Configuration
State of the Art and New Challenges

Lothar Hotz[1] and Thorsten Krebs[2]

[1] HITeC c/o Fachbereich Informatik, Universität Hamburg
Hamburg, Germany, 22527
hotz@informatik.uni-hamburg.de

[2] LKI, Fachbereich Informatik, Universität Hamburg
Hamburg, Germany, 22527
krebs@informatik.uni-hamburg.de

Abstract. In this paper, we give a survey on the AI-field Configuration. Because configuration of software-intensive systems is currently seen as a main challenge for knowledge-based configuration, this topic is emphasized. Main challenges related to this topic are next to others: software-intensive applications, combining of configuration, modeling and evolution of software components, integration of knowledge-based configuration with existing software engineering approaches, and representation of software variability.

1 Introduction

In this paper, we give a detailed insight in currently seen applications and challenges of knowledge-based configuration (Section 2). In the field of configuration, a focus is set to configuration of combined hardware/software systems (i.e. *software-intensive systems*) where main topics are: modeling of software, modeling of software states and state transitions, configuring in the context of product lines, evolution, innovative configuration, planning aspects within the development of software, configuring of software-supported processes (e.g. IT-based business processes), configuration problems in combination with web services.

2 Knowledge-based Configuration

The configuration of technical systems is one of the most successful application areas of knowledge-based systems [17]. The general task in this area is to assemble components (*domain objects*) for getting a description (a *configuration*) of a *product*. The configuration process takes the probably large number of dependencies between domain objects into account. This process is described by means of procedural knowledge. Domain objects, dependencies and procedural knowledge are modeled in the *configuration model*. Beside having a declarative representation the mode is processed by reasoning mechanisms for deriving a configuration. This is done by using logical languages where the semantics are well-defined [32].

In this paper, we describe the current situation in the research area of configuration, by giving the basic methodology which is used in configuration systems (Section 2.1), and new application areas which came up in the recent years (Section 2.2). One area, which gives new challenges for configuration methods is *software configuration*. The main difference of software configuration compared to "traditional" technical application areas is the evolving character of constantly changing software products. The different challenges, requirements, and problems raised by configuration of software-intensive products are a main subject of the Planning, Scheduling and Configuration Workshop 2003 [22]. As an overview, those topics are discussed in Section 2.3. In Section, 3 we summarize the derived research topics Thus, despite of the contribution in [38] where mainly knowledge-based configuration methods are presented, we focus on challenges and upcoming methodologies.

A frequently maintained collection of announcements concerning design and configuration is available on the internet[3]. One main workshop on configuration alternates between IJCAI and ECAI. Conferences and workshops concerning software variability can be found at ICSE (International Conference on Software Engineering), OOPSLA and ECOOP.

2.1 Methods, Applications and Tools in Knowledge-based Configuration

Rule-based Configuration Early configuration applications were based on rule-based configuration methods [31]. The probably most well-known rule-based configuration system is *XCON*. It was a very successful tool for the configuration of DEC computers but also was claimed to be not maintainable for large domains. Because of deficiencies of such systems (see e.g. [17]), we do not further elaborate on this topic.

Resource-based Configuration In resource-based configuration there is no explicit representation of compositional relations between domain objects. Exchange of resources is used to abstract from the direct relations. Components are selected because they supply abstract services that are needed by systems or components of systems.

According to [20], two kinds of relationships between domain objects can be distinguished. There are objects that *consume* resources and there are objects that *provide* them. The configuration starts with consuming a resource (i.e. a lack of this resource in the current partial configuration). During the configuration process, it is tried to balance the consumption and provision of resources. The configuration task is fulfilled when every resource consuming can be balanced with an according resource provision. [33, 37] use the term *functionality* and thereby emphasize the modeling of "functions" while using resources.

Case-based Configuration A major part of human expertise is based on past experiences. Case-based configuration provides a model for representing

[3] http://www.cs.wpi.edu/Research/aidg/AIinD-announce.html

the experiences and using them for problem solving. The knowledge base in case-based reasoning is a set of former cases. A reasonable definition of a case is "a contextualized piece of knowledge representing an experience" [35]. The processing required to analyze the experience is delayed until the time the case is retrieved for solving a new problem. However, at that time the problem is less haunting, because it is only needed to understand how the differences between the problem in the former case and the current problem affect the solution proposed in the recalled experience. Thus, the solution proposed for the former case is examined and applied to the current problem with suitable modifications. Two types of cases that can be distinguished are: *Normative cases* capture the typical situation that occurs more than once and in the same or a similar ways (e.g. reference configurations). *Special cases* on the other hand represent "out-of-the-normal" situations (e.g. customer specific product extensions).

Constrained-based Configuration The constraint-based approach consists of representing restrictions between objects and / or their properties with means of constraints. The configuration problem is explicitly described within the model. Conditions and coherence for the existence of possible system components in the product solution and value assignments are specified in constraints. Description of expert procedures is not necessarily part of the problem specification. This is optional for interactive navigation within the configuration process [24].

Constraint-based problem solving includes a constraint solver in the configuration process. Through constraint propagation, the value assignments of participating objects and their properties are restricted and thus, the search space is reduced. This has the impact of a more efficient configuration process and therefore provides a better performance for configuration systems. Constraint-based approaches are able to include customer decisions in an incremental process for generating a solution that is not only correct with respect to the configuration goal, but that is also in line with the customer's idea of a good product.

Structure-based Configuration In structure-based configuration, ontologies are generated for representing the configuration knowledge [14]. Three central knowledge types concerned with configuration tasks can be identified:

Conceptual Knowledge Objects in the application domain (*domain objects*) are described by means of *concepts*. Concepts are represented through a name and their properties (i.e. parameters and relations to other concepts). Taxonomic structuring is achieved by *specializations*. Compositional structuring is described by aggregation. Restrictions between multiple concepts and / or their properties are expressed by means of *constraints*.

Procedural Knowledge Knowledge about the configuration process describes the ordering and execution of configuration decisions, the focus on particular concepts and conflict resolution methods.

Task Specification The task specification describes the configuration goal. This specifies the demands a created configuration has to accomplish.

Hybrid Approaches When different kinds of configuration methods are combined, this is called *hybrid configuration* (compare [38]). Hybrid systems aim at the largest possible degree of flexibility by combining methods suitable for different configuration tasks and domains. Those methods can be combined orthogonally because distinct aspects of the configuration process are handled by the different methods. Thus, they are not contradictory. Especially for heterogeneous application domains where for example components and their relations are used as well as resource consumption, structure- and resource-based approaches can be used (compare [15, 29, 36]).

Tools and Applications Methods used in the area of configuration have well-defined, system-independent semantics which are manifested in implementations termed *configuration systems*. Such configuration systems for example guarantee that the configuration model is well-defined and consistent.

A domain-specific configuration system can be obtained by implementing an application user interface (UI) over the configuration model. Such an UI maps the knowledge types and the inference process to interface artifacts that are suitable for the supported domain. Configuration systems map the configuration process to an operational computer supported process. As such, configuration methods give a kernel technology for distinct application domains, but which always needs an appropriate surrounding. The *Drive Solution Designer (DSD)* [34] for example is a domain-dependent application of the tool *EngCon*.

In the following we give an overview over selected commercial configuration tools [17, 24].

Cosmos is a resource-based configurator. It is suitable for modular systems with only resource-based components. But when the configuration problem also includes other (i.e. not only based on resources) components or in combination with constraints that do not focus on balancing of resources, using *Cosmos* is problematic. The *ILOG Configurator* is a powerful example for a constraint-based configuration systems. It is a C++ library based on the *ILOG Solver*. The *ILOG Configurator* provides the possibility of generating new components at runtime. It is among other domains deployed for solving configuration problems in the telecommunication sector. Another configuration system based on a constraint solver is *Selectica*. Next to a component taxonomy, constraints and rules can be modeled for usage this tool. *Internet Pricing and Configurator* is part of the customer relationship management (SAP) [18]. Configuration problems can be defined in a class hierarchy containing parameters and relations between classes. Relations can be defined as procedures, rules and constraints. The user is interactively included in the configuration process. The configuration tools *KONWERK* [15] and *EngCon* [34] are examples for hybrid and structure-oriented configurators respectively. They guide a user incrementally through the configuration process in order to create a user-specific product configuration. A flow of configuration steps can be modeled as procedural knowledge and used to enhance the quality of possible solutions. This methodology is able to deal with taxonomic relations, decompositions, component-specific parameters and constraints between components and their properties. For *EngCon*, also the integration of a case-based approach is planned.

2.2 New Application Areas for Knowledge-based Configuration

Software-intensive Domains Car Periphery Supervision (CPS) systems monitor the local environment of a car. CPS systems comprise a family of automotive systems that are based on sensors installed around a vehicle. The measurement and evaluation of sensor data enables different kinds of applications. These can be grouped into safety-related applications like pre-crash detection, blind spot detection and adaptive control of airbags and seat belt tensioners, and comfort-related applications like parking assistance and adaptive cruise control [39]. Because of the wide variety of the involved components and the presence of distinct customer wishes CPS becomes an interesting area for applying knowledge-based configuration methods.

Nokia launches between 30 and 40 products every year [28]. Different challenges in coping with this market, put pressure on the development. Multiple languages need different input methods, e.g. Western languages, Arabic, Chinese, Thai or Hebrew. Next, the diversity of reused hardware components like display or keypad is growing with every new generation of products. Selection of features out of an high and increasing amount is hard to handle. A presented solution is a client-server architecture: a server represents a basic service in a product, while a client implements a feature.

Software Product Families The Linux Familiar project[4] is a next generation PDA operating system. In [42], a configuration modeling language aimed at representing the structure of physical products is used to model and configure a Linux Familiar system. In contrast to operating systems on PCs, in this case resource limitations such as the available amount of memory have to be faced.

[2] address the possibility of applying techniques developed for configuring mechanical and electronic products to configuring software. A mapping between architecture description languages (ADLs) and configuration ontologies is proposed. This shows that configuration languages can be used for representing architectural knowledge. Furthermore, the possibility of applying configurator tools developed for configuring mechanical and electronic products to representing and managing the variation points in a software product family is studied in [2]. Capturing aspects of ADLs seems to require extending the configuration ontology. These aspects include function binding and binding the connection points of compound components with connection points in its inner parts and the modeling of behavior.

Web-based Scenarios Web services have a huge potential in business application integration. A configuration web service is currently being developed by [10] in the research project CAWICOMS[5]. An ontology-based approach allows

[4] http://handhelds.org/familiar
[5] http://www.cawicoms.org

the advertisement of services and a configuration-specific protocol defines the operational processes. Goal of the CAWICOMS project is to enable configuration systems to deal simultaneously with configurations of multiple suppliers over the web. Personalization techniques are applied to CAWICOMS, that personalize the interaction of customers, supporting individual needs during the configuration task and the presentation of solutions [1]. Suggestions of suitable choices assist the customer during the configuration process and provides the information needed for making decisions.

The presentation of personalized web page content is addressed in [8]. In the configuration process it is attempted to determine the optimal presentation of a web page while taking into account the preferences of the web author as well as viewer interaction with the browser. Preferences are encoded in CP-networks.

In [9] it is shown how to apply Semantic Web ontology representation languages for configuration knowledge representation and integration. A description logic-based definition of a configuration problem is given and its equivalence with corresponding consistency-based definitions is shown.

Differences between Hardware and Software Configuration According to [6] the hard part of building software is the specification, design and testing of the conceptual construct. Four major differences between "traditional" technical and software components can be seen, that make it hard to build, and thus to configure, software systems:
Complexity Software entities are more complex than any other human construct, because no two parts are alike. If they are, a subroutine can be called handling the task of the two parts.

Conformity Complexity of software systems forces developers to design conform interfaces.
Changeability Software is constantly subject to pressure for change. It can be changed more easily than technical constructs because it is purely thought-stuff.
Invisibility Software is invisible. This makes it hard to geometrically abstract its functionality - i.e. making it visible for design and implementation.

2.3 Challenges and Research Topics concerning Configuration of Software-intensive Products

Representing Product Variability Because software is very flexible, it easily varies and thus a large variability is present in software-intensive products. For managing this, the knowledge about variability has to be structured accordingly. Beside this, for configuring software-intensive products different kinds of aspects have to be handled which all include variability and therefore are *configurable assets*. E.g. customer requirements, features [12, 19], design of the product architecture, standard code artifacts, software and hardware components and their relations and mappings in between (see e.g. [41]). The union of the components functionality has to provide the desired functionality corresponding to the customer requirements.

Because knowledge-based configuration methods are generic, i.e. domain-independent, in principle they can be applicable for representing those aspects. Thus, concept description, specialization and decomposition hierarchies, constraints, resources and cases can be used for describing variability aspects and restrictions (see e.g. [2] for architecture configuration). Furthermore compositional relations can be used to express diverse types of relations like *has-subfeatures*, or *requires* (see e.g. [13]).

The Product Derivation Process The process of configuring software-intensive systems using different types of knowledge is complex. Not only the mapping between features and software components is of importance, but also the point in time when a variability decision is made (*binding time* [4]). A distinction is made e.g. between development time, compile time, and runtime.

In general, for representing aspects of binding time, knowledge-based configuration techniques provide the representation of procedural knowledge with phases and strategies [16]. In each phase a distinct kind of knowledge can be focussed. For example, first describe the desired features, compute by automated configuring the relations between those and the mappings to software components. Second, switch to a phase where those software components are further configured and parameterized. However, if variability is bound later than at development time, and if complex dependencies have to be considered, the total configuration machinery has to be present in the application program. This might not be feasible according to resource limitations or not adequate as a problem solution.

Representing Functionality of Software Components When assembling software components, the problem is to identify suitable components specify their interfaces, and infer the overall state description of the aggregate. For the last task the *functionality of software components* in terms of states and their dependencies is important. Methods like state charts and automaton can possibly be used for modeling [27]. Further on, techniques like *simulating* totally or partly configured components should be considered.

The question how deep a functional modeling of software components should be is domain-dependent. For some configuration application it might suffice to represent functions only by names for referring to their behavior, which yields to a finite number of behavior based configurations. By using behavior models for function descriptions, an infinite number of configurations with different previously unknown behavior can be expressed. Criteria for deciding what approach is applicable are still missing.

Combining Product Derivation and Product Development When using knowledge-based configuration for configuring software-intensive systems an already existing development methodology has to be considered. This is due to the fact, that the reuse and development process for software is not that separated from the configuration process as this is the case for hardware configuration (see Section 2.2). Thus, the product configuration process has to be coupled with the product development and a configuration system. Moving from a more separating approach where first a configuration model is specified and then configurations are inferred by use of that model, to a more integrated approach where modeling and configuring are interchanged. To support a product developer, one has to combine all three tasks: modeling, configuring, and implementing, because all those tasks have to be executed for realizing a specific product. First, a suitable product is configured for new customer requirements by using the already existing knowledge (model). Then specific software modules, which are not yet incorporated in the model have to be implemented. After that, those new modules have to be included in the model. How this can be supported during the development phase, which is as a further difficulty probably distributed over multiple developers, is an open issue. Furthermore, in Software Engineering the development of a reusable platform and of customer specific products is highly organized, sometimes standardized in processes like the Rational Unified Process (RUP). If knowledge-based configuration technologies are used, they should fit in those processes.

Mappings to Domain-specific Notations While reasoning techniques are developed within knowledge-based configuration, *notation* is domain-specifically developed in Software Engineering. Examples are given by the Unified Modeling Language (UML) [3], or by more specific notations for features like FODA [25]. The general approach for handling such situations is by mapping knowledge-based configuration aspects to those notations, which is e.g. done for UML in [11].

Usage of the Configuration Model A further integration aspect is the usage of the configuration model for tasks other than configuration itself. For example, designing software components can be done with the configuration model for identifying variation points in the software component, by first introducing those points in the model, configuring some products and evaluating the configuration result.

Combining Knowledge-based Configuration and SCM Techniques On the technical side Software Configuration Management (SCM) systems already handle dependencies of software components, mostly seen as files. Thus, the well known versioning problem is solved by SCM systems. A main issue with SCM systems is the missing of an abstract, declarative model of the source code being configured [30]. Each task is done directly on source code and files. Knowledge-based systems provide a more formal notion of consistency and completion than SCM systems [29] and could give support in this issue. Further aspects are discussed in [41].

Integration of two Research Communities In the area of Software Engineering, the aspect of managing variability is seen as a main prerequisite for reuse of software components [4]. Knowledge-based configuration on the other side offers methods for configuring (which also could be called variability management) in a domain-independent way. One distinction is that knowledge-based configuration is more related to configuration and variability management, i.e. how to represent and using existing configurable assets (software, hardware components, documents, test suites etc.). Other aspects like the developing, designing and implementing software components, which *allow* a high degree of variability, are more covered by Software Engineering methods (e.g. design patterns, product families, meta programming, model-driven architecture). However, understanding each others similar topics is a further challenge and interdisciplinary aspect. A project in this direction is the EU-project ConIPF (Configuration of Industrial Product Families) [26], others are [40, 29].

Explainability For supporting transparency, traceability, and knowledge acquisition, sophisticated explainability mechanisms should be developed. These should include managing explanations concerning complex restrictions and alternative configurations. Methods like assumption-based truth maintenance systems (ATMS) are already applied for configuration problems but not included in existing tooling.

Quality Management and Cost Estimation Incorporating optimality methods is partly addressed in the research community but not included in existing tooling. But those aspects, especially cost estimation of realizing configured products, probably containing not yet developed parts, are of major importance for industrial realistic applications (see also [2]).

Integration of Configuration in Business Processes Aspects that are important for further moving configuration out of the research corner into industrial realistic processes are the request of integrating the methods in existing environments, i.e. not to develop stand-alone applications and using configuration results (which are descriptions of products) as input for successive processes, like logistics, planning, maintaining, diagnosing.

Knowledge Elicitation Because of the large variability, complex dependencies and probably low maturity of existing software implementations (less modularization, less documentation, less explicitness of variability [5]), it might be hard to acquire the necessary configuration knowledge from experts like software developers. Thus, for software-intensive products, the process of getting and representing the knowledge has to address those issues, e.g. by providing knowledge acquisition tools for supporting the modeling process.

Evolution In traditional applications where knowledge-based systems are used (e.g. aircrafts, drive systems, elevators, etc.) the manufacturing and development of the components to be configured are final. This means, all components are present, but have to be selected and parameterized by processing dependency structures. Due to easy changeable artifacts in the software domain, a different situation exists. During the engineering of customer-specific products, even when existing software artifacts are reused, some changes on those artifacts have to be implemented [26].

All kinds of configurable assets are subject of evolution: e.g. customer requirements, features, software and hardware components. Currently a copy-and-modify-approach often exists for dealing with evolution in system development.

New versions of components are created by copying code of similar systems and modifying it. For this, knowing of the newest and / or all versions is necessary. The selection of suitable versions for given requirements is done manually. An individual development of customer specific products is realized. Developing the devices for specific customer requirements is fast at first sight, but it becomes slow and expansive in contrast to reuse of such devices.

For making use of knowledge-based configuration, a model of the configurable assets has to be present for allowing logical inferences on the current partial configuration. Thus, when evolving those assets, the suitable parts of the model have to be evolved, too.

Besides handling such model changes in a consistent way, the current configuration process of a specific product should integrate evolution steps of certain software artifacts in a coherent way [30, 21, 23, 7]. For supporting this, besides model evolution, versioning of models, reconfiguration of existing configurations and products [24], distributed configuration and cost estimations for realizing configurations have to be taken into account.

3 Summary

In this paper, we give an overview of existing configuration methods and focus on configuration of software-intensive systems. By looking at applications recently developed, we reflect several challenges in the area of configuration of software-intensive systems. For this applications, the main benefit provided by knowledge-based configuration methodology is seen as the possibility to reason about software components. Thus, research topics concerning configuration methods can be derived when analyzing the challenges thrown by software-intensive systems – e.g. explainability of configuration results, requirement modeling in the language of customer wishes, uncertain reasoning, distributed configuration, general decomposition relations for modeling diverse knowledge types, functional and behavior aspects, integration in the overall production process, quality management, cost estimations of the product to be configured and integration of configuration in business processes.

References

1. L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, M. Meyer, R. Schäfer, W. Schütz, and M. Zanker, 'Customizing the Interaction with the User in Online Configuration Systems', in *Proc. of the Configuration Workshop on 15th European Conference on Artificial Intelligence (ECAI-2002)*, pp. 119–124, Lyon, France, (July 21-26 2002).
2. T. Asikainen, T. Soininen, and T. Männistö, 'Towards Managing Variability using Software Product Family Architecture Models and Product Configurators', in *Proc. of Software Variability Management Workshop*, pp. 84–93, Groningen, The Netherlands, (February 13-14 2003).
3. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, October 1998.

4. J. Bosch, *Design & Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Addison-Wesley, May 2000.
5. J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, and K. Pohl, 'Variability Issues in Software Product Lines', in *Proc. of the Fourth International Workshop on Product Family Engineering(PFE-4)*, Bilbao, Spain, (October 3-5 2001).
6. Frederick P. Brooks, 'No Silver Bullet: Essence and Accidents of Software Engineering', *Computer, Vol 20, No. 4*, 10–19, (April 1987).
7. D.C. Brown and Chandrasekaran B., *Design Problem Solving Knowledge Structures and Conrtol Strategies*, Pitman Publishing, London, 1989.
8. C. Domshlak, S. Genaim, and R. Brafman, 'Preference-based Configuration of Web Page Content', in *Proc. ECAI-Workshop Configuration*, pp. 19–22, Berlin, Germany, (August 21-22 2000).
9. A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker, 'A Joint Foundation for Configuration in the Semantic Web', in *Proc. of the Configuration Workshop on 15th European Conference on Artificial Intelligence (ECAI-2002)*, pp. 89–94, Lyon, France, (July 21-26 2002).
10. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, 'Semantic Configuration Web Services in the CAWICOMS Project', in *Proc. of the Configuration Workshop on 15th European Conference on Artificial Intelligence (ECAI-2002)*, pp. 82–88, Lyon, France, (July 21-26 2002).
11. A. Felfernig, G. E. Friedrich, and D. Jannach, 'UML as Domain Specific Language for the Construction of Knowledge-based Configuration Systems', *International Journal of Software Engineering and Knowledge Engineering*, **10**(4), 449–469, (2000).
12. A. Ferber, J. Haag, and J. Savolainen, 'Feature Interaction and Dependencies: Modeling Features for Re-engineering a Legascy Product Line', in *Proc. of 2nd Software Product Line Conference (SPLC-2)*, Lecture Notes in Computer Science, pp. 235–256, San Diego, CA, USA, (August 19-23 2002). Springer Verlag.
13. L. Geyer, 'Configuring Product Families using Design Spaces', *Integrated Design and Process Technology (IDPT-2002)*, (June 2002).
14. T. Gruber, 'Towards Principles for the Design of Ontologies used for Knowledge Sharing', *International Journal of Human-Computer Studies*, **44**, 907–928, (1995).
15. A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.
16. A. Günter and R. Cunis, 'Flexible Control in Expert Systems for Construction Tasks', *Journal Applied Intelligence*, **2(4)**, 369–385, (1992).
17. A. Günter and C. Kühn, 'Knowledge-based Configuration - Survey and Future Directions', in *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, ed., F. Puppe, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg, (March 3-5 1999).
18. A. Haag, 'Sales Configuration in Business Processes', *IEEE Intelligent Systems*, (July/August 1998).
19. A. Hein, M. Schlick, and R. Vinga-Martins, 'Applying Feature Models in Industrial Settings', in *Software product lines - Experience and research directions*, ed., Donohoe P., pp. 47–70. Kluwer Academic Publishers, (2000).
20. M. Heinrich and E. Jüngst, 'A Resource-based Paradigm for the Configuring of Technical Systems from Modular Components', in *Proc. of 7th IEEE Conf. on Artificial Intelligence for Applications (CAIA'91)*, pp. 257–264, Miami Beach, Florida, USA, (February 24-28 1991).
21. L. Hotz, A. Günter, and T. Krebs, 'A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development (position paper)', in

Proc. of Software Variability Management Workshop, pp. 136–140, Groningen, The Netherlands, (February 13-14 2003).

22. L. Hotz, T. Krebs, J. Sauer, and J. Köhler, *17th Workshop, New Results in Planning, Scheduling and Configuration (PuK2003)*, LKI, Univeristy of Hamburg, Hamburg, Germany, 2003.
23. L. Hotz and T. Vietze, 'Innovatives Konfigurieren in technischen Domänen', in *Proceedings: S. Biundo und W. Tank (Hrsg.): PuK-95 - Beiträge zum 9. Workshop Planen und Konfigurieren*, Kaiserslautern, Germany, (February 28 - March 1 1995). DFKI Saarbrücken.
24. U. John, *Konfiguration and Rekonfiguration mittels Constraint-basierter Modellierung*, Infix, St. Augustin, 2002. In German.
25. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, 'Feature-oriented Domain Analysis (FODA) Feasibility Study', *Technical Report CMU/SEI-90-TR-021*, (1990).
26. T. Krebs, L. Hotz, and A. Günter, 'Knowledge-based Configuration for Configuring Combined Hardware/Software Systems', in *Proc. of 16. Workshop, Planen, Scheduling und Konfigurieren, Entwerfen (PuK2002)*, ed., J. Sauer, Freiburg, Germany, (October, 10-11 2002).
27. C. Kühn, 'Modeling Structure and Behaviour for Knowledge Based Software Configuration', in *14th Workshop, New Results in Planning, Scheduling and Design (PuK2000)*, ed., http://www-is.informatik.uni oldenburg.de/sauer/puk2000/paper.html, (2000).
28. A. Maccari and A. Heie, 'Managing Infinite Variability', in *Proc. of Software Variability Management Workshop*, pp. 28–34, Groningen, The Netherlands, (February 13-14 2003).
29. T. Männistö, T. Soininen, and R. Sulonen, 'Modeling Configurable Products and Software Product Families', in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-2001) - Workshop on Configuration*, Seattle, USA, (August, 6th 2001).
30. T. Männistö and R. Sulonen, 'Evolution of Schema and Individuals of Configurable Products', in *Proc. of ECDM'99 - Workshop on Evolution and Change in Data Management*, Versailles, France, (November 15-18 1999). Springer Verlag.
31. J. McDermott, 'R1: A Rule-based Configurer of Computer Systems', *Artificial Intelligence Journal*, **19**, 39–88, (1982).
32. R. Möller, C. Schröder, and C. Lutz, 'Analyzing Configuration Systems with Description Logics: a Case Study', in *http://kogs-www.informatik.uni-hamburg.de/~moeller/publications.html*, University of Hamburg, (1997).
33. B. Neumann, *Ein Ansatz zur Wissensbasierten Auftragsprüfung für technische Anlagen des Breitengeschäfts*, Ph.D. dissertation, Universität Duisburg, 1990.
34. K.C. Ranze, T. Scholz, T. Wagner, A. Günter, O. Herzog, O. Hollmann, C. Schlieder, and V. Arlt, 'A Structure-based Configuration Tool: Drive Solution Designer DSD', *14. Conf. Innovative Applications of AI*, (2002).
35. M. Sasikumar, 'Case-based Reasoning for Software Reuse', in *Knowledge Based Computer Systems-Research and Applications (International Conference on Knowledge-Based Computer Systems)*, pp. 31–42, Bombai, India, (December 12-15 1996). Narosa Publishing House, London.
36. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, 'Towards a General Ontology of Configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), 12*, 357–372, (1998).
37. B.M. Stein, *Functional Models in Configuration Systems*, Ph.D. dissertation, Universität Paderborn, 1995.

38. M. Stumptner, ‘An Overview of Knowledge-based Configuration’, *AI Communications*, **10(2)**, 111–126, (1997).
39. S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick, ‘A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems’, in *Proceedings of In-Vehicle Software 2001 (SP-1587)*, pp. 43–55, Detroit, Michigan, USA, (March, 5-8 2001).
40. J. Tiihonen, T. Lehtonen, T. Soininen, A. Pulkkinen, R. Sulonen, and A. Riitahuhta, ‘Modelling Configurable Product Families’, in *Proc. of the 4th WDK Workshop on Product Structuring*, Delft, The Netherlands, (October 1998).
41. A. van der Hoek, D. Heimbigner, and L.W. Wolf, ‘Does Configuration Management Research Have a Future?’, in *Proceedings of the 5th Inter. Conf. on Software Configuration Management, LNCS 1005*, Berlin, (1995). Springer-Verlag.
42. K. Ylinen, T. Männistö, and T. Soininen, ‘Configuring Software Products with Traditional Methods - Case Linux Familiar’, in *Proc. of 15th European Conference on Artificial Intelligence (Configuration Workshop)*, pp. 5–10, Lyon, France, (July 21-26 2002).