.

In der Summe jedoch ist das Projekt INDIA für die ServiceXpert als Erfolg zu werten, da wertvolle neue Impulse für die Produktweiterentwicklung gewonnen werden konnten und ein nicht zu unterschätzender Kenntnis- und Erfahrungsschatz um die Möglichkeiten der modellbasierten Diagnose aufgebaut werden konnte.

4 Bewertung der Ergebnisse aus Anwendersicht

Stefan Volke – STILL GmbH Hamburg

In den folgenden Abschnitten wird eine Bewertung der in den Kapiteln II-2 und II-3 dargestellten Forschungsergebnisse der INDIA-Partner LKI und SXP vorgenommen. Es wird hierbei unterschieden zwischen der prinzipiellen Anwendbarkeit dieser Resultate und der praktischen Anwendbarkeit bei STILL, wobei im letzteren Fall der bei Projektabschluß erreichte Stand der entwickelten Software als Bewertungsmaßstab zu Grunde gelegt wird.

4.1 Prinzipielle Anwendbarkeit der INDIA-Ergebnisse

Das in Kapitel II-2 beschriebene MAD-System erfüllt im wesentlichen die Anforderungen an Diagnose, wie sie im Kapitel II-1 formuliert wurden. Die wichtigsten Stichworte sind hierbei:

- Qualitative Modellierung. Dieser Ansatz reduziert die prinzipiell unendliche Menge von quantitativ unterscheidbaren Fehlern auf eine endliche Anzahl von Fehlermodellen.
- Eine geringe Anzahl von Elementarkomponenten ist ausreichend, um eine große Anzahl realer Komponenten modellieren zu können. Hierdurch wird u.a. die Bildung von Aggregaten auf einer für den jeweiligen Fall angemessenen Abstraktionsebene ermöglicht. Komplexe Komponenten können alternativ durch Beschreibung ihres Ein- und Ausgangsverhaltens mittels abstrakter Tabellenmodelle beschrieben werden.
- Physikalische nicht sinnvolle Lösungen werden unterdrückt.
- Ermöglichung einer kostenoptimierten Fehlersuche durch Berücksichtigung der Meßkosten (unter Verwendung von Algorithmen wie ID-3, A* etc.).
- Geringerer Aufwand beim Erstellen der eigentlichen Fehlersuchlogik (des Fehlerbaums) als bei der manuellen Bearbeitung bei gleichzeitiger Vermeidung typischer Eingabe- und Logikfehler.
- Import der erzeugten Fehlerbäume in das existierende System DiaMon.

Im Rahmen des Projektes wurden Software-Werkzeuge entwickelt, die eine durchgängige Bearbeitung einer zu diagnostizierenden Baugruppe von der Modellierung bis hin zum Import des erzeugten Fehlerbaumes in das System DiaMon ermöglichen:

- „Modeling, Analyzing and Diagnosing“ (MAD) als Gesamtsystem zur Modellierung, Verhaltensbeschreibung und Erzeugung der Fehlerklassenstruktur (Fehlerbaum). Die interne Darstellung des Modells bleibt vor dem Bediener weitgehend verborgen, so daß in weiten Teilen eine intuitive Bedienung des Systems möglich ist. Als Bediener-Interface dient der
- „Component Modeling EDItor“ (COMEDI) zum graphischen Erstellen von Schaltungsmodellen (s. Kapitel II-2.3.4) inklusive der zu betrachtenden Betriebszustände. Weiterer Bestandteil dieses Bediener-Interface ist der
- „Component Model Builder“ (CMB) zur Erstellung von abstrakten Tabellenmodellen für komplexe Komponenten (s. Kapitel II-2.3.5).
- Das Diagnosesystem DiaMon wurde dahingehend erweitert, daß von MAD generierte und in textueller Form exportierte Fehlerrelationen als Ergebnis der Verhaltensbeschreibung in das DiaMon-System importiert werden können. Die eigentliche Erzeugung der Fehlerklassenstruktur findet dann in DiaMon selbst statt.

Die Konzepte zur Anbindung des Diagnosesystems an andere serviceunterstützende Softwarepakete (Kundendienstinformations- und steuerungssysteme, Online-Servicedokumentation und -Ersatzteilkataloge) waren ebenfalls vielversprechend (s. Kapitel II-3.4).

4.2 Anwendbarkeit in der Praxis

Die tatsächliche Anwendbarkeit in der betrieblichen Praxis wird unmittelbar beeinflußt von den Möglichkeiten, die die im Verlauf des Projektes einwickelte Software bietet. Solange diese den Ansprüchen eines potentiellen Anwenders in punkto Bedienung und Leistungsumfang für die zu bearbeitenden Aufgaben nicht oder noch nicht in vollem Umfang entspricht, wird die Einführung eines solchen Systems kaum durchzusetzen sein. In diesem

Fall wäre eine Weiterentwicklung des Softwarepaketes nach Projektabschluß erforderlich. Dieses wird jedoch nur dann erfolgen, wenn sich der Anwender einen klaren Nutzen davon verspricht.

Ähnlich wie bei der Formulierung der Anforderungen an Diagnose (s. Kapitel II-1) wird auch die Bewertung der Anwendbarkeit wieder aus Sicht der vier wesentlichen Anwendergruppen behandelt.

4.2.1 Anwendbarkeit aus Sicht des STILL-Kunden

Da, wie bereits festgestellt wurde, die mit den INDIA-Methoden erzeugten Fehlerbäume in das existierende Diagnosesystem DiaMon integriert und dort an die vorhandenen Gegebenheiten angepaßt werden können, wird der Kunde bei einem Einsatz des STILL-Servicetechnikers an seinen Geräten keinen Unterschied zum bisherigen System bemerken. Wie schon in Kapitel II-1 angedeutet, wird allerdings für den Kunden in den meisten Fällen die interne Struktur des vom Servicetechniker verwendeten Diagnosesystems ohnehin von eher geringem Interesse sein.

Der Kunde profitiert jedoch trotzdem von den Vorteilen, die die automatisch erstellten gegenüber den manuell erzeugten Fehlerbäumen besitzen. Insbesondere die Garantie der Vollständigkeit und Korrektheit einer Diagnose, zusammen mit dem Ansatz der kostenoptimierten Fehlersuche, verkürzen die Einsatzzeiten des Servicetechnikers bei zu erwarteten geringeren Aufwänden. Wie groß die Ersparnis allerdings tatsächlich ist, läßt sich an dieser Stelle nicht abschätzen.

4.2.2 Anwendbarkeit aus Sicht des Servicetechnikers

Für den Servicetechniker gelten im Grunde dieselben Argumente, wie sie im vorigen Abschnitt für den Kunden formuliert wurden, da das Diagnosesystem nach außen hin in seiner Bedienung und den für den Anwender sichtbaren Eigenschaften unverändert bleibt. Aus diesem Grund entfällt auch zusätzlicher Schulungsaufwand.

Die Vermeidung der Fehlermöglichkeiten, die beim manuellen Erstellen der Fehlerbäume gegeben sind und die Vollständigkeit der Diagnose schaffen günstige Voraussetzungen, die Akzeptanz des Systems beim Servicetechniker weiter zu erhöhen. Auch die von SXP entwickelten Konzepte zur offenen Kommunikation des Diagnosesystems mit anderen Softwarekomponenten wie Online-Servicedokumentation und -Ersatzteilkatalogen (um nur zwei Anwendungsmöglichkeiten zu nennen) tragen dazu bei, die Effizienz des Diagnoseeinsatzes beim Kunden zu verbessern.

4.2.3 Anwendbarkeit aus Sicht der zentralen Serviceabteilung

Auch der zentralen Serviceabteilung bei STILL kommt zugute, daß das Diagnosessystem DiaMon in seiner bisherigen Form weiterhin verwendet werden kann und auf diese Weise keine Zusatzaufwände für Schulungen der Servicetechniker entstehen. Die verbesserten Eigenschaften der Diagnose werden diesen sozusagen „untergeschoben“. Auch die entstehenden Möglichkeiten der Interaktion des Diagnosesystems mit anderen serviceunterstützenden Softwarekomponenten wird hier als positiv angesehen.

Nicht zu unterschätzen ist allerdings der nicht unerhebliche Nachbearbeitungsaufwand, der entsteht, um die mit dem modellbasierten Ansatz automatisch erstellten Fehlerbäume in das System DiaMon integrieren zu können. Dies betrifft in der Hauptsache das Hinzufügen von Anweisungstexten und erläuternden Graphiken für den Bediener (Servicetechniker) sowie die Mechanismen, die zum Datenaustausch mit dem Fahrzeug bei automatischen Messungen verwendet werden. Zudem zeigt die Erfahrung, daß nicht alle für die Diagnose benötigten Fehlerbäume unter Verwendung des modellbasierten Ansatzes erstellt werden können. Die Erzeugung von Fehlerbäumen für noch nicht modellierte oder ggf. nicht modellierbare Komponenten muß zusätzlich erfolgen.

Alles in allem ist eine deutliche Verbesserung der Diagnose-Qualität zu erwarten, da die Fehlerbaumstruktur automatisch mit Verfahren erzeugt wird, die Vollständigkeit und Korrektheit der Diagnose garantieren. Die Nachbearbeitung der so erzeugten Fehlerbäume bedeutet jedoch (wie bisher) einen nicht unerheblichen Aufwand.

4.2.4 Anwendbarkeit aus Sicht der Entwicklungsabteilung

Der Einsatz der in INDIA entwickelten modellbasierten Verfahren hat von allen beschriebenen Anwendergruppen für die Entwicklungsabteilung die größten Auswirkungen.

Positiv zu bewerten ist in jedem Fall die gegenüber dem bisherigen System verbesserte Qualität der Diagnose. Eine zuverlässige und vollständige Diagnose kann beim Prozeß der Produktverbesserung im Rahmen der Seri-

enbetreuung von Gabelstaplerbaureihen hilfreich sein. Nur wenn garantiert ist, daß die Ergebnisse der Diagnosesitzungen, die bei den Einsätzen der Servicetechniker beim Kunden gewonnen werden, aufgrund der zu Grunde liegenden Qualität der Diagnose den tatsächlichen Gegebenheiten entsprechen, werden diese Ergebnisse gezielt bei der Produktverbesserung eingesetzt werden. Die entwickelten INDIA-Methoden stellen diese Voraussetzung sicher.

Nachteilig zu bewerten ist jedoch der hohe zusätzliche Aufwand, der für die Modellierung der Komponenten anzusetzen ist. Mit dieser als Grundlage der letztlich zu erzeugenden Fehlerbäume verlagert sich der Hauptanteil der erforderlichen Arbeiten aus der Serviceabteilung zurück in die Entwicklungsabteilung, da nur hier das für die Modellbildung erforderliche Wissen in vollem Umfang vorhanden ist. Insbesondere die Entscheidung, für welche Komponenten die Modellierung sinnvoll erscheint und mit welcher Granularität diese zu erfolgen hat, kann nur hier getroffen werden. Hierbei ist vom Entwickler festzulegen, in welcher Art und Weise eine Schaltungsbaugruppe zu modellieren ist. In den meisten Fällen wird es aus Gründen der Übersichtlichkeit nicht möglich sein, eine Modellierung ausschließlich mit den Standardkomponenten des MAD-Systems durchzuführen. Vielmehr werden umfangreiche Erweiterungen der Komponentenbibliothek des Modelleditors COMEDI erforderlich sein, um die Eigenschaften der bei STILL verwendeten elektronischen Bauteile bzw. Baugruppen hinreichend gut beschreiben zu können. Hierbei werden die Kombination von elementaren zu zusammengesetzten Komponenten ebenso eine Rolle spielen wie die Darstellung in abstrakten Tabellenmodellen unter Verwendung des in COMEDI integrierten Component Model Builders (s. Kapitel II-2.3.4 u. II-2.3.5). Die Entscheidung, auf welcher Abstraktionsebene die Modellbildung sinnvoll ist, wird dabei vom Einzelfall abhängen.

Um die Möglichkeit der Erzeugung meßkostenoptimierter Fehlerbäume einsetzen zu können, müssen zunächst die Kosten für die erforderlichen Messungen definiert werden. Es ist hierbei z.B. denkbar, die für die Durchführung manueller Messungen benötigte Zeit als Kriterium zu verwenden. Soweit hierfür bereits festgelegte Vorgaben existieren, können diese für die Bewertung der Kosten übernommen werden. Andernfalls sind im Vorfeld der Kostenoptimierung noch erhebliche Aufwände nötig, um eine Klassifizierung der manuellen Messungen zu erstellen. Automatische Messungen sind dagegen immer mit einem sehr niedrigen Kostenfaktor anzusetzen, da sie in der Regel in Bruchteilen einer Sekunde ablaufen und nur in Ausnahmefällen eine Dauer von einigen Sekunden aufweisen. Hier stellt sich die Frage, ob eine weitere Unterteilung den Aufwand rechtfertigt oder ob automatische Messungen nicht generell mit einem pauschalen Mittelwert angesetzt werden sollten.

Wie bereits am Anfang dieses Kapitels angemerkt, ist die praktische Anwendbarkeit des modellbasierten Ansatzes auch von dem bei Projektabschluß erreichten Stand der entwickelten Software abhängig. Hierbei ist festzustellen, daß trotz der prinzipiellen Möglichkeiten der Software, was die CAD-ähnliche Eingabe des Modells, die Erzeugung der Fehlerrelationen und letztlich den Import der Fehlerbäume in das vorhandene System DiaMon betrifft, der tatsächliche Stand der Software nach wie vor ein prototypischer ist. Um hieraus ein Softwarepaket abzuleiten, das den üblichen Anforderungen nach Stabilität, Bedienungskomfort und Erscheinungsbild der graphischen Oberfläche entspricht, wäre ein erheblicher zusätzlicher Entwicklungsaufwand erforderlich, bevor überhaupt an eine gezielte Einführung des Systems im Hause STILL zu denken ist.

Weiterhin läßt sich derzeit keine Einschätzung abgeben, ob die Komplexität des Gesamtsystems „elektrische Komponenten des Gabelstaplers“ grundsätzlich mit den entwickelten Methoden beherrschbar ist. Sämtliche Evaluierungen wurden aus Gründen der Übersicht anhand im Vorfeld festgelegter Leitbeispiele vorgenommen, um die prinzipielle Leistungsfähigkeit zu zeigen. Alle Leitbeispiele bestanden jedoch aus nur wenigen und dazu einfachen Bauteilen. Aus den für diese Beispiele gewonnenen Ergebnissen Rückschlüsse ziehen zu wollen für Baugruppen mit Hunderten von Komponenten, die ihrerseits teilweise sehr komplex sind (man denke nur an Mikrocontroller und ähnliche Bauteile), erscheint fragwürdig. Auch die Möglichkeit der Abstraktion auf höherer Ebene durch Verwendung abstrakter Tabellenmodelle einer nach außen als „Black Box“ erscheinenden Komponente garantiert ohne weitere eingehende Untersuchungen letztendlich nicht den Erfolg. Für periphere Baugruppen geringer Komplexität (vgl. das Beispiel Fahrgeber in den Kapiteln II-1 u. II-2) scheint das Verfahren dagegen anwendbar zu sein. Immerhin enthalten auch die manuell für solche wenig komplexen Baugruppen erstellten Fehlerbäume oft bereits einige Dutzend Objekte (Fehler, Meßgrößen, Messungen, Anweisungen usw., s. Kapitel II-3.2.1), so daß auch eine Erleichterung der Bearbeitung solcher Komponenten willkommen wäre. Der modellbasierte Ansatz ist als Ergänzung des bisherigen manuellen Verfahrens bei der Erstellung von Fehlerbäumen durchaus vielversprechend.

Der entscheidende Nachteil des modellbasierten Ansatzes ist jedoch die Tatsache, daß die für die Modellbildung erforderlichen Daten nicht, wie in den Anforderungen an Diagnose formuliert, im laufenden Entwicklungsprozeß quasi als Nebenprodukt automatisch anfallen bzw., daß die zur Verfügung stehenden Daten (Schaltpläne, CAD-Daten etc.) in ihrer vorliegenden Form nicht für die Verwendung zusammen mit dem MAD-System

geeignet sind. Dieser Nachteil ist also weniger bei den im Rahmen von INDIA entwickelten Methoden zu suchen als bei den im Hause STILL in der Entwicklung verwendeten Softwarepakete und Entwicklungsverfahren. Er würde jedoch zur Folge haben, daß für einen Einsatz der modellbasierten Verfahren eine auf dem üblichen Weg entwickelte Baugruppe ein zweites Mal auf einer ggf. völlig anderen Abstraktionsebene nunmehr mit Hilfe des Modelleditors COMEDI bearbeitet werden müßte. Der Bearbeitungsaufwand würde sich schlichtweg verdoppeln. Diese Tatsache zusammen mit dem bereits erwähnten Sachverhalt, daß diese Modellierungsarbeit nach derzeitigem Stand nur in der Entwicklungsabteilung stattfinden könnte, ist ein klares Argument gegen den Einsatz der in INDIA entwickelten Verfahren bei STILL. Erst wenn Ansätze zur Verfügung stehen, die diesen Konflikt zu entschärfen vermögen, könnte eine Einführung modellbasierter Verfahren in Erwägung gezogen werden.

4.3 Zusammenfassung

Die Ergebnisse der Hamburger Säule im Rahmen des Forschungsprojektes INDIA haben gezeigt, daß die modellbasierten Verfahren einen leistungsfähigen Ansatz für eine verbesserte Diagnose darstellen. Insbesondere die Partner LKI und SXP konnten umfangreiches Wissen erlangen, das sich für die Arbeit an zukünftigen Projekten als wertvoll erweisen kann. Für den Anwender STILL zeichnet sich aus den angegebenen Gründen derzeit kein konkreter Ansatzpunkt für eine Verwendung der entwickelten Verfahren ab. Jedoch ist für die Zukunft durchaus denkbar, die Methoden von INDIA in kleinerem Umfang auf abgeschlossene, wenig komplexe Baugruppen als Ergänzung der bisherigen manuellen Fehlerbaumerstellung anzuwenden. Erfolgversprechend sind jedoch besonders die Ergebnisse, die bei den Untersuchungen zur Integration von Produktdaten (wie sie bei Online-Servicedokumentationen und -Ersatzteilkatalogen vorkommen) in das Diagnosesystem gewonnen wurden.

5 Literatur Teil II

[Cascio et al. 1999]

Cascio, F., Console, L., Guagliumi, M., Osella, M.,Panati, A., Sottano, S., and Theseider Dupré, D.: On-board diagnosis of automotive systems: from dynamic qualitative diagnosis to decision trees, *IJCAI-99, Workshop on Qualitative Reasoning for Complex Systems and their Control*,1999.

[deKleer et al. 1984]

de Kleer, J., and Brown, J. S.: A Qualitative Physics Based on Confluences, in: *AI Journal*, 1984.

[Faure et al., 1999]

Faure, P.-P., Trave-Massuyes, L., and Poulard, H.: An Interval Model-Based Approach for Optimal Diagnosis Tree Generation, in*: Proc. DX-99, 10th International Workshop on Principles of Diagnosis*, 1999.

[Guckenbiehl et al., 1999]

Guckenbiehl, T., Milde, H., Neumann, B., and Struss, P.: Meeting Re-use Requirements of Real-Life Diagnosis Applications, in: Proc. *XPS-99: Knowledge-Based Systems*, Springer, LNAI 1570, pp. 90-100, 1999. .

[Malik et al., 1996]

Malik, A., and Struss, P.: Diagnosis of Dynamic Systems Does Not Necessarily Require Simulation. In*: Workshop Notes of the 10th International Workshop on Qualitative Reasoning QR-96*, AAAI Press, pp. 127-136, 1996. .

[Mauss et al., 1996]

Mauss, J., and Neumann, B.: How to Guide Qualitative Reasoning about Electrical Circuits by Series-Parallel Trees. In *Proceedings of the tenth International Workshop on Qualitative Reasoning (QR'96)*, 1996.

[Mauss 1998]

Mauss, J.: *Analyse kompositionaler Modelle durch Serien-Parallel-Stern Aggregation*, DISKI 183, Dissertationen zur Künstlichen Intelligenz, 1998.

[Milde et al. 1997]

Milde, H., Hotz, L., Möller, R., and Neumann, B.:Resistive Networks Revisited: Exploitation of Network Structures and Qualitative Reasoning about Deviation is the Key. *8th International Workshop on Principles of Diagnosis (DX'97*), 1997.

[Milde et al. 1999]

Milde, H., Hotz, L., Kahl, J., Neumann, B., and Wessel, S.: Qualitative Analysis of Electrical Circuits for Computer-based Diagnostic Decision Tree Generation, in*: Proc. DX-99, 10th International Workshop on Principles of Diagnosis*, 1999.

[Milde et al. 2000]

Milde, H., and Hotz, L.: Facing Diagnosis Reality - Model-based fault Tree Generation in Industrial Application, submitted *to DX-00, 11th International Workshop on Principles of Diagnosis*, 2000.

[Murthy 1997]

Murthy, K. V. S.: *On Growing Better Decision Trees from Data.* PhD. Diss., Johns Hopkins University, 1997.

[Norton 1989]

Norton, S. W.: Generating better decision trees. *In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 800-805. Michigan: International Joint Conferences on Artificial Intelligence, Inc., 1989.

[Pugh et al. 1996]

Pugh, d., and Snooke, N.: Dynamic Analysis of Qualitative Circuits, in*: Proc. of Annual Reliability and Maintainability Symposium*, 37 - 42, IEEE Press, 1996.

[Price et al. 1996a]

Price, C., and Pugh, D.: Interpreting Simulation with Functional Labels. In *Proceedings of the tenth International Workshop on Qualitative Reasoning about Physical Systems (QR'96.)*, 1996.

[Price et al. 1996b]

Price, C., Wilson M., Timmis, J., and Cain, C.: Generating Fault Trees from FMEA. In *Proceedings of the seventh International Workshop on Principles of Diagnosis (DX'96)*, S. Abu-Hakima, eds., 1996.

[Quinlan 1986]

Quinlan J. R.: Induction of Decision Trees, *Machine Learning*, I:81-107, 1986.

[Russel et al. 1995]

Russel, S., and Norvig, P.: *Artificial Intelligence A Modern Approach*, Prentice Hall, 1995.

[Struss 1990]

Struss, P.: Problems of Interval-based Qualitative Reasoning: in*: Readings in Qualitative Reasoning about Physical Systems*, Weld, D., de Kleer, J. (Eds.), Morgan Kaufmann, 1990.

[Struss 1994]

Struss, P.: Testing for Discrimination of Diagnoses. In Working Papers of the fifth International Workshop on Principles of Diagnosis (DX-94), New Paltz, USA, 1994.

[Struss et al. 1995]

Struss, P., Malik, A., and Sachenbacher, M.: Qualitative Modelling is the Key. In *Proceedings of the sixth International Workshop on Principles of Diagnosis (DX'95)*, 1995.

[Zink 1996]

Zink, B.: *Manual for developing a DIM knowledge base* (in German), Report, Mikroelektronik Anwendungszentrum Hamburg GmbH (MAZ), 1996.

Teil III

Überwachung und Diagnose in Färbereianlagen

Der Maschinen- und Anlagenbau ist einer der größten deutschen Industriezweige, mit hohem Exportanteil. Diagnosesystemen kommt hier eine wachsende Bedeutung zu, da sich störungsbedingte Ausfälle direkt auf die Produktion auswirken. Aufgabe der „THEN-Säule“ in INDIA war es daher, die Möglichkeiten wissensbasierter Diagnose in diesem Bereich zu demonstrieren, vorhandene Probleme zu erfassen und geeignete Lösungsansätze zu entwickeln. Anwendungsgegenstand waren dabei Maschinen und Versorgungsanlagen für Färbereien, die von der THEN GmbH hergestellt werden. Die R.O.S.E Informatik GmbH und das Fraunhofer-Institut für Informations- und Datenverarbeitung IITB entwickelten hierfür prototypische Diagnosesysteme. Weiterhin stand die Reduktion der Kosten zur industriellen Einführung solcher Systeme im Vordergrund. Dieser dritte Teil stellt Anwendungsfeld und Ergebnisse der THEN-Säule vor.

1 Die „THEN-Säule“ im Überblick

Thomas Guckenbiehl - Fraunhofer Institut für Informations- u. Datenverarbeitung

Im Kampf um Kostenvorteile in der Produktion gewinnt die Reduktion von Qualitätsmängeln und Ausfallzeiten für die Produzenten immer stärkere Bedeutung. In zunehmendem Maß verlangen sie dabei Unterstützung durch die Hersteller der Produktionsmaschinen und -anlagen. Für diese wird daher die Unterstützung bei der Fehlersuche und -beseitigung ein immer wichtigerer Wettbewerbs-, aber auch Kostenfaktor. Gleichzeitig fordern auch verschärfte Vorschriften zur Produkthaftung von den Anlagenherstellern die Validierung der Betriebs- und Ausfallsicherheit ihrer Systeme. Die Integration leistungsfähiger Überwachungs- und Diagnosesoftware in ihre Produkte ist damit ein wichtiger Schritt zur Stärkung ihrer Wettbewerbsfähigkeit auf den internationalen Märkten.

Aufgrund dieses Bedarfs und der großen Bedeutung des Maschine n-und Anlagenbaus in Deutschland war es für INDIA besonders interessant, die Verwendung wissensbasierter Überwachungs- und Diagnosetechniken in dieser Branche zu demonstrieren. Anwendungspartner war dabei die THEN Maschinen- und Apparatebau GmbH, Schwäbisch-Hall, ein Hersteller von Versorgungseinrichtungen und Maschinen für Färbereien mit ca. 240 Mitarbeitern und über 70 Mio DM Jahresumsatz. Als Systemhaus war die R.O.S.E. Informatik GmbH in Heidenheim (kurz: RIG) beteiligt. Das Fraunhofer Institut für Informations- und Datenverarbeitung IITB in Karlsruhe begleitete die Arbeiten aus Sicht der anwendungsorientierten Forschung. Anwendungsgegenstand war zum einen die Färbemaschine THEN Airflow AFS, zum anderen der Chemikaliendistributor THEN CHD.

Die Airflow AFS ist eine Maschine zum Färben von Stoffbahnen, die aufgrund ihrer Konstruktion sehr viel weniger Wasser verbraucht als herkömmliche Strangfärbemaschinen. Die Maschine ist mit relativ vielen Sensoren ausgestattet, die zwar eine Diagnose erleichtern, aber die wissensbasierte Überwachung aufgrund der großen Menge anfallender Daten erschweren. Bei der Realisierung eines entsprechenden Überwachungs- und Diagnosesystems stand daher insbesondere die rechtzeitige Erfassung aller relevanter Prozeßwerte im Vordergrund.

Der Chemikaliendistributor THEN CHD steuert das Abmessen der für eine Färbung benötigten Chemikalien und den Transport von den Chemikalientanks über Rohrleitungssysteme bis zur richtigen Färbemaschine. Im Gegensatz zur Färbemaschine steht hier für die Diagnose im wesentlichen nur ein Sensor zur Verfügung, bei Bedarf müssen Beobachtungen des Bedienpersonals miteinbezogen werden. Ein zweites Problem ergibt sich daraus, daß jeder Chemikaliendistributor im Prinzip ein Einzelstück ist, weil die Topologie der Rohrleitungen in jeder Färberei anders ist. Das Diagnosesystem muß also auf jede Anlage zugeschnitten werden ohne daß die Entwicklungskosten den Rahmen des wirtschaftlich Sinnvollen sprengen. Gerade hier zeigte der modellbasierte Ansatz seine Stärken.

Beide Anwendungen, die verwendeten Lösungsansätze und die erzielten Ergebnisse werden in den folgenden Abschnitten noch näher beschrieben. Abschnitt III-2 skizziert zunächst das Anwendungsfeld „Färbereianlagen“ und die spezifischen Anforderungen an Überwachungs- und Diagnosesysteme in diesem Bereich. Abschnitt III- 3 schildert dann die Arbeiten an der Anwendung Airflow AFS, die vorwiegend von THEN und RIG durchgeführt wurden. Die bei der Diagnose des Chemikaliendistributors erzielten Ergebnisse werden in Abschnitt III-4 dargestellt. Federführend war hier Fraunhofer IITB.

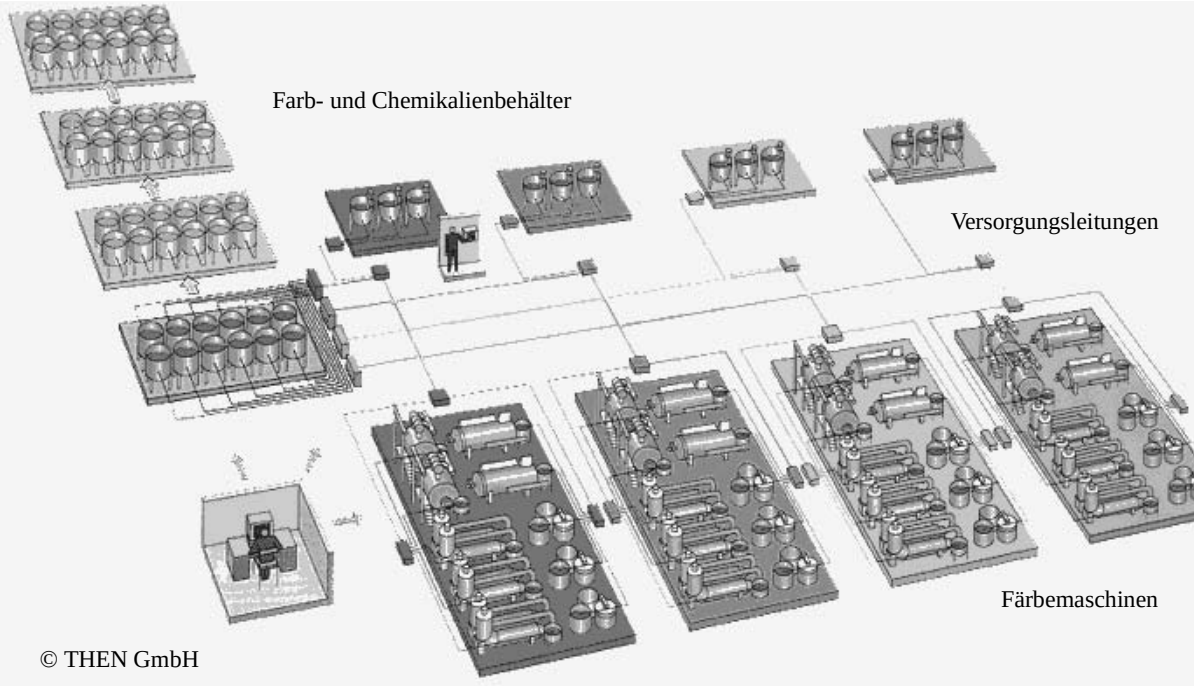


Abbildung 2-1: Schema einer Färberei

2 Das Anwendungsfeld „Färbereianlagen“

Thomas Guckenbiehl - Fraunhofer Institut für Informations- u. Datenverarbeitung

Abbildung 2-1 zeigt das Grundschema einer Färberei. Gefärbt werden Stoffe oder Garne dabei in hochtechnisierten Maschinen, wie sie THEN herstellt. Ein Beispiel dafür ist die THEN Airflow AFS, in der die zu färbende Ware als endloser Strang durch einen Luftstrom stetig umlaufend bewegt wird. Über eine Düse wird dabei die sogenannte *Flotte* auf die Ware aufgetragen, ein Gemisch aus Farbe, Wasser und Chemikalien. Um bestimmte Farbeffekte oder Griffeigenschaften zu erzielen, müssen Druck und Temperatur der Flotte jeweils entlang spezifischer Kurven gesteuert werden. Dabei werden in der Regel drei- bis viermal verschiedene Mischungen von Farben oder Chemikalien zugesetzt, z. B. um die Textilfasern für die Färbung vorzubereiten oder nach der Färbung zu versiegeln.

Aufgabe des Chemikaliendistributors CHD ist es, die richtige Mischung zum richtigen Zeitpunkt an der richtigen Maschine bereitzustellen. Den Auftrag dazu (*Chemikalienabruf*) erhält er entweder direkt von einem Leitsystem oder über ein Bedienpult vom Färbereipersonal. Die CHD mißt die einzelnen Chemikalienmengen aus den betreffenden Tanks ab, stellt sie zu einem „Chemikalienzug“ zusammen und transportiert sie durch das Leitungsnetz zur betreffenden Färbemaschine. Durch entsprechende Stellsignale an die Ventile an den Vereinigungspunkten (Sammler) und Verzweigungspunkten (Verteiler) im Leitungsnetz schaltet es dazu jeweils den richtigen Weg frei. Zum Transport wird teilweise Wasser und teilweise Druckluft verwendet. Abbildung 2-2 skizziert die Struktur des Weges von den Tanks über verschiedene Sammler und Verteiler bis zu den Färbemaschinen.

Störungen im Färbeprozeß führen z. B. zu ungleichmäßiger oder falscher Färbung. Häufig ist dann die gesamte Ware im Wert von bis zu mehreren zehntausend DM unbrauchbar. In anderen Fällen sind Reparaturschritte möglich, die jedoch in der Regel teuer sind, eine zusätzliche Umweltbelastung bedeuten und trotzdem Qualitätseinbußen oft nicht vollständig vermeiden können.

THEN ist daher zum einen an Überwachungssystemen interessiert, mit deren Hilfe Störungen möglichst vermieden oder zumindest frühzeitig erkannt werden können. Verschleißerscheinungen sollen z. B. im Idealfall automatisch erkannt und über WAN an das Kundendienstzentrum weitergemeldet werden, das dann noch vor Ausfällen entsprechende Instandhaltungsmaßnahmen einleiten kann. Verglichen mit zeitorientierten Instandhaltungsstrategien nutzt eine solche zustandsorientierte Strategie das Betriebspotential einer Anlage sehr viel besser aus und steigert dadurch deren Verfügbarkeit.

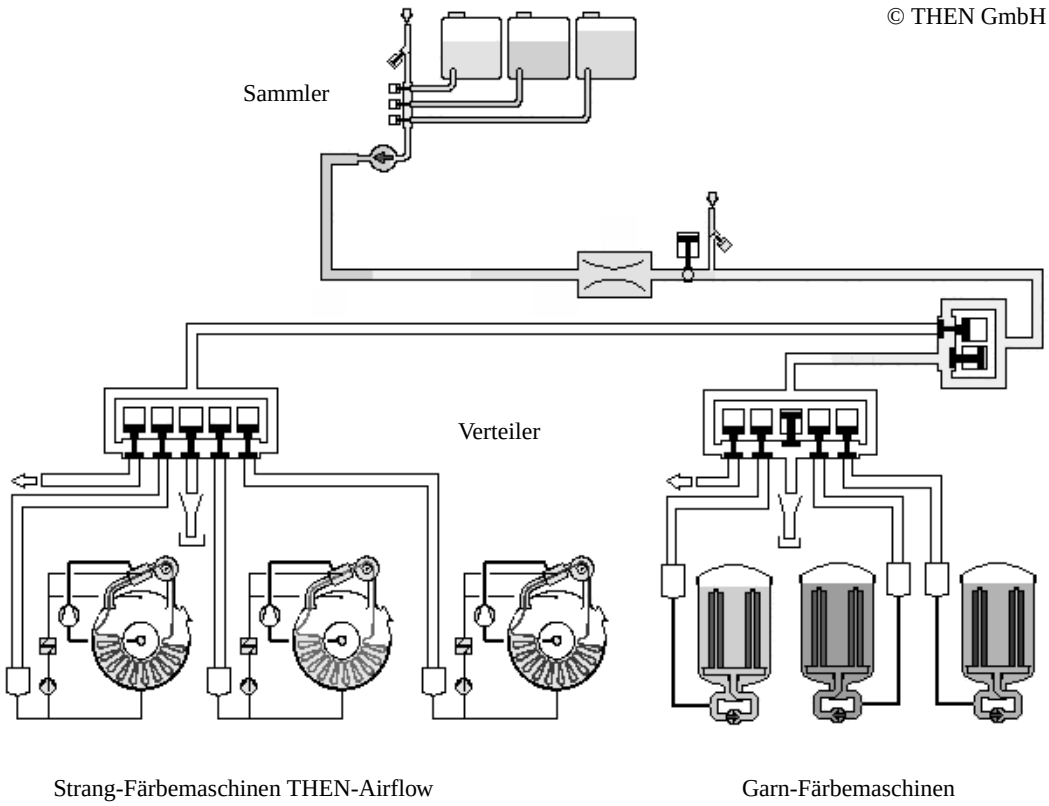


Abbildung 2-2: Struktur des Netzes zur Versorgung von Färbemaschinen mit Chemikalien

Zum anderen möchte THEN Diagnoseverfahren, mit deren Hilfe die Störungsursache so schnell wie möglich gefunden und behoben werden kann, um den Produktionsausfall durch Maschinenstillstand zu minimieren. Zeit wird insbesondere dann gespart, wenn schon das Bedienpersonal den Fehler finden kann und nicht erst ein auswärtiger Servicetechniker gerufen werden muß. Da aber viele Färbereien aus Kostengründen dazu übergehen, vermehrt ungeübtes oder schlechter ausgebildetes Bedienpersonal einzusetzen, müssen die zur Diagnose benötigten Informationen durch wissensbasierte Assistenzsysteme zurVerfügung gestellt werden.

Aufgrund dieser Anforderungen wurde das in Abbildung 2-3 skizzierte Einsatzszenario entworfen, das in diesem Umfang zwar nicht realisiert werden sollte, aber als Orientierungsrahmen diente. In diesem Rahmen entstanden

- ein Überwachungs- und Diagnosesystem für die Färbemaschine THEN Airflow AFS mit Online-Verbindung zur Datenaufnahme an der Anlage,
- ein Diagnoseassistent für den Chemikaliendistributor CHD, der über Internet aus einem Servicezentrum heraus bedient werden kann.

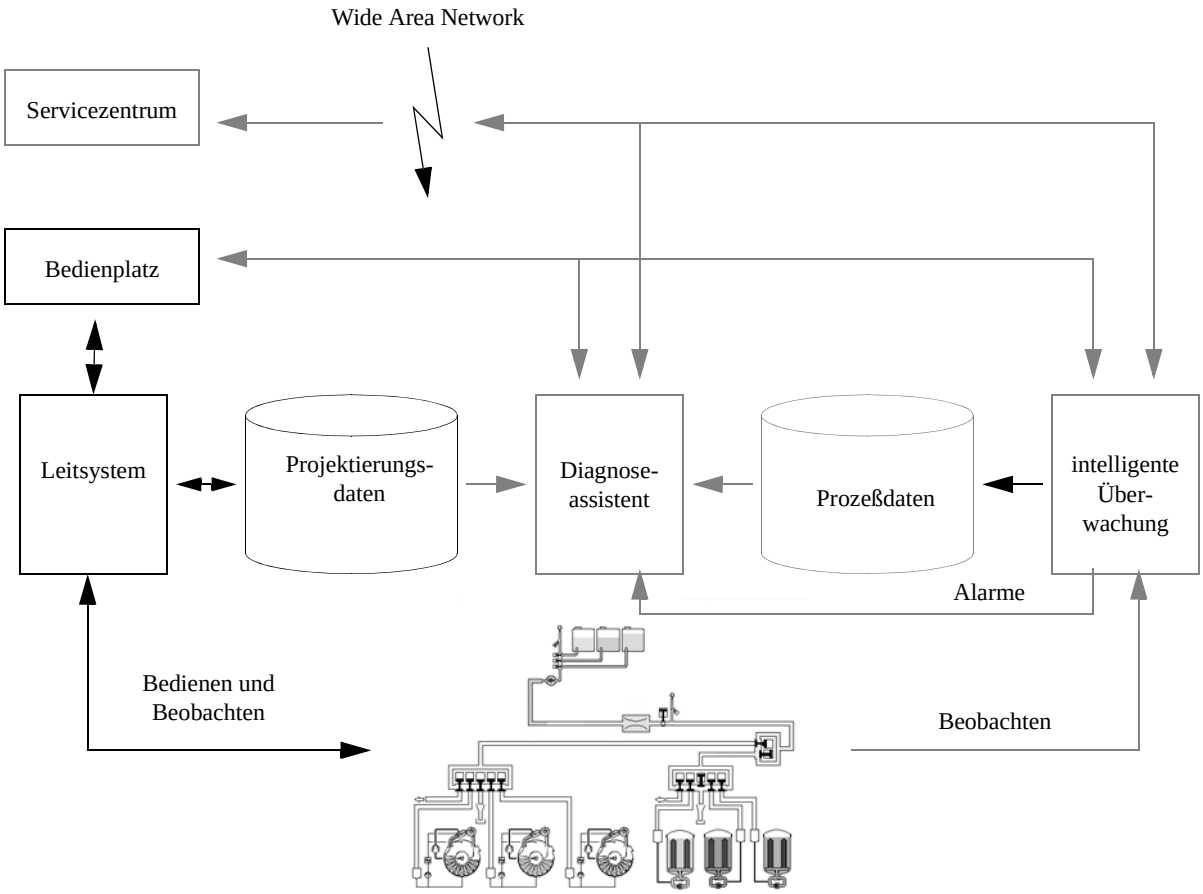


Abbildung 2-3: Einsatz-Szenario in der THEN-Anwendung (Gegenstand in INDIA waren die hellgrauen Teile)

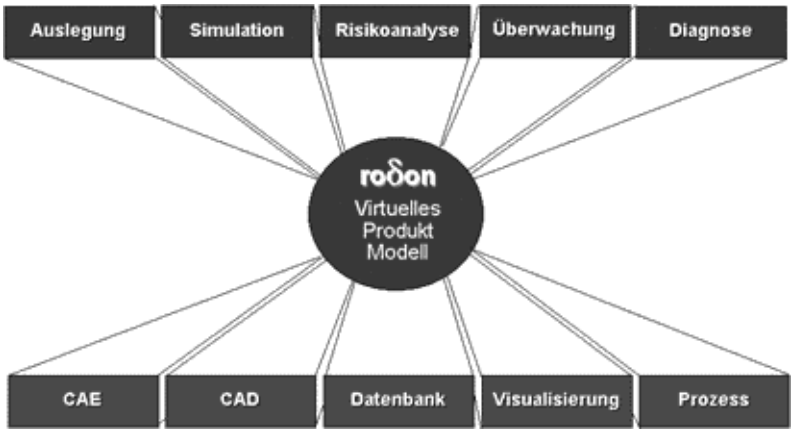


Abbildung 3-1: Einordnung eines virtuellen Produktmodells

3 Diagnose an einer Färbemaschine mit RODON[1]

Burkhard Münker - R.O.S.E. Informatik GmbH, Heidenheim

3.1 Das Diagnosewerkzeug RODON

3.1.1 Motivation und Anwendungsfelder

Die modellbasierte Analysesoftware RODON der Firma R.O.S.E. Informatik GmbH (RIG) dient der Simulation, Auslegung, Risikoanalyse, Überwachung und Diagnose komplexer technischer Systeme. Sie bietet dem Ingenieur eine produktlebenszyklusbegleitende Unterstützung vom Entwurf über die Qualitätssicherung bis hin zur Wartung. Wissen aus verschiedenen Bereichen der Entwicklung wird dabei zu einem virtuellen Produktmodell (VPM) aggregiert (Abbildung 3-1), das sowohl Struktur als auch Verhalten beschreibt. Dieses Modell erlaubt mittels Simulationsrechnungen automatisch fundierte Aussagen über das Zusammenspiel der Komponenten in unterschiedlichen Betriebs- und Fehlerzuständen herzuleiten.

RODON automatisiert und integriert u.a. folgende Bereiche:

- Entwurfsverifikation
- Risikoanalyse (FMEA, Fehlerbäume)
- Prozeßüberwachung
- Modellbasierte Diagnose
- Automatische Generierung von Entscheidungsbäumen zur Offboard-Diagnose
- Automatische Generierung von Diagnoseregelsystemen zur Onboard-Diagnose

Die erste Pilotanwendung von RODON lag im Anlagenbau.Von dort kommen auch wichtige Anforderungen wie

1. modellbasierte Diagnose in Echtzeit,
2. Modellerzeugung aus Bauunterlagen,
3. Konfigurieren des Modells „auf der Baustelle“ und
4. Präventive Diagnose.

1.

Wie sich gezeigt hat sind diese Technologien auch für unsere Kunden im Automobil- und im Luft- und Raumfahrtbereich von rasch wachsender Bedeutung. Das INDIA-Projekt hat maßgeblich dazu beigetragen, diese Anwendungen zu erschließen (siehe Kapitel 3.2). RODON kann darüberhinaus aber prinzipiell in allen Branchen genutzt werden, „von der Waschmaschine bis zur Raumstation“.

3.1.2 Technologie und Funktionalität

RODON ist ein Expertensystem der zweiten Generation. Es unterscheidet sich von den meisten zur Zeit auf dem Markt angebotenen Diagnose- und Analysetools durch die konsequente Verarbeitung funktionalen Verhaltenswissens über das zu untersuchende System. Funktionales Wissen wird in RODON an Komponenten gebunden, die über Ports und Verbindungen miteinander kommunizieren. Das Abstraktionsniveau der funktionalen Beschreibung liegt im Ermessen des Modellierers und kann physikalische Gesetze oder auch vom Übertragungsprotokoll abstrahierende Signalwerte beinhalten. Die kontextfreie bauteilorientierte Modellierung erlaubt die Erzeugung wiederverwertbarer Komponentenbibliotheken, die in RODON in objektorientierter Weise hierarchisch verwaltet werden (vgl. Abschnitt 3.2.2). Steht eine geeignete Bibliothek zur Verfügung, reduziert sich die Modellierung eines konkreten Systems auf das Zusammenfügen der Bibliothekskomponenten entsprechend der Systemstruktur, was wiederum mit Hilfe von CAD-Interfaces automatisiert werden kann.

Das Simulationsmodul von RODON verbindet Intervallmengen-Arithmetik mit Constrainttechnologie und numerischen Iterationsverfahren. Dieses Lösungsverfahren erlaubt eine ungerichtete Modellierung, die vom Kontrollfluß der Berechnung abstrahiert und in der Lage ist, mit unterschiedlichen Wertvorgaben Berechnungen durchzuführen. Der Einsatz eines leistungsstarken ATMS ermöglicht dem Simulationsmodul eine effektive Fehlereingrenzung bei Auftreten eines Widerspruchs zwischen vorgegebenen und berechneten Werten. Neben dem Einsatz des ATMS zum Zweck der Kandidateneinschränkung in der modellbasierten Diagnose oder zum Aufspüren von Modellierungsfehlern dient es dem Simulationsmodul auch zur Berechnungsbeschleunigung, wenn die Aufgabe darin besteht, eine Menge von ähnlichen Zuständen zu bearbeiten.

Für die Zuverlässigkeit von Aussagen, die auf der Basis einer Modellvorstellung über ein reales System hergeleitet werden, spielt die Behandlung von Unsicherheit und Unschärfe eine entscheidende Rolle. RODON kommt diesem Umstand nach, indem es auf eine flexible Intervallmengenarithmetik zurückgreift. Sie erlaubt neben der Berechnung scharfer Werte auch die Verarbeitung von Wertebereichen, Mengen verschiedener möglicher Werte, sowie der Information, daß über eine bestimmte Größe gar nichts bekannt ist.

RODON bietet die Möglichkeit, Simulationen für einen zuvor spezifizierten Zustandsraum automatisch durchzuführen und die Ergebnisse in einer Verhaltenswissensbasis bzw. Zustands-Datenbank abzulegen. Ist die Vorgabe hinreichend allgemein, so kann aus den vorhandenen (beispielhaften) Simulationsergebnissen induktiv auf Gesetzmäßigkeiten im Gesamtverhalten des Systems geschlossen werden. RODON verwendet zur Auswertung Algorithmen aus dem Forschungsgebiet des Maschinellen Lernens. Diese Eigenschaft wird vor allem bei der Generierung von Diagnosewissen für die On- und Offboard-Diagnose nutzbringend eingesetzt.

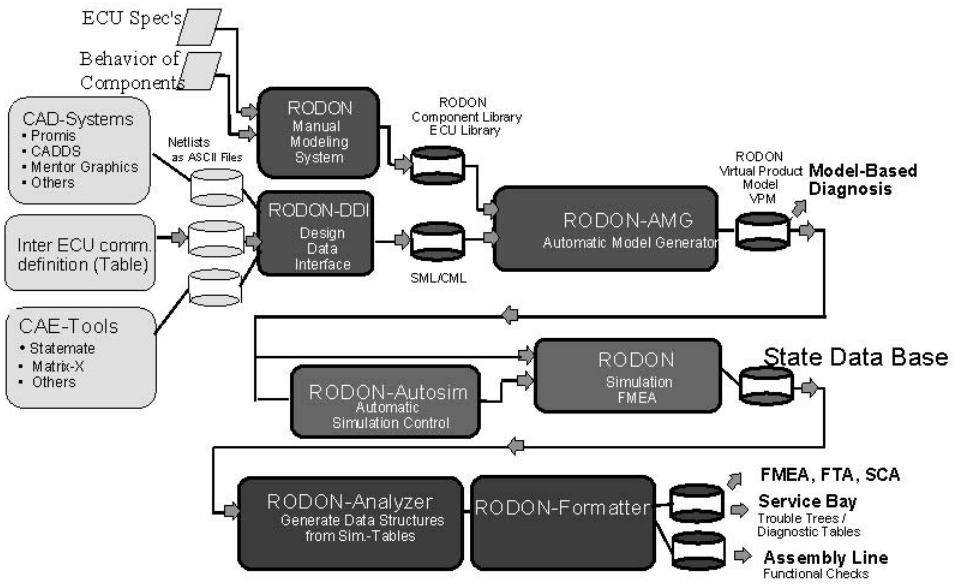


Abbildung 3-2: Zusammenspiel der verschiedenen RODON -Komponenten

3.1.3 Eigenschaften und Verfügbarkeit

Die folgende Aufstellung faßt die wesentlichen Merkmale von RODON zusammen. Abbildung 3-2 veranschaulicht darüberhinaus das Zusammenspiel der verschiedenen Teilmodule innerhalb des Systems.

- RODON verfügt über eine zeitgemäße graphische Oberfläche mit zweidimensionaler System-Blockdarstellung und zahlreichen Bedienelementen und Editoren.
- RODON unterstützt deklaratives Modellieren auf hohem Abstraktionsniveau.
- Für RODON sind zahlreiche Schnittstellen zu CAD- und CAE-Tools (z.B. CADDS, MatrixX) erhältlich.
- RODON ist mit einem Application Programming Interface (API) ausgestattet, welches neben der interaktiven Arbeitsweise die Programmierung und vollautomatische Steuerung ganzer Arbeitsgänge ermöglicht.
- Die RODON Ausgabeformate für Analyseergebnisse (z.B. Entscheidungsbäume, FMEA-Tabellen) können vom Anwender über eine flexible Definitionssprache den individuellen Wünschen angepaßt werden.
- RODON ist zur Zeit verfügbar auf Solaris, HP-Unix und Linux.

3.2 Anwendung an der AFS-Färbemaschine

Im Rahmen des INDIA-Projektes bezogen sich die Arbeiten konkret auf eine neuartige Färbemaschine der Firma THEN, Schwäbisch Hall.

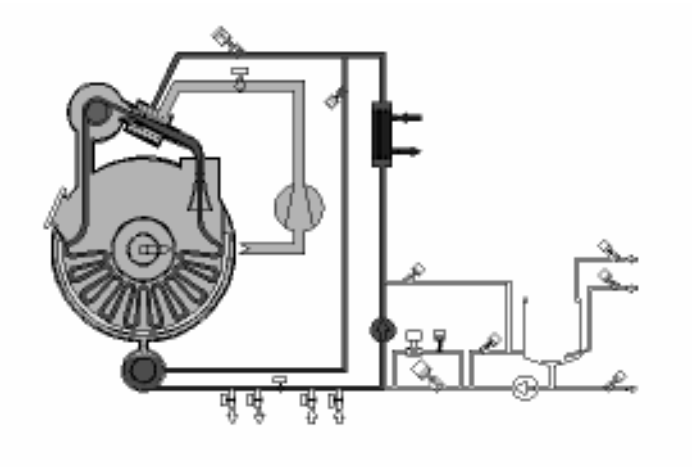


Abbildung 3-3: Schematischer Aufbau der Färbemaschine Airflow AFS

3.2.1 Funktionsweise der AFS-Färbemaschine

Bei der Maschine Airflow AFS handelt es sich um eine Hochtemperatur-Düsenfärbemaschine. Die zu färbende Ware wird darin als endloser Strang durch einen Luftstrom stetig umlaufend bewegt und dabei gleichzeitig mit über eine Düse injizierten Farbstoff durchsetzt. Abbildung 3-3 zeigt schematisch den Aufbau dieser Anlage

Man erkennt in der Abbildung links den kreisrunden Färbekessel, in dem die Ware umläuft, rechts einen kleineren Zusatzbehälter sowie vereinfacht das Rohr- und Ventilsystem der Anlage. Im Rahmen eines Färbeprozesses wird in dem Zusatzbehälter das Gemisch von Wasser und Farbstoff bzw. Chemikalien auf die richtige Zusammensetzung eingestellt und mittels eingeleitetem Dampf auf die erforderliche Temperatur gebracht. Durch entsprechende Ansteuerung der Pumpen und Ventile wird dieses Gemisch, die sog. „Flotte“, über das Rohrleitungssystem zu einer Düse oben im Färbekessel gefördert und dort direkt und unter hohem Druck auf die zu färbende Ware aufgebracht. Die überschüssige Flotte sammelt sich unten im Färbekessel und wird zurück in den Kreislauf gepumpt. Ist eine ausreichende Flottenmenge im Kreislauf-System vorhanden, so wird der Zusatzbehälters im weiteren Verlauf des Färbeprozesses noch zur Korrektur der Gemischzusammensetzung oder aber zur Einbringung weiterer Stoffe für die Faserbehandlung benötigt.

Nicht zuletzt aufgrund der oft sehr wertvollen Färbeware kommt einer zuverlässigen Prozeßführung und Überwachung an der Färbemaschine eine zentrale Rolle zu. Von großer Bedeutung ist eine leistungsfähige Diagnosemethodik im Falle, wenn bereits ein Defekt eingetreten ist, da sich durch eine von der Bedienung her einfache, schnelle und richtige Fehleridentifikation für den Anlagenhersteller Technikerreisen evtl. vermeiden und durch eine schnelle Reparatur für den Anlagenbetreiber teure Stillstandszeiten verkürzen lassen.

Charakteristisch an dieser Anwendung ist die Vielzahl rein maschinenbaulicher Komponenten mit den in ihnen ablaufenden hydraulischen, pneumatischen, thermischen oder mechanischen Vorgängen. Eine scharfe Grenze zwischen diesen Domänen kann jedoch üblicherweise nicht gezogen werden, da in einzelnen physikalischen Bauteilen typischerweise mehrere Effekte auftreten. So handelt es sich beispielsweise bei den hier eingesetzten Ventilen um eine Bauart, bei der die Stellung des hydraulischen Hauptventils durch ein pneumatisch angesteuertes Pilotventil mechanisch vorgegeben wird.

3.2.2 Modellierung des Systems mit RODON

3.2.2.1 Modellierungsprinzip

In RODON erfolgt die Modellierung rein komponentenorientiert. Das bedeutet, das Wissen über eine bestimmte physikalisch-technische Funktion eines realen Systems wird im Modell genau derjenigen technischen Komponente zugeordnet, die für diese Funktion verantwortlich ist. Dabei wird konsequent das sogenannte „No Function In Structure“-Prinzip („Nfis“-Prinzip) [deKleer und Brown, 1984] eingehalten, um eine gute Wiederverwendbarkeit der erstellten Teilmodelle zu gewährleisten.

Das Funktionsverhalten jeder einzelnen Komponente wird in Form von linearen oder nichtlinearen Kennlinien bzw. Kennfelder formuliert, zum einen für die normale, zum anderen aber ggf. auch für die gestörte Arbeitsweise (vgl. Abschnitt 3.2.2.3). Diese Kennlinien beschreiben den Zusammenhang zwischen den Eingangs- und Ausgangswerten einer Komponente auf mathematischer und/oder logischer Ebene. Diese Darstellung ist rein deklarativ und dem Ingenieur z.B. in Form von Ventil- oder Pumpenkennlinien bereits vertraut.

Die so erstellten Komponentenmodelle sind in einer objektorientierten Klassenhierarchie strukturiert. Zur Modellierung des realen Systems werden die benötigten Komponentenobjekte aus diesen Modellklassen instantiiert und in einem speziellen Systemeditor grafisch miteinander verknüpft. Durch die Bildung von Systemhierarchien ist es möglich, die auch real vorzufindende Strukturierung einer technischen Anlage (Stückliste) in austauschbare Teilmodule auch im Modell unmittelbar abzubilden (vgl. Abschnitt 3.2.2.4).

Das modellierte Betriebsverhalten der Gesamtanlage ergibt sich schließlich aus der Vernetzung aller logisch-mathematischen Beziehungen, die alle auftretenden Funktionseinheiten und Komponenten in das Gesamtmodell einbringen. Diese Verkopplung führt RODON vollautomatisch durch, so daß sich das Modell unmittelbar auswerten läßt.

3.2.2.2 Strukturierung der Klassenhierarchie

Zur Modellierung der hier betrachteten Färbemaschine war es erforderlich, zunächst einen Überblick über die zu modellierenden relevanten technischen Bauteile zu gewinnen, diese zu klassifizieren und entlang einer geeigneten Hierarchie zu strukturieren. Da RODON objektorientiert aufgebaut ist und den daraus resultierenden Mechanismus der Vererbung auch bei der Aufstellung undVerwaltung der Modell-Kennlinien unterstützt, werden in einem ersten Schritt die Komponenten nur bezüglich sehr allgemeiner Eigenschaften unterschieden. Strukturierungsmerkmale können zum Beispiel die Ingenieur-Domänen (Hydraulik, Pneumatik, Thermodynamik, Mechanik, Elektrik,...) oder die generelle technische Funktion der betrachteten Bauteile (Ventile, Pumpen, Rohrleitungen, Motoren,...) sein. Trotz der in Abschnitt 3.2.2.1 angesprochenen komponentenorientierten Vorgehensweise in RODON macht es allein aus Strukturierungsgründen Sinn, auch Komponentenklassen einzuführen, von denen bei der späteren Zusammenstellung des Systemmodells kein konkretes Objekt instantiiert wird.

Abbildung 3-4 zeigt einen Ausschnitt aus der Hierarchie der Modellklassen, die im Rahmen des INDIA-Projekts für die AFS-Färbemaschine angelegt worden sind. Man erkennt darin zwei allgemeine Oberklassen für Strömungsmaschinen (`mo-ko_str-masch`) und Konstruktionsteile (`mo-konstr-teile`) als Beispiele solcher Strukturierungsklassen. In der Abbildung rechts daneben, hierarchisch aber darunter ordnen sich die Spezialisierungsklassen ein. In der tiefsten Ebene der Modellhierarchie befinden sich schließlich die Verhaltensabbilder der ganz konkreten Bauteile z.B. eines bestimmten Herstellers (hier z.B. `mo-kreis-hyg1_2` für eine spezielle Kreiselpumpe).

Es stellte sich bei den Klassifizierungsarbeiten für die Färbemaschine heraus, daß der Aufbau einer geeigneten Klassenstruktur für maschinenbauliche Anwendungen sich in manchen Aspekten von denen beispielsweise in der KFZ-Elektrik und -Elektronik unterscheidet. Während auch relativ komplexe elektronische Schaltungen häufig zwar aus einer Vielzahl von Bauteilen bestehen, die aber für sich genommen relativ einfache Funktionen realisieren, bestehen maschinenbauliche Anlagen oft nur aus vergleichsweise wenigen, dafür aber komplizierten Teilen. Die Berechnungsmodelle der hier angewandten Ingenieursdisziplinen Thermodynamik und Strömungslehre sind auch für relativ anschauliche und einfache Vorgänge häufig stark nichtlinear, im Gegensatz zu den „relativ linearen“ Gesetzen der stationären Elektrotechnik oder Elektronik. Beispielsweise lautet der Zusammenhang zwischen Volumenstrom $\dot{V}$ und reibungsbedingtem Druckverlust Δp in einem Rohr:

(3-0)

$$\dot{V} = A \cdot \sqrt{\frac{2\Delta p}{\xi \rho l}}.$$

Unterschiedliche Vereinfachungsannahmen zur Modellierung baulich gleicher, aber an verschiedenen Stellen eingebauter Anlagenteile machen hier eine entsprechende Modelldiskriminierung erforderlich, die beim Entwurf der Modellhierarchie zu berücksichtigen ist (z.B. „reibungsbehaftete Rohre“ und „reibungsfreie Rohre“).

Schließlich stellt die Strukturierung der maschinenbaulichen Komponenten oft auch dadurch besondere Anforderungen, daß bestimmte Bauteile nicht nur im Modell, sondern auch real Einzelstücke oder Sonderanfertigungen darstellen, deren Betriebsverhalten bei Beginn der Modellierungsarbeiten erst unvollständig bekannt ist. Eine zunächst vorgenommene Einordnung in die Klassenstruktur kann durch zwischenzeitlich neu hinzugewonnene Erfahrung oder Information überholt sein. Dies gilt in besonderem Maße für das Modell des Fehlverhaltens einer

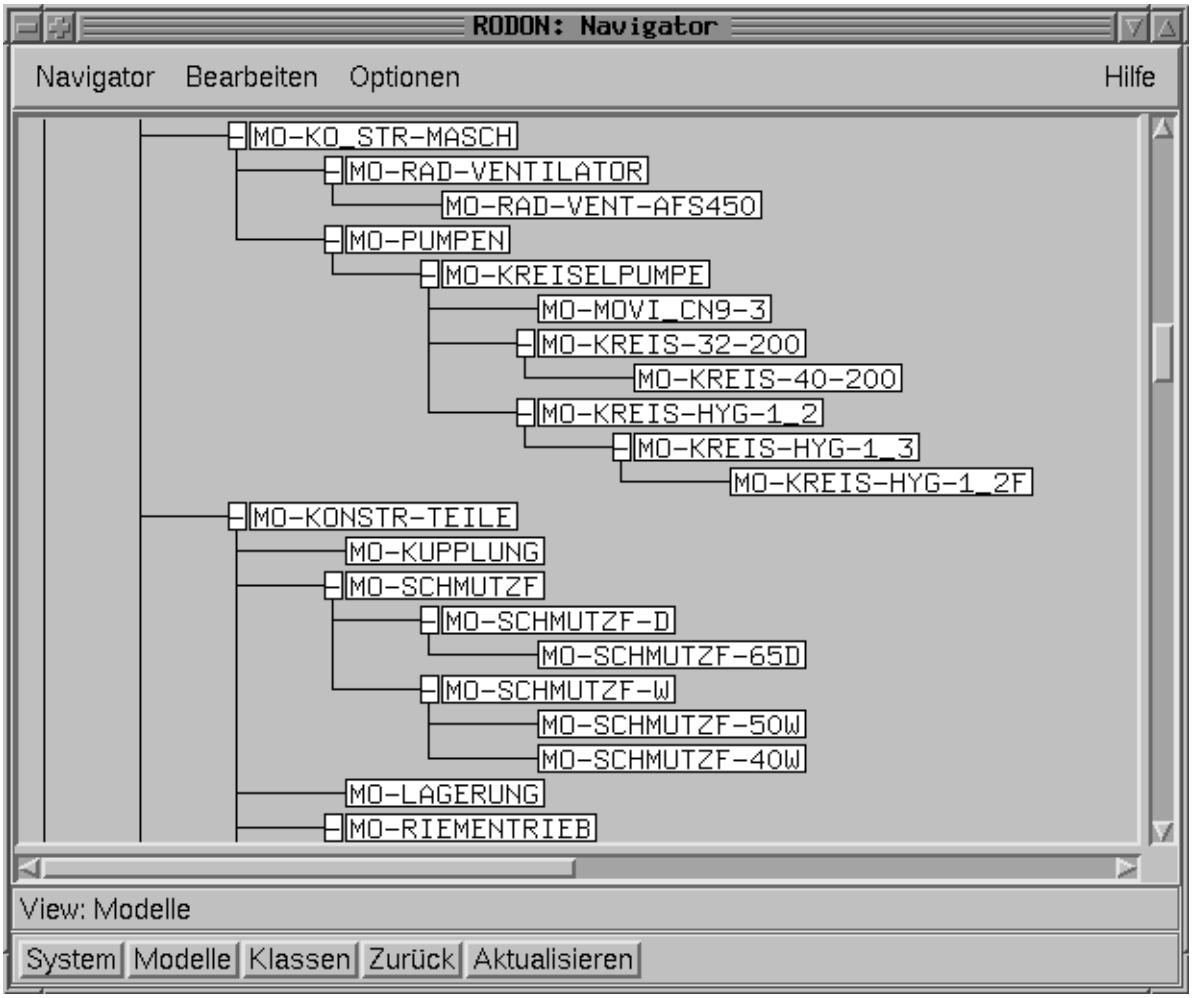


Abbildung 3-4: Hierarchische Strukturierung der Komponentenklassen

Komponente, dessen zugrundeliegenden Informationen noch schwieriger zu erhalten sind als für das Nominalverhalten.

Alle die hier beschriebenen Klassifizierungsprobleme äußerten sich in mehr oder weniger ausgeprägter Form auch bei der Modellierung der Färbemaschine. Sie lassen sich in den Fragen zusammenfassen,

- ob eine eigene Klasse für ein bestimmtes Komponentenmodell, welches nur einmal im Systemmodell benötigt wird, angelegt werden sollte, die genau (und nur) dessen spezifisches Verhalten wiedergibt (geringerer Abstraktions- und Modellierungsaufwand) oder ob sich dieses Modell eher durch ein Objekt einer allgemeineren Klasse und entsprechende Beschaltung der Eingänge realisieren läßt (bessere Wiederverwendbarkeit) und
- ob das globale Verhalten einer Baugruppe als Ganzes modelliert wird (einfacheres Modell) oder ob auch die Einzelkomponenten der Baugruppe zu berücksichtigen sind (genaueres Modell).

Durch die diversen Modellierungsmöglichkeiten in RODON konnten die Fragen aber in jeder Situation individuell beantwortet und Teilmodelle nach genauerem Kennenlernen von deren Wirkungsweise auch im Nachhinein noch ohne Probleme detailliert, ganz ausgetauscht oder neu in der Klassenhierarchie eingeordnet werden. Als Beispiel sei eine Ventil-Baugruppe der Färbemaschine - bestehend aus pneumatischem Pilotventil, einer Schlauchverbindung und dem hydraulischen Hauptventil - genannt, die zunächst als Ganzes, später jedoch als Aggregat ihrer Einzel-Bauteile modelliert wurde. Es hat sich gezeigt, daß diese Flexibilität gerade dann von besonderer Bedeutung ist, wenn an einer konkreten Anlage oder Maschine ein Bauteil-Typ nur in einer sehr geringen Anzahl oder sogar nur als Einzelstück auftritt, wie eben bei Anwendungen des Maschinenbaus.

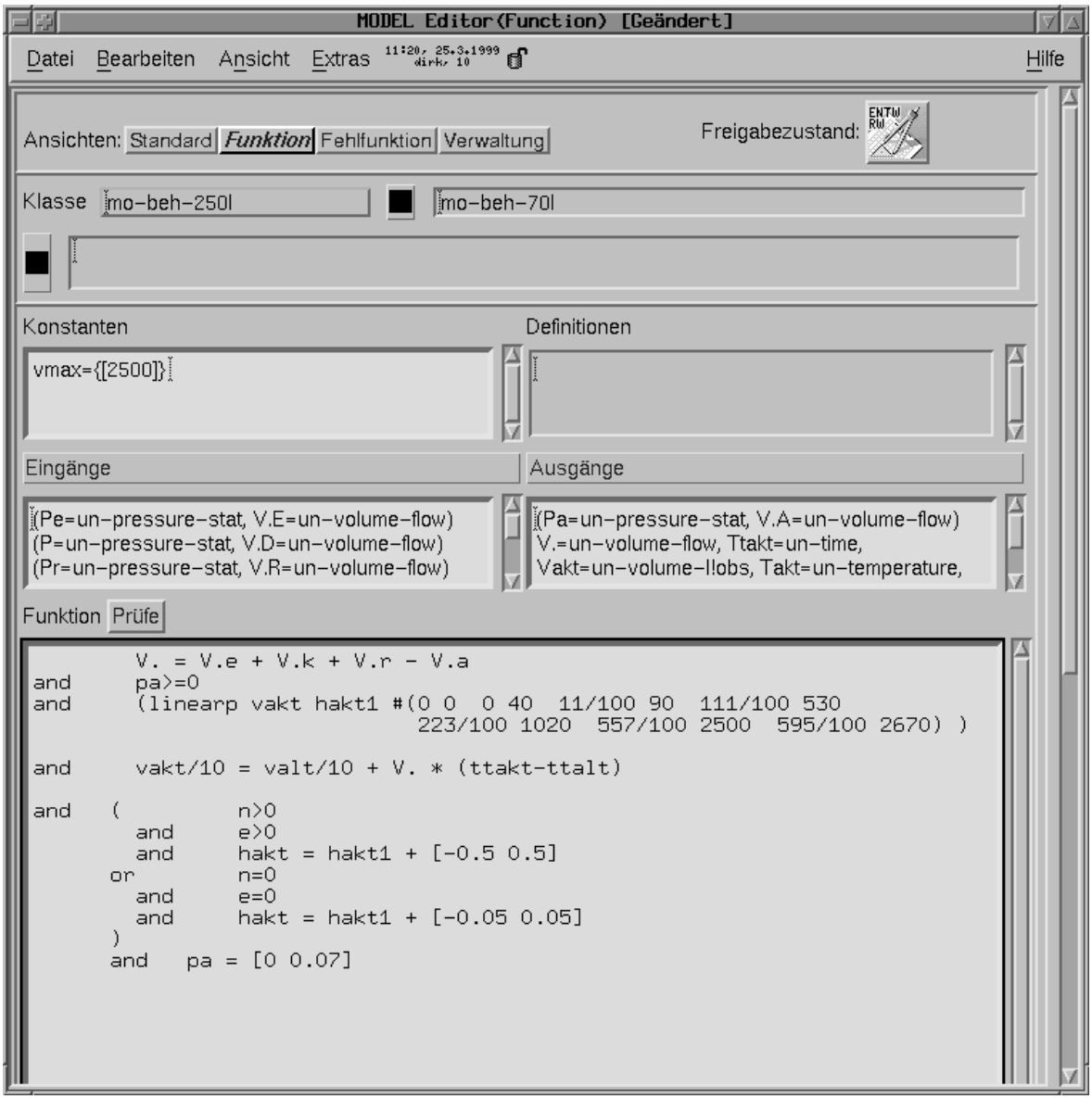


Abbildung 3-5: Erstellung des Modells für die Komponente "250l-Behälter"

3.2.2.3 Formulierung der Komponentenmodelle

Wie bereits in Abschnitt 3.2.2.1 angesprochen, wird in RODON das funktionale Verhalten der einzelnen Komponenten in Form von Kennlinien oder Kennfeldern quantitativ, qualitativ und/oder logisch beschrieben. Der Informationsaustausch der Komponenten nach außen geschieht über komponentenspezifische Eingangs- und Ausgangs-Ports. Weiterhin können in den Kennlinien eines Komponentenmodells noch lokale Größen als Konstanten auftreten, die nach außen nicht sichtbar sind. Hierunter fallen insbesondere die für ein bestimmtes Bauteil typischen Konstruktionsgrößen, wie Durchmesser, Querschnittsflächen, Ventilkoeffizienten usw.

Da jedes Komponentenmodell in die oben beschriebene objektorientierte Klassenhierarchie eingebunden ist, können Unterklassen-Modelle ausgewählte Eigenschaften von ihren Oberklassen erben, d.h. nicht nur die Nominal- und Fehlerkennlinie, sondern auch die Port- und Konstantenlisten werden zunächst mit in die Unterklasse übernommen. Allein die spezifischen Abweichungen brauchen dann dort neu formuliert zu werden.

Erstellt werden die Komponentenkennlinien in RODON mit einem speziellen Modelleditor. Beispielhaft sind in Abbildung 3-5 die Einträge der Klasse `mo-beh-250l` für einen 250l-Behälter mit mehreren Zuflüssen dargestellt, mit denen der Zusatzbehälter der AFS-Färbemaschine modelliert wurde.

Der Modelleditor stellt u.a. Eintragsfelder für folgende Eigenschaften zur Verfügung:

- Konstanten, mit symbolischer Bezeichnung und Wert der Größe für das maximale Füllvolumen, hier angegeben in 0.1 l;
- Eingangs- und Ausgangsgrößen, mit der symbolischen Bezeichnung u.a. für verschiedene Druck- und Volumenstromgrößen sowie die aktuelle Zeit `ttakt`, hier ergänzt durch die jeweiligen physikalischen Einheiten, die RODON mitverwaltet;
- die eigentliche Funktionskennlinie. Im abgebildeten Beispiel ist hier das ganze Wissen um das nominale Verhalten des Behälters in Form von Bedingungen, die zwischen den Eingangs- und Ausgangsgrößen und den Konstanten gelten müssen, zusammengestellt. Man erkennt z.B. in der ersten Zeile eine analytische Bilanzgleichung für die Volumenströme, in der zweiten Zeile eine Ungleichung für den Druck, in der dritten Zeile eine Linearisierungsanweisung für einen experimentell am realen Behälter aufgenommenen Zusammenhang zwischen aktuellem Füllvolumen `vakt` und aktueller Füllhöhe `hakt1`[2] usw. Alle diese Anweisungen sind durch logische Operatoren (`and`, `or`, . . .) miteinander verknüpft. Hinzuweisen ist auch noch auf die Beaufschlagung der Größe `hakt` mit einem Toleranzintervall (Zeilen 11 und 14), was durch die RODON zugrundeliegende Intervallmengen-Arithmetik problemlos möglich ist. Diese Operation wurde an dieser Stelle vorgenommen, um die speziell bei der Befüllung des Behälters unvermeidlichen Schwankungen der Füllstandshöhe mit in das Modell einfließen zu lassen.
- Das Verhalten der Komponente im Fehlerfall wird vollkommen analog der Nominal-Kennlinie in einem weiteren Feld spezifiziert. Da für den Behälter keine Fehlerkennline formuliert wurde, ist das entsprechende Feld hier jedoch nicht mit abgebildet. Ist dies der Fall, so existieren für RODON bei der späteren Auswertung des Constraint-Netzes für den Fehlerfall keinerlei Zwangsbedingungen zwischen den Modellgrößen. Diese können dann vom Algorithmus unabhängig voneinander eingestellt werden, erschweren jedoch eine zuverlässige Diagnose.

Nachdem alle in der Klassenhierarchie auftretenden Modelle auf diese Weise ausimplementiert wurden, stand eine umfangreiche Modellbibliothek zur Verfügung, um damit das Systemmodell der gesamten Färbemaschine zu erstellen.

3.2.2.4 Zusammenstellung des Systemmodells

Durch Instantiierung der in der Komponentenbibliothek vorhandenen Modellklassen in der benötigten Anzahl wird nun im RODON -Systemeditor für jedes technische Bauteil ein geeignetes Modell-Objekt angelegt. Die im Editor dargestellten graphischen Blöcke werden geeignet angeordnet und ihre Ports - die in Abschnitt 3.2.2.3 beschriebenen Schnittstellen der Einzelmodelle - miteinander verbunden. Dabei ist es sinnvoll, geeignete Baugruppen der realen Anlage auch im Modell in Form von Subsystemen zusammenzufassen. Dies ist in beliebiger Tiefe möglich.

In Abbildung 3-6 ist ein Ausschnitt des erstellten Modells zu sehen. Zu erkennen sind (vlnr. und vonu.) Modellblöcke für

- einen hydraulischen Knoten (Rohr-T-Stück),
- den sog. Spritzring, der zur Befüllung des Zusatzbehälters dient und an dessen oberem Rand angebracht ist,
- den 250l-Zusatzbehälter selbst,
- das Dampf-Versorgungsnetz,
- ein Dampfventil zur Dosierung des Heizdampfs, mit dem der Flottenansatz im Zusatzbehälter aufgeheizt wird, sowie
- eine elektrisch angetriebene Pumpe. Diese wurde hier aus mehreren Einzelkomponenten zusammengesetzt und in einem Subsystem zusammengefaßt. Das Subsystem „E-Pumpe“ führt dabei nur wenige der intern auftretenden Größen als eigene Ports nach außen. Implementierungsdetails von Subsystemen bleiben so auf der nächsthöheren Abstraktionsebene verborgen.

---

2. Das Operatorargument besteht aus einer Liste von Zahlenwerten, von denen jeweils zwei aufeinanderfolgende ein *xy*-Wertepaar darstellen.

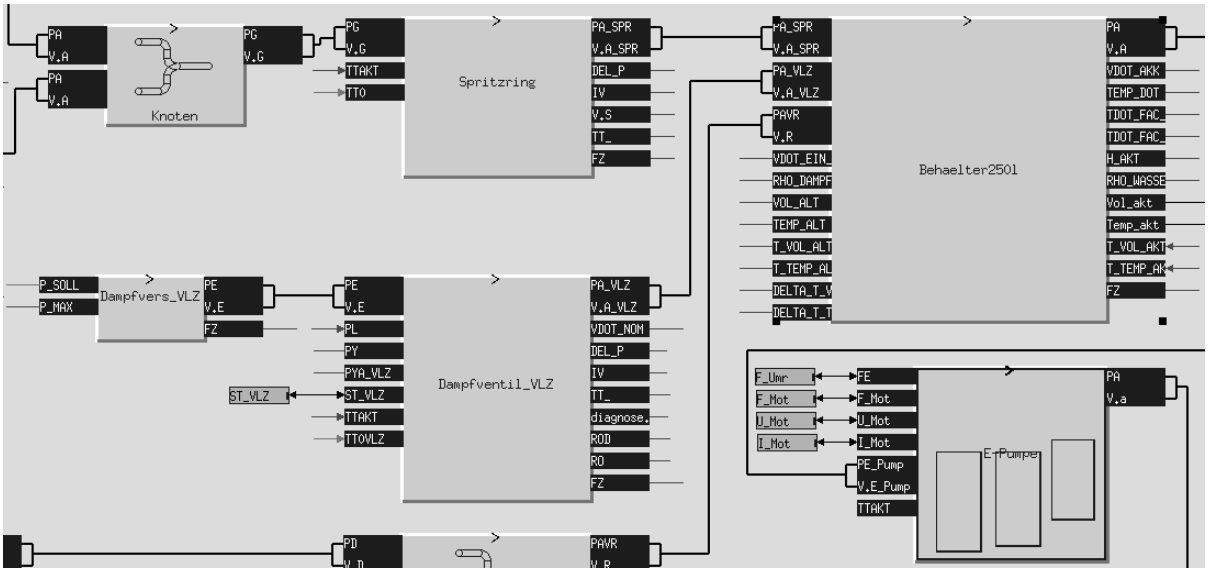


Abbildung 3-6: Kopplung der Komponentenmodelle zum Gesamtmodell

Auch das Zusatzbehältermodell selbst stellt im Rahmen des Modells für die gesamte Färbemaschinen ein Subsystem dar, von dem nur die durch kleine hell hinterlegte Rechtecke explizit angegebenen Größen (sog. Systemports) nach außen geführt werden (`ST_VLZ`, `F_UMR`, `F_MOT`,...).

Insgesamt wurden zur Modellierung der gesamten Färbemaschine über 300 Einzelobjekte instantiiert, die sich auf bis zu 5 hierachischen Systemebenen befinden.

Neben den in der Abbildung sichtbaren Linienzügen deuten auch die an einigen Ports erkennbaren Pfeile Kopplungen zwischen den Komponentenmodellen an. Hierbei handelt es sich um sog. „globale Verbindungen“, auf die jedoch hier nicht näher eingegangen werden soll. Allen im Systemmodell auftretenden und als Linien(ansätze) dargestellten Größen können Werte zugewiesen werden, entweder explizit durch den Modellierer, durch Einlesen aus einer sog. Prozeßdaten-Datei oder durch Zuweisung eines Preprocessing-Programms, welches z.B. aktuelle Werte unmittelbar aus einer Meßwerterfassung von der realen Anlage übernimmt; siehe dazu Abschnitt 3.2.3.2.

3.2.3 Meßdatenverarbeitung

3.2.3.1 Meßdaten-Erfassung an der Anlage

Die Datenkommunikation an der hier betrachteten AFS-Färbemaschine der Fa. THEN erfolgt über ein „Local Operating Network" (LON) [Wiesböck 1994]. Grundprinzip dieser Technologie ist die Dezentralisierung der Steuerungsaufgaben auf sogenannte LON-Knoten - intelligente Aktoren und Sensoren, die zwischeneinander Informationen austauschen. An der Färbemaschine sind dies u.a. sämtliche aktiv geschaltetenVentile als Aktoren sowie verschiedene Fühler für Drücke und Temperaturen als Sensoren.

Die Werte der Stell- und Meßgrößen werden als reine Integerwerte gemeinsam mit der Schlüsselbezeichung (Zeichenkette) der jeweils zugehörigen physikalischen Größe sowie eines Zeitstempels über das LON-Netz übertragen. Diese Übertragung geschieht jedoch nicht kontinuierlich, sondern nur genau dann, wenn sich ein Wert geändert hat oder ein Wert ausdrücklich von einem anderen Knoten angefordert wird, z.B. im Rahmen einer Initialisierung. Weiterhin wurde vom Anlagenbauer eine Zusammenfassung der Stellwerte (zu/auf bzw. 0/1) der Ventile der Färbemaschine zu einem 16Bit-Wort vorgenommen, vgl. Abbildung 3-7.

Ändert sich die Stellung eines einzelnen Ventils und damit auch der numerische Wert dieses 16Bit-Worts, so werden aufgrund dieser Aktualisierung auch die Informationen über die Sollstellungen aller anderenVentile über das LON-Netz geleitet, auch wenn sich deren Stellungen zu diesem Zeitpunkt nicht geändert haben.

Auf weitere Details der LON-Technologie soll an dieser Stelle nicht weiter eingegangen werden.

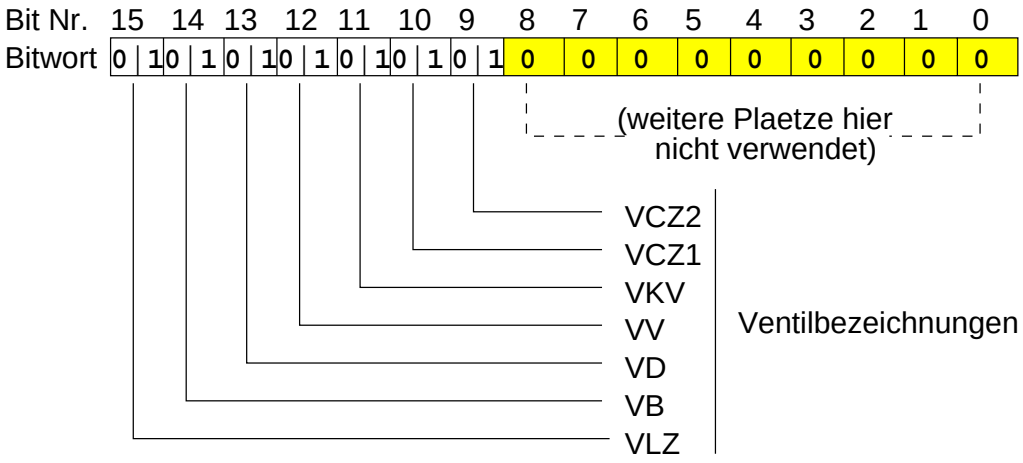


Abbildung 3-7: Verschlüsselung der Ventilstellungen zu einem 16Bit-Wort

```
174 gui1.nviLogDataAt1.bit4: FALSE  Fri Oct 01 09:36:34.430          ; Ventilstellung
175 gui1.nviLogDataAt1.bit5: FALSE  Fri Oct 01 09:36:34.540          ; Ventilstellung
176 gui1.nviLogDataAt1.bit6: FALSE  Fri Oct 01 09:36:34.650          ; Ventilstellung
241 The PollItem field is empty.The PollItem field is empty: 0.000000  Fri Oct 01 09:36:35.310
242 V142_VE2.nviPressRegPar: 0.000000  Fri Oct 01 09:36:35.640
243 gui1.nviValueLevelDT: 346.900000  Fri Oct 01 09:36:37.290
244 gui1.nviVolumePcentDT: 17.000000  Fri Oct 01 09:36:37.450
245 gui1.nviValueTempAT: 26.500000  Fri Oct 01 09:36:38.770          ; ZB-Temperatur in Grad C
246 gui1.nviValueLevelAT: 138.500000  Fri Oct 01 09:36:39.160        ; ZB-Fuellstand in l
247 gui1.nviVolumePcentAT: 27.500000  Fri Oct 01 09:36:39.380
248 The PollItem field is empty.The PollItem field is empty: 0.000000  Fri Oct 01 09:36:39.870
249 V142_VE2.nviPressRegPar: 0.000000  Fri Oct 01 09:36:39.980
250 The PollItem field is empty.The PollItem field is empty: 0.000000  Fri Oct 01 09:36:43.330
251 V142_VE2.nviPressRegPar: 0.000000  Fri Oct 01 09:36:43.880
252 gui1.nviValueTempFK: 37.800000  Fri Oct 01 09:36:44.210
253 gui1.nviValueLevelAT: 139.400000  Fri Oct 01 09:36:44.980
254 gui1.nviVolumePcentAT: 27.500000  Fri Oct 01 09:36:45.250
```

Abbildung 3-8: Auszug aus den protokollierten LON-Daten der AFS-Anlage

Alle über das LON übertragenen Daten werden vom Steuerungsrechner des LON-Netzes empfangen und über eine Ethernet-Verbindung zum Leitstand der AFS-Färbemaschine - einem unter *MS-WindowsNT* laufenden PC - übertragen. Ein dort installierter Client protokolliert schließlich die eintreffenden Bezeichner-Werte-Paare inklusive des Zeitstempels auf einer Datei.

Der in Abbildung 3-8 dargestellte Auszug aus der Protokolldatei vermittelt einen Eindruck vom Format der an der realen Anlage erfaßten Rohdaten, wobei jedoch die Kommentare nach den Semikolons nachträglich eingefügt wurden und nicht Bestandteil der Datei sind[3].

3.2.3.2 Meßdaten-Aufbereitung und Transfer nach RODON

In RODON ist es möglich, eine kontinuierliche Überwachung und Diagnose einer technischen Anlage auf der Basis sukzessive eintreffender Meß- und Ansteuerwerte zu realisieren. Dazu ist eine entsprechende Aufbereitung der Daten erforderlich, bevor diese über das *Process Data Interface* (PDI) von RODON ins Diagnosesystem gelangen können.

Die Zuweisung eines Wertes zu einer modellierten Größe kann in RODON neben der unmittelbaren Zuweisung zum Polygonzug im Systemeditor auch durch Spezifikation eines „Größenname Größenwert“-Paares auf einer

3. `gui1.nvi...` sind die eigentlichen LON-Schlüsselworte, speziell stehen `gui1.nviLogDataAt1.bit1..9` für die Ventile der Anlage.

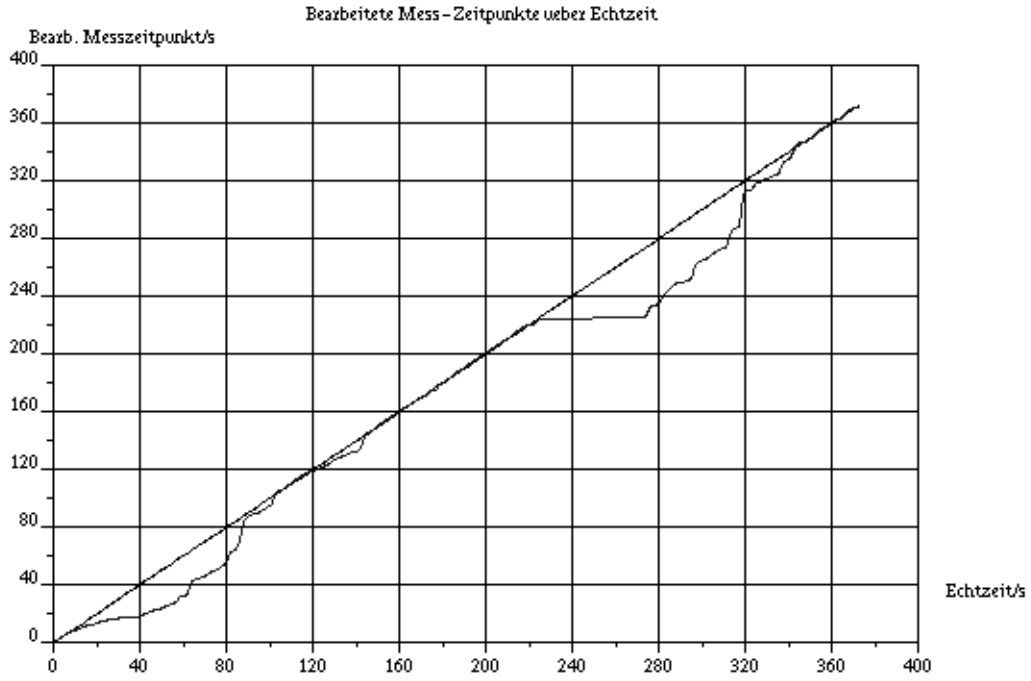


Abbildung 3-9: Zeitpunkt der Abarbeitung von Meßpunkten gegenüber der Echtzeit-Geraden

Prozeßdaten-Datei erfolgen. Insbesondere solche Größen, die nicht ständig neu von der Anlage kommen, aber zur Berechnung notwendig sind, werden typischerweise auf einer solchen Datei angegeben.

Soll das Programm kontinuierlich im Rahmen einer Überwachung mit Daten gespeist werden, so ist auch hier das genannte Format zu beachten.

Im vorliegenden Falle war daher zunächst das in Abbildung 3-8 zu erkennende Format der LON-Werte und ihrer Zeitstempel entsprechend den RODON -Erfordernissen anzupassen. Da die LON-Werte in der Realität beispielsweise aufgrund von Störungen im Datennetz nicht tatsächlich in dem angedeuteten millisekundengenauen Zeitpunkt, sondern eher paketiert am Client bzw. am PDI eintreffen, mußte ein spezielles Preprocessing-Programm (PP) dafür Sorge tragen, daß auch verspätet eintreffende Werte korrekt als „Name Wert“-Zeichenkette nach RODON übergeben werden. Zudem ist es Aufgabe des PP, aus allen über das LON transferierten Werten nur diejenigen auszufiltern, die für die Überwachung des modellierten Teilsystems der gesamten Färbemaschine benötigt werden.

Aufgrund der Vielzahl möglicher Meßdaten-Formate der Anlagen, an denen RODON eingesetzt werden kann, ist das PP vom Anwender speziell gemäß seiner Erfordernisse zu erstellen.

Beim Auslösen der Überwachungsfunktion in RODON wird von diesem die Ausführung des PP als separater Unix-Prozeß gestartet. Während das PP läuft, hält RODON seinen eigenen Prozeß an. Nachdem das PP das erste Datenpaket auf eine Unix-Pipe geschrieben hat, suspendiert es sich seinerseits und die Ausführung von RODON wird wieder aktiviert. Dieses liest nun die übergebenen Daten aus der Pipe und weist sie den entsprechenden Größen im Systemmodell zu. Nach Zuweisung auch der von der Prozeßdaten-Datei eingelesenen ergänzenden Werte wird die Diagnose gestartet (vgl. Abschnitt 3.2.4). Sobald die Diagnoseberechnungen beendet sind, suspendiert RODON sich erneut selbst und aktiviert das PP, damit der nächste Datensatz aufbereitet und übertragen werden kann. Dieser Zyklus wiederholt sich solange, bis der Anwender ihn beendet.

Eine wichtige Anforderung bei dieser Prozeßkommunikation ist die echtzeitnahe Abarbeitung der Meßwerte. Im Diagramm nach Abbildung 3-9 ist auf der x-Achse die tatsächliche Systemzeit des Rechners aufgetragen, zu der ein bestimmter Meßwert bearbeitet wird. Die y-Achse stellt die Zeit dar, zu denen ein Meßwert aufgenommen wurde, verkörpert durch den Zeitstempel in den LON-Daten. Würde das Modell stets exakt zu den Zeitpunkten durchgerechnet, zu dem ein neuer Meßwert erfaßt wird, so ergäbe sich mit der im Diagramm eingezeichneten Echtzeit-Geraden der Idealfall des Rechenzeit-Profils. In der Realität werden die Meßwerte typischerweise etwas

später bearbeitet, als der Zeitstempel vorgibt, d.h. die Kurve schmiegt sich von unten an die Ideallinie heran. Ist das Diagnosesystem überdurchschnittlich lange mit der Abarbeitung von Meßwerten beschäftigt (hier: Sekunde 10 bis 50), so kann ein größerer Rückstand entstehen, der dann in der Folgezeit (Sekunde 50 bis 70) wieder aufgeholt werden muß. Dieselbe Notwendigkeit ergibt sich, wenn Störungen in der Datenübertragung aufgetreten sind (unterbrochen zwischen ca. Sekunde 220 bis 260) und anschließend ein Paket von Daten mehrerer Meßzeitpunkte eintrifft.

Für die im Rahmen des INDIA-Projektes betrachtete Anwendung war zu beachten, daß es sich bei der Färbemaschine um ein Zeitsystem handelt. Das Füllvolumen des Zusatzbehälters oder die Flottentemperatur stellen integrierende Größen des Flüssigkeitszu- und -abstroms bzw. des Wärmestroms dar. In den Kennlinien dieser Komponenten wurden mittels eines Euler-Ansatzes die aufgrund der aktuellen Zu-/Abstrombilanz maximal möglichen Werte der Integralgrößen berechnet und im Zuge der Diagnose mit den gemessenen Füllstands- und Temperaturwerten verglichen. Da in die darin verwendeten Differenzenquotienten auch der jeweilige Wert aus dem vorherigen Berechnungszyklus einging, war es erforderlich, diese Größenwerte von einem zum nächsten Zyklus „hinüberzuretten“. Eine derartige, sogenannte „Sicherung“ von Prozeßdaten war in RODON jedoch relativ einfach zu realisieren, indem die entsprechenden Größenbezeichner in einem Einstellungsfeld spezifiziert wurden.

Dabei war insbesondere auch die jeweils zugehörige Meßzeit `ttakt` zu berücksichtigen, da diese - wie in Abschnitt 3.2.2.3 beschrieben - im Modell analog zu den anderen Größen geführt wurde. Aus diesem Grunde mußte das PP neben den eigentlichen Meßgrößen auch für die jeweils aktuelle Zeit aus dem Zeitstempel und dem Bezeichner `ttakt` in jedem Berechnungszyklus ein „Name Wert“-Paar bauen und mit im Datenpaket übergeben.

Hinzuweisen ist schließlich auf die Toleranzbeaufschlagung der Meßwerte: Diese wird unmittelbar beim Einlesen der Prozeßdaten von der Datei und beim Übernehmen eines Datenpakets aus der Unix-Pipe von RODON automatisch durchgeführt gemäß den Spezifikationen, die der Modellierer im Systemmodell vorgenommen hat

Wie die Diagnose innerhalb von RODON abläuft, beschreibt das nächste Unterkapitel.

3.2.4 Modellbasierte Diagnose der Anlage

3.2.4.1 Abgleich von Werteintervallen

Die Grundlage der Diagnose in RODON ist die Analyse der Abweichung zwischen dem nominalen und dem tatsächlichen Verhalten der zu untersuchenden Anlage. Das nominale Verhalten berechnet sich aus den Ansteuergrößen des Systems gemäß den durch die Nominalkennline ausgedrückten Modellgleichungen. Das Ergebnis dieser Berechnung wird mit dem tatsächlichen Systemverhalten, welches in Form der an der Anlage gemessenen Werte einiger physikalischer Größen vorliegt, abgeglichen. Sobald nur ein Meßwert nicht mit dem berechneten Wert übereinstimmt bzw. - da in realen Systemen stets Toleranzen zu berücksichtigen sind - außerhalb des berechneten Bereichs liegt, ist das Nominalmodell nicht mehr geeignet, das real beobachtete Systemverhalten zu beschreiben. Eine Abweichung vom Nennbetrieb läßt sich aber nur durch einen Defekt an der Anlage erklären, der anschließend zu identifizieren ist. Die modellbasierte Diagnose wird auf der Basis des in [deKleer, Williams 1987] definierten Verfahrens durchgeführt.

Anhand von Abbildung 3-10 soll dies veranschaulicht werden: Ein Behälter wird durch einen oder mehrere Flüssigkeitsströme befüllt. Die Netto-Füllrate sei im Mittel konstant, aber beispielsweise aufgrund von Druckschwankungen im System toleranzbehaftet. Das Füllvolumen werde in bestimmten Zeitabständen gemessen. Ausgehend vom vorherigen Meßzeitpunkt t_0 und der durch die Modellgleichungen bilanzierten Soll-Füllrate ergeben sich das rechnerische Werteintervall $I_1 = [V_{min,theor}, V_{max,theor}]$ des Füllstands zur aktuellen Meßzeit t_{akt} gemäß der Gleichung:

$$(3\text{-}0) \quad V_{min;theor} = V_0 + \dot{V}_{min,theor} \cdot (t_{akt} - t_0) \quad \textit{bzw.}$$

$$V_{max;theor} = V_0 + \dot{V}_{max,theor} \cdot (t_{akt} - t_0)$$

Zur Zeit t_{akt} werde an der Anlage der Füllstandswert V_{mess} gemessen. Bedingt durch die Schwankungen des Flüssigkeitsspiegels gibt auch dieser Wert die Realität nicht exakt, sondern nur mit einer gewissen Bandbreite wieder, so daß zum Abgleich des Meßwerts mit dem berechneten Wert auch für ersteren ein Toleranzintervall $I_2 = [V_{min,\,mess}, V_{max,\,mess}]$ zugelassen werden muß. Gibt es nun eine Schnittmenge zwischen I_1 und I_2, so kann das Modell die Messung erklären, und für weitere rechnerische Analysen sind beide Intervalle durch die Schnittmenge zu ersetzen.

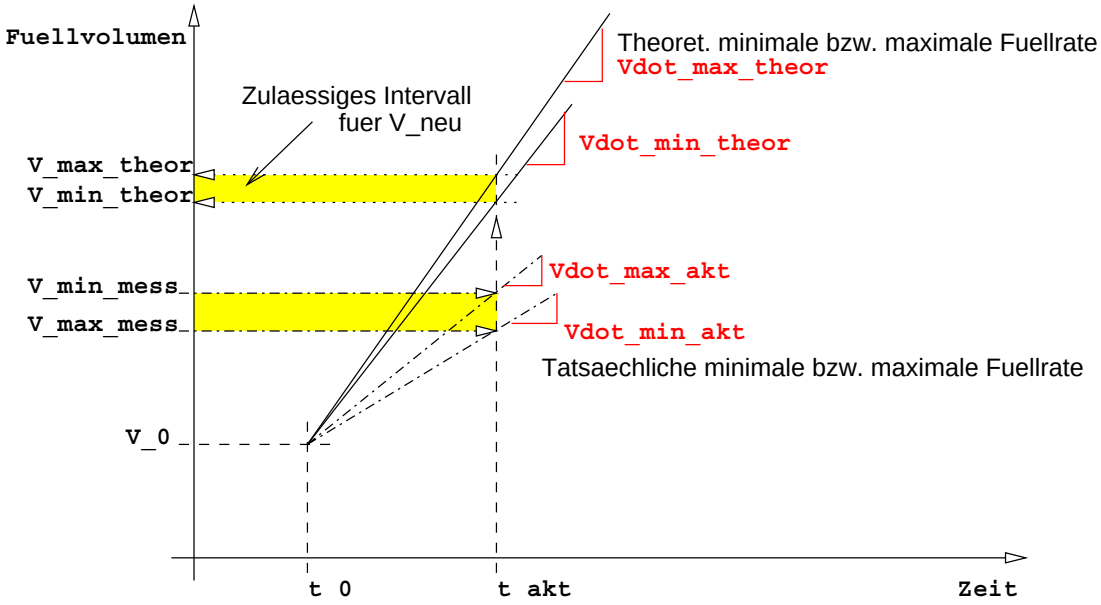


Abbildung 3-10: Befüllung eines Behälters: Berechnung zulässiger Volumenwerte

Existiert jedoch - wie es in der Abbildung angedeutet ist - kein Volumenwert, der sowohl in I_1 als auch in I_2 enthalten ist, so wird die Beobachtung nicht mehr durch das Nominalmodell beschrieben. Es liegt ein rechnerischer Konflikt bzw. ein technischer Defekt innerhalb der Anlage vor, beispielsweise in Form eines Lecks im Behälter oder eines deutlich unterhalb seines Sollwerts liegenden Zulaufstroms (⟶ verstopftes Rohr, klemmendes Ventil, ...).

3.2.4.2 Identifikation verdächtiger Kandidaten

Der soeben beschriebene Abgleich zwischen Nominal-Berechnungs- und Meßwerten stellt in der Diagnosefunktion von RODON einen sogenannten Eingangstest dar. Nach dem Einlesen der Prozeßdaten von Datei bzw. über das PP und dem Zuweisen der u.U. toleranzbehafteten Werte an die entsprechenden Modellgrößen wird das Constraint-Netz mit den übernommenen Werten durchpropagiert. Dabei ist es gleichgültig, ob die eingelesenen Werte Eingangs- oder Ausgangsgrößen der modellierten Kennlinien belegen, da RODON intern diese Unterscheidung aufhebt. Stellt sich ein Konflikt ein, so ist die Anlage nicht in Ordnung und RODON führt automatisch die Lokalisierung und Identifikation des defekten Bauteils weiter. Wie beim ATMS Verfahren üblich, werden die möglichen Fehlerkandidaten auf den Konfliktpfaden schon bei der Propagierung ermittelt.

Für die Überprüfung der Kandidaten werden die Fehlerkennlinien ausgewertet. Falls das restliche Netz mit einer dieser Kennlinien konfliktfrei zu erfüllen ist, kann dieser Kandidat den Fehler erklären.

Die Spezifikation von Fehlermodellen für die einzelnen Komponenten ist anzustreben, da dann keine physikalisch unmöglichen Betriebszustände in Betracht gezogen werden und sich die Diagnose erheblich beschleunigen läßt.

Werden mehrere Versagensfälle für ein Bauteil definiert, so werden diese innerhalb der Fehlerkennlinie typischerweise mit Hilfe einer sog. Fehlerzustandsvariable `fz` mit einfachen Integerwerten unterschieden, die ebenso wie die anderen Größen als Port des Komponentenmodells anliegt (vgl. Abschnitt 3.2.2.3). Es ist zweckmäßig, für diese Variablen „Mappings“ anzugeben, d.h. Zuordnungen zwischen dem Integerwert und einer Zeichenkette mit der Bedeutung des Wertes (z.B. 0="ok", 1="Ventil_hängt_offen", 2="Ventil_hängt_geschlossen", . . .). Dann werden bei der Anzeige des Diagnoseergebnisses von RODON unmittelbar die klartextlichen Fehlerursachen genannt; siehe dazu das Beispiel in Abschnitt 3.2.4.3.

3.2.4.3 Beispiel: „Verstopftes Zulaufventil“

Exemplarisch für die Vielzahl der in den einzelnen Bauteilmodellen der Färbemaschine abgebildeten Fehlerfällen soll die Diagnose mit RODON für das Szenario „verstopftes Zulaufventil“ demonstriert werden. Hierbei handelt es sich um das Ventil mit der Bezeichnung `VCZ1` in der Wasserleitung, die den Zusatzbehälter der Anlage mit Kaltwasser speist. Von der Ansteuerung her seien alle Ventile bis auf das `VCZ1` geschlossen, d.h. es ist lediglich

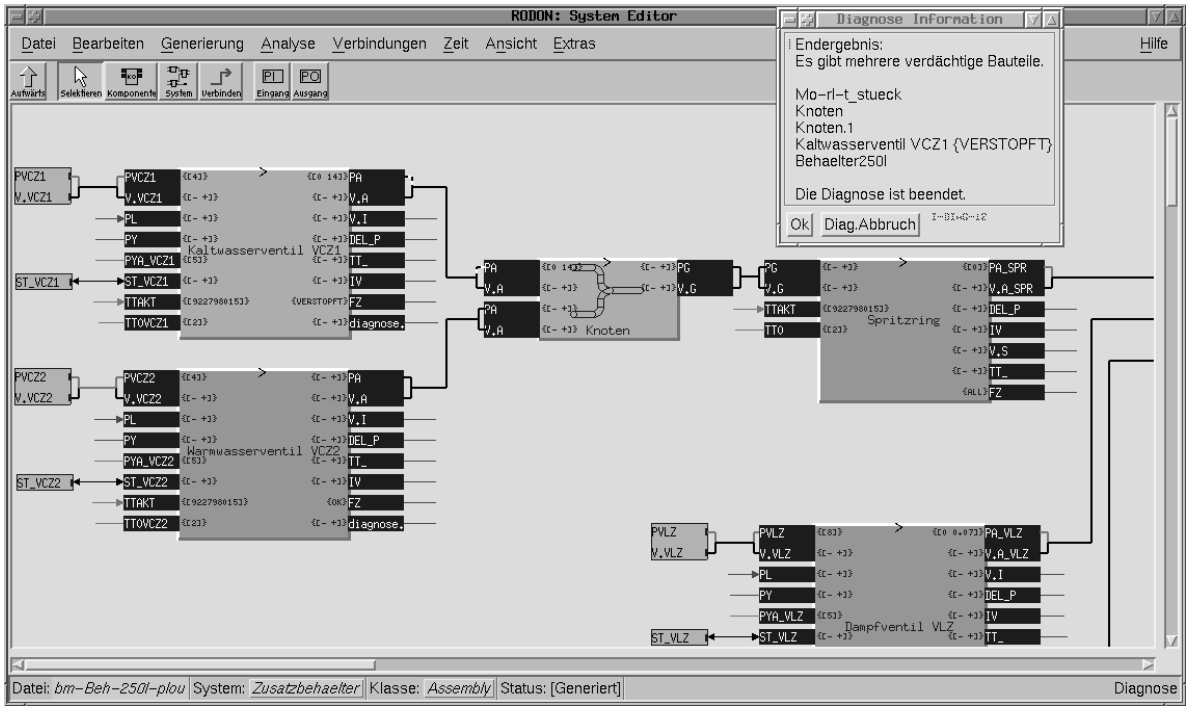


Abbildung 3-11: Diagnose-Ergebnis mit Liste der verdächtigen Bauteile

der Behälter über den Spritzring mit kaltem Wasser zu füllen. Der Wasserdruck in der Versorgungsleitung des Ventils sei mit p_{VCZ1} = 4 bar bekannt und konstant. Aus dem in Reihe mit dem Ventil geschalteten Spritzring, bei dem es sich vereinfacht um ein ringförmiges „Rohr mit Löchern“ handelt, spritzt das Wasser in den Behälter und entspannt sich dabei auf Umgebungsdruck. Aus den Öffnungsquerschnitten und der Durchströmungsgeometrie von Ventil und Spritzring bzw. den vorab berechneten (konstanten) Druckverlustzahlen resultiert ein definierter Zulauf-Volumenstrom $\dot{V}$ = 5 l/s in den Behälter[4]. Zum Zeitpunkt tt_{alt} = 922798014 sei ein Füllstand von $V_{alt,\ ZB}$ = (99.1 ± 1.0) Liter gemessen worden. Der nächste Meßwert werde nun 1 Sekunde später, zum Zeitpunkt tt_{akt} = 922798015 aufgenommen.

Gemäß des Nominalmodells nach Glg (3-0) müßte das Füllvolumen nun $V_{akt,\ ZB}$ = (104.1 ± 1.0) Liter betragen. Der tatsächlich gemessene Wert betrage jedoch lediglich $V_{akt,\ ZB,\ mess}$ = 102.0 Liter. RODON trifft daher während der Propagierung auf einen Konflikt und stößt die Überprüfung der Fehlerkennlinien an. Abbildung 3-11 zeigt das schließlich erhaltene Ergebnis der Diagnose: eine Liste derjenigen Komponenten, bei deren Fehlermodell das Constraint-Netz konfliktfrei erfüllt werden kann.

In dieser Liste treten neben dem korrekten Diagnoseergebnis für die fehlerhafte Komponente „Kaltwasserventil VCZ1“ und der Fehlerart „verstopft“ auch weitere Komponenten auf. Dies ist plausibel, da für die weiteren Komponenten kein spezielles Fehlermodell angegeben wurde. Aufgrund der dann unabhängig voneinander einstellbaren Werte für Ein- und Ausgänge ergibt sich im Constraint-Netz ebenfalls eine konfliktfreie Lösung (vgl. Abschnitt 3.2.4.2). Die Knoten sind ebenfalls reale Bauteile. Sie können deswegen ebenfalls Grund des Problems sein, entweder durch Leckage oder Verstopfung. Damit zeigt dieses Beispiel auch, daß RODON - wie alle echten modellbasierten Diagnosesystem - auch unspezifizierte Fehler in Komponenten hypothetisieren kann.

3.2.5 Weitergehende Modellauswertungen

Neben den bisher beschriebenen Möglichkeiten der Online-Überwachung und der Diagnose defekter Bauteile ist es auch möglich, das Wissen über die betrachtete Anlage systematisch und vollautomatisch auf der Grundlage des erstellten Modells von RODON erschließen zu lassen. Durch strukturierte Ablage der Analyseergebnisse und

4. Eine Volumenstrom-Toleranz ist zwar auch hier vorhanden, jedoch sehr gering und im Beispiel vernachlässigbar.

geeignete Aufbereitungsmechanismen kann dieses Wissen für Anwendungen, wie beispielsweise die Erstellung von Fehlerbäumen oder „Failure Mode and Effect“-Analysen (FMEAn), nutzbar gemacht werden.

3.2.5.1 Systematische Erfassung des Systemwissens

Ein nach den Beschreibungen des Abschnitts 3.2.2.4 erstelltes Modells der Färbeanlage enthält das Wissen um die funktionalen und physikalischen Zusammenhänge des Gesamtsystems, ausgedrückt durch miteinander verbundene Blöcke für die diversen Anlagenbauteile und entsprechende Koppelgrößen. Diese Angaben reichen aus, um die in den Abschnitten 3.2.3 und 3.2.4 vorgestellten Analysen quasi „von Hand“ bzw. interaktiv für bestimmte Betriebszustände auszuführen. Sie reichen jedoch nicht aus, um beispielsweise im Rahmen einer FMEA alle möglichen Ursachen für einen bestimmten Systemfehler zu ermitteln.

Um nun alle möglichen nominalen und fehlerhaften Zustände, in denen sich die betrachtete Anlage prinzipiell befinden kann, systematisch zu erfassen und zu untersuchen, sind zunächst weitere Fragen zu beantworten wie

- Welche Größen sind ansteuerbar?
- Welche Größen sind meß- oder beobachtbar?
- Welche Fehlerzustände sind möglich?
- ...

Diese Informationen, die bei interaktiver Arbeit mit dem Modell vom Modellanwender oft intuitiv berücksichtigt werden, sind im Systemmodell noch explizit zu ergänzen, um die systematische Modellauswertung automatisieren zu können. Sie stellen zusätzliche Merkmale und Eigenschaften bestimmter Systemgrößen dar. Deren Spezifikation ist in RODON durch die Angabe sogenannter *Property Marks* - Kennungen für verschiedene Eigenschaftsgruppen - möglich, die im Eingangs- bzw. im Ausgang-Feld des Modelleditors (vgl. Abschnitt 3.2.2.3) an die entsprechenden Modellgrößen geheftet werden. Tabelle 3-1 zeigt exemplarisch einige solcher Eigenschaftsgruppen und ihre Kennungen, wie sie in RODON verwendet werden.

Tabelle 3-1 Kennung und Bedeutung einiger Property Marks

Kennung	Bedeutung der Größe	Beispiel
!sw	Ansteuergröße	Ventil-Steuerspannungen
!obs	Beobachtung	Volumenströme
		Ist-Ventilstellungen
!fm	Fehlermodus	Fehlerzustands-Variablen

Zur vollständigen Auswertung des Systemmodells werden von RODON prinzipiell *alle* möglichen Nominal- und Fehlermodi *aller* Modellkomponenten (`fz`-Werte; siehe Abschnitt 3.2.4.2 für *alle* Ansteuerungen durchpermutiert und für jeden der sich dabei ergebenden Fälle durch den bereits oben beschriebenen Constraint-Propagierungsmechanismus die Werte aller Systemwerte berechnet[5]. Die resultierenden Werte aller mit einem Property-Mark versehenen Modellgrößen werden nach jeder Analyse in einer Zustands-Datenbank (*state data base; SDB*) abgelegt.

Zwar kann gerade bei komplexeren Modellen die Anzahl durchzuführender Analysen aufgrund vieler Steuergrößen oder zahlreicher Fehlermodi stark ansteigen. Durch eine speziell entwickelte komfortable Benutzeroberfläche und die vollkommene Automatisierung des Auswerteprozesses in RODON ist es aber für den Anwender sehr einfach (wenige Mausklicks) und schnell (wenige Minuten) möglich, die Erstellung einer derartigen Datenbank, die praktisch das komplette im Systemmodell abgebildete Wissen über das Verhalten der realen Anlage enthält, anzustoßen. Die Erstellung einer solchen Datenbank kann je nach Systemgröße von wenigen Minuten bis zu mehreren Stunden oder Tagen (interaktionsfreie (!)) Rechenzeit beanspruchen; im Anschluß daran steht deren Inhalt aber für beliebige Auswertungen zu Verfügung.

Da die soeben beschriebene Funktionalität von RODON ursprünglich sehr stark aus Anforderungen des Automobilbaus und hier speziell der KFZ-Elektrik motiviert war und entwickelt wurde, spiegelt sich dieser Ursprung in

5. Aus diesem Grunde ist es notwendig, auch die Ansteuergrößen vorher zu diskretisieren.

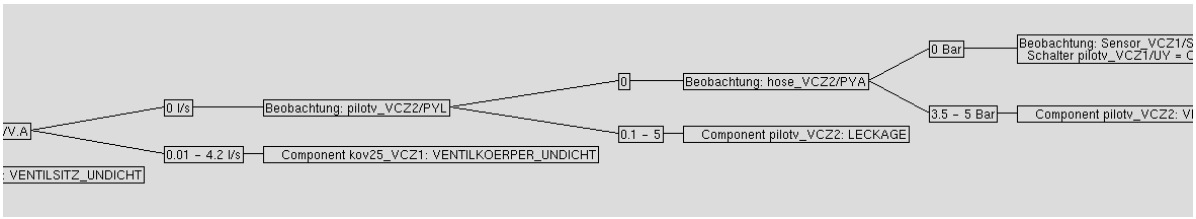


Abbildung 3-12: Fehlersuchbaum-Ausschnitt für Befüllungssystem des Zusatzbehälters

einigen speziell dafür zugeschnittenen Funktionen innerhalb des Programms wider. So sind einige Property-Marks nur für elektrische oder elektronische Systeme sinnvoll. Für die Anwendung am Modell der Färbemaschine war es daher erforderlich, hier geeignete Parallelen zu ziehen bzw. die Unterschiede der Domänen herauszuarbeiten. Beispielsweise läßt sich im elektrischen Bereich ein Stecker auftrennen, um die an seinen Pins anliegende Spannung zu messen. Das Analogon im hydraulischen Bereich - Öffnen einer Flanschverbindung zur Messung des Rohr-Innendrucks - ist allerdings nicht ohne Weiteres möglich. Dieser Umstand hat unmittelbar Auswirkungen auf die Klassifizierung der Systemgrößen mithilfe der Property-Marks (Anzahl beobachtbarer Größen und Konditionierung des Problems).

3.2.5.2 Automatische Erstellung von Fehlersuchbäumen und FMEA-Tabellen

Eine Art der Auswertung der im vorherigen Abschnitt beschriebenen Zustands-Datenbank bilden die Darstellung in Form eines sogenannten Fehlersuchbaums oder allgemeiner Entscheidungsbaums. Ein Entscheidungsbaum hat zum Ziel, Objekte anhand einer Reihe von Merkmalen zu klassifizieren. Dabei ist eine Abarbeitung der Merkmale in einer Reihenfolge anzustreben, die in möglichst wenigen Testschritten zum Ziel führt. Ausgehend von der Wurzel wird pro Schritt ein neuerVerzweigungsknoten an einen der offenen Äste des Baums angehängt.

Im vorliegenden Falle der technischen Diagnose bilden die im modellierten System berücksichtigten Fehler die Klassen, zu denen zugeordnet werden soll. Eine bestimmte Werte-Konstellation der Fehlerzustands-Variablen ist bei definierten System-Eingangsgrößen fest mit bestimmten Werte oder Wertebereichen der beobachtbaren Größen korreliert. Die bei einem bestimmten Betriebszustand gemachten Beobachtungen stellen somit die Klassifizierungsmerkmale für die Fehlerzustandsvektoren dar.

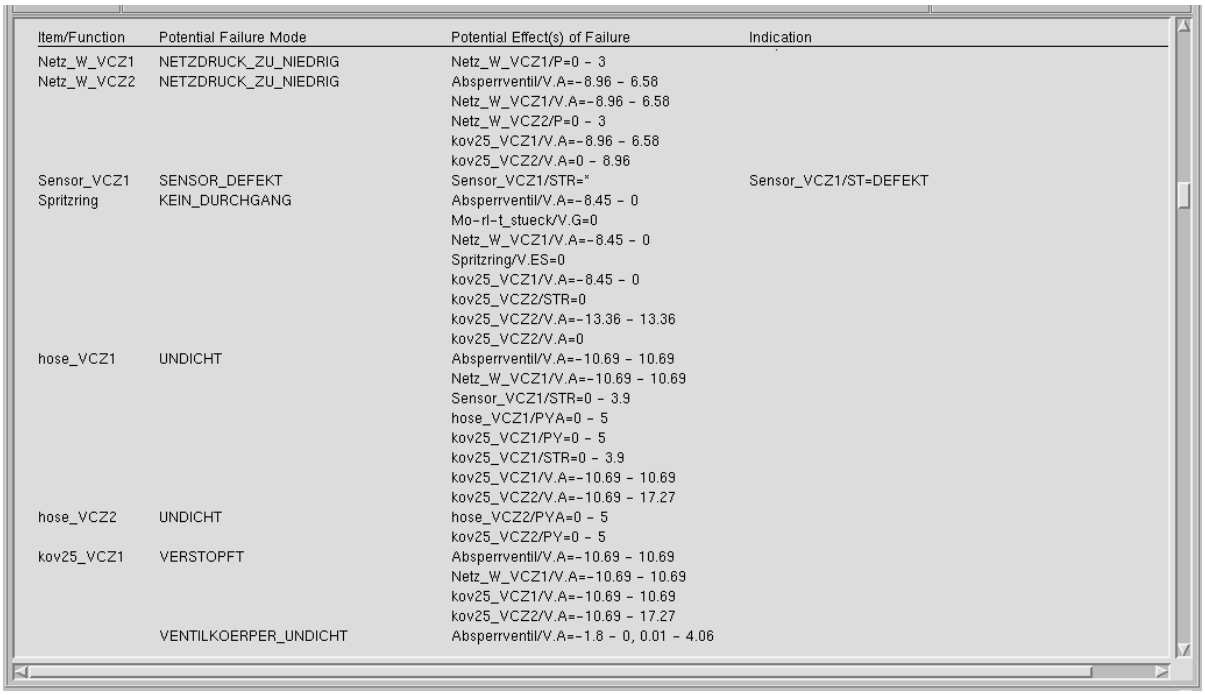
Da der Zusammenhang zwischen den Werten von Eingangs- und Beobachtungsgrößen bei der Erstellung der Zustands-Datenbank bereits erfaßt wurde, ist es allein eine Frage der Ergebnisaufbereitung, einen Entscheidungsbaum zur Fehlerklassifikation zu erstellen. Diese Auswertung und die Umsetzung in eine grafische Baumstruktur ist in RODON quasi auf Knopfdruck möglich.

Abbildung 3-12 zeigt einen Ausschnitt eines derart für die Färbemaschine erstellten Fehlerbaums bzw. für ein Teilsystem davon (Befüllungssystem des Zusatzbehälters). Es ist zu erkennen, wie aufgrund diskreterWertebereiche von Systemgrößen auf bestimmte Systemzustände geschlußfolgert wird. Beispielsweise kann aufgrund eines beobachteten Ausgangs-Luftdrucks *PYL* eines pneumatischen Pilotventils (`pilotv_VCZ2`) in Höhe von 0.l. . . 5 bar bereits gesagt werden, daß dieses Ventil leckt. Ist der Ausgangsdruck Null, so ist der Abgleich weiterer Beobachtungen notwendig, um zu einer Aussage über Vorhandensein und Art von Komponentendefekten im Gesamtsystem zu gelangen.

Eine andere Sichtweise auf die berechneten und in der Zustands-Datenbank abgelegten Ergebnisse stellt eine FMEA-Tabelle dar. Hierin sind potentielle Defekte der modellierten technischen Anlage tabellarisch den resultierenden Auswirkungen in den Werten der beobachtbaren bzw. meßbaren Systemgrößen gegenübergestellt (siehe Abbildung 3-13). Die Erstellung solcher Tabellen in verschiedenen Formaten ist in RODON mittlerweile auf Knopfdruck möglich. Auch zur Realisierung dieser Funktionalität des Programms hat das INDIA-Projekt maßgeblich beigetragen.

3.3 Aktueller Projektstand und Ausblick

Im Rahmen der bisherigen Arbeiten im Zuge des INDIA-Projekts wurde eine Reihe physikalischerVorgänge und Bauteile analysiert, die für maschinenbauliche Anwendungen typisch sind, und in einer Bibliothek von Kompo-



Item/Function	Potential Failure Mode	Potential Effect(s) of Failure	Indication
Netz_W_VCZ1	NETZDRUCK_ZU_NIEDRIG	Netz_W_VCZ1/P=0 - 3	
Netz_W_VCZ2	NETZDRUCK_ZU_NIEDRIG	Absperrventil/V.A=-8.96 - 6.58	
		Netz_W_VCZ1/V.A=-8.96 - 6.58	
		Netz_W_VCZ2/P=0 - 3	
		kov25_VCZ1/V.A=-8.96 - 6.58	
		kov25_VCZ2/V.A=0 - 8.96	
Sensor_VCZ1	SENSOR_DEFEKT	Sensor_VCZ1/STR=*	Sensor_VCZ1/ST=DEFEKT
Spritzring	KEIN_DURCHGANG	Absperrventil/V.A=-8.45 - 0	
		Mo-rl-t_stueck/V.G=0	
		Netz_W_VCZ1/V.A=-8.45 - 0	
		Spritzring/V.ES=0	
		kov25_VCZ1/V.A=-8.45 - 0	
		kov25_VCZ2/STR=0	
		kov25_VCZ2/V.A=-13.36 - 13.36	
		kov25_VCZ2/V.A=0	
hose_VCZ1	UNDICHT	Absperrventil/V.A=-10.69 - 10.69	
		Netz_W_VCZ1/V.A=-10.69 - 10.69	
		Sensor_VCZ1/STR=0 - 3.9	
		hose_VCZ1/PYA=0 - 5	
		kov25_VCZ1/PY=0 - 5	
		kov25_VCZ1/STR=0 - 3.9	
		kov25_VCZ1/V.A=-10.69 - 10.69	
		kov25_VCZ2/V.A=-10.69 - 17.27	
hose_VCZ2	UNDICHT	hose_VCZ2/PYA=0 - 5	
		kov25_VCZ2/PY=0 - 5	
kov25_VCZ1	VERSTOPFT	Absperrventil/V.A=-10.69 - 10.69	
		Netz_W_VCZ1/V.A=-10.69 - 10.69	
		kov25_VCZ1/V.A=-10.69 - 10.69	
		kov25_VCZ2/V.A=-10.69 - 17.27	
	VENTILKOERPER_UNDICHT	Absperrventil/V.A=-1.8 - 0, 0.01 - 4.06	

Abbildung 3-13: Ausschnitt aus FMEA-Tabelle für Befüllungssystem Zusatzbehälter

nentenmodellen in RODON abgebildet. In einer standardisierten Form enthalten die Klassen der Bibliothek derzeit Modellgleichungen aus den Bereichen

- Strömungsmaschinen (z.B. Kreiselpumpen und Gebläse)
- Thermodynamik (z.B. Wärmetauscher- oder Mischungsvorgänge),
- Hydraulik (z.B. reibungsbehaftete Rohr- oder Ventilströmung),
- Pneumatik (z.B. verlustbehaftete Druckluftleitungen und Magnetventile),
- Mechanik (z.B. Verschluß- oder Passungseigenschaften von Ventilen und Dichtungen),
- Elektrik (z.B. Schlupf bei elektromotorischen Antrieben) und
- Optik (z.B. Lichtleitung in Faserkabeln).

Durch einheitliche Schnittstellen sind die Klassenobjekte beliebig miteinander koppel-, austausch- und bei Bedarf durch Subklassenbildung modifizierbar. Es bestehen darüberhinaus RODON-Modelle zur Abbildung der gesamten AFS-Färbemaschine oder Teilsystemen davon, mit denen diverse Simulations- und Diagnosestudien im stationären und instationären Anlagenbetriebs durchgeführt wurden. Es ist mit der aktuellen Implementierung möglich, an der realen Anlage aufgenommene Ansteuer- und Meßdaten kontinuierlich RODON zuzuführen, um dort den Abgleich des nominalen und des beobachteten Systemverhaltens durchzuführen. Eine bei Diskrepanz ggf. notwendige Identifikation eines fehlerhaften Bauteils und dessen genauer Fehlerart wird von RODON vollautomatisch und ohne weitere Benutzer-Interaktion durchgeführt und dem Anwender angezeigt. Wird das Diagnosesystem unmittelbar an der realen Anlage, z.B. im Leitstand, betrieben, kann so der Maschinenbediener aufgrund der klartextlichen Anzeige des Fehlers unverzüglich geeignete Schritte z.B. zum kontrollierten Herunterfahren der Anlage und zur Reparatur oder Austausch des Bauteils einleiten.

Die derzeit noch laufenden Arbeiten haben u.a. zum Ziel, eine *präventive Diagnose* mit RODON zu realisieren. Um das Verschleißverhalten einzelner Bauteile der Färbemaschine in den Modellklassen abzubilden, wird dieses derzeit sowohl analytisch als auch experimentell-statistisch anhand historischer Daten untersucht. Durch den Abgleich angemessen geglätteter aktueller Meßdaten auch an diesen Modellen ist zu erwarten, daß Verschleißerscheinungen noch vor dem eigentlichen Ausfall des Bauteils erkannt und so die Verfügbarkeit der Anlage deutlich erhöht werden kann.

Weiterhin soll neben dem modellbasierten Ansatz eine weitere Diagnose-Technologie prototypisch an der Färbemaschine eingesetzt werden. Grundlage des Verfahrens ist die Erstellung von Diagnoseregeln aus einem Modell heraus und die strukturierte Ablage von deren Ergebnissen „imVoraus“ vor einer Anwendung in einer Datenbank, so daß dann im Betrieb aufgrund der abgelegten Informationen eine beschleunigte Schlußfolgerung und Identifikation defekter Komponenten möglich ist.

Schließlich laufen Bestrebungen, ein neues, verbessertes modellbasiertes Verfahren zu entwickeln, welches auf einer hochkompakten Modelldarstellung und -auswertung beruht und eine sehr schnelle Diagnose verspricht. Ziel dieser Entwicklung ist u.a. eine gegenüber dem heutigen Stand deutliche Herabsetzung der Systemanforderungen des Diagnosesystems, um dieses letztendlich auf einem Mikroprozessor integrieren zu können. Dadurch wird bei potentiellen Anwendern der Einsatz modellbasierter Diagnoseverfahren erheblich vereinfacht.

Es ist zu erwarten, daß sich auf Grundlage gerade auch der letztgenannten prototypischen Realisierung nach entsprechender Weiterentwicklung die beschriebene modellbasierte Diagnosetechnologie mittelfristig auch in kleinen und mittleren Firmen des Maschinen- und Anlagenbaus neue Anwendungsbereiche erschließt.

4 Ansätze zur Senkung der Einführungskosten von Diagnosesystemen

Thomas Guckenbiehl - Fraunhofer Institut für Informations- u. Datenverarbeitung

4.1 Einleitung

Ausgangssituation

Leitanwendung für die Arbeit des Fraunhofer-Instituts für Informations- und Datenverarbeitung IITB in INDIA war die Post-Mortem-Diagnose des von THEN entwickelten Chemikaliendistributors (CHD). Er mißt die für eine Färbung benötigten Chemikalien ab und transportiert sie von den Chemikalientanks über ein System von Rohrleitungen bis zur richtigen Färbemaschine.

Wenn Ausfälle dieser Anlage auch selten sind, so können sie doch die gesamte Produktion lahmlegen und sowohl Material als auch die Umwelt wesentlich schädigen. Es ist daher sehr wichtig, die Ursache eines solchen Ausfalls so schnell wie möglich zu finden und zu beseitigen. Ein Post-Mortem-Diagnoseassistent kann diese Ausfallzeit wesentlich verringern. Mit seiner Hilfe kann das Färbereipersonal die Anlage häufig selbst wieder instandsetzen und muß nicht auf die Ankunft von THEN-Servicetechnikern warten.

Trotz dieses hohen Nutzens stattet THEN seine Produkte bisher nicht standardmäßig mit wissensbasierter Diagnosetechnik aus. Einer der Hauptgründe dafür ist ein altbekanntes, aber immer noch offenes Problem: die Erfassung des benötigten Wissens, insbesondere die Modellierung des Geräts für die Diagnose ist zu teuer.

Diese Situation von THEN ist typisch für viele, gerade mittelständische Unternehmen im deutschen Maschinen- und Anlagenbau. Zwar wächst der Druck des Marktes, die Störungsüberwachung und Fehlerdiagnose besser zu unterstützen; doch sind die Unternehmen häufig nicht in der Lage, die hohen Einführungskosten zu tragen. Nur wenn dieses Hindernis überwunden wird, werden wissensbasierte Diagnosesysteme in diesen Branchen breiter eingesetzt werden. Das Fraunhofer-Institut für Informations- und Datenverarbeitung IITB stellte sich daher in INDIA der Aufgabe, ausgehend von der THEN-Anwendung Lösungsansätze für dieses Problem zu entwickeln.

Ansatz und Ergebnisse

Der von THEN produzierte Chemikaliendistributor CHD schien aus verschiedenen Gründen als Ausgangspunkt gut geeignet: Erstens gibt es eine gemeinsame Plattform für alle Varianten und Installationen und damit eine Basis für die Wiederverwendung von Diagnosewissen durch den Hersteller THEN. Zweitens finden sich die beteiligten Komponenten, wie Rohrleitungen, Pumpen, Ventile und Steuerungen, überall in der Versorgungstechnik, aber z. B. auch im Schiffsbau wieder. Damit besteht Aussicht für einen Hersteller von Diagnosesystemen, die hierfür entwickelten Modelle und Diagnosemethoden in anderen Branchen wiederverwenden zu können. Drittens schließlich ist der Chemikaliendistributor ein typisches Beispiel für ein *reaktives System*, das einen technischen Ablauf steuert, indem es auf asynchron auftretende Ereignisse in seiner Umgebung mit Änderungen seines Betriebszustandes reagiert. Diese Klasse von Systemen findet mit der fortschreitenden Automatisierung in Produktion und Dienstleistung immer weitere Verbreitung.

Auf Basis der modellbasierten Diagnoseshell MuDia entstand mit CHD-DIAS (*Ch*emical *D*istributor - *Di*agnostic *A*ssistant) zunächst der erste Prototyp eines Post-Mortem-Diagnoseassistenten für den Chemikaliendistributor. Das Black-Box-Verhalten der Komponenten wurde dabei im wesentlichen durch erweiterte Zustandsdiagramme beschrieben. Dieser Formalismus wird industriell gerade zur Beschreibung reaktiver Systeme immer häufiger eingesetzt. CHD-DIAS vergleicht diese Modelle mit gesammelten Meßdaten und nötigenfalls auch mit Beobachtungen des Bedienpersonals, um mittels konsistenzbasierter Diagnose den Fehler zu identifizieren.

Die bei dieser Entwicklung gewonnenen Erkenntnisse wurden dann zu einer Methodik verallgemeinert, nach der in einem Anwendungsbereich existierende Zustandsdiagramme durch die Fachexperten für die Diagnose erweitert und verwendet werden können. Die Fachexperten werden dabei durch gezielte Fragen angeleitet. Um diese Diagramme zu repräsentieren und aus ihnen Diagnosemodelle zu erzeugen, wurden die Sprache AML und der Übersetzer SCI (*S*tate *C*hart *I*ntegrator) entwickelt. Mit dieser Vorgehensweise können Anwendungsexperten große Teile des zur Diagnose benötigten Wissens mit Hilfe ihnen bekannter Werkzeuge selbständig spezifizieren. Dies senkt die Kosten für externe Diagnoseexperten.

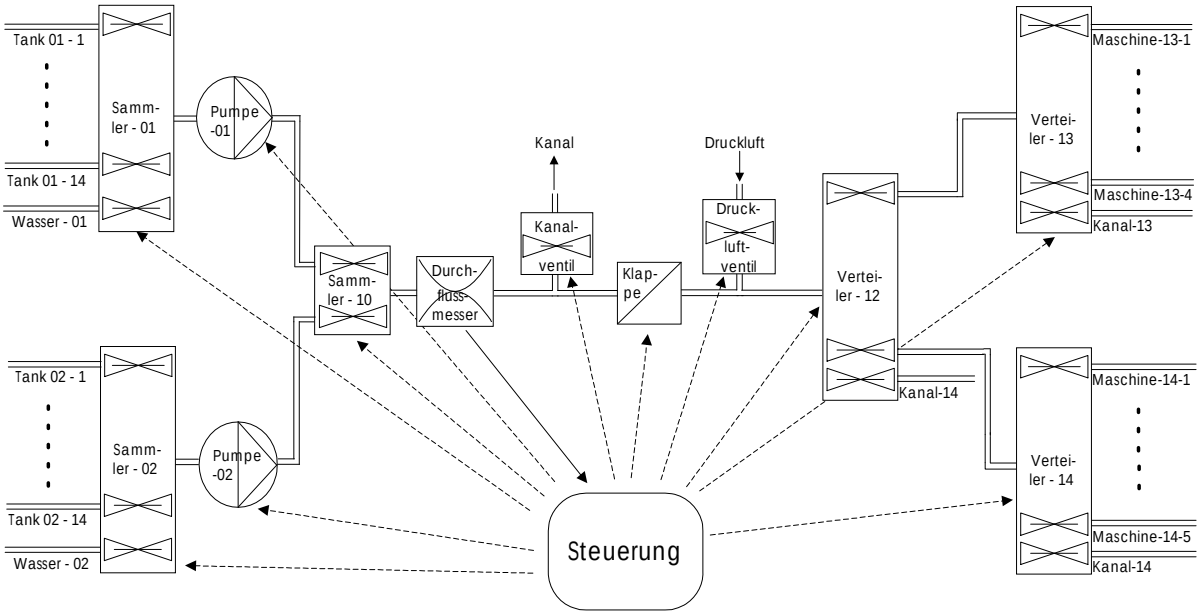


Abbildung 4-1: Anlagenkomponenten zum Chemikalientransport

Gliederung

Die beiden folgenden Abschnitte stellen zunächst den Chemikaliendistributor CHD und das Diagnosesystem CHD-DIAS näher vor. Die sich daraus ergebenden Ansatzpunkte zur Senkung der Einführungskosten werden in Abschnitt 4.4 diskutiert und in den weiteren Abschnitten dargestellt. Als Grundlage dafür dient das in Abschnitt 4.5 vorgestellte Referenzmodell der Entwicklung von Diagnosewerkzeugen. Die erwähnte Verwendung erweiterter Zustandsdiagramme setzt dabei während der Wissensakquisition beim Anlagenhersteller an (Abschnitt 4.6). In INDIA wurden aber auch Möglichkeiten zur Kostenreduktion während der Werkzeugentwicklung (Abschnitt 4.7) und während der Generierung von Laufzeitsystemen (Abschnitt 4.8) untersucht. Eine Zusammenfassung und Diskussion der Ergebnisse bildet den Abschluß.

4.2 Der Chemikaliendistributor CHD

Während eines Färbevorgangs müssen in der Regel drei- bis viermal verschiedene Chemikalienmischungen zur Färbemaschine geleitet werden. Der Chemikaliendistributor THEN CHD steuert das Abmessen der Chemikalien und den Transport von den Chemikalientanks durch Rohrleitungen bis zur richtigen Färbemaschine. Den Auftrag dazu (*Chemikalienabruf*) erhält er entweder direkt von einem Leitsystem oder über ein Bedienpult vom Färbereipersonal.

Die Chemikalien sind in Behältern gelagert, die in Gruppen zusammengefaßt werden können. Die Lagerbehälter sind über sogenannte *Sammler* an das Leitungsnetz angeschlossen, über die gezielt einzelne Chemikalien eingeleitet werden können. Mehrere Behältergruppen werden über weitere Sammler hierarchisch gruppiert an das Leitungssystem angeschlossen. Über diese Sammler können auch andere Anlagenkomponenten, beispielsweise Farblösestationen, integriert werden. Hinter dem Sammler werden die Chemikalien über eine Leitung zur Färberei transportiert, wo sie über *Verteiler* zu den einzelnen Färbeapparaten geleitet werden.

Die Grundversion des Chemikaliendistributors besteht aus einem Sammler, an den die Chemikalienbehälter und ein Wasseranschluß angeschlossen sind, einer nachgeschalteten Pumpe, einem Durchflußmesser und einem Durchflußregler. Abbildung 4-1 zeigt eine komplexere Variante, wie sie bei der MEZ GmbH in Freiburg installiert ist und vom IITB in INDIA betrachtet wurde. Im folgenden soll ihre Arbeitsweise beim Chemikalientransport dargestellt werden[6].

6. vgl. [Brinkop und Höfer, 1996]

1. *Ruhezustand*:

   Die Ventile an den Sammlern zu den Lagerbehältern der Chemikalien sind geschlossen. Durch eine geschlossene Absperrklappe nach dem Durchflußmesser ist das Rohrleitungsnetz in zwei entkoppelte Systeme aufgeteilt. Die Komponenten zwischen den Chemikalienbehältern und der Absperrklappe sind mit Wasser gefüllt, die Komponenten zwischen Absperrklappe und Färbemaschinen dagegen mit Luft.

2. *Zusammenstellung eines Chemikalienzugs*:

   Zunächst werden alle Ventile und Klappen geöffnet, die sich zwischen dem Tank mit der ersten Chemikalie und der anfordernden Färbemaschine befinden. Dann läuft die Pumpe an. Wenn der Durchflußmesser das Erreichen der Sollmenge meldet, schließt das der Chemikalie zugeordnete Ventil, und die Pumpe stoppt. Dieser Vorgang wird für alle angeforderten Chemikalien wiederholt.

3. *Chemikalientransport*:

   Ist die Dosierung der einzelnen Chemikalien abgeschlossen, so befindet sich mindestens ein Teil der Chemikalien noch im Leitungssystem vor der Absperrklappe. Der Durchflußregler wird geöffnet, das Wasserventil bleibt offen, und es wird soviel Wasser in das Leitungssystem gepumpt, bis sich die komplette Chemikalienmischung hinter der Absperrklappe befindet. Dann wird die Absperrklappe geschlossen, ein Druckluftventil geöffnet und der komplette „Zug“ zu den Apparaten geblasen.

   Anschließend wird das komplette Leitungssystem gespült. Die Wasserventile der Sammler werden geöffnet und die Pumpen aktiviert. Die Sammler werden entsprechend angesteuert. Die Absperrklappe ist geöffnet und eine vorgegebene Menge Wasser wird durch das System gepumpt. Anschließend wird das Leitungssystem hinter der Absperrklappe mittels Druckluft geleert.

Kommt es bei dieser extrem zuverlässigen Anlage tatsächlich einmal zu einer Störung, dann ist die Ursache in der Regel eine Fehlbedienung, wie z. B. das Verwenden zu zäher Chemikalien. Rein theoretisch kann es auch zum Verklemmen von Ventilen oder zu Fehlfunktionen der Dosiereinheit kommen. Aufgrund der zentralen Stellung dieser Anlage innerhalb des Färbereibetriebs sind mächtige Diagnosefunktionen daher eine Beruhigung für die Betreiber und ein wichtiges Verkaufsargument für THEN.

4.3 Unterstützung des Färbereipersonals bei der CHD-Diagnose

Anforderungen

Ausgangspunkt für die Gestaltung von CHD-DIAS war eine zusammen mit THEN durchgeführte Analyse der Anwendung, auf deren Grundlage das IITB zunächst einen Mock Up erstellte. Wenn Färbereimitarbeiter Unregelmäßigkeiten bei der Funktion des Chemikaliendistributors feststellen, dann sollte ihnen dieses System bei der Fehlersuche helfen. Solche Unregelmäßigkeiten können insbesondere ein kompletter Stillstand der Anlage, ein endloser Fluß in den Abwasserkanal oder den Zielbehälter an einer Färbemaschine sein, aber auch, daß eine angeforderte Chemikalienmischung nicht an der Zielmaschine ankommt. Allerdings kann das Personal in der Regel nicht entscheiden, ob eine Mischung alle angeforderten Chemikalien in der richtigen Menge enthält. Viele Dosierfehler führen zu Qualitätsmängeln, die teilweise nicht sofort erkennbar sind.

Abbildung 4-2 zeigt, wie diese Anfangssymptome beim Mock-Up spezifiziert werden können. Dabei kann der Bediener auch angeben, bei welchen der aktuell laufenden oder kürzlich beendeten Färbevorgänge (Partien) und welchen der dazu gehörenden Abrufe er die Symptome beobachtet hat.

Zur Diagnose benötigt das Diagnosesystem einerseits Informationen zur Konfiguration der Anlage, z. B. in welchen Tanks sich eine angeforderte Chemikalie befindet, wo welche Maschine angeschlossen ist und wie das Leitungsnetz aussieht. Diese Daten sind in der Projektierungsdatenbank des Färbereileitsystems bzw. des Chemikaliendistributors vorhanden. Um den Wartungsaufwand zu verringern und Inkonsistenzen zu vermeiden, sollen sie von dort direkt in das Diagnosesystem übernommen werden.

Andererseits werden zur Diagnose Daten über den Ablauf der gestörten Chemikalienabrufe benötigt. Da aus Kostengründen für die Diagnose keine zusätzlichen Sensoren eingeführt werden, stehen hier lediglich die in der Steuerung verfügbaren Signale zurVerfügung. Dies sind der Zustand der Steuerung, die Stellsignale an die Aktoren und die Rückmeldungen der Dosiereinheit. Die Historien dieser binären Signale sollten von einer Überwachungseinheit vollständig erfaßt und für eine vorgegebene Zeit gespeichert werden.

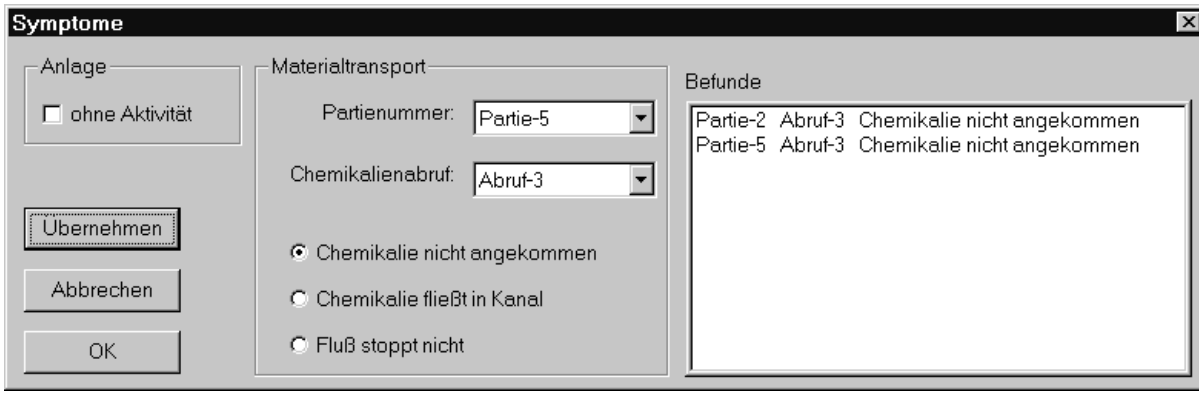


Abbildung 4-2: Eingabe der Anfangssymptome (Mock Up)

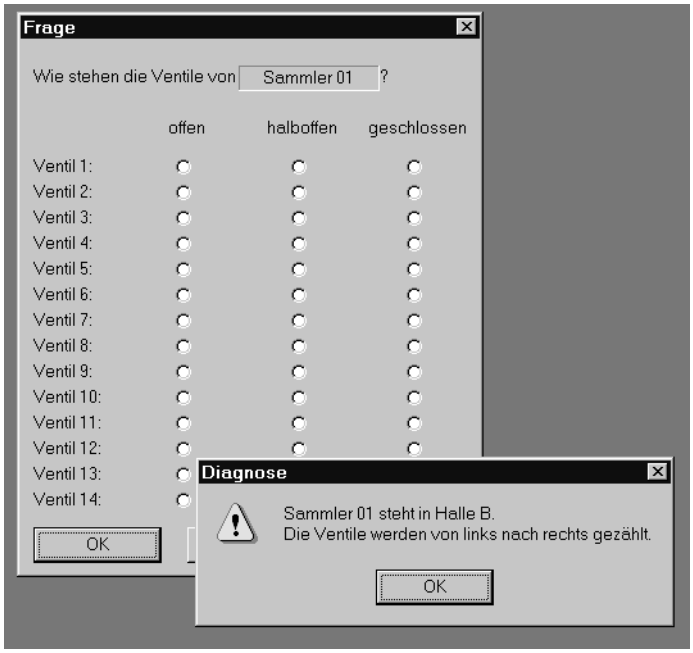


Abbildung 4-3: Dialog zur Eingabe von Ventilständen mit Hilfetext (Mock Up)

Diese minimale Sensorik stellt eine besondere Herausforderung für die Diagnose dar. Wo sie nicht ausreicht, kann das Bedienpersonal auf Anfrage zusätzliche Beobachtungen spezifizieren, die sich aber nur auf den aktuellen Zustand der Anlage, nicht auf einen beliebigen Zustand in der jüngsten Vergangenheit beziehen können. Neben den schon genannten Anfangssymptomen handelt es sich dabei im wesentlichen um aktuelle Ventilstände (Abbildung 4-3).

Das Diagnosesystem sollte PC-basiert sein, um auch auf dem PC des Leitsystems lauffähig zu sein. Abbildung 4-4 skizziert die sich aus diesen Anforderungen ergebende Einbettung in das informationstechnische Umfeld der Färberei.

CHD-DIAS

Auf der Grundlage der ermittelten Anforderungen wurde bei Fraunhofer IITB das System CHD-DIAS (*Ch*emical *D*istributor - *Dia*gnostic *A*ssistant) entwickelt. Dabei handelt es sich um einen Prototypen, bei dem zwar insbesondere die Schnittstellen zu Projektierungsdatenbank und Bedienpersonal noch nicht den Anforderungen genügen, der aber die Tragfähigkeit des Diagnoseansatzes demonstriert.

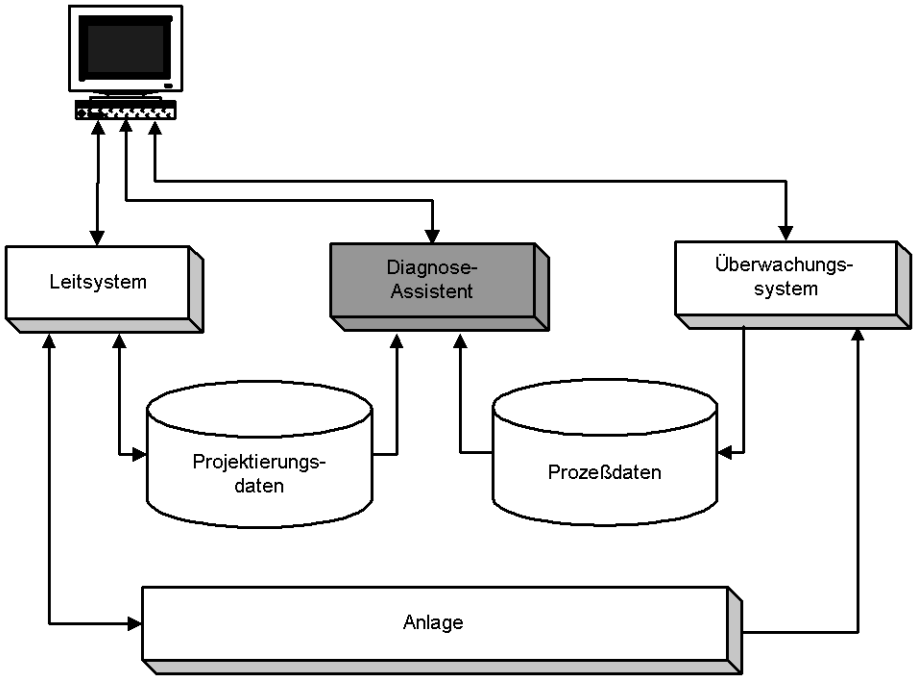


Abbildung 4-4: Szenario: Einbettung des Diagnoseassistenten in das informationstechnische Umfeld

CHD-DIAS verwendet die am IITB entwickelte Diagnosemaschine MuDia. Es ist damit ein modellbasiertes Diagnosesystem in der Tradition von GDE [deKleer, Williams 1987], das Inkonsistenzen zwischen Beobachtungen und Anlagenmodellen analysiert. Nach dem Start der Diagnose liest CHD-DIAS zunächst aus einer Datei die zum verdächtigten Chemikalienabruf gehörenden Prozeßdaten ein und prüft sie auf Konsistenz mit Modellen des Chemikaliendistributors und seiner Komponenten. In dieser Phase werden noch nicht die einzelnen Ventile, sondern Sammler und Verteiler als ganzes betrachtet.

Bestimmte Fehlfunktionen des Durchflußmessers werden in dieser Phase schon eindeutig lokalisiert. Meldet der Durchflußmesser jedoch keinen Strom obwohl laut Historien der Steuersignale ein Weg freigeschaltet sein müßte, dann müssen mangels weiterer Sensoren alle Komponenten auf dem Weg verdächtigt werden: Eine von ihnen hat offenbar auf das Steuersignal nicht in der gewünschten Weise reagiert. Um zwischen diesen Komponenten zu diskriminieren, wählt CHD-DIAS eine der komplexeren Komponenten aus und untersucht sie näher. Im allgemeinen kann dabei zwar der Fehler weiter eingeschränkt werden, ohne ihn aber eindeutig lokalisieren zu können (Abbildung 4-5). In dieser Situation wird CHD-DIAS den Bediener nach Beobachtungen an einer der verdächtigten Komponenten fragen[7].

CHD-DIAS lokalisiert auf diese Weise defekte Pumpen, geschlossen verklemmte Ventile und Fehlfunktionen der Dosiereinheit.

Getestet wurde das System bisher mit Daten, die durch Simulation von Anlagenfehlern erzeugt wurden. Die Datenaufnahme an einem realen Chemikaliendistributor erwies sich für THEN als zu aufwendig, da dazu Hard- und Software der CHD-Steuerung hätten erweitert werden müssen. Dies unterstreicht die Notwendigkeit, die Entwicklung des Diagnosesystems stärker mit der Entwicklung der Anlage zu integrieren, um frühzeitig die Anforderungen der Diagnose berücksichtigen zu können (vgl. Abschnitt 4.4).

Bei der Erzeugung der Simulationsdaten kam uns zustatten, daß es sich jeweils um binäre Signale handelt. Real auftretende Zufälligkeiten zeigen sich daher nur im Zeitbereich, verursacht z .B. durch schwankende Schaltzeiten der Ventile. Im Simulationsmodell wurden diese Zufälligkeiten mit Hilfe stochastischer Elemente berücksichtigt.

7. Genauer: jede Ursache entspricht einer Menge von Strukturelementen, z. B. Komponenten. CHD-DIAS wählt dann nach verschiedenen Heuristiken ein Strukturelement aus, wobei jene aus einelementigen Mengen bevorzugt werden.

Abbildung 4-5: Oberfläche von CHD-DIAS. Das Kandidatenfenster zeigt verschiedene mögliche Ursachen und ihre Wahrscheinlichkeit an. Die Komponenten der ausgewählten Alternative werden im Anlagenbild farblich gekennzeichnet.

Modellierung

Modellbasierte Diagnosemaschinen wie MuDia analysieren Inkonsistenzen zwischen Anlagenmodellen und Beobachtungen. Grob enthalten die für CHD-DIAS benutzten Modelle folgende Informationen:

- Sowohl für die komplexen Sammler und Verteiler als auch für Ventile, Klappen und Pumpen beschreiben die Modelle, wie die zugehörige Steuergröße den Durchfluß durch die Komponente ermöglicht oder verhindert.
- Das Strukturmodell des Gesamtsystems beschreibt, welche Wege es gibt. Ist für alle Komponenten auf einem Weg der Durchfluß möglich, dann findet nach diesem Modell auch ein Fluß statt.
- Das Modell des Durchflußmessers beschreibt, daß die Komponente regelmäßig einen Impuls an die Steuerung sendet, wenn ein Durchfluß stattfindet.
- Die Steuerung der Anlage wurde zwar für Simulationszwecke ebenfalls modelliert; für die Diagnose ist sie aber nicht nötig, da Ausfälle der Steuerung nicht betrachtet werden und alle ihre Ein- und Ausgangsgrößen vollständig beobachtbar sind.

Im Gegensatz zu anderen INDIA-Anwendungen werden von CHD-DIAS dynamische Modelle verwendet[8]. Warum ist der damit verbundene Aufwand nötig?

Für die konsistenzbasierte Diagnose genügen statische Modelle offensichtlich dann, wenn sich mit ihrer Hilfe Beobachtungen zu einem Zeitpunkt eindeutig einem Fehler zuordnen lassen[9]. Umgekehrt benötigt man dynamische Modelle, wenn es keinen beobachtbaren Systemzustand gibt, der für Fehlverhalten allgemein oder für einen

8. Ein Modell ist dynamisch, wenn es die Werte seiner Modellvariablen zu verschiedenen Zeiten in Beziehung setzt.

9. Vgl. auch [Struß 1997]

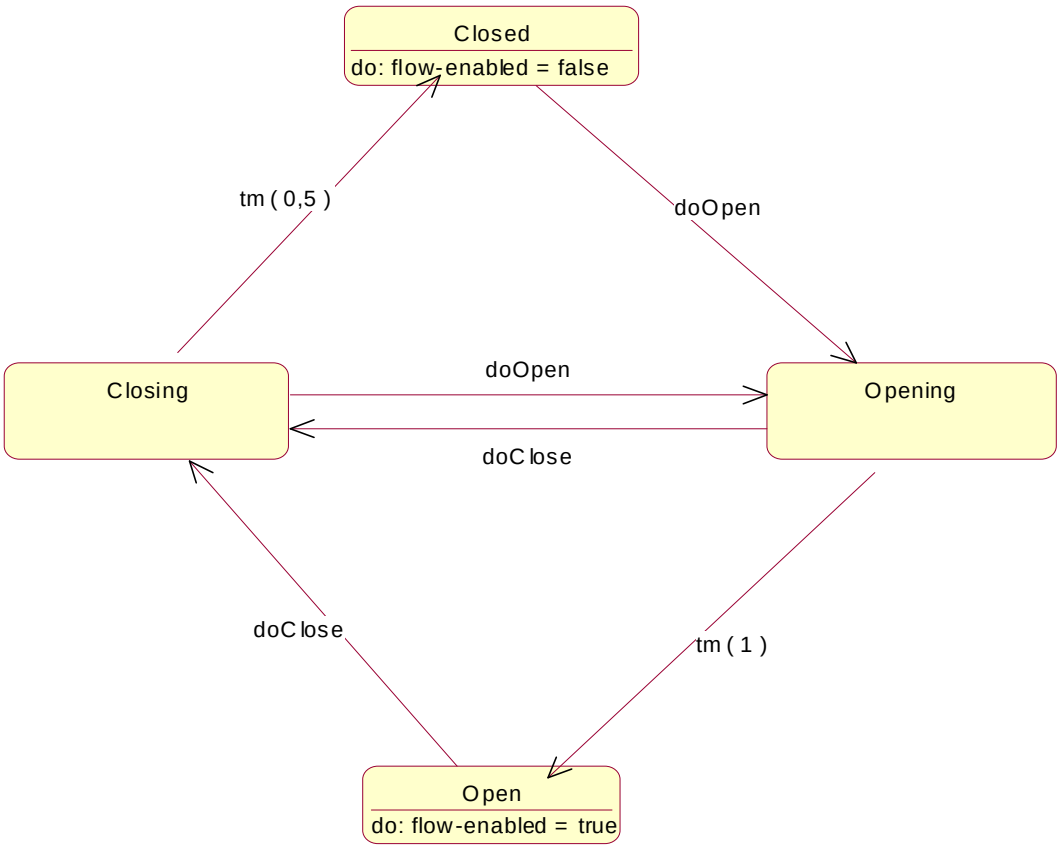


Abbildung 4-6: Erweitertes Zustandsdiagramm für einVentil:

bestimmten Fehler charakteristisch ist, wenn sich Fehler also nur im Verlauf des Systemzustandes über der Zeit zeigen. Dies ist beim Chemikaliendistributor der Fall.

Betrachten wir z. B. den typischen Fehler, daß sich ein Ventil nicht mehr öffnen läßt. Letztlich führt dies zu einem Zustand, in dem das Binärsignal des Durchflußmessers auf 0 liegt, obwohl die Steuersignale für die Pumpe und für alle Ventile auf einem bestimmten Weg gesetzt sind. Dies ist jedoch für sich allein kein Fehlercharakteristikum! Zum einen zeigt sich dieser Zustand auch direkt nach dem Setzen der Steuersignale, da die Ventile Zeit zum Öffnen und die Pumpe Zeit zum Anlaufen benötigen. Zum anderen signalisiert der Durchflußmesser einen Durchflusses nicht durch eine dauerhafte 1, sondern sendet kurzzeitig einen Impuls, wenn seit dem letzten Impuls eine bestimmte Menge Flüssigkeit geflossen ist. Charakteristisch für das Vorliegen eines Fehlers ist also nicht der genannte Zustand selbst, sondern seine lange Dauer.

Die von CHD-DIAS verwendeten Modelle nutzen dieses Charakteristikum zur Diagnose. Das Verhalten von Sammlern, Verteilern, Ventilen, Klappen und Pumpen wird dabei im wesentlichen durch erweiterte Zustandsdiagramme beschrieben.

Abbildung 4-6 zeigt beispielsweise das Zustandsdiagramm für ein Ventil. Neben den beiden stabilen Zuständen *open* und *closed* werden die beiden Übergangszustände *opening* und *closing* spezifiziert. Während stabile Zustände im wesentlichen dadurch charakterisiert werden, ob in ihnen ein Durchfluß möglich oder unmöglich ist, wird in Übergangszuständen darüber nichts ausgesagt. In der Realität wird z. B. gegen Ende des Öffnens eines Ventils schon ein Fluß erfolgen können.

Die Steuerereignisse „*doOpen*“ bzw. „*doClose*“ entsprechen den positiven Flanken in den entsprechenden Steuersignalen des Controllers an das Ventil. Nur aufgrund solcher Steuerereignisse wechselt das Ventil von einem stabilen in einen Übergangszustand.

Falls ein Übergangszustand nicht vorzeitig aufgrund eines Steuerereignisses verlassen wird, sorgt nach einer für diesen Zustand parametrierten maximalen Zeitspanne ein Timeout-Ereignis für den Wechsel in einen stabilen Zustand. Das Modell fordert dabei nicht, daß Öffnen bzw. Schließen desVentils in der Realität genau bei Ablauf dieser maximalen Dauer beendet werden. Es legt nur fest, daß ohne zwischenzeitliche Steuersignale das Ventil spätestens nach Ablauf dieser Zeit offen bzw. geschlossen sein wird. Auf diese Weise können auch die unvermeidbaren Schwankungen in der Dauer solcher Abläufe auf einfache Weise ausreichend berücksichtigt werden.[10]

Dieses Verhaltensmodell wurde nicht nur für dasVentil, sondern nach Anpassung der maximalen Übergangsdauern auch für Sammler, Verteiler und die Klappe verwendet. Auch das Pumpenmodell ist analog aufgebaut, wobei Öffnen und Schließen desVentils dem An-und Abschalten der Pumpe entsprechen.

Damit diese Modelle aber von unserer Diagnosemaschine MuDia verwendet werden konnten, mußten sie in dessen Modellierungssprache DEEP[11] spezifiziert werden. Insbesondere mußten dabei im Diagramm implizit enthaltene Informationen explizit formuliert werden, z. B. daß Ereignisse folgenlos bleiben, wenn im aktuellen Zustand keine Reaktionen auf das Ereignis spezifiziert sind. Im ersten Anlauf geschah diese Übersetzung von Hand. Diese handoptimierten Modelle wurden zum „Goldstandard“ für eine automatische Übersetzung (vgl .Abschnitt 4.7).

Realisierung

CHD-DIAS besteht aus zwei Teilen: dem in Lisp implementierten eigentlichen Kernsystem und der in Java implementierten Oberfläche. Beide Teile kommunizieren über TCP-IP-Sockets und können daher auf verschiedenen, über Internet gekoppelten PCs ablaufen. Das Kernsystem setzt sich aus der anwendungsunabhängigen Diagnoseschale MuDia und anwendungsabhängigen Informationen zum Aufbau des Chemikaliendistributors, verfügbaren Sensoren und der Diagnosestrategie zusammen. Auch die Oberfläche läßt sich in ein anwendungsunabhängiges und ein CHD-spezifisches Modul aufteilen. Ersteres wickelt im wesentlichen die Kommunikation mit dem Kernsystem ab, letzteres enthält die bedienbaren Anlagenbilder und die Anlagendokumentation.

4.4 Erfahrungen aus der Entwicklung von CHD-DIAS

CHD-DIAS bestätigte erneut, daß sich mit den vorhandenen Techniken zur intelligenten Diagnose Systeme realisieren lassen, die den Betreibern von Maschinen und Anlagen einen echten Nutzen schaffen. An CHD-DIAS zeigten sich aber auch wieder zwei bekannte Probleme: die hohen Einführungskosten und die Trennung von Diagnosesystem- und Produktentwicklung.

Hohe Kosten zur Einführung wissensbasierter Diagnose

Zwar lohnt sich mittelfristig für viele Unternehmen im Maschinen- und Anlagenbau die Entwicklung gerade modellbasierter Diagnosesysteme durch Möglichkeiten zur Wiederverwendung und zur Integration verschiedener Arbeitsprozesse. CHD-DIAS zeigte jedoch, daß zunächst einmal erhebliche Kosten entstehen, um diese Technologie einzuführen.

Eine wesentliche Ursache dieser Kosten war, daß das zur Diagnose benötigte Wissen auf verschiedene Personengruppen verteilt vorlag. Die Domänenexperten von THEN besaßen das Wissen über Struktur und Arbeitsweise des Chemikaliendistributors sowie die vorhandenen Unterlagen. Fraunhofer IITB dagegen besaß das notwendige Wissen aus Informatik und Künstlicher Intelligenz, um die vorhandenen Wissensrepräsentationswerkzeuge und Diagnosetechniken optimal nutzen zu können. Zur Abstimmung zwischen beiden Gruppen war erheblicher Aufwand notwendig. Damit CHD-DIAS dann letztlich von Fraunhofer IITB entwickelt werden konnte, mußten sich die Wissenschaftler in das Anwendungsgebiet einarbeiten. Dies erforderte auf beiden Seiten ebenfalls erheblichen Aufwand.

Eine zweite Kostenquelle war die fehlende Methodik zur Entwicklung der zur Diagnose benötigten Modelle. Die Modelle mußten daher zum überwiegenden Teil zeitraubend durch Versuch und Irrtum entwickelt werden: man testete, ob mit Hilfe der aktuellen Modelle alle relevanten Fehler in akzeptabler Zeit gefunden werden. War dies nicht der Fall, dann wurden die Modelle ergänzt. Dauerte dagegen die Diagnose zu lange, dann suchte man die Modelle zu vereinfachen ohne an Diagnosefähigkeit zu verlieren.[12] Trotz der Erfahrung aus vorangegangenen Entwicklungsprojekten entstand hier für Fraunhofer IITB erheblicher Aufwand.

10. Dieser Ansatz wurde auch in [Zimmermann-Sturm 1996] erfolgreich verwendet.

11. Mit 'DEEP' wird die Kombination der Modellierungssprachen für die beiden Werkzeuge DAM und EEP bezeichnet.

Trennung von Diagnosesystem- und Produktentwicklung

CHD-DIAS wurde für ein existierendes Produkt entwickelt. Da die in der Steuerung vorhandenen Signalwerte an diesem Produkt nicht protokolliert werden konnten, mußte CHD-DIAS mit simulierten Daten getestet werden. Eine entsprechende Erweiterung der Steuerung des bei MEZ betriebenen Chemikaliendistributors war für THEN jedoch zu aufwendig und zu riskant. Dieses Problem ist typisch für Anlagenhersteller. Viele von ihnen verfügen nur über kleine eigene Testanlagen und testen daher Innovationen bei speziellen Pilotkunden. Ein solcher Test ist in der Regel zwar aufgrund der Praxisnähe sehr aussagekräftig; allerdings steigt der Aufwand zur Durchführung, und ein Fehlschlag kann zu wesentlich schwerwiegenderen Konsequenzen führen. Billiger und weniger riskant ist es in der Regel, wenn von der Diagnose benötigte Funktionen schon bei der Entwicklung der Anlage erkannt und berücksichtigt werden („Design for Diagnosis“).

Lösungsansätze

Nach Auffassung von Fraunhofer IITB müssen die genannten Probleme gelöst werden, damit sich wissensbasierte Diagnosesysteme industriell durchsetzen können. Von Anfang an war daher unsere zweite Aufgabe in INDIA, ausgehend von den Erfahrungen mit CHD-DIAS nach geeigneten Lösungsansätzen zu suchen.

Um den Rahmen für diese Suche abzustecken wurde zunächst ein Referenzmodell der Diagnosesystementwicklung formuliert, das in Abschnitt 4.5 vorgestellt wird. Es identifiziert die verschiedenen Phasen von der Modulentwicklung bei Systemhäusern wie der ROSE Informatik GmbH bis zur Wartung des Diagnoselaufzeitsystems.

Innerhalb dieses Rahmens konzentrierten wir uns auf drei Ansätze, um Kosten im Entwicklungsprozeß zu reduzieren:

- Die Experten beim Hersteller des zu diagnostizierenden Produkts, also z. B. bei THEN, sollten das zur Diagnose benötigte Wissen soweit wie möglich selbständig akquirieren. Dazu benötigen sie Unterstützung durch geeignete Werkzeuge in der Entwicklungsumgebung. Als Basis für solche Werkzeuge entwickelten wir eine Menge von Heuristiken. Sie sind anwendbar, wenn für das Produkt Verhaltensbeschreibungen in Form von Zustandsdiagrammen vorliegen. Sie scheinen aber auch auf andere Formen von Flußdiagrammen übertragbar. (Abschnitt 4.6).
- Die benötigten Flußdiagramme sollten möglichst aus kommerziellen CAx-Systemen in Diagnoseentwicklungsumgebungen importiert werden können. Wir suchten nach Ansätzen, mit denen Diagnosewerkzeuganbieter wie die ROSE Informatik die dazu benötigten Importfilter kostengünstig entwickeln können (Abschnitt 4.7). Dadurch besteht zum einen die Chance, für andere Zwecke erstellte Modelle als Basis für die Diagnose wiederverwenden zu können. Zum anderen können Domänenexperten zusätzlich benötigte Modelle mit den vertrauten Werkzeugen erstellen.
- Die Kosten der Wissensakquisition sollten möglichst auf die Entwicklung verschiedener Laufzeitsysteme verteilt werden. So kann es z. B. zur Diagnose des Chemikaliendistributors unterschiedliche Diagnosesysteme für die Post-Mortem-Diagnose durch das Färbereipersonal, für die Ferndiagnose durch THEN-Techniker und die Diagnose vor Ort durch THEN-Techniker geben. Wir untersuchten, wie die Generierung solcher unterschiedlichen Diagnosesysteme aus derselben Wissensbasis unterstützt werden kann (Abschnitt 4.8).

4.5 Referenzmodell für die Diagnosesystementwicklung

4.5.1 Motivation

Die Diagnoseaufgabe kann in vielen Fällen nicht getrennt von Monitoring und Fehlerbehandlung gelöst werden. Im folgenden werden die drei Aufgaben daher unter dem Akronym *MDF* zusammengefa8t.

Der Gestaltung von Entwicklungswerkzeugen für wissensbasierte MDF-Funktionen muß ein Modell des Entwicklungsprozesses zugrundeliegen, das beschreibt,

- welche Funktionen

12. Verbesserungen können auch durch Verwendung weiterer Beobachtungen erzielt werden - sofern der Anlagenhersteller die Kosten für entsprechende Sensoren akzeptiert.

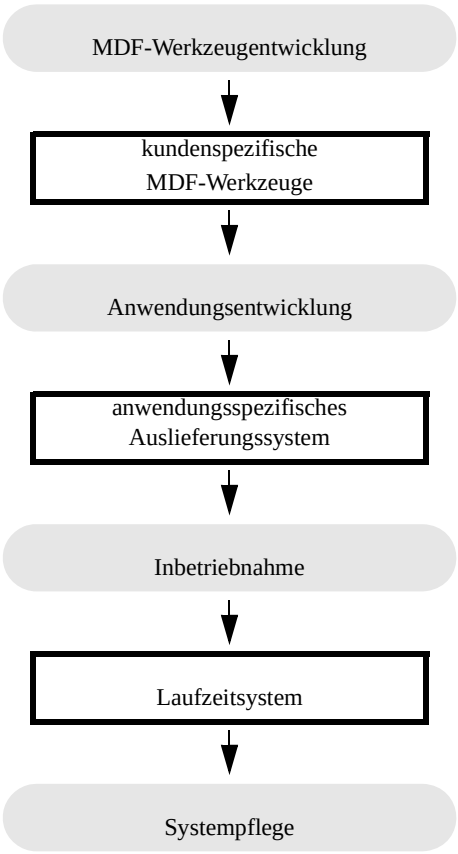


Abbildung 4-7: Modell des Entwicklungsprozesses für MDF-Systeme

- wer
- auf welchem Weg

entwickelt. Daraus ergeben sich Hinweise auf Möglichkeiten zur Kostensenkung sowie Anforderungen an die jeweils benötigten Werkzeuge.[13] Dieser Abschnitt schlägt ein solches Modell vor (vgl. Abbildung 4-7). Es bezieht vier Entwicklungsphasen ein, die sich in ähnlicher Weise auch in anderen Entwicklungsprozessen finden:

- Die *Entwicklung von MDF-Werkzeugen*, z. B. durch Systemhäuser wie der ROSE Informatik GmbH. Das Spektrum reicht hier von monolithischen Diagnoseschalen bis zu Modulbibliotheken, in denen Basismodule selbst wieder Grundlage für speziellere Module bilden.
- Die *Entwicklung einer Anwendung* mit Hilfe der Werkzeuge, die als Datei oder als Hardwareprogramm in die Laufzeitumgebung ausgeliefert werden kann.
- Die *Inbetriebnahme* der ausgelieferten Anwendung in der Laufzeitumgebung. In dieser Phase werden die Anwendung in die Laufzeitumgebung eingepaßt und spezifische Parameter der Laufzeitumgebung festgelegt.
- Die *Systempflege* des Laufzeitsystems, z. B. durch Ändern von Einstellungen oder durch Einspielen von Patches und neuen Versionen.

Diese vier Phasen werden im folgenden näher besprochen.

13. Dieser Blickwinkel unterscheidet sich z. B. von der Sicht der im Software Engineering entwickelten Modelle (Wasserfallmodell, Spiralmodell, V-Modell, usw.). Mit diesen Modellen lassen sich aber einzelne Stufen weiter strukturieren.

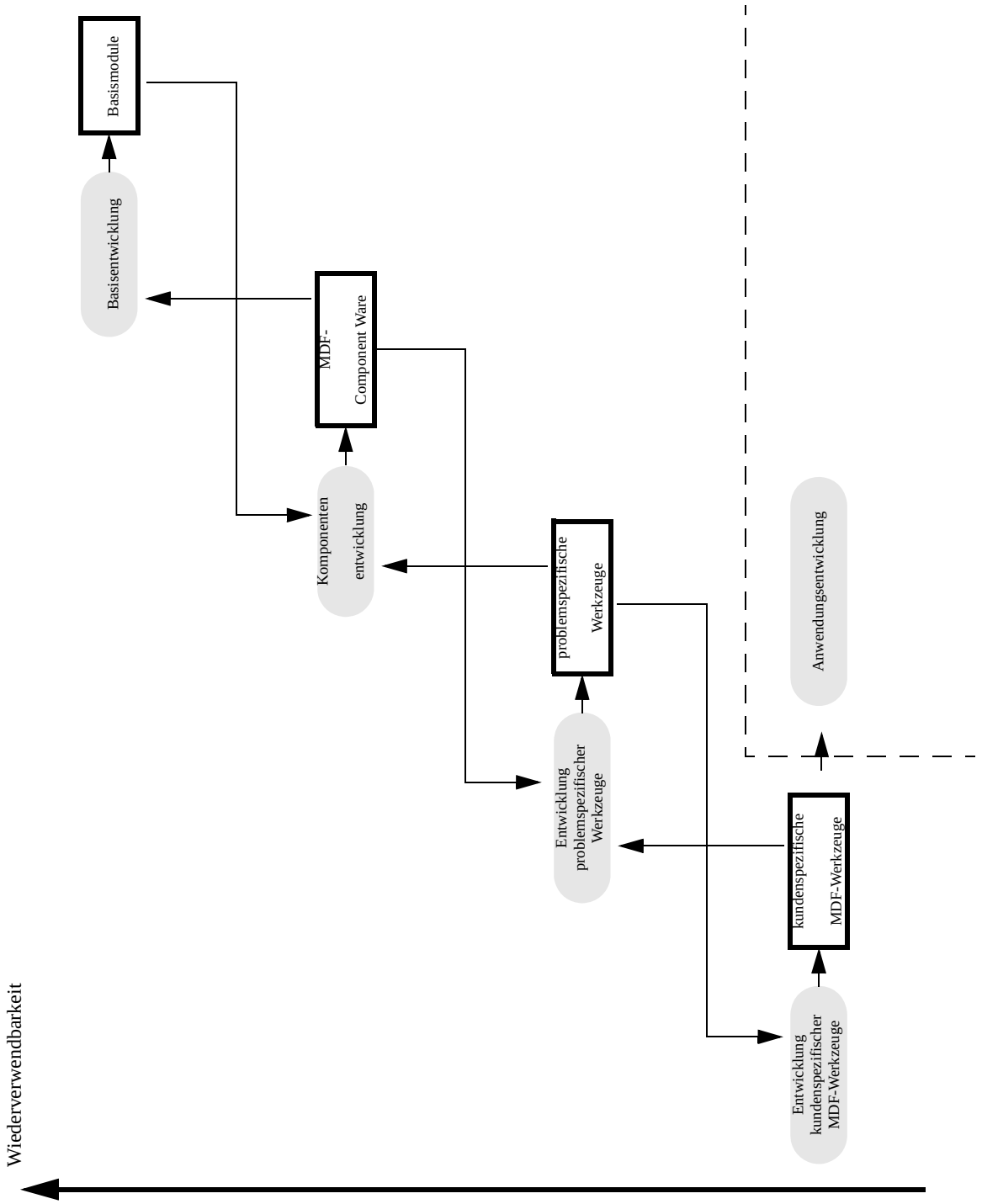


Abbildung 4-8: Referenzmodell für die MDF-Werkzeugentwicklung

4.5.2 Entwicklung von MDF-Werkzeugen

Abbildung 4-8 zeigt ein grobes Modell der verschiedenen Phasen der Werkzeugentwicklung, wie sie Fraunhofer IITB aus heutiger Sicht für die Zukunft erwartet. Es skizziert das Wechselspiel von Abstraktionsprozessen und Konfigurations- oder Anpassungsprozessen.

Entwicklung problemspezifischer MDF-Werkzeuge

Zur Realisierung wissensbasierter MDF-Systeme gibt es verschiedene Möglichkeiten. Falls die benötigten Mechanismen zur Wissensrepräsentation und zum Schlußfolgern sehr ungewöhnlich oder recht einfach sind, wird man die Anwendung unter Umständen direkt in einer höheren Programmiersprache implementieren. Im allgemeinen ist dies aber sehr aufwendig und damit teuer, so daß man in der Regel auf Werkzeugen aufzubauen sucht, die möglichst viel der benötigten Funktionalität preisgünstig zur Verfügung stellen.

Solche problemspezfischen, aber kundenunabhängigen Werkzeuge basieren auf den Gemeinsamkeiten, die bei der Entwicklung verschiedener kundenspezifischer MDF-Werkzeuge deutlich wurden. Der Markt bietet hier heute zwei Gruppen von Werkzeugen:

1. Aufgabenunabhängige Expertensystemschalen wie z .B. G2 oder RT-Works:

   Solche Systeme stellen auf der einen Seite Repräsentationsformalismen wie Fuzzy-Werte, Neuronale Netze, Regeln, Constraints oder Objekte, auf der anderen Seite Schlußfolgerungsverfahren wie Propagierung durch Neuronale Netze, Vorwärts- oder Rückwärtsverkettung zur Verfügung.

   Werkzeuge dieser Art werden insbesondere für Überwachungs- und einfache Diagnosesysteme verwendet, deren Aufgabe sich als Zuordnung von Handlungen oder Diagnosen zu Alarmen oder anderen Symptomen beschreiben läßt. Zur Realisierung komplexerer MDF-Methoden reichen die verfügbaren Schlußfolgerungsverfahren in der Regel nicht aus.

2. MDF-spezifische Expertensystemschalen:

   Diese Systeme bieten Methoden zur Lösung von Aufgaben aus dem Bereich Überwachung, Diagnose und Fehlerbehandlung sowie Primitive zur Repräsentation des dazu benötigten Wissens über Symptome, Fehler, Anlagenmodelle u. ä. Im Hinblick auf Überwachungsaufgaben gehören dazu SCADA[14]- und Leitsysteme, aber auch z. B. wissensbasierte Systeme zur Interpretation bestimmter Typen von Beobachtungen (z. B. wissensbasierte Schwingungsdiagnostik). Im Hinblick auf Diagnoseaufgaben handelt es sich um Systeme wie RODON[15] oder DIAMON [Marburger 1994]. Dazwischen stehen Systeme wie z .B. G2-GDA.

   Die Werkzeuge dieser Gruppe gewinnen zunehmend an Bedeutung, decken aber meist jeweils nur Teile des gesamten MDF-Aufgabenspektrums ab.

Diese Werkzeuge werden heute in der Regel vollständig unter Federführung eines einzelnen MDF-Systemhauses entwickelt. Für die Anforderungen größerer MDF-Projekte werden sie oft mit spezifischen Zusatzfunktionen ausgestattet, z. B. mit speziellen Oberflächen oder Schnittstellen zu anderen Informationssystemen. Für diesen Anpassungsprozeß hat sich der englische Begriff 'Customization' eingebürgert.

Entwicklung von Softwarekomponenten

Zwar existieren Ansätze zur Modularisierung aufgabenspezifischer Werkzeuge, z. B. bei G2; doch sind diese Systeme häufig eher monolithisch aufgebaut. Dies wirkt sich negativ auf die Gesamtkosten eines MDF-Systems aus, das mit einem solchen Werkzeug erstellt wird. Zum einen kann man keine Kosten sparen, indem man sich zunächst auf jene Funktionalität beschränkt, die auf absehbare Zeit benötigt wird. Zum anderen lassen sich kundenspezifische Anpassungen und Ergänzungen häufig nicht durch Verfeinern vorhandener Module oder Ergänzen neuer Module realisieren, sondern nur mit Hilfe von Änderungen am Werkzeug. Sie können daher in der Regel auch nicht von Drittunternehmen, sondern nur vom Werkzeughersteller durchgeführt werden. Wartung und abwärtskompatible Weiterentwicklung des Werkzeugs werden dabei aufwendig, denn verschiedene Kunden benötigen verschiedene Änderungen. Zwar werden sich die MDF-Werkzeughersteller bemühen, für einen Kunden realisierte Lösungen zu verallgemeinern. Erfahrungsgemäß entstehen dabei aber aufgrund des ständigen Termin- und Kostendrucks häufig keine voll befriedigenden Lösungen.

Das Modell deutet einen Weg an, der sich z. B. bei kaufmännischer (SAP R/3) oder mathematischer Software (Matlab / Simulink) bewährt hat: auf der dritten Stufe werden die problemspezifischen Werkzeuge in einzelne Module mit öffentlichen Schnittstellen zerlegt, die möglichst gut wiederverwendbar sind und interessierten Unternehmen als eigenständige Softwarekomponenten (*Component Ware*) angeboten werden können. Insbesondere kann dabei ein Kern aus problemspezifischen, aber domänenübergreifenden Modulen mit Modulen erweitert werden, die auf bestimmte Domänen (Elektrik, Hydraulik, usw.) oder Branchen im Maschinenbau zugeschnitten sind.

---

14. Supervisory Control and Data Acquisition

15. Vgl. Abschnitt 3

Am Beispiel der Domäne „Rohrleitungen und Armaturen“ erkennt man, daß diese unternehmensunabhängigen Anteile des Domänenwissens nicht unbedeutend sind. Zunächst sind z. B. die unterschiedlichen Typen und Funktionen von Ventilen in der Verfahrenstechnik geradezu paradigmatisch.[16] Weiterhin spielt z. B. Wissen über die typischen Fehler in Rohrleitungssystemen unter verschiedenen Betriebsbedingungen (verklemmte Ventile, Lecks,...) oder die Strategien zur Fehlersuche in solchen Systemen in vielen Teilen des Anlagenbaus eine große Rolle. Schließlich wird vieles durch Normen festgelegt, wie z. B. DIN 2429 für die Darstellung von Leitungssystemen. Ähnliches gilt auch für andere Domänen, wie z. B. „Elektrische Schaltkreise“. Die Kosten für die Entwicklung solcher domänenspezifischen Module können wesentlich verringert werden, wenn man die zwischen vielen Domänen existierenden physikalischen und technischen Analogien in einem gemeinsamen Kern repräsentiert.

Neben dem Domänenwissen sollte ein domänenspezifisches Modul z. B. eine objektorientierte CAD-Oberfläche zur Verfügung stellen, mit der ein Fachexperte die Typen der Anlagenteile aus einer Datenbank auswählen, die Kennwerte der einzelnen Instanzen sowie der Betriebsumgebung (Temperatur, Schwankungen in der Versorgung usw.) über Masken spezifizieren und ihre Kopplung graphisch beschreiben kann. Dadurch schwindet die Bedeutung des Wissensingenieurs, denn der Fachexperte kann nun das für ein MDF-Projekt relevante unternehmens- oder anlagentypspezifische Wissen in der Terminologie und Denkweise seines Fachgebiets formalisieren.

Ein Modul muß nicht unbedingt branchenspezifisches Wissen oder MDF-Funktionen anbieten, sondern kann auch andere Dienste bereitstellen:

- angepaßte oder erweiterte Fremdsoftware für Zusatzaufgaben (z. B. Signalaufbereitung, Datenbank, Dokumentenverwaltung);
- domänenspezifische Bibliotheken mit Modelltypen, Modellen, Strategien oder Symbolen;
- Treiber für Schnittstellen zu im Entwicklungs- oder Laufzeitumfeld vorhandener Software;
- Rahmen und „Infrastruktur“ zur Integration aller Module.[17]

Das kundenspezifische MDF-Werkzeug ist nun Ergebnis einer Auslegung, bei der die Module mit den in der Anwendungsentwicklung benötigten Funktionen ausgewählt, angepaßt und integriert werden. Diese Systemintegration kann entweder der MDF-Werkzeughersteller, der Anlagenhersteller oder auch ein Ingenieurbüro durchführen.

Die Anwendungsentwicklung wird so potentiell billiger. Zum einen muß man bei der Auslegung des Werkzeugs keine Module kaufen, die nicht benötigt werden. Zum anderen verringern sich die Kosten der Wissensakquisition, weil auch ein Großteil des domänenspezifischen Wissens in Form von Modulen zugekauft werden kann, eventuell auch nachträglich.

Auch für den Anbieter eines MDF-Werkzeugs bietet dieser modulare Ansatz Vorteile. Zum einen kann er sich auf die Entwicklung von Modulen mit seinen Kernkompetenzen konzentrieren und für andere Aufgaben Module anderer Hersteller einbinden oder Schnittstellen zu solchen Modulen anbieten.[18] Zum anderen kann er seine Module unter Umständen nicht nur Entwicklern von MDF-Systemen anbieten, sondern auch den Herstellern anderer Werkzeuge, z. B. für Anlagenentwurf oder FMEA. Denkbar ist auch, daß ein auf Technische Diagnostik spezialisiertes Unternehmen Standardsoftware und domänen*unabhängige* MDF-Module anderer Hersteller zur Entwicklung domänen*spezifischer* MDF-Module oder -systeme verwendet.

Entwicklung von Basismodulen

Die letzte Stufe im Modell deutet einen weiteren Abstraktionsschritt an, der sich z. B. in Diagnose-Frameworks wie GenDE [Leitch et al. 1991] oder Expertensystembaukästen wie D3 [Puppe et al. 1996] findet. Wissensbasis und Problemlösungsprozeß werden dort soweit zerlegt, daß Basismodule für verschiedene Softwarekomponenten

16. Eine gute Darstellung dieser Begriffe gibt [Hemming 1993].

17. Ein solches Modul wird häufig als 'Framework' bezeichnet:
„A framework embodies a generic design, comprised of a set of cooperating classes, which can be adapted to a variety of specific problems within a given domain. (...) A framework represents a generic design solution. It is a meta-solution encompassing a set of possible solutions, rather than any one solution, within a particular problem domain.“ [Taligent 1996].
Vgl. auch [Lewis et al. 1995].

18. Man spricht in diesem Zusammenhang auch von „ComponentWare“.

wiederverwendet werden können, z. B. für verschiedene domänenspezifische Bibliotheken oder verschiedene Inferenzmaschinen.

4.5.3 Anwendungsentwicklung

Dieser Abschnitt skizziert die Entwicklung einer MDF-Anwendung bis zur Auslieferung, z. B. als Teil eines ASIC[19] oder als Menge von Dateien. Die Inbetriebnahme dieses anwendungsspezifischen *Auslieferungssystems* wird im nächsten Abschnitt dargestellt.

Im Hinblick auf ihre Funktionalität lassen sich heutige wissensbasierte MDF-Anwendungen nach Spezialisierungsgrad und Benutzungskontext einordnen. Beim Spezialisierungsgrad reicht die Bandbreite von Systemen für eine bestimmte Anlage bis zu Systemen für eine Klasse von Anlagen eventuell unterschiedlicher Hersteller. CHD-DIAS ist ein Beispiel für die erste Kategorie. Auftraggeber ist hier im allgemeinen der betreffende Hersteller, die Entwicklung selbst kann von einem MDF-Systemhaus wie RIG übernommen werden. Am anderen Ende des Spektrums findet sich z. B. das System ROMEX zur Unterstützung der Instandhalter bei der Post-Mortem-Diagnose von Rotationsmaschinen, wie Zentrifugalpumpen, Zentrifugalkompressoren oder Gasturbinen.[20] Es wurde von einem MDF-Systemhaus entwickelt.

Unterschiedliche Benutzungskontexte für MDF-Anwendungen ergeben sich aus dem unterschiedlichen Ressourceneinsatz in verschiedenen Phasen der Diagnose und Behandlung von Fehlern. Auf diese Weise wird versucht, Störungen in Dauer und Kosten zu minimieren.

- Im allgemeinen wird zunächst das Bedien- oder Instandhaltungspersonal des Maschinen- bzw. Anlagenbetreibers[21] versuchen, die Störungsursache selbständig zu finden und zu beheben. Hilfsmittel dazu können Onboard-MDF-Systeme sein, aber auch z. B. bei Bedarf aktivierbare Post-Mortem-Systeme.
- Falls ihnen das nicht gelingt, kontaktiert man ein Servicezentrum des Herstellers der Maschine oder Anlage. Gerade bei telefonischem Kontakt kann die erste Anlaufstation ein Call Center sein. Hauptaufgabe der dortigen Mitarbeiter ist es, leichtere Probleme (z. B. Bedienungsfehler) zu lösen und schwerere an die Experten weiterzuleiten.
- Die Experten in der Servicezentrum werden zunächst im Telefonkontakt mit dem Betreiber eine Ferndiagnose versuchen, die u. U. durch Datenzugriff auf die gestörte Maschine / Anlage unterstützt wird.
- Erst wenn die Störung auch auf diesem Weg nicht behoben werden kann, muß entweder ein Servicetechniker anreisen oder - z. B. bei Pkw - die Maschine ins Servicezentrum gebracht werden. Auf dieser Stufe hat man die besten Chancen zur Diagnose und Fehlerbehebung, aber auch die höchsten Kosten.

Aus diesen Benutzungskontexten ergeben sich unterschiedliche Anforderungen an die Funktionalität von MDF-Anwendungen, die sinnvollerweise durch verschiedene MDF-Anwendungen für dasselbe Produkt erfüllt werden. Um Kosten zu minimieren, sollten diese Anwendungen nicht unabhängig voneinander entwickelt, sondern möglichst viele Teile wiederverwendet werden. Daraus ergibt sich das in Abbildung 4-9 dargestellte Referenzmodell der MDF-Anwendungsentwicklung. Die beiden dort dargestellten Phasen der MDF-Funktionsentwicklung und der Generierung des Auslieferungssystems werden im folgenden näher diskutiert.

Entwicklung anwendungsspezifischer MDF-Funktionen

Zur Lösung von MDF-Aufgaben in einem bestimmten Benutzungskontext werden außer den Hauptfunktionen für Monitoring, Diagnose und Fehlerbehandlung zusätzliche Nebenfunktionen benötigt, z. B. zur Interaktion mit Benutzern oder anderen IT-Systemen, zur Archivierung der Ergebnisse oder zur Systempflege. Einen Teil dieser Funktionen wird schon das verwendete MDF-Werkzeug bereitstellen. Andere Funktionen wie z. B. Schnittstellentreiber können häufig aus früheren Entwicklungen übernommen werden.

Um die übrigen Funktionen zu realisieren, werden einerseits Wissen über die Maschine bzw. Anlage und die dort ablaufenden Prozesse, andererseits Problemlösungswissen benötigt. Zu letzterem gehören bei einem modellbasierten Ansatz neben MDF-Strategien auch Strategien zur Modellierung des Anwendungsbereichs. Einen Teil die-

19. ASIC: Application Specific Integrated Circuit

20. vgl. [Schahn 1994].

21. Im weiteren Sinne kann man hierzu auch den Fahrer eines Autos zählen.

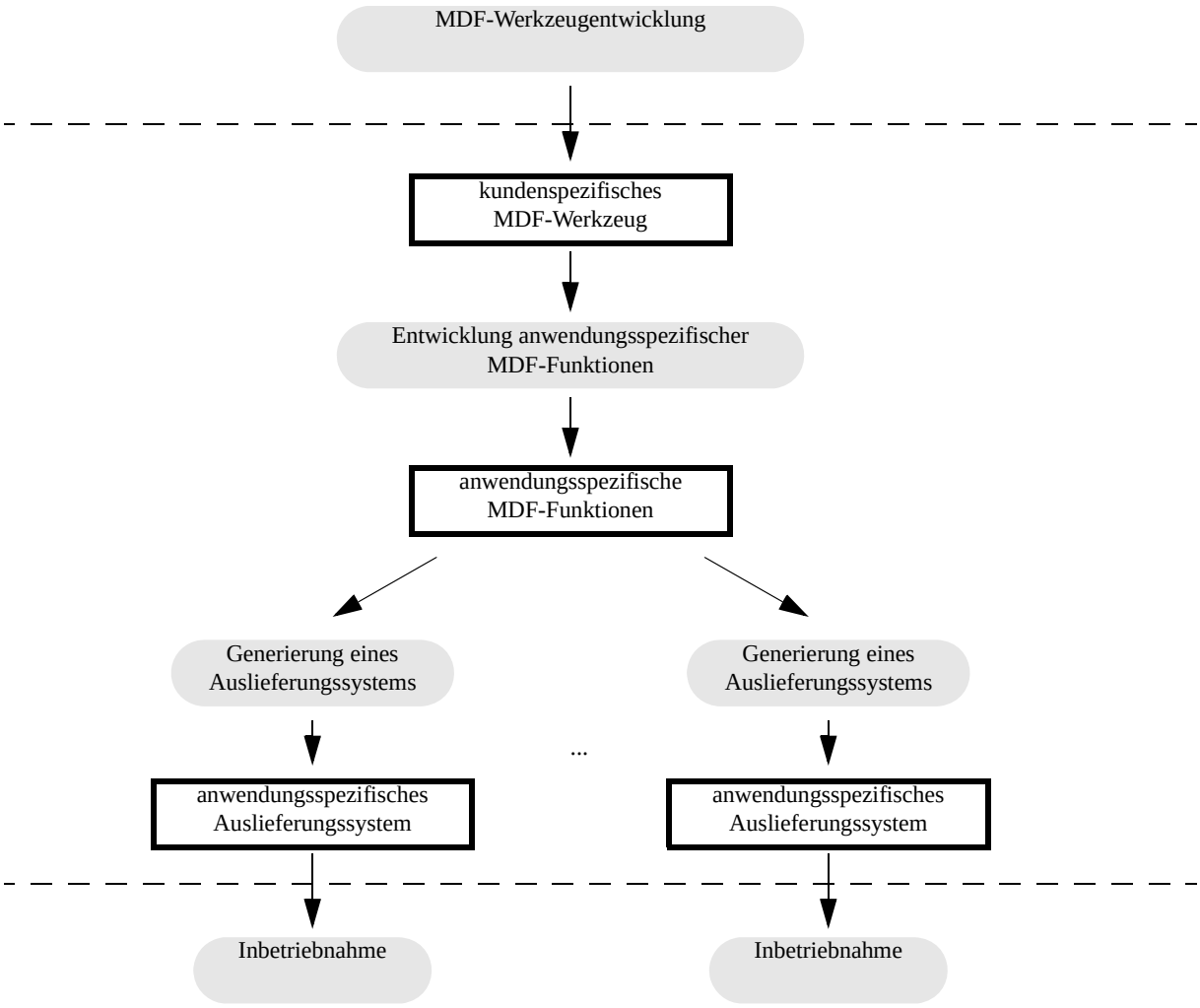


Abbildung 4-9: Referenzmodell der MDF-Anwendungsentwicklung

ses Wissens wird ebenfalls das MDF-Werkzeug bereitstellen, z. B. in Form domänenspezifischer Bibliotheken. Darüberhinaus enthält es Module, um die Erarbeitung und Formalisierung des übrigen Wissens zu unterstützen. Dabei verursachen die schon bei der Entwicklung von CHD-DIAS[22] festgestellten Probleme wesentliche Kosten.

Das erste Problem besteht darin, daß die von heutigen Werkzeugen gebotene Unterstützung in vielen Fällen noch nicht ausreicht, um den Aufwand für die Wissensakquisition auf ein für den MDF-Entwickler akzeptables Maß zu senken. Insbesondere benötigt man Konzepte aus der Informatik, der Künstlichen Intelligenz oder der Mathematik, um angebotenen Wissensrepräsentations- und Diagnosetechniken optimal nutzen zu können. Dieses Wissen ist heute in der Regel nur bei Experten mit Informatik-Hintergrund verfügbar. Andererseits ist das Anlagenwissen in der Regel nur bei den Domänenexperten des Herstellers vorhanden. Und domänenspezifisches Diagnosewissen, z. B. über spezielle Diagnoseverfahren wie Körperschallanalysen, besitzen heute nur relativ wenige Spezialisten aus dem Maschinen- und Anlagenbau. So müssen sich heute entweder die Domänenexperten entsprechendes Diagnostikwissen aneignen oder Diagnostikexperten müssen sich in die Domäne einarbeiten - beides teure Alternativen.

Ein weiteres Defizit heutiger Werkzeuge ist die fehlende Verbindung mit anderen Geschäftsprozessen. Weil z. B. geeignete Austauschformate fehlen, kann vorhandenes Wissen bei der Entwicklung nur in geringem Umfang automatisch importiert und das für MDF erfaßte Wissen nur schwer exportiert werden. Zum einen entstehen hier hohe Kosten, weil dieselben Informationen über Produkte oft mehrfach repräsentiert und mit entsprechend hohem

22. Vgl. Abschnitt 4.4

Aufwand konsistent gehalten werden müssen. Zum anderen können die Kosten zur Wissensakquisition nicht auf verschiedene Aufgaben, wie FMEA oder Generierung von Servicedokumenten, verteilt werden.[23]

Das zweite Problem bei der Entwicklung anwendungsspezifischer MDF-Funktionen ist die mangelhafte Integration der Entwicklung von Überwachungsfunktionen und der Entwicklung wissensbasierter Diagnosefunktionen. Während erstere im Rahmen der Produktentwicklung erfolgt, geschieht letztere meist in Verantwortung der Serviceabteilung für bestehende Produkte. Diese zeitliche und organisatorische Trennung hat erhebliche Nachteile. Erstens werden die Anforderungen der Diagnose bei der Auslegung der Sensorik oft nur unzureichend berücksichtigt. Zweitens wächst der Aufwand zur Entwicklung von Diagnosefunktionen, weil dazu Geschäftsprozesse in zwei Unternehmensbereichen koordiniert werden müssen. Erschwerend kommt häufig dazu, daß der Import von Daten aus den beiden Bereichen in das Diagnosesystem schwierig wird, weil beide unterschiedliche Informationssysteme oder -formate verwenden. Dieser Aufwand schwächt natürlich auch die Bereitschaft in einem Unternehmen, wissensbasierte Diagnosetechniken schrittweise in bestehende Arbeitsabläufe einzubringen.[24]

Generierung anwendungsspezifischer Auslieferungssysteme

Als *Auslieferungssystem* wird jene Form eines MDF-Systems bezeichnet, in der es in die Laufzeitumgebung gebracht wird. Die wissensbasierten Funktionen können darin z. B. in Form spezieller integrierter Schaltkreise (ASICs) vorliegen oder in Form einer oder mehrerer Dateien, die auf einem Speichermedium oder durch Herunterladen von einem Server ausgeliefert werden.

Bei der Generierung dieser Komponenten des Auslieferungssystems werden in der Regel zunächst letzte Optimierungsmaßnahmen durchgeführt, z. B. die Kompilation der Wissensbasis.[25] Anschließend werden zur Laufzeit zusätzlich benötigte Module (z. B. Schnittstellen oder Programmbibliotheken) eingebunden, während zur Laufzeit nicht benötigte Funktionen und Wissensfragmente der Entwicklungsumgebung sowie meist auch der Quellcode entfernt werden. Bei der Auslieferung als Hardware folgt der entsprechende Hardwareentwurf. Bei der Auslieferung als Software wird diese Konfigurationsaufgabe heute durch Standardwerkzeuge wie InstallShield oder InstallAnywhere unterstützt.

4.5.4 Inbetriebnahme

Bei der Inbetriebnahme oder Installation wird aus dem MDF-Auslieferungssystem das *MDF-Laufzeitsystem*, also jene Form, in der das MDF-System betrieben wird. In diesem Rahmen werden zunächst Parameterwerte bestimmt, die von der Laufzeitumgebung abhängen. In manchen Anwendungen läßt sich das System anschließend aufgrund der neu festgelegten Werte weiter optimieren.

Die Spezifikation der Parameterwerte kann in mehreren Schritten erfolgen. In der *Vorinstallation* von Onboard-Systemen für Maschinen werden beim Hersteller zunächst jene Parameterwerte bestimmt, die nur von der Maschine abhängen. In der *Endinstallation* beim Betreiber werden dann zusätzlich die vom Betriebsort abhängigen Parameter bestimmt.

Typische Parameterwerte eines Werkstattdiagnosesystems sind z. B. Namen und Adresse der Werkstatt oder die für Messungen und Tests verfügbaren Geräte. Häufige Parameter von Onboard-Überwachungssystemen sind das Schwingungsverhalten oder die Form von Spektrogrammen im fehlerfreien Anlagenbetrieb. Falls solche Werte benötigt werden, kann die Inbetriebnahme entsprechende Lernphasen enthalten. Weitere Beispiele sind Pfadnamen für benötigte Dateien oder Hilfsprogramme, soweit sie nicht Teil des Auslieferungssystems sind, gegebenenfalls auch das Verzeichnis im Dateibaum, in dem das MDF-System installiert werden soll. Parameter des in INDIA betrachteten Chemikaliendistributors sind z. B. die Anzahl der Behältergruppen oder die zum Spülen der Leitung benötigte Zeit.

4.5.5 Systempflege

Wie auch bei anderen Softwaresystemen ist die Entwicklung eines MDF-Systems mit der Auslieferung nur vorläufig abgeschlossen. Auch danach werden immer wieder Änderungen notwendig, z .B. an der anwendungsspezifischen Wissensbasis. Ursachen dafür sind insbesondere

23. Vgl. [Guckenbiehl et al. 1999].

24. Vgl. [Milde et al .2000].

25. Vgl. Abschnitt 4.8.1

- Verbesserungen in den verwendeten Werkzeugen, z. B. im Rahmen von Software-Updates;
- Verbesserungen des MDF-Systems selbst, z. B. durch bessere Strategien, mehr Erfahrungen aus dem Betrieb von Anlagen oder einer Erweiterung der abgedeckten Anlagenklasse;
- Änderungen in der Laufzeitumgebung, aufgrund deren sich die vom System benötigten Parameterwerte ändern.

Je stärker ein MDF-System auf eine bestimmte Anlage oder Anlagenklasse zugeschnitten ist, desto mehr Probleme verursacht gerade der letzte Punkt. Denn dann können auch kleine Änderungen an einer Anlage die Konsistenz mit der Wissensbasis verletzen und damit die Leistungsfähigkeit des MDF-Systems beeinträchtigen. Diese Konsistenz zu wahren ist besonders schwierig, wenn die Anlage nicht vom MDF-Systementwickler verändert wird. So kann z. B. ein Anlagenbetreiber in der Regel nicht überschauen, inwieweit sich seine Reparaturen oder Erweiterungen der Anlage auf MDF-relevante Parameter auswirken. Hier wären Ansätze für eine Verwaltung der Änderungen (engl. 'Change Management') sinnvoll.

4.6 Verwendung von Zustandsdiagrammen für die Diagnose

Motivation

Wesentliches Ziel von Fraunhofer IITB in INDIA war es, ausgehend von den Erfahrungen mit CHD-DIAS Ansätze zur Senkung der Einführungskosten wissensbasierter Diagnoseansätze zu entwickeln. Das im letzten Abschnitt vorgestellte Referenzmodell der MDF-Entwicklung steckt dafür den Rahmen ab. Dieser Abschnitt stellt einen Ansatz vor, mit dem die Kosten der Entwicklung von MDF-Funktionen reduziert werden sollen.

Kernidee dabei ist, daß die Domänenexperten des Maschinen- bzw. Anlagenherstellers die zur Diagnose benötigten Informationen weitgehend selbstständig erarbeiten und in das System einbringen sollen. Dies gilt insbesondere für die zur Diagnose benötigten Modelle. Zum einen sinken so der Bedarf an externen Wissensingenieuren und die damit verbundenen Kosten zur Einarbeitung in das Anwendungsfeld. Zum anderen können sich Entwicklungsingenieure stärker an der Entwicklung des Diagnosesystems beteiligen.

Voraussetzung dafür ist einerseits, daß die Domänenexperten in ihrer eigenen Begriffswelt modellieren können. Zum anderen müssen sie in die Lage versetzt werden, auch ohne das heute benötigte Detailwissen über den Diagnosealgorithmus und ohne entsprechende Modellierungserfahrung rasch geeignete Diagnosemodelle entwickeln zu können. Vorstellbar ist z. B., daß eine Entwicklungsumgebung für Diagnosesysteme Assistenzmodule („Wizards“) bereitstellt, die entweder die gesamte Modellierung steuern oder auf Wunsch nach Lücken im aktuellen Modellierungsstand suchen und diese durch gezielte Fragen an die Experten schließen.

Leider gibt es bisher keine domänenunabhängige Methodik zur Modellierung für Diagnose, aus der sich domänenspezifische Methodiken ableiten ließen. Wie in Abschnitt 4.4 geschildert entwickeln auch Diagnoseexperten Modelle häufig mittels Versuch und Irrtum neu. Nach Meinung von Fraunhofer IITB reichen die bisher gesammelten Erfahrungen auch noch nicht aus, um eine allgemeine Methodik abzuleiten.

In INDIA wurde daher ein Bottom-Up-Ansatz gewählt: ausgehend von CHD-DIAS wurde versucht, die dort verwendeten domänenspezifischen Modellierungs- und Diagnosestrategien zu identifizieren und anschließend soweit wie möglich zu verallgemeinern. Eine solche domänenunabhängige Methodik sollte jedoch durch domänen- oder branchenabhängige Bibliotheken ergänzt werden, in denen Modelle von Standardkomponenten oder domänentypische Diagnosestrategien zur Verfügung gestellt werden.

Modellierungsmethodik bei CHD-DIAS

Am Anfang der Entwicklung von CHD-DIAS wurden die vorhandenen Unterlagen gesichtet und die Experten bei THEN befragt. Beide Quellen beschrieben das Verhalten der Aktorkomponenten (Ventile, Pumpe, Klappe usw.) im wesentlichen als Zustandswechsel aufgrund von Steuerereignissen. Die Steuerung selbst reagiert auf Timeouts, auf Signale des Durchflußmessers und auf externe Steuerereignisse.

Diese Sicht spiegelt letztlich auch das Zustandsdiagramm von Abbildung 4-6 wider. Insbesondere stehen dabei die Zwischenzustände *opening* und *closing* für Phasen, in denen sich aufgrund kontinuierliche Prozesse der Betriebszustand des Ventils qualitativ ändert. Die Transitionen in diese Zwischenzustände bzw. aus ihnen heraus kennzeichnen deshalb keine unendlich schnelle Änderung der Realität. Sie legen vielmehr mit gewissen Sicherheitsmargen Zeitpunkte fest, an denen diese Veränderungen sicher noch nicht begonnen haben bzw. sicher been-

det sind. Die Zwischenzustände verbergen also sowohl den genauen Verlauf als auch die tatsächliche Dauer der kontinuierlichen Prozesse.

Parallelen zum Diagnosesystem für eine Montagestation

Hier zeigten sich Parallelen zu einer Entwicklung, die das IITB in der Vergangenheit in einer ganz anderen Domäne durchgeführt hatte: dem System PAS-DIAS zur Diagnose einer Station zur Montage von Robotergreifern [Zimmermann-Sturm 1996]. Auch dort beschrieben die Experten das Verhalten mit Hilfe von Zustandsdiagrammen. Dabei war nur der Ausgangszustand stabil. Alle anderen Zustände repräsentierten jeweils die Ausführung eines bestimmten Montageschritts und damit Übergangszustände. Sensoren signalisierten das Ende eines solchen Schritts und veranlaßten den Beginn des nächsten Schritts. Dies entsprach genau der Transition zwischen zwei Zuständen. Für jeden Montageschritt konnten die Experten eine maximale Dauer angeben. Fehler zeigte sich darin, daß diese Zeit ablief ohne daß das Ende des Schritts signalisiert worden war. Dies wurde in die Diagnosemodelle übernommen.

Gemeinsamkeit: Reaktive Systeme

Diese Gemeinsamkeiten von CHD-DIAS und PAS-DIAS sind nicht zufällig. Sowohl der Chemikaliendistributor und die Montagestation als auch viele ihrer Komponenten sind Beispiele einer Kategorie von Systemen, die nach [Harel, Pnueli 1985] als *reaktive Systeme* bezeichnet werden. Charakteristische Merkmale solcher Systeme sind:[26]

- Sie interagieren ständig mit ihrer Umgebung, wobei Ein- und Ausgabegrößen sowohl zeitkontinuierlich als auch zeitdiskret sein können. Viele dieser Größen können ihre Werte asynchron verändern, also zu nicht vorhersehbaren Zeiten.
- Sie müssen auf Unterbrechungen durch Ereignisse mit hoher Priorität reagieren können, selbst wenn sie gerade mit etwas anderem beschäftigt sind.
- Ihr Betrieb und ihre Reaktion auf Eingaben unterliegt häufig strikten Zeitanforderungen.
- Es gibt verschiedene Betriebsszenarien, die vom aktuellen Betriebszustand, von den aktuellen Ein- und Ausgaben und von der Vergangenheit abhängen können.
- Sie sind häufig aus interagierenden Prozessen aufgebaut, die parallel ablaufen.

Damit unterscheiden sie sich von sogenannten *transformierenden Systemen*, deren Verhalten in der Regel als Schleife mit folgenden Stationen beschrieben wird: Warten auf alle Eingaben, Berechnung der Ausgaben aus Eingaben und aktuellem Zustand, Ausgabe der Ergebnisse. Dieser Typ von Systemen wird gerade in der Regelungstechnik häufig verwendet.

Zur Verhaltensmodellierung reaktiver Systeme finden *Statecharts* wachsende Verbreitung.[27] Dies wird auch durch die Aufnahme von Statecharts in UML [UML-RTF 1999] demonstriert und weiter gefördert. Statecharts wurden 1984 von Harel als graphische Beschreibungssprache für Zustandsdiagramme vorgeschlagen. Sie erweitern herkömmliche Mealy-Moore-Zustandsdiagramme insbesondere um Ausdrucksmittel für:[28]

- Systemattribute, die ihren Wert innerhalb eines bestimmten Rahmens ändern können ohne daß sich der Zustand des Systems ändert;
- Bedingungen für Zustandsübergänge;
- Aktionen, die bei jedem Ein- bzw. Austritt in den Zustand ausgeführt werden;
- Aktionen, mit denen auf ein Ereignis reagiert werden soll ohne den aktuellen Zustand zu verlassen und dabei Ein-/Austrittsaktionen durchzuführen;
- Aktivitäten, die ausgeführt werden, solange sich das System in einem bestimmten Zustand befindet;

26. Nach [Harel, Politi 1997], S. 3f.

27. Vgl. z. B. [Douglass 1999].

28. Vgl. [Harel, Politi 1997]

- Timeout-Ereignisse, die den Ablauf einer spezifizierten Frist nach einem bestimmten Vorgängerereignis signalisieren, z. B. dem letzten Eintritt in den aktuellen oder einen anderen Zustand.
- komplexe Zustände, die sich aus alternativen oder unabhängigen Teilzuständen zusammensetzen und damit eine Menge qualitativ ähnlicher Zustände repräsentieren;
- Pseudozustände zur Kennzeichnung von Verbindungsstellen, Entscheidungsknoten u .ä.

Diagnosemodelle für Reaktive Systeme

Im Rahmen von INDIA erarbeitete Fraunhofer IITB Heuristiken, um als Statecharts repräsentierte Verhaltensmodelle reaktiver Systeme im Hinblick auf die Diagnose zu ergänzen. Diese Heuristiken sollen die Grundlage bilden für Werkzeuge, die Domänenexperten bei der Entwicklung von Diagnosemodellen anleiten.

- Für jeden Zustand spezifiziere *Invarianten*, d. h. Systemmerkmale, die sich während des gesamten Zustands nicht ändern. Im Zustandsdiagramm für CHD-DIAS (Abbildung 4-6) ist z. B. „*flow-enabled = false*“ eine Invariante des Zustands *closed*.
- Identifiziere Zustände, in deren Verlauf sich der Betriebsmodus der Maschine/Anlage oder bearbeitete Werkstücke qualitativ ändern. Zu diesen *Übergangszuständen* gehören in Abbildung 4-6 *opening* und *closing*.
- Für jeden Übergangszustand identifiziere *Nachbedingungen*, d. h. hinreichende Bedingungen für den erfolgreichen Abschluß eines Übergangs.
- Für jeden Übergangszustand identifiziere *Vorbedingungen*, d. h. Bedingungen, die beim Eintritt in den Übergangszustand notwendig erfüllt sein müssen, um den Übergang erfolgreich abzuschließen. Dies entspricht z. B. einem anfänglichen Qualitätstest auf den in einem solchen Zustand bearbeiteten Werkstücken oder Betriebsmitteln.
- Falls möglich spezifiziere auch hinreichende Vorbedingungen für den erfolgreichen Abschluß eines Übergangs. Dies sind Merkmale, aufgrund derer schon zu Beginn des Zustands der erfolgreiche Abschluß feststeht - sofern das System korrekt funktioniert. Wenn während des Schritts eine Störung auftritt obwohl diese hinreichenden Bedingungen erfüllt sind, dann muß die Ursache dafür im Schritt selbst liegen.
- Für jeden Übergangszustand suche nach unteren und oberen Schranken für die Zeitdauer.
- Suche nach weiteren *Betriebsbedingungen* des Systems. Dazu gehört z. B. auch, in welchen Zuständen und wie oft externe Ereignisse erwartet werden.

Alle spezifizierten Kriterien müssen anhand der anhand der verfügbaren Betriebsdaten überprüfbar sein. Da deren Menge häufig schon vor Beginn der Diagnoseentwicklung festgelegt wurde, lassen sich Fragen nach der Detaillierung von Vorbedingungen, Invarianten und Auswirkungen[29] relativ leicht beantworten. Beim Chemikaliendistributor z. B. fanden sich aufgrund der beschränkten Menge verfügbarer Betriebsdaten keine überprüfbaren Vor- oder Betriebsbedingungen.

In manchen Methodiken zur Entwicklung technischer Systeme werden Informationen, die hier als diagnoserelevant genannt wurden, schon standardmäßig erhoben. Dies gilt insbesondere für Anforderungen, wie Vor- und Nachbedingungen einer Operation oder Zeitbedingungen. Bei der Verwendung für die Diagnose kommt es darauf an, diese Kriterien so zu formulieren, daß sie anhand der verfügbaren Betriebsdaten überprüft werden können.

Vor- und Nachbedingungen eines Übergangszustands werden teilweise mit den Invarianten in vorangehenden bzw. nachfolgenden stabilen Zuständen übereinstimmen. So ist z. B. die Invariante „*flow-enabled = false*“ des Zustands *closed* gleichzeitig Nachbedingung für den Übergangszustand *closing*. Eine Modellierungsumgebung kann hier Werkzeuge zur Modelloptimierung bereitstellen.

Erweiterung der Methodik auf Flußdiagramme

Die genannten Heuristiken zur Entwicklung von Diagnosemodellen setzen voraus, daß das Systemverhalten bereits in Form von Zustandsdiagrammen beschrieben wurde. Zustandsdiagramme beschreiben den Kontrollfluß.

---

29. Diese Fragen entsprechen den in der Künstlichen Intelligenz berüchtigten Qualification, Frame- und Ramification-Problemen (vgl. z. B. [Davis, 1990].

Orthogonal dazu beschreiben *Flußdiagramme* häufig den Material-, Energie und / oder Informationsfluß zwischen den Schritten eines Verfahrens, den Phasen eines technischen Prozesses oder den Komponenten einer Anlage. Ein Flußdiagramm ist ein Graph, in dem Teilfunktionen, Teilprozesse, Teilanlagen oder Maschinenteile durch Knoten repräsentiert werden. Sie werden im folgenden *aktive Knoten* genannt. Die Energie-, Materie und Informationsströme zwischen aktiven Knoten werden in der Regel durch Kanten repräsentiert. In sogenannten *Produktnetzen* liegt auf jeder dieser Kanten ein Stromknoten, an dem die Art und die Merkmale des Stroms festgehalten werden. In diesem Fall entsteht ein bipartiter Graph.

Flußdiagramme werden in vielen Disziplinen von den jeweiligen Experten als Beschreibungsmittel verwendet, z. B. Fließbilder in der Verfahrenstechnik [DIN-28004, 1988], Funktionsstrukturen im Maschinenbau [Pahl, Beitz 1997], Funktionsgraphen in der Automatisierungstechnik [Ahrens et al., 1997] oder Phasenmodelle zur Beschreibung von Produktionsprozessen generell [Buchner et al., 1994]. Daher stellt sich die Frage, inwieweit die für Zustandsdiagramme entwickelten Heuristiken übertragen werden können, um Domänenexperten bei der Ableitung von Diagnosemodellen aus Flußdiagrammen zu unterstützen.

Um Fließbilder im Hinblick auf die Diagnose zu ergänzen, sucht man nach Merkmalen für die Qualität der Ströme und die Qualität des Betriebs aktiver Knoten. In der Tat kann man bei dieser Suche viele der oben genannten Heuristiken mit wenigen Änderungen übernehmen.

- Für jeden aktiven Knoten spezifiziere *Invarianten* in Form von Qualitätsmerkmalen der Eingangsströme, die sich auch an den Ausgangsströmen wiederfinden.
- Für jeden aktiven Knoten identifiziere *Nachbedingungen* in Form von Anforderungen an die Qualität der Ausgangsströme.
- Für jeden aktiven Knoten identifiziere *Vorbedingungen* in Form von Anforderungen an die Qualität der Eingangsströme, die notwendig erfüllt sein müssen, um Ausgangsströme in der geforderten Qualität zu liefern.
- Falls möglich spezifiziere Vorbedingungen, die hinreichend für Ausgaben in der geforderten Qualität sind - sofern das System korrekt funktioniert. Wenn die Ausgangsströme ihre Qualität nicht erreichen obwohl diese hinreichenden Bedingungen erfüllt sind, dann muß die Ursache dafür im aktiven Knoten selbst liegen.
- Für jeden aktiven Knoten suche nach unteren und oberen Schranken für die Zeit die benötigt wird, bis sich Änderungen der Eingangsströme in den Ausgangsströmen wiederfinden.
- Für jeden aktiven Knoten suche nach weiteren *Betriebsbedingungen*, d. h. Anforderungen an seinen Betrieb, die erfüllt sein müssen, um die Ausgangströme in der erforderlichen Qualität zu liefern.

Auch hier gilt, daß die spezifizierten Anforderungen anhand der verfügbaren Daten überprüfbar sein müssen.

Gerade bei aktiven Knoten, die komplexe Einheiten repräsentieren, können sich komplexe Zusammenhänge zwischen Eingangsströmen, Betriebsbedingungen und Ausgangsströmen ergeben. Hier kann es sogar sinnvoll sein, diese Zusammenhänge wieder durch ein Zustandsdiagramm zu beschreiben.

Ein Vorteil dieser Diagramme ist, daß die aktiven Knoten eine Modularisierung des technischen Prozesses bzw. der Anlage vorgeben, die im Rahmen einer konsistenzbasierten Diagnose für eine erste zeitliche und/oder räumliche Eingrenzung des Fehlers benutzt werden kann. Dazu rechtfertigt man z. B. jede Inferenz über die Qualität eines Ausgangsstroms mit der Annahme, daß der liefernde aktive Knoten korrekt arbeitet und die benötigten Eingangsströme die geforderte Qualität besitzen. Daß ein Eingangsstrom die geforderte Qualität besitzt, wird gerechtfertigt mit der Qualität des „gleichnamigen“ Ausgangsstroms und mit der Korrektheit des Flußdiagramms. Auf diese Weise lassen sich bei Qualitätsmängeln Mengen von aktiven Knoten ermitteln, deren Fehlfunktion für alle Mängel verantwortlich sein können. Für eine genauere Diagnose werden dann Modelle der Implementierung eines verdächtigten Knotens benötigt. Die Standardresultate zur Vollständigkeit und Korrektheit konsistenzbasierter Diagnose gelten weiterhin.

Baut man Diagnosemodelle auf Flußdiagrammen auf, dann bezahlt man den geringeren Modellierungsaufwand allerdings häufig mit einer geringeren Diagnosefähigkeit. Denn einerseits werden Störungen in den Eingangsbedingungen eines aktiven Knotens Fehlern in den stromaufwärts liegenden Knoten zugeschrieben. Andererseits enthalten Flußdiagramme im allgemeinen nur die intendierten Ströme aber keine ungewollten Wärmeströmen, Schwingungen o. ä.. Aufgrund solcher Interaktionen können aber die Eingangsbedingungen eines aktiven Knotens auch durch stromabwärts liegende Knoten beeinflußt werden. Falls sich die Diagnosefähigkeit der aus Flußdiagrammen benötigten Modelle als nicht ausreichend erweist, müssen sie durch feinere Modelle der möglichen Interaktionsmöglichkeiten ergänzt werden.

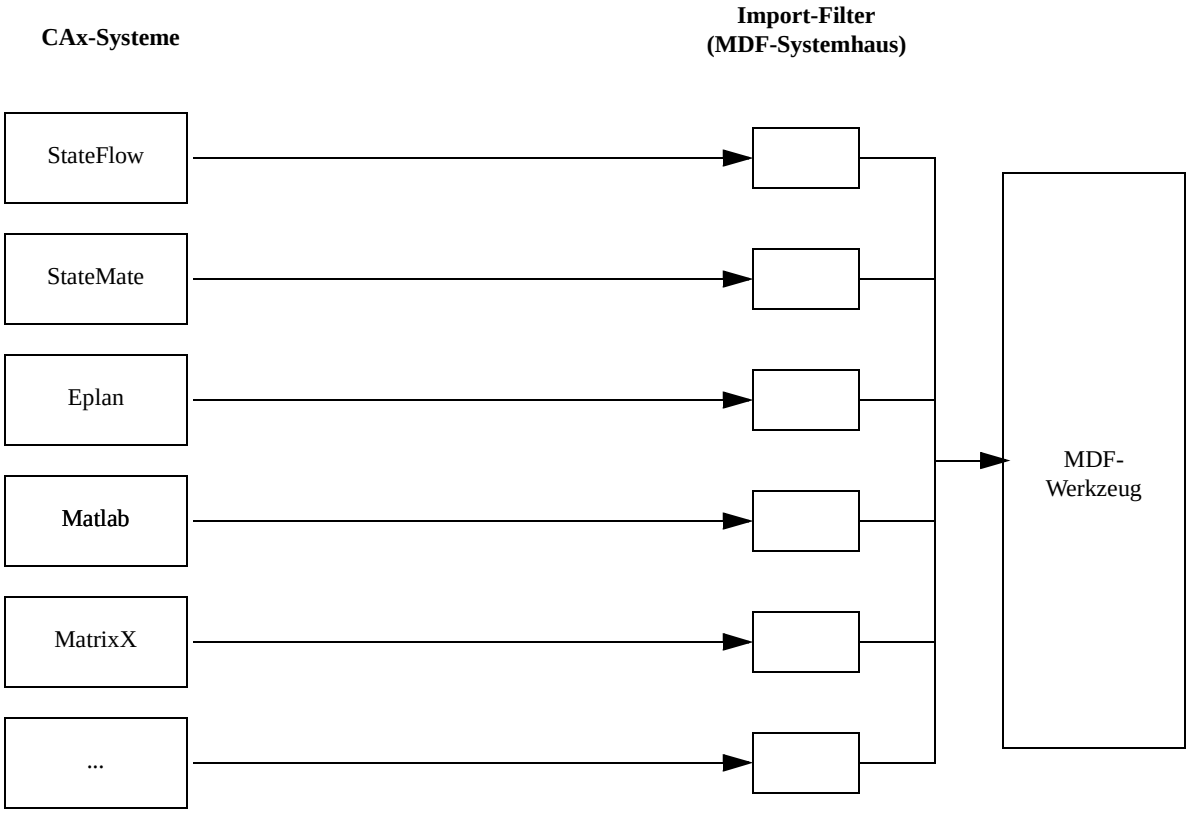


Abbildung 4-10: Import von Modellen aus CAx-Systemen in Diagnosewerkzeuge: Aktuelles Vorgehen

4.7 Import von Verhaltensmodellen in Diagnosesysteme

4.7.1 Überblick

Bei der Suche nach Ansätzen, um den Aufwand für die Entwicklung von MDF-Anwendungen zu reduzieren, spielen Importfilter eine wesentliche Rolle. Dabei handelt es sich um Module des MDF-Werkzeugs, mit denen Informationen über ein Produkt oder eine Produktreihe aus anderen Informationssystemen in das MDF-Werkzeug importiert werden können. Zu diesen Informationen gehören insbesondere Modelle von Struktur, Funktion und Verhalten, die z .B. mit herkömmlichen CAx-Systemen erstellt wurden.

Dieser Import hat mehrere Vorteile. Erstens müssen bereits vorhandene Modelle nicht erneut formuliert werden. Zweitens können Fachexperten Diagnosemodelle zumindest teilweise mit den ihnen vertrauten CAx-Werkzeuge formulieren. Drittens läßt sich die Konsistenz zwischen Diagnoseinformationen und den übrigen Produktinformationen wesentlich einfacher wahren.

Schon heute bieten MDF-Werkzeuge die Möglichkeit, mit gebräuchlichen CAx-Systemen erstellte Modelle in das MDF-System zu importieren. In der Regel wird dazu jedoch vom MDF-Systemhaus für jedes CAx-System ein eigener Import-Filter implementiert (Abbildung 4-10) - ein teures Vorgehen. Fraunhofer IITB suchte daher im Rahmen von INDIA nach alternativen Ansätzen, wobei der Import von Funktions- und Verhaltensmodellen im Vordergrund stand. Geplant war dabei, auf einem weitverbreiteten Format zum Produktdatenaustausch aufzusetzen und dieses gegebenenfalls im Hinblick auf den Austausch von Funktions- und Verhaltensmodellen zu erweitern. Im Laufe der Arbeit erschien es jedoch sinnvoller, auf verschiedenen Sprachen zum Austausch von Verhaltensmodellen aufzusetzen, die jeweils einem Modellierungsparadigma folgen.

Um diesen neuen Ansatz zu demonstrieren, wurde mit AML eine Sprache zum Austausch von erweiterten Zustandsdiagrammen entworfen, in der das Black-Box-Verhalten der verschiedenen Komponenten des Chemika-

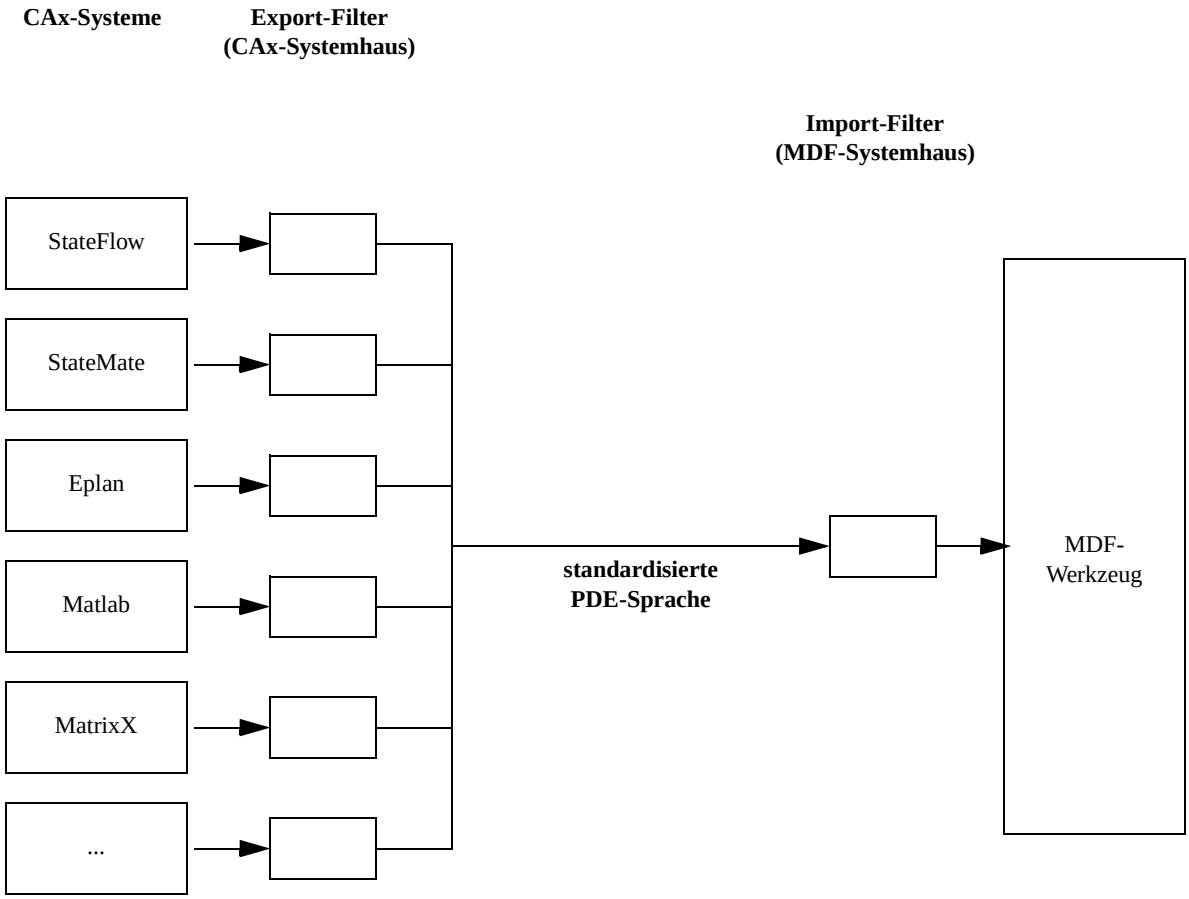


Abbildung 4-11: Import von Modellen aus CAx-Systemen in Diagnosewerkzeuge: Verwendung standardisierter Produktdatenaustauschsprachen

liendistributors modelliert wurde. Mit Hilfe des Programms SCI können diese AML-Modelle dann im Hinblick auf Diagnose ergänzt und in die Modellierungssprache DEEP unseres Diagnosewerkzeugs übersetzt werden.

Im folgenden werden zunächst der Wechsel des Ansatzes begründet, dann AML und SCI vorgestellt.

4.7.2 Konzeptentwicklung

Ziel unserer Arbeit war eine Analyse, inwieweit gebräuchliche Formate zum Produktdatenaustausch zum Austausch von Funktions- und Verhaltensmodellen ausreichen und wie sie gegebenenfalls ergänzt werden können. Gedacht war dabei an formale Standards, Industriestandards oder zumindest De-Facto-Standards wie STEP, VHDL oder KIF. Ein MDF-Systemhaus könnte sich dann darauf beschränken, Importfilter für Modelle in dieser Sprache anzubieten und den Export aus einem herkömmlichen CAx-System in diese Sprache dem betreffenden Hersteller überlassen (vgl. Abbildung 4-11)

STEP

In der Diskussion mit THEN zeigte sich, daß im Anlagenbau kein Format als Standardaustauschformat in diesem Sinn angesehen werden kann. Auch die bei THEN verwendeten CAx-Systeme lieferten keine Hinweise auf ein solches Format. Dagegen schien STEP um 1996 ein Standard zu sein, in dessen Rahmen sich ein solches Format würde entwickeln können. Unsere Arbeiten orientierten sich daher zunächst in diese Richtung [Brinkop,1997].

STEP (Standard for the Exchange of Product data) ist ein formaler Standard für den Austausch von produktbezogenen Daten während des gesamten Produktlebenszyklus [ISO 1993]. Dazu werden neben Hilfsmitteln zum Konformanztest insbesondere Terminologien für bestimmte Anwendungsbereiche sowie entsprechende Beschreibungsmittel spezifiziert.

Die Terminologie für einen Anwendungsbereich wird im Rahmen eines Anwendungsprotokolls (engl. Application Protocol) festgelegt. Zur Zeit sind über 30 AP vorgesehen, die teilweise bereits fertiggestellt sind. Begriffe, die in verschiedenen AP vorkommen, sollen dabei im Rahmen spezieller „Integration Information Resources“ spezifiziert werden.

STEP enthält neben den textuellen Beschreibungssprachen EXPRESS und EXPRESS-I auch die bildliche Beschreibungssprache EXPRESS-G. Mit EXPRESS-I können einzelne Objekte beschrieben werden. In EXPRESS und EXPRESS-G werden Hierarchien von Objekttypen beschrieben, wobei jeder Typ eine Menge von Attributen spezifiziert. Die Werte verschiedener Attribute eines Typs können durch komplexe statische Constraints eingeschränkt werden. Dynamische Abhängigkeiten oder Wertänderungen lassen sich aber nicht direkt beschreiben.

Aufgrund dieser eingeschränkten Ausdruckskraft eignet sich EXPRESS primär zur Beschreibung von Struktur und statischem Verhalten. Um dynamisches Verhalten zu beschreiben, müssen Tricks angewendet werden, z. B. indem man die Historie eines Attributwerts als einzelnen Wert auffaßt. Eine Standardmethode gibt es dabei nicht. Zwar wurden verschiedene Erweiterungen von EXPRESS zur Repräsentation dynamischer Abhängigkeiten vorgeschlagen, doch auch hier hat sich bisher kein Vorschlag durchgesetzt.

Als Grundlage eines Austauschformats für Verhaltensmodelle ist EXPRESS nach Einschätzung von Fraunhofer IITB auf absehbare Zeit noch nicht geeignet. Die wesentlichen Gründe dafür sind die fehlende Standardisierung bei der Repräsentation von Verhalten und das frühe Stadium bei der Spezifikation von Anwendungsprotokollen und „Integration Information Resources“.

Alternative: Import über Standard-Modellierungsparadigmen

Da auf absehbare Zeit kein Standardformat für den Austausch von Verhaltensmodellen erkennbar ist, entwickelten wir einen alternativen Ansatz (Abbildung 4-12). Er beruht auf der Beobachtung, daß es zwar viele Modellierungswerkzeuge, aber nur relativ wenige Modellierungsparadigmen gibt. Dazu gehören insbesondere Netzlisten zur Strukturmodellierung, Blockdiagramme für zeitkontinuierliche Modelle und erweiterte Zustandsdiagramme für zeitdiskrete Modelle. Statt daher für jedes CAx-Werkzeug einen kompletten Importfilter zu realisieren, sollte ein MDF-Systemhaus für jedes relevante Modellierungsparadigma

- eine Austauschsprache wählen oder entwerfen,
- Filter realisieren, um Modelle in dieser Sprache in das MDF-Werkzeug zu importieren,
- für jedes relevante CAx-Werkzeug, das auf dem Modellierungsparadigma beruht, einen Filter realisieren, der Modelle aus dem proprietären Format des Werkzeugs in die Austauschsprache übersetzt.

Durch die semantische Nähe zwischen dem CAx-Werkzeug und der entsprechenden Austauschsprache wird erwartet, daß der Aufwand für letztere relativ niedrig ist und sich dadurch gegenüber dem heutigen Ansatz erhebliche Kostenersparnisse ergeben.

Demonstrator

In Abschnitt 4.3 wurde deutlich, wie gut sich Zustandsdiagramme zur Modellierung des Black-Box-Verhaltens der CHD-Komponenten eignen. Dieses Modellierungsparadigma wurde daher zur Demonstration des geschilderten Ansatzes verwendet (Abbildung 4-13). Da es für solche Diagramme noch keine standardisierte Austauschsprache[30] gibt, wurde mit AML eine neue Sprache entworfen. Mit Hilfe des neu entwickelten Filtermoduls SCI können in dieser Sprache formulierte Modelle in das Diagnosesystem MuDia von Fraunhofer IITB importiert werden. Schließlich wurde ein Modul realisiert, das mit dem kommerziellen Werkzeug StateMate spezifizierte Modelle nach AML übersetzt. Allerdings besitzt AML zur Zeit noch nicht die Ausdruckskraft von StateMate.

AML und SCI werden in den beiden folgenden Abschnitten näher dargestellt.

30. UML [UML-RTF 1999] spezifiziert u. a. ein XML-Format für Statecharts, das sich längerfristig zu einem Quasi-Standard entwickeln könnte. Zur Zeit wird es unseres Wissens aber noch von keinem Werkzeug unterstützt.

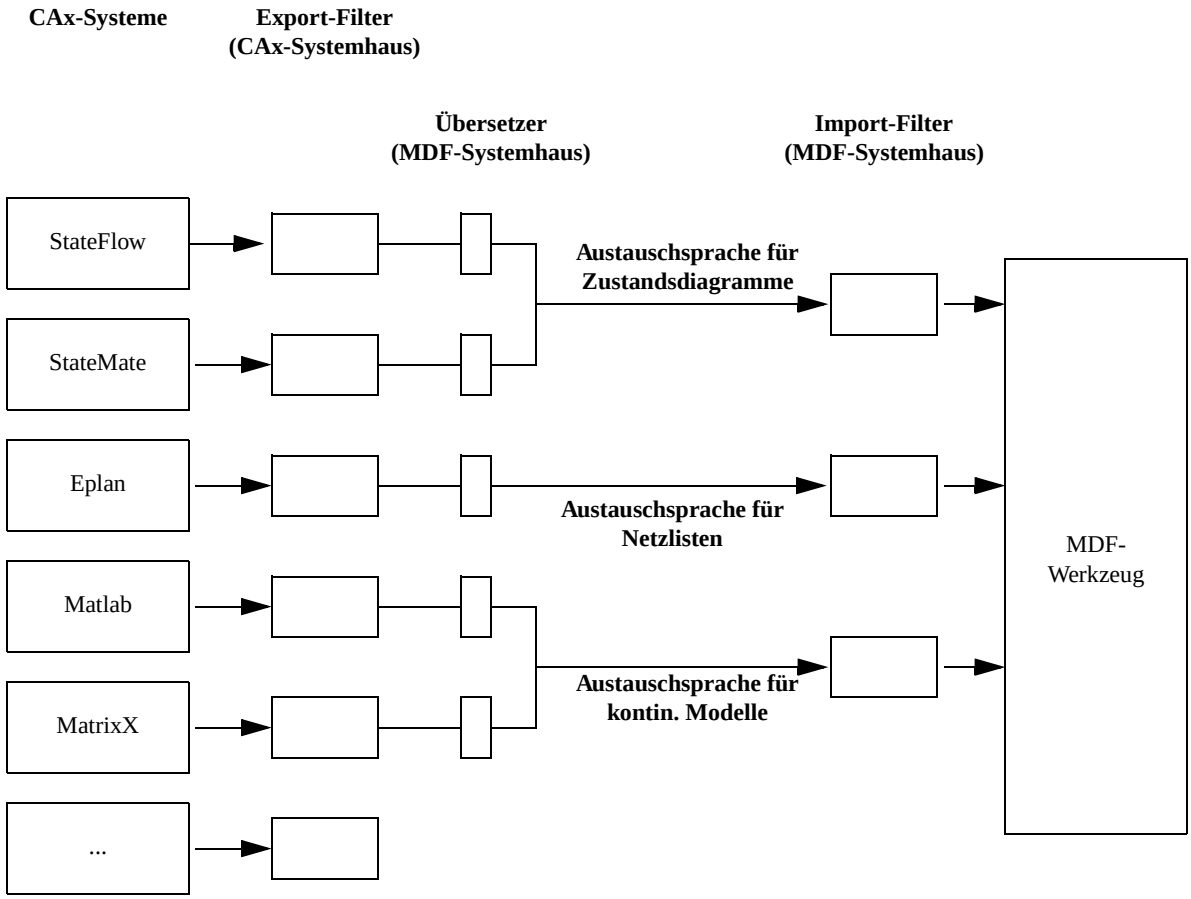


Abbildung 4-12: Import von Modellen aus CAx-Systemen in Diagnosewerkzeuge: Verwendung von Austauschsprachen für verschiedene Modellierungsparadigmen

4.7.3 AML - ASCII Modelling Language

AML (ASCII Modelling Language) ist eine Sprache zur textuellen Beschreibung erweiterter Zustandsdiagramme. In Ausdruckskraft und Semantik orientiert sie sich an der in [Harel, Politi 1997] beschriebenen Variante von Statecharts, die auch in StateMate verwendet wird. Einige Ausdrucksmittel, insbesondere komplexe Zustände und Pseudozustände, wurden dabei bisher nicht berücksichtigt. Darüberhinaus sind als Aktionen nur Ereignisse und Zuweisungen an Systemattribute zugelassen. Aktivitäten schließlich müssen durch Aussagen über Attributwerte repräsentiert werden, die auch als Lisp-Ausdrücke formuliert sein können.

Als Beispiel zeigt Abbildung 4-14 die AML-Beschreibung des in Abbildung 4-6 dargestellten Zustandsdiagramms, welches das Black-Box-Verhalten eines Ventils modelliert. Eine detaillierte Darstellung sowohl von AML als auch der Modellierung des Chemikaliendistributors findet sich in [Brinkop, 1999].

4.7.4 SCI - Import von AML-Modellen in den Diagnoseformalismus

SCI[31] ist ein Lisp Programm, das in AML repräsentierte Zustandsdiagramme in die Repräsentationssprache DEEP des Diagnosesystems MuDia übersetzt. Diese Übersetzung ist zugeschnitten auf die Anforderungen der modellbasierten Diagnose. Ziel sind Aussagen über Attributwerte, die anhand von Beobachtungen überprüft werden können. Dabei dürfen außer der korrekten Arbeitsweise des Strukturelements und der Korrektheit der beobachteten Werte keine weiteren Annahmen gemacht werden. Damit scheidet insbesondere die in herkömmlichen ereignisorientierten Simulationen übliche Annahme aus, daß sich Attributwerte nicht verändern, wenn nichts

31. Abk. für *S*tatechart *I*ntegrator, vgl. [Brinkop, 1999].

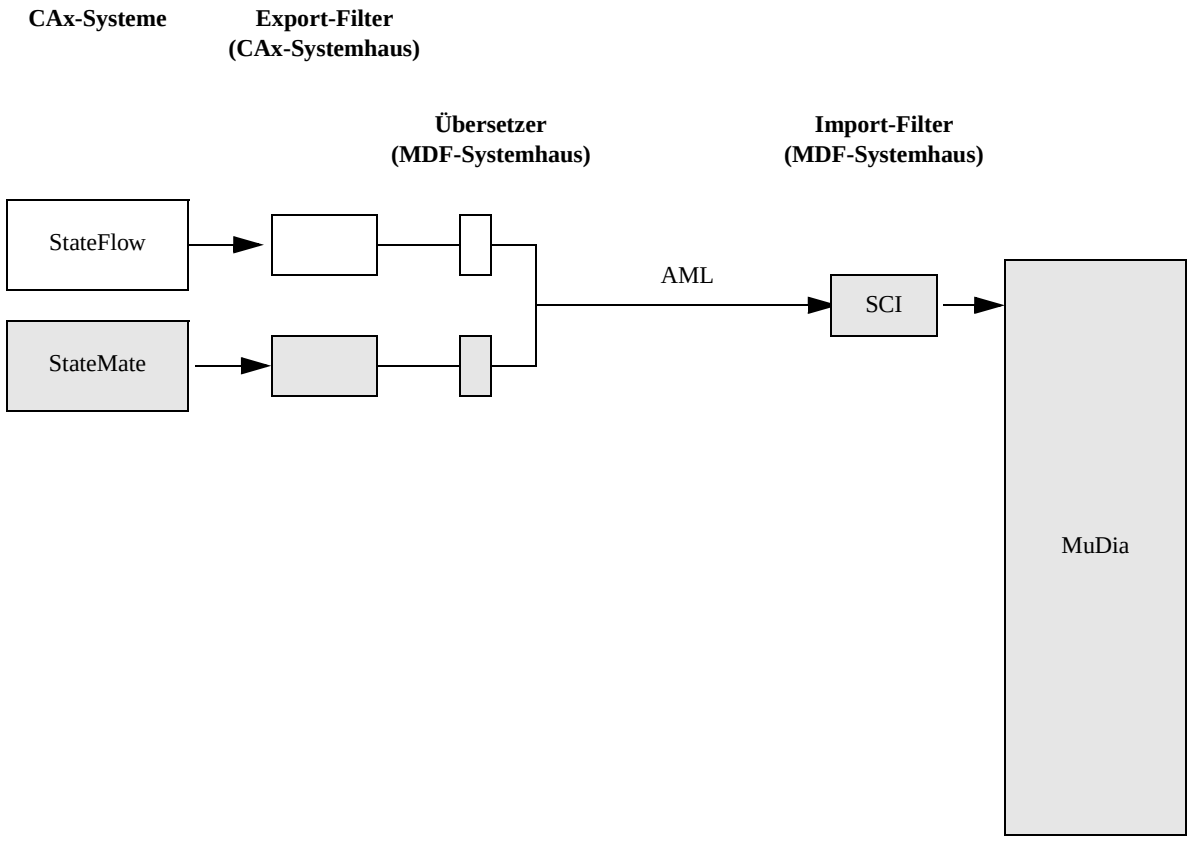


Abbildung 4-13: Demonstrator mit AML und SCI

anderes hergeleitet werden kann („Persistence by default“). Stattdessen muß nicht nur die Änderung, sondern auch die Persistenz eines Attributwerts explizit hergeleitet werden.

Zielsprache der Übersetzung ist DEEP. Diese Repräsentationssprache wird durch die Kombination der Modellierungswerkzeuge DAM und EEP definiert, die beide bei Fraunhofer IITB entwickelt wurden. DasVerhalten dynamischer Systeme wird in DEEP im Prinzip durch Regeln der folgenden Form beschrieben:

```
(rule-xy
   (  :IF     P [ att1(f1 (t)) , ... , attn(fn (t)) ]
      :THEN   Q [ att'1(f'1 (t)) , ... , att'm(f'm (t))])
```

Dabei stehen *t* für eine Zeitpunktvariable, f_i und f'_j für Funktionen, die Zeitpunkte auf Zeitintervalle abbilden, att_i und att'_j für im Modell definierte Attribute. *P[ ]* und *Q[ ]* stehen für Konjunktionen atomarer Aussagen auf den spezifizierten Termen. Typische Beispiele für atomaren Aussagen sind

```
(:TRUE-IN  (  [  (?t - 1)   (?t + 2)   [  )       (flow-enabled = nil))
(:TRUE-IN              ?t                         (doOpen = :occuring))
(:NIL      T)
(:NON-NIL  sexpr [ att1(f1 (t)) , ... , attn(fn (t)) ])
```

Die erste repräsentiert, daß `flow-enabled` im gesamten Intervall `[(?t-1) , ?t+2]` den Wert `nil` besitzt. Die zweite repräsentiert, daß `doOpen` im Punktintervall `[?t , ?t]` den Wert `:occuring` hat. Die dritte drückt aus, daß das Ergebnis des Lisp-Ausdrucks '`T`' den Wert `nil` hat. Sie wird verwendet, um Inkonsistenzen zu repräsentieren. Die letzte schließlich versichert, daß der Wert des Lisp-Ausdrucks *sexpr* von NIL verschieden ist. *sexpr* kann sich dabei auf Attributwerte der Form *a(f(t))* beziehen.

Eine erste Version von SCI wurde in [Brinkop, 1999] beschrieben. Sie besaß jedoch eine Reihe von Schwächen und wurde daher im letzten Abschnitt von INDIA überarbeitet. Im folgenden werden die Grundprinzipien der neuen Version skizziert. Sie erzeugt für ein Zustandsdiagramm fünf Arten von Regeln:

- *Transitionsregeln*[32] beschreiben, wann ein Zustandswechsel stattfindet und welche Aktionen dadurch ausge-

```
ENTITY  valve-ok OF valve;
 ATTRIBUTES
    flow-enabled: boolean;
    doOpen: event;
    doClose: event;
 STATES
    closed:    do / flow-enabled = false;
    opening;
    open:      do / flow-enabled = true;
    closing;
 TRANSITIONS
    FROM closed TO opening
       doOpen;
    FROM opening TO closing
       doClose;
    FROM opening TO open
       tm(entry(opening) , 1);
    FROM open TO closing
       doClose;
    FROM closing TO closed
       tm(entry(closing) , 0.5);
    FROM closing TO opening
       doOpen;
END-ENTITY
```

Abbildung 4-14: AML-Repräsentation des Zustandsdiagramms von Abbildung 4-6

löst werden.

- *Zustandsinterne Aktionsregeln* beschreiben Aktionen, die innerhalb eines Zustands ausgeführt werden. Dazu gehören Eintrittsaktionen, Reaktionen auf Ereignisse innerhalb eines Zustands sowie Austrittsaktionen.
- *Aktivitäts-Regeln* beschreiben Aktivitäten und Beziehungen, die während des gesamten Zustands andauern. Sie entsprechen den klassischen Intra-State-Constraints.
- *Definitionsregeln* definieren bestimmte Typen von Ereignissen, insbesondere Timeouts und Wertänderungen.
- *Begründungsabschluß-Regeln* sagen aus, unter welchen Umständen der Zustand und andere lokale Attribute unverändert bleiben. Für lokal definierte Ereignisse geben sie z. B. hinreichende Bedingungen für das Ausbleiben des Ereignisses an.

Die fünf Regelgruppen werden weiter unten näher erläutert. Dabei repräsentiert **SCI** den Übersetzungsoperator für Aktionen und Aktivitäten. Für eine Aktion *a* ist er folgendermaßen definiert:

- Ist *a* ein atomares Ereignis *e*, dann besitzt **SCI**(*a*) die Form

  `(:TRUE-IN ?t (`*e* `= :OCCURING))`

- Ist *a* ein abgeleitetes Ereignis, z. B. ein Timeout, dann besitzt **SCI**(*a*) die Form

  `(:TRUE-IN ?t (`*att* `= :OCCURING))`

  wobei *att* ein neues Attribut vom Typ `:EVENT` ist, das durch Definitionsregeln (s. u.) aus *a* abgeleitet wird.

- Ist *a* eine Wertzuweisung der Form '*att* = *f* (att_1 , ... , att_n)', dann besitzt **SCI**(*a*) die Form

  `(:TRUE-IN (:after ?t) (`*att* `=` *f*`(`att_1`(:before ?t) , ... ,` att_n`(:before ?t)))`

32. In [Brinkop, 1999] werden Transitionsregeln als Effektaxiome bezeichnet.

Wertzuweisungen werden in der Übersetzung also nicht explizit repräsentiert, sondern nur implizit: als Aussage über den Wert nach Ausführen der Aktion. Dieser wird gegebenenfalls aus Attributwerten *vor* der Aktion berechnet. Ob sich der Wert durch die Aktion tatsächlich ändert, bleibt dabei offen.

Für Aktivitäten ist SCI wie folgt definiert:

- Ist *ac* ein Lisp-Ausdruck *sexpr [att_1 , ... , att_n]*, dann besitzt **SCI**(*ac*) die Form

  `(:NON-NIL` *sexpr [* att_1`(?t)` *, ... ,* att_n`(?t)` *]*`)`

- Ist *ac* eine Spezifikation der Form '*att = f (att_1 , ... , att_n)*', dann besitzt **SCI**(*ac*) die Form

  `(:TRUE-IN ?t (`*att = f(*att_1`(?t)` *, ... ,* att_n`(?t))))`

Bei der Übersetzung wird also davon ausgegangen, daß sich die Aktivität höchstens auf Attributwerte zum selben Zeitpunkt beziehen kann. Bei Abhängigkeiten von vergangenen Werten müßte sich die Aktivität in einem Zustand nämlich entweder auf Werte in anderen Zuständen beziehen oder könnte nicht direkt beim Eintritt in den Zustand, sondern nur verzögert beginnen. Dies widerspricht nach unserer Meinung der zugrundeliegenden Idee.

Transitionsregeln

Transitionsregeln repräsentieren AML-Konstrukte der Form

`FROM` S_1 `TO` S_2
e `[`*P [* att_1 *, ... ,* att_n *]* `]` `/` a_1 `,` `...` `,` a_m

durch

(S_1`-->`S_2
`(` `:IF` `(:AND` `(:TRUE-IN (:BEFORE ?t) (state =` S_1`))`
`(:TRUE-IN ?t (`*e* `= :OCCURING))`
`(:TRUE-IN (:BEFORE ?t) (`att_1 `= ?v1)`
…
`(`att_m `= ?`v_n`))`
`(:NON-NIL` *P [* `?v1 , ... , ?`v_n*]*`))`
`:THEN (:AND (:TRUE-IN ?t (exit-`S_1 `= :OCCURING)`
`(entry-`S_2 `= :OCCURING))`
SCI(a_1)
…
SCI(a_m)`)))`

Beispielsweise wird aus der Spezifikation

```
FROM closed TO opening
     doOpen;
```

des Ventilmodells in Abbildung 4-14 die Regel

```
(closed-->opening
   (  :IF   (:AND   (:TRUE-IN  (:BEFORE ?t) (state  = closed))
                    (:TRUE-IN  ?t           (doOpen = :OCCURING)))
      :THEN (:TRUE-IN  ?t    (exit-closed    = :OCCURING)
                             (entry-opening  = :OCCURING))))
```

Zustandsinterne Aktionsregeln

Zustandsinterne Aktionsregeln repräsentieren AML-Konstrukte, die Aktionen beim Betreten eines Zustands, beim Verlassen des Zustands und bei zustandsinternen Reaktionen auf Ereignisse spezifizieren. Aus

S`:` `entry` `[`*P [* att_1 *, ... ,* att_n *]* `]` `/` a_1 `,` `...` `,` a_m

wird dabei

```
(Entry-S-nr
   (  :IF   (:AND    (:TRUE-IN  ?t             (entry-S = :OCCURING))
                     (:TRUE-IN  (:BEFORE ?t)  (att1     = ?v1)
                                              ...
                                              (attn     = ?vn))
                     (:NON-NIL  P [?v1 , ... , ?vn]))

      :THEN(:AND     SCI(a1)
                     ...
                     SCI(am)    )))
```

Analog wird aus

```
S:   exit    [P [ att1 , ... , attn ] ]   /  a1 , ... , am
```

die DEEP-Regel

```
(Exit-S-nr
   (  :IF   (:AND    (:TRUE-IN  ?t             (exit-S = :OCCURING))
                     (:TRUE-IN  (:BEFORE ?t)  (att1     = ?v1)
                                              ...
                                              (attn     = ?vn))
                     (:NON-NIL  P [?v1 , ... , ?vn]))

      :THEN(:AND     SCI(a1)
                     ...
                     SCI(am)    )))
```

Schließlich wird aus

```
S:   on    e     [P [ att1 , ... , attn ] ]   /  a1 , ... , am
```

die Regel

```
(Reaction-S-nr
   (  :IF   (:AND    (:TRUE-IN  ?t    (state  = S)
                                      (e      = :OCCURING))
                                      (att1   = ?v1)
                                      ...
                                      (attn   = ?vn))
                     (:NON-NIL  P [?v1 , ... , ?vn]))

      :THEN(:AND     SCI(a1)
                     ...
                     SCI(am)    )))
```

Bei der Interpretation der Entry-Konstrukte folgt SCI nicht der in UML, sondern der in [Harel, Pol i ti1997] spezifizierten Semantik: die Wächterbedingungen werden *vor* der Transition und damit auch *vor* Änderungen ausgewertet, die durch die Austrittsereignisse des vorausgehenden Zustands oder den mit der Transition verbundenen Ereignissen bewirkt werden.

Die AML-Spezifikation aus Abbildung 4-14 enthält weder zustandsinterne Reaktionen noch Eintritts- oder Austrittsereignisse, so daß zustandsinterne Aktionsregeln in der Übersetzung nicht auftreten.

Aktivitätsregeln

Das AML-Konstrukt

```
S:   do / ac1 , ... , acm
```

spezifiziert, daß die Aktivitäten $ac_1 , ... , ac_m$ während des gesamten Zustands *S* ausgeführt werden. SCI übersetzt dies in die Aktivitäts-Regel

```
(Activity-S-nr
   (  :IF   (:TRUE-IN  ?t    (state  = S))

      :THEN(:AND     SCI(a1)
                     ...
                     SCI(am)    )))
```

Beispielsweise wird aus der Spezifikation

```
closed:   do / flow-enabled = false;
```

des Ventilmodells in Abbildung 4-14 die Regel

```
(ACTIVITY-closed-1
   (  :IF     (:TRUE-IN  ?t    (state  = CLOSED))
      :THEN   (:TRUE-IN  ?t    (flow-enabled = false))))
```

Definitionsregeln

In AML können Attribute nicht nur durch Bezeichner, sondern auch durch komplexe Terme repräsentiert werden. Dazu gehören '`entry(`*S*`)`' und '`exit(`*S*`)`', die den Eintritt in bzw. Austritt aus einem Zustand S repräsentieren. Dazu gehört auch ein Ereignisterm wie '`tm(`Δt, *e*`)`', der einen Timeout zur Zeit Δt *nach dem letzten* Auftreten von *e* bezeichnet.[33] SCI definiert für solche komplexen Terme zunächst ein lokales Attribut und führt dann Definitionsregeln ein, mit denen die Werte dieses Attributs aus den Werten der Teilterme hergeleitet werden.

Beispielsweise wird für jeden Timeout-Term '`tm(`Δt, *e*`)`' zunächst ein lokales Ereignis '`timeout-`*e*`-`*nr*' definiert, wobei *nr* eine fortlaufende Nummer ist. Dann werden folgende Regeln erzeugt:

- Falls *e* die Form '`entered(`*S*`)`' besitzt und nur beim Timeout verlassen wird:

```
(Timeout-entry-S-nr
   (  :IF     (:TRUE-IN    ?t       (entry-S              = :OCCURING)))
      :THEN   (:TRUE-IN  (?t + Δt)  (timeout-entry-S-nr   = :OCCURING))))
```

- Falls *e* die Form '`entered(`*S*`)`' besitzt und auch bei anderen Ereignissen verlassen werden kann:

```
(Timeout-entry-S-nr
   (  :IF   (:AND   (:TRUE-IN          ?t         (entry-S   = :OCCURING))
                    (:TRUE-IN  [] ?t (?t + Δt) [] (state     = :constant)))
      :THEN   (:TRUE-IN  (?t + Δt)  (timeout-entry-S-nr = :OCCURING))))
```

  Mit der zweiten Bedingung wird sichergestellt, daß der Timeout nur auftritt, wenn das System sich noch im Zustand *S* befindet.

- Falls e ein Ereignis anderenTyps ist:

```
(Timeout-e-nr
   (  :IF     (:AND   (:TRUE-IN          ?t         (e = :OCCURING))
                      (:TRUE-IN  [] ?t (?t + Δt) [] (e = :NOT-OCCURING)))
      :THEN   (:TRUE-IN  (?t + Δt)  (timeout-e-nr = :OCCURING))))
```

Begründungsabschluß-Regeln

Begründungsabschlußregeln[34] spezifizieren Bedingungen dafür, daß ein Attribut seinen Wert beibehält. Dies ist im Hinblick auf die Diagnose besonders wichtig, da hier auch die Persistenz von Werten begründet werden muß, um die Ursache für unvorhergesehene Änderungen zu identifizieren. Da die Werte lokaler Attribute wie Timeouts, Zustand, Ein- und Austrittsereignisse nicht von außen gesetzt werden können, können Begründungsabschlußregeln für sie direkt aus dem Zustandsdiagramm abgeleitet werden. Bei öffentlichen Attributen wäre dies nur für Ausgangs-Attribute möglich, deren Wert nur vom modellierten System abhängen. Diese werden jedoch in der Schnittstellenspezifikation nicht explizit von Eingangsattributen unterschieden.

Die Begründungsabschlußregeln für ein Timeout-Ereignis `timeout-`*e*`-`*nr* haben folgende Form:

```
(EC-timeout-e-nr-1
   (  :IF     (:TRUE-IN  ?t         (e              =  :NOT-OCCURING))
      :THEN   (:TRUE-IN  (?t + Δt)  (timeout-e-nr   =  :NOT-OCCURING))))

(EC-timeout-e-nr-2
   (  :IF     (:TRUE-IN          ?t         (e              =  :OCCURING))
      :THEN   (:TRUE-IN  [] ?t (?t + Δt) [] (timeout-e-nr   =  :NOT-OCCURING))))
```

33. Tritt *e* vor dem Ablauf von Δt erneut auf, dann beginnt auch der Timeout von vorne.

34. engl.: Explanation Closure, vgl. [Schubert 1994].

Ein Zustand kann nur über Transitionen verlassen werden. Er bleibt also unverändert, solange kein Ereignis auftritt, das eine solche Transition auslöst oder solange bei solchen Ereignissen die Wächterbedingungen nicht erfüllt sind. Daraus ergeben sich zwei Arten von Begründungsabschlußregeln für das Zustandsattribut:

- *Lokale* Regeln drücken aus, daß ein bestimmter Zustand nicht verlassen wird, solange die Vorbedingungen für alle Transitionen aus diesem Zustand unerfüllt sind. Keine Rolle spielt dabei, ob Transitionen möglich wären, wenn sich das System in einem anderen Zustand befände. Für den Zustand closing in der AML-Beschreibung aus Abbildung 4-14 ist dies z. B.:

```
(EC-local-closing
   (  :IF     (:AND   (:TRUE-IN  (:before ?t)  (state  = closing))
                      (:TRUE-IN       ?t       (doOpen = :NOT-OCCURING)
                                               (timeout-entry-opening-1
                                                       = :NOT-OCCURING)))
      :THEN   (:TRUE-IN  (:around ?t)  (state  = closing))))
```

- *Globale* Regeln drücken aus, daß kein Zustand verlassen werden kann, solange die Vorbedingungen aller im System definierten Transitionen unerfüllt sind. Solche Regeln sind nützlich, wenn die Historie des Systemzustands nicht vollständig bekannt ist. Für das Diagramm aus Abbildung 4-14 ist dies z .B. die Regel:

```
(EC-global-state
   (  :IF     (:TRUE-IN  ?t    (doOpen    =  :NOT-OCCURING)
                               (doClose   =  :NOT-OCCURING)
                               (timeout-entry-opening-1   = :NOT-OCCURING)
                               (timeout-entry-closing-1   = :NOT-OCCURING))
      :THEN   (:TRUE-IN  (:around ?t)  (state  = :CONSTANT))))
```

Das von MuDia verwendete Inferenzsystem EEP besitzt zur Zeit noch Schwächen bei der Verarbeitung lokaler Regeln. Für CHD-DIAS wurden daher fast ausschließlich globale Regeln verwendet.

Auf die Angabe der Begründungsabschlußregeln für andere lokale Ereignistypen wird aus Platzgründen verzichtet. Für Eintritts- und Austrittsereignisse decken diese Regeln im wesentlichen folgende Fälle ab:

- Wenn der Zustand konstant bleibt, dann finden keine Ein- oder Austrittsereignisse statt.
- Wenn ein Ereignis `entry-`S stattfindet, dann findet kein Ereignis `entry-`S' für $S \neq S'$ statt.
- Wenn ein Ereignis `exit-`S stattfindet, dann findet kein Ereignis `exit-`S' für $S \neq S'$ statt.

Optimierung

Nach der eigentlichen Übersetzung versucht SCI, die entstandene Menge von DEEP-Regeln zu vereinfachen. Ansatzpunkt dafür sind die lokalen Attribute, insbesondere Timeout-, Eintritts- und Austrittsereignisse, da ihre Werte nicht nach außen propagiert werden können. SCI verwendet hier im wesentlichen folgende Heuristik:

> *Eliminiere Mengen lokaler Attribute, deren Werte nur verwendet werden, um Werte anderer Attribute der Menge herzuleiten.*

Sind z. B. mit Austrittsereignissen keine Aktionen verbunden, dann werden ihre Werte nur in Begründungsabschlußregeln anderer Austrittsereignisse verwendet. In diesem Fall kann man alle Austrittsereignisse eliminieren.

4.8 Generierung anwendungsspezifischer Diagnosesysteme

In Abschnitt 4.5.3 wurde deutlich, daß aus denselben MDF-Funktionen Laufzeitsysteme für verschiedene Anwendungen generiert werden können. Die wesentlichen Schritte sind dabei die Generierung und die Inbetriebnahme des jeweiligen Auslieferungssystems. Sie werden von kommerziellen Softwarewerkzeugen wie InstallShield oder InstallAnywhere unterstützt. Diese Hilfe erstreckt sich jedoch naturgemäß nicht auf diagnosespezifische Teilaufgaben wie die Optimierung im Hinblick auf das Laufzeitverhalten. Darüberhinaus sollte eine geeignete Modularisierung der Diagnosesysteme vorgegeben sein, um den anwendungsunabhängigen Kern möglichst kostengünstig um anwendungsspezifische Elemente ergänzen zu können. Fraunhofer IITB befaßte sich im Rahmen von INDIA mit beiden Problemkreisen.

4.8.1 Wissenskompilation zur Verbesserung des Laufzeitverhaltens

Fraunhofer IITB untersuchte, welcher Teil der im bisherigen System zur Laufzeit durchgeführte Berechnungen sich mit Hilfe von Wissen über die betrachtete technische Anlage in die Phase der Laufzeitsystemgenerierung vorziehen lassen. Im einzelnen wurden drei Ansätze verfolgt: Mittels *Partieller Auswertung* können aus allgemein formulierten Modellen und aus Informationen über konkrete Anlageninstanzen spezialisierte Modelle erzeugt werden. *Memorierung* wird eingesetzt, um sich wiederholende Situationen bei der Modellanalyse zu erkennen und Mehrfachberechnungen zu vermeiden. *Inkonsistenztests*, die durch Vorab-Analyse des bisher zur Laufzeit aufgebauten Netzes von Abhängigkeiten zwischen Modellattributen erzeugt werden, sollen die Suche nach Inkonsistenzen beschleunigen. Im folgenden werden die wesentlichen Ergebnisse skizziert. Eine ausführlichere Darstellung findet sich in [Schmitz 1997].[35]

Partielle Auswertung

Die Grundidee der Partiellen Auswertung[36] reicht zurück in die frühen 60er Jahre. Sie besteht darin, aus einer allgemeinen Funktion und einem Ausschnitt ihres Definitionsbereichs vorab eine *spezialisierte* Funktion zu erzeugen, die für Eingaben im spezifizierten Ausschnitt in weniger Zeit dasselbe Ergebnis wie die Ausgangsfunktion liefert.[37] Ein wichtiges Beispiel sind Funktionen, bei denen sich der Definitionsbereich eines Arguments auf einen einzelnen oder auf wenige Werte beschränkt.

Für die Generierung anwendungsspezifischer Diagnosesysteme ist diese Technik von Bedeutung, weil viele Modelle Parameter enthalten, deren Wert eine anwendungsspezifische Konstante ist. Besitzt z. B. ein Parameter *prm* für eine bestimmte Anlage den Wert `false`, dann kann *prm* überall im Modell durch `false` ersetzt werden. Enthält das Modell darüberhinaus eine Regel der Form

```
(  :IF     (:TRUE-IN  ?t    (prm     = false))
   :THEN   (:TRUE-IN  ?t    (att     = 0))))
```

dann kann überall im Modell *att* durch 0 ersetzt und die Regel eliminiert werden. Besitzt *prm* dagegen den Wert `true`, dann kann die obige Regel zumindest eliminiert werden.

Generell müssen dabei zwei Parametergruppen unterschieden werden. Zur ersten Gruppe gehören alle Parameterwerte, die im Rahmen der Anwendung festgelegt werden. Dazu gehört beim Chemikaliendistributor z. B., ob ein konstruktiv möglicher Sammler oder Verteiler in der speziellen Anlage vorhanden ist. Die zweite Gruppe besteht aus jenen Parameterwerten, die erst bei der Inbetriebnahme spezifiziert werden und sich im Lebenslauf der Anlage ändern können. In unserem Beispiel gehört dazu z .B., welche Chemikalie sich in welchem Tank befindet. Partial Evaluation spielt daher sowohl bei der Generierung des Auslieferungssystems als auch bei der Systemkonfiguration eine Rolle.

Memorierung

Falls zwar vorab nicht bekannt ist, daß der Definitionsbereich eines Funktionsarguments nur wenige Werte enthält, die Funktion aber zur Laufzeit häufig mit denselben Argumenten aufgerufen wird, dann kann Memorierung[38] helfen. Ist die Berechnung nämlich aufwendig, aber ohne Seiteneffekte, dann ist es sinnvoll, Argumente und zugehöriges Funktionsergebnis zu speichern. Bei Folgeaufrufen mit denselben Argumenten kann dann der gespeicherte Wert zurückgegeben werden statt das Ergebnis neu zu berechnen. Memorierung setzt also bei der Modellanalyse zur Laufzeit an und ist daher strenggenommen keine Methode zur Wissenskompilation.

Für Lisp-Funktionen existieren inzwischen Programmpakete, welche die zur Memorierung benötigte Verwaltung von Funktionen, Aufrufparametern und Rückgabewerten weitgehend automatisieren. Programmierer oder Modellierer bezeichnen lediglich die Funktionen, die memoriert werden sollen. In einem Modell sind es typischerweise jene Funktionen, die nur auf Modellparametern, Attributen mit kleinem Definitionsbereich und auf solchen Attributen aufgerufen werden, deren Wert über längere Zeit konstant bleibt.

---

35. [Cadoli und Donini, 1997] ist eine eher theoretische Darstellung verschiedener Techniken zur Wissenskompilation.

36. engl.: *Partial Evaluation*

37. Vgl. [Jones 1996].

38. engl.: *memoization*, vgl. z. B. [Mayfield et al. 1995]

Inkonsistenztests

Bei der konsistenzbasierten Diagnose mit MuDia wird versucht, eine Inkonsistenz zwischen Beobachtungen und Modell aufzudecken. Das kann grundsätzlich auf zwei Arten geschehen: Entweder ergeben sich für dasselbe Attribut und denselben Zeitpunkt zwei unterschiedliche Werte (z. B. einmal aus dem Modell abgeleitet und einmal direkt am System beobachtet) oder im Modell ist eine Regel angegeben, die für eine bestimmte Wertebelegung eine Inkonsistenz meldet. Aus den Annahmen, die den an einer Inkonsistenz beteiligten Werten zugrundeliegen, können dann Kandidaten für die Diagnose berechnet werden. In MuDia werden dazu die Annahmen, die den Attributwerten zugrundeliegen, zur Laufzeit berechnet und in einem ATMS verwaltet. Dieser Prozeß ist jedoch sehr zeitaufwendig.

Es wurde daher ein Algorithmus entwickelt, mit dem im Rahmen der Laufzeitsystemgenerierung zunächst jene Pfade in einem Modell identifiziert werden, auf denen sich Inkonsistenzen ergeben können. Aus dieser Menge werden dann Funktionen auf den beteiligten Attributen generiert, welche die Attributwerte auf Konsistenz abtesten. Jeder dieser Funktionen wird dabei die Menge der Annahmen zugeordnet, die den betreffenden Pfaden zugrundeliegen.

Zur Laufzeit wird dann die bisherige Form der Modellanalyse ersetzt durch die Auswertung der Tests auf den verfügbaren Beobachtungen. Die dabei festgestellten inkonsistenten Annahmenmengen werden wie bisher zur Kandidatengenerierung verwendet.

Vorteile dieses Verfahrens sind zum einen, daß nur solche Modellzweige ausgewertet werden, die tatsächlich zu einer Inkonsistenz beitragen können. Zum anderen wird die zeitaufwendige Berechnung der Annahmen, die einem Attributwert zugrundeliegen von der Laufzeit in die Phase der Systemgenerierung verlagert. Beides spricht für eine wesentliche Verbesserung des Laufzeitverhaltens verglichen mit dem bisherigen Vorgehen von MuDia.

Bewertung der Ansätze

Partielle Auswertung und Memorierung bieten in der Praxis den Vorteil, daß sie mit relativ geringem Aufwand in existierende Diagnose- und Modellanalysewerkzeuge integriert werden können. Die Laufzeitverbesserungen waren in den untersuchten Beispielen mit weniger als 5% aber recht gering.

Der Algorithmus zur Generierung von Inkonsistenztests wurde nur auf dem Papier getestet, aber nicht implementiert. Er verspricht ein gegenüber der existierenden MuDia-Implementierung wesentlich verbessertes Laufzeitverhalten. Darüberhinaus ermöglichen die Vorabuntersuchungen des Abhängigkeitsnetzes auch Rückschlüsse auf die Eignung der verfügbaren Sensorik im Hinblick auf die Diagnose.

4.8.2 Modularisierung von Anwendungssystemen

Wie in Abschnitt 4.5.3 dargestellt, können im Rahmen der Anwendungsentwicklung verschiedene MDF-Anwendungssysteme generiert werden, die sich im Nutzungskontext und den dort benötigten Haupt -und Nebenfunktionen unterscheiden. Nach Auffassung von Fraunhofer IITB erweist es sich als sinnvoll, im Rahmen einer solchen Generierung ein problem- und nutzungskontextspezifisches Framework um Module für die Domäne und die benötigten Nebenfunktionen zu ergänzen. Bei einem Anlagenhersteller wie THEN gäbe es damit z. B. Frameworks für

- die Onboard-Diagnose im laufenden Betrieb,
- die Post-Mortem-Diagnose,
- die Diagnose im Call Center,
- die Ferndiagnose,
- ein Diagnosesystem im Servicekoffer des Servicetechnikers.

Im Rahmen von INDIA untersuchte Fraunhofer IITB, welche Funktionen eines Post-Mortem-Diagnosesystems vom Framework selbst bereitgestellt und welche ausgelagert werden sollten. Als Ausgangspunkt dienten die Erfahrungen mit MuDia bei der Entwicklung von Diagnosesystemen für drei verschiedene Anlagen: für den Chemikaliendistributor, für ein Beballastungssystem auf Offshore-Plattformen [Böttcher et al., 1994] und für eine Montagestation [Zimmermann-Sturm 1996]. Dabei erwies sich, daß ein Framework zur Post-Mortem-Diagnose im Rahmen der Generierung eines Anwendungssystems insbesondere um folgende anwendungsspezifischen Informationen ergänzt werden muß:

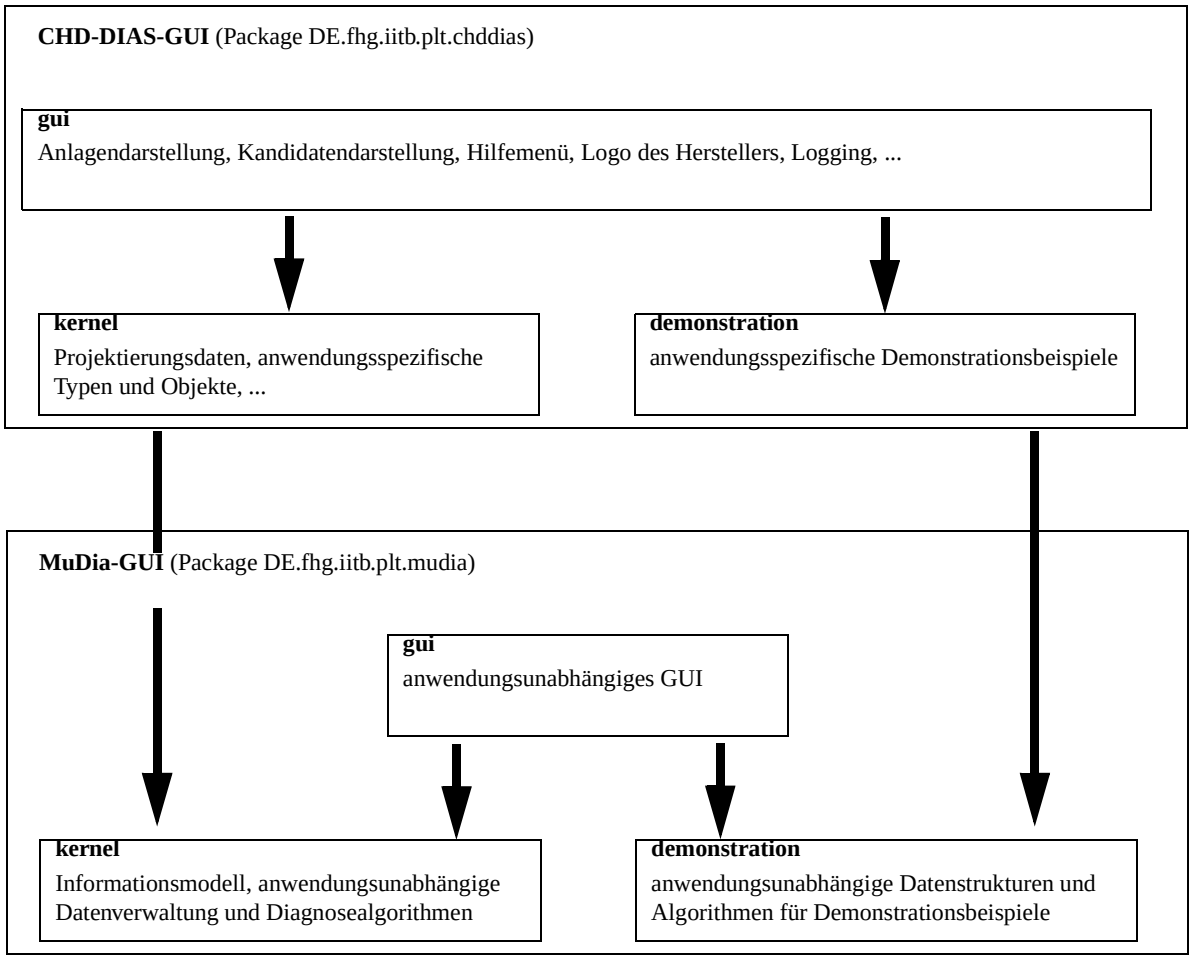


Abbildung 4-15: Modulstruktur der Graphischen Benutzungsoberfläche einer MuDia-Anwendung

- die Anlagenstruktur,
- die Bedienoberfläche, insbesondere bedienbare Bilder der Anlagenstruktur,
- die verfügbaren Datenquellen und die Methoden zum Zugriff auf diese Quellen,
- die Diagnosestrategie.

Schon in der alten Version von MuDia konnte eine deklarative Spezifikation der Anlagenstruktur zum Auslieferungssystem hinzugenommen und vom Laufzeitsystem verarbeitet werden. Als Konsequenz der genannten Analyse wurde MuDia nun umstrukturiert, um dies auch für die anderen Informationen zu ermöglichen. Im folgenden werden die dazu definierten Module und ihre Ankopplung an den Kern von MuDia skizziert.

Bedienoberfläche

Eine MuDia-Anwendung besteht aus der in Lisp implementierten Diagnosemaschine und der in Java geschriebenen graphischen Oberfläche. Beide kommunizieren über TCP/IP-Sockets. Abbildung 4-15 zeigt beispielhaft die Struktur der graphischen Oberfläche von CHD-DIAS. Sie besteht aus einem anwendungsunabhängigen Kern MuDia-GUI, der Teil des MuDia-Frameworks ist, und einem anwendungsspezifischen Modul CHD-GUI.

CHD-GUI implementiert die in Abschnitt 4.3 beschriebene graphische Oberfläche von CHD-DIAS in einem eigenen Java-Modul, das die Klassen von MuDia-GUI importiert. Diese stellen anwendungsunabhängige Dienste bereit, insbesondere

- zur internen Repräsentation der Anlagenstruktur;
- zur Bedienung und Beobachtung des Diagnoseprozesses über ein API;

- zur Bedienung und Beobachtung des Diagnoseprozesses über eine anwendungsunabhängige graphische Oberfläche, in der die Anlagenstruktur durch ein TreeControl repräsentiert wird;
- zur Verwaltung von Fehlerfällen, die zur Demonstration verwendet werden.

Die von MuDia-GUI bereitgestellte anwendungsunabhängige Oberfläche ist insbesondere während der Systementwicklung hilfreich. Mit ihrer Hilfe können Modelle getestet werden bevor Bedienbilder vorliegen.

Datenakquisition

Werte (Episoden) von Variablen werden aus *Datenquellen* (*Sources*) akquiriert. Quellen sind z. B. Dateien, Datenbanktabellen, online auslesbare Sensoren oder der Mensch. Quellen können Werte zu einer oder zu mehreren Variablen liefern. Jede Quelle spezifiziert, zu welchen Variablen sie mindestens Werte liefern kann.

Ein *Quellen-Manager* verwaltet eine Menge von in der Regel gleichartigen Quellen. Beispiele sind Datenbanken, Leitsystem, Datendateimanager oder Manager für die Datenakquisition durch den Menschen. Meßkosten werden dabei zunächst nur indirekt berücksichtigt.

MuDia unterscheidet zwei Typen von Quellenmanagern. Manager des ersten Typs fragen alle ihre Quellen ab, sobald bestimmte Strukturelemente im Rahmen der Modellanalyse betrachtet werden. Diese Quellen liefern im allgemeinen relativ kostengünstig Werte von Attributen dieses Strukturelements. Ein typisches Beispiel ist ein SCADA-System, das solche Werte routinemäßig erfaßt und ablegt. Manager des zweiten Typs verwalten Quellen, die nur zu relativ hohen Kosten Werte liefern können und daher gezielt abgefragt werden sollten. Dazu gehören insbesondere Beobachtungen des Bedienpersonals.

Eine spezielle Klasse von Datenquellen sind Datendateien. Sie werden von speziellen Datafile-Managern verwaltet. Die erste Eintrag jeder Datendatei spezifiziert das Format der übrigen Einträge. Die für dieses Format spezifizierten Methoden erlauben den Zugriff auf den Wert und die Zeitmarke einer Episode.

Während der Generierung anwendungsspezifischer Auslieferungssysteme wird das MuDia-Framework um ein Lisp-Modul ergänzt, bei dessen Laden Manager zur Verwaltung der verfügbaren Quellen erzeugt und dem Framework bekanntgemacht werden.

Strategiekomponente

Die Strategiekomponente versetzt MuDia in die Lage, die Diagnose bis zu einem gewissen Punkt automatisch durchzuführen und damit den Programmbenutzer zu entlasten. Nach dem Aufruf von der graphischen Oberfläche aus analysiert die Strategiekomponente zunächst die aktuelle Diagnosesituation. Auf dieser Grundlage wählt sie nach einer anwendungsspezifischen Strategie eine der von MuDia angebotenen Diagnoseaktionen aus und startet sie. Situationsbewertung, Aktionswahl und Aktionsdurchführung werden solange wiederholt, bis entweder ein von der Strategie festgelegtes Abbruchkriterium erfüllt wird oder kein Strukturelement mehr verdächtigt wird, auf das sich eine der verfügbaren Diagnoseaktionen noch sinnvoll anwenden ließe. In jedem dieser Fälle wird die Kontrolle an den Menschen zurückgegeben.

Strategien werden deklarativ in einer speziellen Sprache spezifiziert. Abbildung 4-16 zeigt beispielhaft die Spezifikation der von CHD-DIAS verwendeten Strategie. Die Reihenfolge der Schlüsselwörter in der Liste für `:next-component-precedence` ordnet die Kriterien zur Auswahl des nächsten Strukturelements, auf dem eine Aktion durchgeführt wird. Entsprechend ordnet die Reihenfolge der Schlüsselwörter für `:next-component-actions` die Auswahl der Aktionen auf dem gewählten Strukturelement. Das Fehlen expliziter STOP-Conditions führt dazu, daß die automatische Diagnose beendet wird, wenn es keine geeigneten verdächtigen Strukturelemente mehr gibt.

MuDia gibt eine Reihe möglicher Strategien vor, neue können leicht ergänzt werden. Das anwendungsspezifische Strategiemodul besteht im wesentlichen aus einer Datei mit den Einstellungen für den Strategen. Bei der Generierung des Auslieferungssystems wird diese Datei in das Auslieferungssystem übernommen und dafür gesorgt, daß sie zur Laufzeit gefunden wird.

4.9 Zusammenfassung und Ausblick

Ziel der Arbeiten des Fraunhofer-Instituts für Informations- und Datenverarbeitung IITB in INDIA war die Entwicklung von Methoden, um die Kosten der Einführung wissensbasierter Diagnosesysteme in die industrielle Pra-

```
((:initial-actions (:acquire-observations
                    :generate-candidates))
 (:diagnostic-loop-actions
  (:next-component-precedence( :highest-plausibility-candidate
                               :lowest-candidate-cardinality
                               :lowest-component-level
                               :structured-components
            ))
  (:next-component-actions (:check-known-faults
                            :focussed-diagnosis
                            :add-observations
            )))
 (:stop-conditions ()
            ))
```

Abbildung 4-16: Von CHD-DIAS verwendete Strategie

xis zu senken. Als Leitbeispiel diente die Entwicklung von CHD-DIAS, einem Diagnoseassistenten für Chemikaliendistributoren der Firma THEN.

Um Potentiale für Kostensenkungen zu identifizieren wurde zunächst ein Modell der Entwicklung von Diagnosesystemen erstellt. Es lieferte ein wesentlich differenzierteres Bild dieses Prozesses, den es in Werkzeugentwicklung, Anwendungsentwicklung, Inbetriebnahme und Systempflege einteilt.

Ein wesentlicher Kostenfaktor bei der Anwendungsentwicklung ist die Modellierung für Diagnosezwecke. Hierfür existiert bisher keine Methodik, so daß geeignete Modelle in der Regel von fachfremden Diagnoseexperten erstellt werden. Gleichzeitig existiert bisher auch keine Methodik, um für die Simulation eines Prozesses oder einer Anlage erstellte Modelle für Diagnosezwecke zu modifizieren. Auf Basis der von Fraunhofer IITB in INDIA entwickelten Heuristiken für Flußdiagramme scheint es nun möglich, den teuren Beitrag fachfremder Experten zu verringern und stattdessen Fachexperten bei der Ergänzung vorhandener und Entwicklung neuer Modelle für Diagnosezwecke anzuleiten.

In der Literatur gibt es verschiedene andere Ansätze zur Diagnose mit ereignisorientiert modellierten Systemen [Baroni et al., 1999], [Chen und Provan, 1997], [Cordier und Thiébaux, 1994], [Lunze, Schröder 1999], [Sampath et al. 1996], [Sampath et al. 1998]. Ein genauer Vergleich mit dem hier vorgestellten Ansatz für Zustandsdiagramme steht noch aus. Darüberhinaus bestehen Parallelen zur systematischen Spezifikation der Anforderungen an ein technisches System (Requirements Engineering) sowie zur Analyse der möglichen Fehler eines technischen Systems, z. B. im Rahmen einer FMEA. Auch diese müssen noch genauer herausgearbeitet werden.

Bei der Kostensenkung im Bereich der Werkzeugentwicklung erwies sich die Orientierung an den Anwendern vertrauten Modellierungsparadigmen ebenfalls als wesentlich. Durch Spezifikation einer auf ein solches Paradigma ausgerichteten Zwischensprache, die automatisch in die Repräsentationssprache des Diagnosesystems übersetzt werden kann, können im Rahmen der Entwicklung von Diagnosewerkzeugen die Kosten für Importfilter zwischen CAx-Systemen und Diagnosewerkzeug wesentlich verringert werden. Für die Modellierung mit erweiterten Zustandsdiagrammen wurde dieser Ansatz anhand der Zwischensprache AML und des Übersetzers SCI demonstriert. Mit diesem Ansatz können zum einen bestehende Modelle in die Diagnoseentwicklungsumgebung importiert werden. Zum anderen können Fachingenieure Modelle mit Hilfe der ihnen vertrauten Werkzeuge entwickeln, so daß der Beitrag teurer (weil fachfremder) Wissensingenieure verringert wird.

Schließlich wurden Ansätze vorgestellt, um die Generierung von Anwendungssystemen zu unterstützen. Bei der Wissenskompilation zur Verbesserung des Laufzeitverhaltens wurde bisher nicht ausgeschöpftes Potential deutlich. Die vorgestellte Modularisierung von MuDia demonstriert eine Alternative zur Entwicklung monolithischer Diagnoseschalen: hier wird ein Framework zur Post-Mortem-Diagnose ergänzt um Module, die anwendungsspezifische Informationen enthalten oder anwendungsspezifische Aufgaben lösen.

Ziel der weiteren Arbeit muß es sein, die Praxistauglichkeit der vorgestellten Ansätze zu bewerten und ihre Tragweite abzustecken. So müssen z .B. die Ausdruckskraft von AML und die Mächtigkeit von SCI erweitert werden, um einen größeren Teil der von StateMate o .a. Systemen bereitgestellten Ausdrucksmittel abzudecken. Die Heuristiken zur Modellierung für Diagnosezwecke müssen in entsprechende Werkzeuge umgesetzt und anhand größerer Beispiele erprobt werden. Schließlich muß die präsentierte Modularisierung von Post-Mortem-Diagnosesystemen verfeinert und auch auf andere Anwendungskontexte wie die Onboard-Diagnose übertragen werden.

Die erfolgreiche Vermarktung wissensbasierter Diagnosesysteme wird jedoch entscheidend von der Verfügbarkeit branchen- oder domänenspezifischer Bibliotheken abhängen. Mit ihrer Hilfe können Herstellern und Betreibern von Maschinen und Anlagen die Vorteile solcher Systeme sehr viel besser als bisher demonstriert werden. Darüberhinaus tragen sie wesentlich zur Kostensenkung im Entwicklungsprozeß bei. Die Auswahl geeigneter Branchen, die Entwicklung entsprechender Bibliotheken und ihre Erprobung zusammen mit Fachexperten sollte daher ein zentrales Anliegen zukünftiger FuE-Projekte im Diagnosebereich sein.

5 Literatur Teil III

[Ahrens et al., 1997]

W. Ahrens, H.-J. Scheurlen, G.-U. Spohr. *Informationsorientierte Leittechnik*. Oldenbourg-Verlag. 1997.

[Baroni et al., 1999]

P. Baroni, G. Lamperti, P. Pogliano, M. Zanella. „Diagnosis of large active systems“. In: *Artificial Intelligence*. Vol. 110, 1999. 135-183.

[Böttcher et al., 1994]

C. Böttcher, A. Brinkop, M. Zimmermann-Sturm. „Modellbasierte Fehlerdiagnose eines Ballastwassertanksystems“. In: *Proceedings Anwenderkongreß zur KI-94*. Springer-Verlag. 1994. 140-154.

[Brinkop, 1997]

A. Brinkop. *Anforderungen an standardisierte Repräsentationsformalismen zur Beschreibung eines Chemikalien-Distributors*. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB, Karlsruhe, INDIA-Report 97-02. 1997.

[Brinkop, 1999]

A. Brinkop. *Statecharts zur Simulation und Diagnose*. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB, Karlsruhe, INDIA-Report 99-01. 1999.

[Brinkop und Höfer, 1996]

A. Brinkop und U. Höfer. *THEN Farbküchensysteme - Der Farbküchenmanager AMC-CKM*. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB, Karlsruhe, INDIA-Report 96-02. 1996.

[Buchner et al., 1994]

H. Buchner, J. Lauber, M. Polke. „Das Informationsmodell: Basis für die interdisziplinäre Prozeßbeschreibung“. In *AT - Automatisierungstechnik*, 42 (1994) 1, 5-10.

[Cadoli und Donini, 1997]

M. Cadoli und F. M. Donini. „A survey on knowledge compilation“. In *AI Communications*, Vol 10, 1997. 137-150.

[Chen und Provan, 1997]

Y. L. Chen und G. Provan. „Modeling and Diagnosis of Timed Discrete Event Systems - A Factory Automation Example“. In *Proceedinsg of the American Control Conference, Albuquerque, New Mexico (USA), June 1997*. 31-36.

[Cordier und Thiébaux, 1994]

M.-O. Cordier und S. Thiébaux. *Event-Based Diagnosis for Evolutive Systems*. Internal Report 819, IRISA. Mai 1994.

[DIN-28004, 1988]

DIN Deutsches Institut für Normung e.V (Hrsg.). *DIN-28004, Teil 1-4: Fließbilder verfahrenstechnischer Anlagen*. Beuth-Verlag, Berlin, 1988.

[Davis, 1990]

E. Davis. *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers. 1990.

[deKleer, 1982]

J. deKleer and J. S. Brown. „Foundations of Envisioning“. In *Proceedings of the 2nd National Conference on Artificial Intelligence*, 18-20 August 1982, Pittsburgh (Pa), USA. 434-437.

[deKleer und Brown, 1984]

J. de Kleer and J.S. Brown. „A qualitative physics based on confluences“. In: *Artificial Intelligence*, 24:7-83, 1984.

[deKleer, Williams 1987]

J. de Kleer and B. C. Williams. „Diagnosing multiple faults“. In: *Artificial Intelligence*, 32:97-130, 1987.

[Douglass 1999]

B. P. Douglass. *Doing Hard Time - Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*. Addison-Wesley. 1999.

[Guckenbiehl et al. 1999]

T. Guckenbiehl, H. Milde, B. Neumann, P. Struss. „Meeting Re-use Requirements of Real-life Diagnosis Applications“. In F. Puppe (Hrsg.). *Knowledge Based Systems - Survey and Future Directions*. Springer Verlag, Lecture Notes in Artificial Intelligence 1570, 1999. 90-100.

[Harel, Pnueli 1985]

D. Harel und A. Pnueli. „On the Development of Reactive Systems“. In: K. R. Apt (Hrsg.). *Logics and Models of Concurrent Systems*. NATO ASI Series,Vol. F13. 1985.

[Harel, Politi 1997]

D. Harel und M. Politi. *Modeling Reactive Systems with Statecharts*. McGraw-Hill. 1997.

[Hemming 1993]

W. Hemming. *Verfahrenstechnik* (7., überarbeitete und erweiterte Auflage). Vogel, Kamprath-Reihe. 1993.

[ISO 1993]

Internationale Standardisierungs Organisation - TC184/SC4 (Hrsg.). *ISO-10303-1 Product Data Representation And Exchange - Part 1: Overview and Fundamental Principles*. ISO TC184/SC4 N193. März 1993.

[Jones 1996]

N. D. Jones. „An introduction to partial evaluation“. In *ACM Computing Surveys*. Vol 28, No. 3, Sept. 1996. 480-503.

[Leitch et al. 1991]

R. Leitch, H. Freitag, Q. Shen, P. Struss, G. Tornielli. „ARTIST A Methodological Approach to Specifying Model-based Diagnostic Systems“. In: *European Conference on Industrial Applications of Knowledge-based Diagnosis*, 1991.

[Lewis et al. 1995]

T. Lewis, L. Rosenstein, W. Pree, A. Weinand, E. Gamma, P. Calder, G. Andert, J. Vlissides, K. Schmucker. *Object-Oriented Application Frameworks*. Prentice-Hall. 1995.

[Lunze 1994]

J. Lunze. *Künstliche Intelligenz für Ingenieure, Bd 1: Methodische Grundlagen und Softwaretechnologie*. Oldenbourg, 1994.

[Lunze, Schröder 1999]

J. Lunze und J. Schröder. „Process Diagnosis Based on a Discrete-Event Description“. In: *at - Automatisierungstechnische Praxis*, Vol. 47, No. 8, 1999. 358-365.

[Marburger 1994]

H. Marburger. „Überwachung und Diagnose - Gemeinsam geht es besser“. In: Barth, G. et al. (Hrsg.): *KI-94: Anwendungen der Künstlichen Intelligenz*. Informatik Aktuell, Springer Verlag, Berlin. 1994. 84-92.

[Mayfield et al. 1995]

J. Mayfield, M. Hall und T Finin. „Using automatic memoization as a software engineering tool in real-world AI systems“. In *Proceedings of the Eleventh Conference on Artificial Intelligence for Applications*, Los Alamitos, Februar 1995. 8-93.

[Milde et al .2000]

H. Milde, T. Guckenbiehl, A. Malik, B. Neumann, P. Struß. *Integrating Model-based Diagnosis Techniques into Current Work Processes - Three Case Studies from the INDIA-project*. Zur Veröffentlichung eingereicht.

[Pahl, Beitz 1997]

G. Pahl und W. Beitz. *Konstruktionslehre - Methoden und Anwendungen*. 4. Auflage. Springer Verlag. 1997.

[Puppe et al. 1996]

F. Puppe, U. Gappa, K. Poeck, S. Bamberger. *Wissensbasierte Diagnose- und Informationssysteme*. Springer Verlag. 1996.

[Sampath et al. 1996]

M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. C. Teneketzis. „Failure Diagnosis using Discrete-Event Models“. In: *IEEE Transactions on Control Systems Technology*, Vol 4, No. 2, März 1996. 105-124.

[Sampath et al. 1998]

M. Sampath, S. Lafortune, D. Teneketzis. „Active Diagnosis of Discrete-Event Systems“. In *IEEE Transactions on Automatic Control*. Vol. 40, No. 7, July 1998. 908-929.

[Schahn 1994]

M. Schahn. „ROMEX - Ein Expertensystem zur Diagnose von Rotationsmaschinen“. In: *Marktreport 1994 - Intelligente Software-Technologien*. Oldenbourg-Verlag. 1994.

[Schmitz 1997]

R. Schmitz. *Laufzeitsystemgenerierung - Wissenskompilation*. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB, Karlsruhe, INDIA-Report 97-01. 1997.

[Schubert 1994]

L. K. Schubert. „Explanation Closure, Action Closure, and the Sandewall Test Suite for Reasoning about Change“. In: *Journal of Logic and Computation*, 4 (5). 1994. 679-699.

[Struß 1997]

P. Struß. „Fundamentals of Model-Based Diagnosis of Dynamic Systems“. In: *Proc. 15th International Joint Conference on Artificial Intelligence IJCAI-97, Nagoya, Japan*. 1997. 480-485.

[Taligent 1996]

Taligent Inc. (Hrsg.). *The Power of Frameworks*. Addison-Wesley. 1996.

[UML-RTF 1999]

UML Revision Task Force. *OMG Unified Modeling Language Specification, Vers. 1.3*. Document ad/99-06-08. Object Management Group. Juni 1999

[Wiesböck 1994]

Johann Wiesböck (Hrsg.). *Vorträge und Begleittexte zum Entwicklerforum: ``Local Operating Network LONWORKS``*, Hans-Pinsel-Straße 2, D-85540 Haar bei München, März 1994. Design & Elektronik.

[Zimmermann-Sturm 1996]

M. Zimmermann-Sturm. „Model-based Diagnosis of PLC Controlled Assembly Equipment“. In: *Proc. 6th International Conference on Data and Knowledge Systems for Manufacturing and Engineering (DKSME'96)*, Tampe (Arizona), USA. October 24-25, 1996. 11-22.

Anhang[1]

INDIA-Veröffentlichungen mit Beteiligung des IITB

Artikel:

Lothar Hotz, Peter Struss, Thomas Guckenbiehl. *INDIA - Intelligente Diagnose in der Anwendung*. Shaker-Verlag. 2000.

Heiko Milde, Thomas Guckenbiehl, Peter Struss. Integrating Model-based Diagnosis Techniques into Current Work Processes - Three Case Studies from the INDIA-Project. Erscheint in: *AI Communications*, Nr. 13(2)/2000.

Thomas Guckenbiehl, Heiko Milde, Bernd Neumann, Peter Struss. „Meeting re-use requirements of real-life diagnosis applications“. In: F. Puppe(Hrsg.). *Knowledge-Based Systems - Survey and Future Directions*. Springer Verlag, Lecture Notes in Artificial Intelligence 1570, 1999. 90-100.

Axel Brinkop. „Integrating Function and Behavior for Model-Based Diagnosis“. In: Yasusi Umeda (Hrsg.): *IJCAI-97 Workshop on Modeling and Reasoning about Function*, Nagoya, Japan, August 25, 1997. 1-10.

Thomas Guckenbiehl. „Intelligentes Diagnosesystem für prototypische Anwendungen“. In: *COMPUTERWOCHE* Nr. 8 vom 21.02.1997. 72.

Claudia Böttcher, Axel Brinkop, Thomas Guckenbiehl, Rainer Schmitz, Matthias Zimmermann- Sturm. "INDIA - Intelligente Diagnose für Industrielle Anwendungen". In: *IITB-Jahresbericht 1995*. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Karlsruhe, 1996.

Axel Brinkop. „Modeling in Standards - Requirements to Represent a Chemical Distributor“. In: Rogelio Soto, Jose M. Sanchez, Moraima Campbell and Francisco J. Cantu (Hrsg.). *Proc. ISAI / IFIS 1996, Mexico - USA Collaboration in Intelligent Systems Technology*, Cancún, México, November 12-15, 1996. 108-116.

Thomas Guckenbiehl. „Einbindung in das informationstechnische Umfeld: Herausforderung für die technische Diagnostik". In: *Proceedings 4. Kolloquium Technische Diagnostik 1996, Dresden, 14.-15. März 1996*. 314-320.

Forschungsberichte:

Axel Brinkop. *Statecharts zur Simulation und Diagnose*. INDIA Report 99-01. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Karlsruhe, 1999.

Michael Kappert. *Charakterisierung von MDF-Aufgaben*. INDIA Report 97-05. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Report 10456. Karlsruhe, 1997.

Rainer Schmitz. *Erweiterung von MuDia um eine abduktive Komponente*. INDIA Report 97-03. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Report 10454. Karlsruhe, 1997.

Axel Brinkop. *Anforderungen an standardisierte Repräsentationsformalismen zur Beschreibung eines Chemikalien-Distributors*. INDIA Report 97-02. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Karlsruhe, 1997.

Rainer Schmitz. *Laufzeitsystemgenerierung - Wissenskompilation*. INDIA Report 97-01. Fraunhofer- Institut für Informations- und Datenverarbeitung IITB. Report 10440. Karlsruhe, 1997.

Claudia Böttcher. *Strukturierung von Diagnosemaschinen - Auf dem Weg zu maßgeschneiderten Diagnosemaschinen von der Stange*. INDIA Report 96-04. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Report 10438. Karlsruhe, 1996.

Rainer Schmitz. *Laufzeitsystemgenerierung - Stand der Technik*. INDIA Report 96-03. Fraunhofer- Institut für Informations- und Datenverarbeitung IITB. Report 10431. Karlsruhe, 1996.

Axel Brinkop, Ulrike Höfer. *THEN Farbküchensysteme - Der Farbküchenmanager AMC-CKM*. INDIA Report 96-02. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB. Karlsruhe, 1996.

[1] Einige der hier aufgelisteten Berichte können auf der Seite: http://lki-www.informatik.uni-hamburg.de/~india abgerufen werden.

INDIA-Veröffentlichungen mit Beteiligung der TUM

Artikel:

Lothar Hotz, Peter Struss, Thomas Guckenbiehl. *INDIA - Intelligente Diagnose in der Anwendung*. Shaker-Verlag. 2000.

Milde, H., Guckenbiehl, T., Malik, A., Neumann, B., and Struss, P.: Integrating Model-based Diagnosis Techniques into Current Work Processes - Three Case Studies from the INDIA Project, to appear in AI Communications, Vol. 13(2), 2000.

Sachenbacher, M., and Struss, P.: Automated Determination of Qualitative Distinctions: Theoretical Foundations and Practical Results, to appear in Working Papers of the 14th International Workshop on Qualitative Reasoning (QR-00), Morelia, Mexico, 2000.

Sachenbacher, M., Struss, P., and Carlén, C.: Insights from Building a Prototype for Model-based On-Board Diagnosis of Automotive Systems, to appear in Working Papers of the 11th International Workshop on Principles of Diagnosis (DX-00), Morelia, Mexico, 2000.

Guckenbiehl, T., Milde, H., Neumann, B., and Struss, P.: Meeting Re-use Requirements of Real-Life Diagnosis Applications. In: Puppe, F. (ed.), XPS-99: Knowledge-Based Systems, Survey and Future Directions. Lecture Notes in Artificial Intelligence No. 1570, Springer, Heidelberg, 1999, ISBN 3-540-65658-8, pp. 90-100.

Mauss, J., and Sachenbacher, M.: Conflict-Driven Diagnosis using Relational Aggregations, Working Papers of the 10th International Workshop on Principles of Diagnosis (DX-99), Loch Awe, Scotland, 1999.

Struss, P.: There Are no Hybrid Systems - A Multiple-Modeling Approach to Hybrid Modeling In: Hybrid Systems and AI: Modeling, Analysis and Control of Discrete and Continuous Systems, AAAI Technical Report SS-99-05, March, 1999, p. 180-185, ISBN 1-57735-101-0.

Struss, P. and Heller, U.: Model-based Support for Water Treatment. In: Milne, R. (ed.): Qualitative and Model Based Reasoning for Complex Systems and their Control, Workshop KRR-4 at the 16th International Joint Conference on Artificial Intelligence, Stockholm, 1999, pp. 84-90.

Struss, P., Sachenbacher, M.: Significant Distinctions Only: Context-Dependent Automated Qualitative Modeling. Working Papers of the 13th International Workshop on Qualitative Reasoning (QR-99), Loch Awe, Scotland, 1999 Also in: Working Papers of the 10th International Workshop on Principles of Diagnosis (DX99), Loch Awe, Scotland, 1999.

Cunis, R., Struss, P: INDIA — Intelligente Diagnose in der industriellen Anwendung (in German). In: Projektträger Informationstechnik des BMBF bei dem DLR e.V. (ed.): Intelligente Systeme, Statustagung des BMBF in Neu-Ulm, 1998, pp. 75-85.

Mauss, J.: Analyse kompositionaler Modelle durch Serien-Parallel-Stern Aggregation. infix-Verlag, Dissertationen zur Künstlichen Intelligenz, DISKI 183, ISBN 3-89601-183-9, Sankt Augustin, 1998 (in German).

Mauss, J.: Local Analysis of Linear Networks by Aggregation of Characteristic Lines. Working Papers of the 9th International Workshop on Principles of Diagnosis (DX-98), Sea Crest Resort, Cape Cod, MA, USA, pp. 78-85, 1998.

Sachenbacher, M., Malik, A., Struss, P.: From Electrics to Emissions: Experiences in Applying Model-based Diagnosis to Real Problems in Real Cars. Working Papers of the 9th International Workshop on Principles of Diagnosis (DX-98), Sea Crest Resort, Cape Cod, MA, USA, pp. 246-253, 1998. Also in: Working Papers of the ECAI-98 Workshop on Model-based Systems and Qualitative Reasoning, Brighton, UK, 1998.

Struss, P., Heller, U., Malik, A., Mauss, J., Sachenbacher, M.: Model-based Tools for the Automotive Industry - Extended Abstract, Working Papers of the ECAI-98 Workshop on Model-based Systems and Qualitative Reasoning, Brighton, UK, 1998.

Struss, P., Heller, U.: Process-oriented Modeling and Diagnosis - Revising and Extending the Theory of Diagnosis from First Principles. Working Papers of the 9th International Workshop on Principles of Diagnosis (DX-98), Sea Crest Resort, Cape Cod, MA, USA, pp. 110-117, 1998.

Sachenbacher, M. and Struss, P.: Fault Isolation in the Hydraulic Circuit of an ABS: A Real-World Reference Problem for Diagnosis. Working Papers of the 8th International Workshop on Principles of Diagnosis (DX-97), Mont-Saint-Michel, France, 1997, pp. 113-119.

Struss, P.: Eine strategische Diagnose-Initiative ist nötig. Beitrag zu einer Podiumsdisskussion auf der XPS-97. In: KI 2 / 97, Juni 97, pp. 15-19, 1997.

Struss, P.: Fundamentals of Model-Based Diagnosis of Dynamic Systems. International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan, August 23-29, pp. 480-485, 1997.

Struss, P.: Model-based and qualitative reasoning: An introduction. In: Annals of Mathematics and Artificial Intelligence 19 (1997) III-IV, Baltzer Science Publishers, pp. 355 - 381, 1997.

Struss, P.: Model-based Diagnosis for Industrial Applications. Colloquium - Applications of model based reasoning, Institute of Electrical Engineers (IEE), Savoy Place, London, November 11, 1997.

Struss, P., Malik, A.: Automated Diagnosis of Car-Subsystems Based on Qualitative Models. In: Expertensysteme 97, Beiträge zur 4. Deutschen Tagung Wissensbasierter Systeme (XPS-97), 5.-7. März 1997, Bad Honnef am Rhein. PAI Proceedings in Artificial Intelligence, Vol. 6, infix-Verlag, pp. 157-166, 1996.

Struss, P., Sachenbacher, M., and Dummert, F.: Diagnosing a Dynamic System with (almost) no Observations. Workshop Notes of the 11th International Workshop on Qualitative Reasoning (QR-97), Cortona, Italy, 1997, pp. 193-201. Also in: Working Papers of the Third IJCAI Workshop on Engineering Problems for Qualitative Reasoning, Nagoya, Japan, 1997.

Heller, U., Struss, P.: Transformation of Qualitative Dynamic Models - Application in Hydro-Ecology. Workshop Notes of the 10th International Workshop on Qualitative Reasoning, AAAI Press, pp. 83-92, 1996.

Malik, A., Struss, P.: Diagnosis of Dynamic Systems Does Not Necessarily Require Simulation. In: Workshop Notes of the 10th International Workshop on Qualitative Reasoning QR-96, AAAI Press, pp. 127-136, 1996.

Malik, A., Struss, P.: Diagnosis of Dynamic Systems does Not Necessarily Require Simulation.7th International Workshop on Principles of Diagnosis (DX-96), Montreal, Canada, pp. 147-156, 1996.

Struss, P.: "Modellierung, qualitative", "Schließen, qualitatives" und "System, modellbasiertes". Drei Artikel im Wörterbuch der Kognitionswissenschaft (Hrsg.: Gerd Strube), Klett-Cotta, 1996.

Struss, P., Malik, A.: Automated Diagnosis of Car-Subsystems Based on Qualitative Models. In: Proceedings of the CESA'96 IMACS Multiconference: Symposium on Modelling, Analysis and Simulation, Lille, France, July 9-12, 1996, ISBN 2-9502908-5-X, Vol. 1, pp. 558-563, 1996.

Struss, P., Malik, A., Sachenbacher, M.: Qualitative Modeling is the Key to Automated Diagnosis. 13th World Congress of IFAC, San Francisco, CA, USA, Pergamon, 1996.

Struss, P., Malik, A., Sachenbacher, M.: Case Studies in Model-based Diagnosis and Fault Analysis of Car-Subsystems. In: 1st International Workshop on Model-based Systems and Qualitative Reasoning, ECAI'96 Workshop W23, pp. 17-25, 1996.

Montag, M., Struss, P.: Qualitatives und modell-basiertes Schließen. In: Einführung in die Künstliche Intelligenz, 2. Auflage. Görz, G. (ed.), Oldenbourg, ISBN 3-486-24367-5, pp. 87-128, 1995. (Revised Version of [Montag/Struss 93]).

Struss, P., Malik, A., and Sachenbacher, M.: Qualitative Modeling is the Key. Working Papers of the 6th International Workshop on Principles of Diagnosis (DX-95), Goslar, Germany, 1995, pp. 99-106.

INDIA-Veröffentlichungen mit Beteiligung des LKI

Artikel:

Lothar Hotz, Peter Struss, Thomas Guckenbiehl. *INDIA - Intelligente Diagnose in der Anwendung*. Shaker-Verlag. 2000.

Heiko Milde, Thomas Guckenbiehl, Peter Struss. Integrating Model-based Diagnosis Techniques into Current Work Processes - Three Case Studies from the INDIA-Project. Erscheint in: *AI Communications*, Nr. 13(2)/2000.

Andreas Günter, Lothar Hotz. KONWERK - A Domain Independent Configuration Tool. *In Proc. Of the Workshop on Configuration*, AAAI-99, Orlando 1999.

Heiko Milde, Lothar Hotz, Jörg Kahl, Bernd Neumann, Stephanie Wessel: Qualitative Analysis of Electrical Circuits for Computer-based Diagnostic Decision Tree Generation, in*: Proc. DX-99, 10th International Workshop on Principles of Diagnosis*, 1999.

Thomas Guckenbiehl, Heiko Milde, Bernd Neumann, Peter Struss. Meeting re-use requirements of real-life diagnosis applications. In: F. Puppe(Hrsg.). *Knowledge-Based Systems - Survey and Future Directions*. Springer Verlag, Lecture Notes in Artificial Intelligence 1570, F. Puppe (Eds.) , 1999. 90-100.

Lothar Hotz, Michael Trowe. NetCLOS and Parallel Abstractions – Actor and Structure-Oriented Programming on Workstation Clusters with Common Lisp. In Proc. ELUGM `99, *European Lisp Users Group Meeting*, Amsterdam, 1999.

Lothar Hotz, Michael Trowe. NetCLOS – Parallel Programming in Common Lisp. In Proc. PDPTA `99, *The 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, 1999.

Jörg Kahl, Lothar Hotz, Heiko Milde, Stephanie Wessel. Improving Reasoning Efficiency for Subclasses of Allen's Algebra with Instantiation Intervals. In *IJCAI-Workshop KRR-3, Hot Topics in Spatial and Temporal Reasoning*, Stockholm, 1999.

Jörg Kahl, Lothar Hotz, Heiko Milde, Stephanie Wessel. Improving Reasoning Efficiency for Subclasses of Allen's Algebra with Instantiation Intervals. In *Proc. 23. Jahrestagung Künstliche Intelligenz* (KI'99), Bonn, 1999.

Heiko Milde, Lothar Hotz, Jörg Kahl, Bernd Neumann, Stephanie Wessel. MAD: A Real World Application of Qualitative Model-based Decision Tree Generation for Diagnosis. In *Proc. IEA/AIE-99, The Twelfth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, Imam, I., Kodratoff, Y., El-Dessouki, A., Ali, M. (Eds), 1999.

Jörg Kahl, Heiko Milde, Lothar Hotz, Stephanie Wessel. A More Efficient Knowledge Representation for Allen's Algebra and Point Algebra. In Proc. *IEA/AIE-99, The Twelfth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, 1999.

Heiko Milde, Lothar Hotz, Jörg Kahl, Bernd Neumann, Stephanie Wessel. Qualitative Model-based Decision Tree Generation for Diagnosis in Real World Industrial Application, *KI-Zeitung*, 3/99, 1999.

Jörg Kahl, Heiko Milde, Lothar Hotz, Stephanie Wessel. MAD: Modellierungs- Analyse- und Diagnosesystem. *Fachausstellungsführer zur 23. Jahrestagung Künstliche Intelligenz* (KI'99), Bonn, 1999.

Jakob Mauss. *Analyse kompositionaler Modelle durch Serien-Parallel-Stern Aggregation.* infix-Verlag, Dissertationen zur Künstlichen Intelligenz, DISKI 183, ISBN 3-89601-183-9, Sankt Augustin, 1998 (in German).

Jörg Kahl, Heiko Milde, Lothar Hotz, Stephanie Wessel. MAD: Modellierungs- Analyse- und Diagnosesystem. *Fachausstellungsführer zur 22. Jahrestagung Künstliche Intelligenz* (KI'98), Bremen, 1998.

Jörg Kahl, Heiko Milde, Lothar Hotz, Stephanie Wessel. Automatic Generation of Decision Trees for Diagnosis: The MAD-System. *Int. Conference on Information Technology and Knowledge Systems*. Wien, Budapest, August-September 1998.

Lothar Hotz, Michael Trowe. Parallel Programming in Common Lisp using Actors and Parallel Abstractions. In Proc. WS 13 der KI´98, *Deklarative KI-Methoden zur Implementierung und Nutzung von Systemen in Netzen*, KI-98, Bremen, 1998.

Heiko Milde, Lothar Hotz, Ralf Möller, Bernd Neumann: Resistive Networks Revisited: Exploitation of Network Structures and Qualitative Reasoning about Deviation is the Key. *8th International Workshop on Principles of Diagnosis (DX'97*), 1997.

Lothar Hotz, Michael Trowe. Distributing Constraints on Workstation Clusters by using a Structure-Oriented Programming Model. *Parallel Computing '97* (ParCo97), Bonn, September 1997.

Lothar Hotz. Überlegungen zur parallelen Verarbeitung in Konfigurierungssystemen. *In Proc. Workshop Planen und Konfigurieren, PUK '96*, April 1996.

Jakob Mauss, Bernd Neumann: How to Guide Qualitative Reasoning about Electrical Circuits by Series-Parallel Trees. In *Proceedings of the tenth International Workshop on Qualitative Reasoning (QR'96)*, 1996.

Lothar Hotz, Heiko Milde, Ralf Möller, and Bernd Neumann. Using Behavior Deviations and an Interval-Based Calculus for Modeling Electronic Circuits, in*: Proc. Workshop der Fachgruppe Modellbasiertes und Qualitatives Schliessen*, also published as LKI-Memo No. 97/1, 1997.

Forschungsberichte:

Lothar Hotz. *Deklarative Beschreibung von Diagnose-Kontrollprinzipien.* INDIA Report 99-03. Universität Hamburg, August 1999.

Lothar Hotz, Heiko Milde, Joerg Kahl, Stephanie Wessel. *General Specification of the Modelling, Analysing and Diagnosing System(MAD) - Second Prototype* - INDIA Report 99-02. Universität Hamburg, August 1999.

Stephanie Wessel. *Kontrollprinzipien für Diagnose unter Ausnutzung von Kostengesichtspunkten.* INDIA Report 98-06. Universität Hamburg, August 1998.

Heiko Milde. *Integration lokaler und globaler Methoden zur qualitativen Analyse linearer Netze.* INDIA Report 98-05. Universität Hamburg, August 1998.

Stephanie Wessel. *Entwicklung von theoretischen Grundlagen für diagnostische Kosten- und Nutzenmodelle.* INDIA Report 98-04. Universität Hamburg, August 1998.

Jörg Kahl. *Reduktion des Meßaufwandes mittels funktionaler Modellierung.* INDIA Report 98-03. Universität Hamburg, August 1998.

Heiko Milde. *Separation Method - Multiple faults analysed by the SDSP method.* INDIA Report 98-02. Universität Hamburg, Dezember 1997.

Lothar Hotz, Jörg Kahl, Heiko Milde, Bernd Neumann, Stephanie Wessel. *Entwurf eines Verfahrens zur automatischen Generierung von Fehlerstrukturen unter Einbeziehung des Wartungskontextes.* INDIA Report 98-01. Universität Hamburg, Dezember 1997.

Jakob Mauss. *Analyse kompositionaler Modelle durch Serien-Parallel-Stern (SPS) Aggregation*. INDIA Report 97-06. Universität Hamburg, September 1997.

Sabine Kockskämper, Lothar Hotz, Heiko Milde, Ralf Möller, Bernd Neumann, Stephanie Wessel, *Pflichtenheft für den 1. Prototyp.* INDIA Interner Report 97. Universität Hamburg, 1997.

Lothar Hotz, Heiko Milde, Bernd Neumann, Stephanie Wessel, *Spezifikation des 1. Prototyps für das Modul zur Erzeugung der Fehlerklassenstruktur.* INDIA Interner Report 97. Universität Hamburg, 1997.

Heiko Milde. *Qualitative Analyse von Widerstandsnetzwerken für die Diagnose.* LKI-Memo LKI-M-97/2.

Heiko Milde. *Untersuchung der SDSP-Analyse auf Vollständigkeit und Korrektheit.* LKI-Memo LKI-M-97/4.

Die Autoren

Cunis, Roman – ServiceXpert GmbH, Hamburg

Heller, Ulrich – Technische Universität München, Institut für Informatik

Hotz, Lothar – Universität Hamburg, Labor für Künstliche Intelligenz

Kahl, Jörg – Universität Hamburg, Labor für Künstliche Intelligenz

Guckenbiehl, Thomas – Fraunhofer-Institut für Informations- und Datenverarbeitung, Karlsruhe

Mauss, Jakob – Technische Universität München, Institut für Informatik

Malik, Andreas – Robert Bosch GmbH, Stuttgart

Milde, Heiko – Universität Hamburg, Labor für Künstliche Intelligenz

Münker, Burkhard – R.O.S.E. Informatik GmbH, Heidenheim

Neumann, Bernd – Universität Hamburg, Labor für Künstliche Intelligenz

Sachenbacher, Martin – Technische Universität München, Institut für Informatik

Struss, Peter – Technische Universität München, Institut für Informatik

Volke, Stefan – Still GmbH, Hamburg

Weber, Reinhard – Robert Bosch GmbH, Stuttgart

Weiler, Harald – Robert Bosch GmbH, Stuttgart

Wessel, Stephanie – Universität Hamburg, Labor für Künstliche Intelligenz

Adressen der Projektpartner

Fraunhofer Institut für Informations- und Datenverarbeitung (IITB)
Dr. H.-W. Früchtenicht
Fraunhoferstraße 1
D-76131 Karlsruhe

Robert Bosch GmbH
Dr. Reinhard Weber
Abt. FV/FLI
PF 10 60 50
D-70049 Stuttgart

R.O.S.E. Informatik GmbH
Dr. W. Seibold
Schloßstraße 34
D-89518 Heidenheim a.d. Brenz

ServiceXpert GmbH
Dr. Roman Cunis
Harburger Schloßstraße 6-12
D-21079 Hamburg

Still GmbH
Dr. G. Fromme
Berzeliusstraße 10
D-22113 Hamburg

Technische Universität München
Prof. Dr. Dr. Peter Struss
Institut für Informatik
Orleansstrasse 34
D-81667 München

Then Maschinen- und Apparatebau GmbH
V. Palm
Postfach 40 01 71
D-74510 Schwäbisch-Hall (Hessental)

Universität Hamburg
Prof. Dr. Bernd Neumann
Fachbereich Informatik, Labor für Künstliche Intelligenz
Vogt-Kölln-Straße 30
D-22527 Hamburg

Modellbasierte Diagnosetechniken werden seit fast 20 Jahren theoretisch und praktisch entwickelt. Gegenwärtig dringen diese Techniken verstärkt in die Anwendung im industriellen Umfeld. Diese Entwicklung sollte in dem vom BMBF geförderten Verbundprojekt INDIA – intelligente Diagnose in der industriellen Anwendung – weiter vorangetrieben werden. Dies geschah in drei Anwendungsbereichen:

- mechatronische Fahrzeugsysteme,
- Flurförderzeuge (z.B. Gabelstapler)
- Färbeanlagen.

Modellbasierte Systeme, wie sie in der Künstlichen Intelligenz entwickelt wurden, erlauben die Repräsentation von Wissen über einen Bereich, wie z.B. eine Klasse technischer Systeme als Gegenstand von Diagnoseproblemen und die Anwendung automatischer Schlußfolgerungsverfahren für die computergestützte Problemlösung.

Das Buch beschreibt sowohl Anwendungsanforderungen und -nutzen, als auch die technischen Grundlagen der modellbasierten Lösungen. Damit stellt es für forschungsinteressierte Industrievertreter, anwendungsinteressierte KI-Wissenschaftler und Studenten, sowie Ingenieure aus dem akademischen und industriellen Bereich eine wichtige Quelle dar, um an konkreten Fallstudien das Potential dieser Technologie und die Richtung weiterer Forschung zu ermessen.