Recognizing User Intentions in Incremental Configuration Processes

Thorsten Krebs
Labor for
Artificial Intelligence
University of Hamburg
Germany
krebs@informatik.uni-hamburg.de

Thomas Wagner
Center for
Computing Technology
University of Bremen
Germany
twagner@tzi.de

Wolfgang Runte
Center for
Computing Technology
University of Bremen
Germany
woru@tzi.de

Abstract

Increasing market demands concerning customer requirements and market pressure from competitors force several enterprises to diversify their product range. Offering more and more product variants has the implicit impact of growing development process complexity. Although many configuration problems can be solved with knowledge-based configuration, the usage of configuration tools is very often restricted to domain experts. In this paper we extend knowledge-based configuration with a novel plan recognition approach. We show that the combination of configuration and plan recognition simplifies the use of configuration tools for non-configuration experts and non-domain experts.

1 Introduction

Increasing market demands concerning customer requirements and market pressure from competitors force enterprises to diversify their product range. Offering more and more product variants has the implicit impact of growing development process complexity [Pulm, 2002].

Configuration is perhaps one of the most successful areas of application for AI-methods. Many real world problems were solved by using a wide range of approaches varying from consistency checking methods, intelligent heuristic search to ontology-based methods, according to the requirements of the problem domain [Stumptner, 1997]. Although many configuration problems can be solved using knowledge-based configuration, the use of the configuration engine is very often restricted to domain experts. This is due to the fact that relation between the selection of components and the related functionality is often very complex to represent. These requirements are usually met e.g. in B2B-scenarios[1] where the costumer also is a domain expert. However in B2C-scenarios[2] the customer usually has very restricted knowledge about the problem domain and almost none concerning configuration methods. Moreover, it is difficult for the customer to define his requirements in terms of the solution. Given he wants to configure a multimedia-PC for watching DVD movies and game playing, without expert assistance he will not be able to choose appropriate attribute value pairs for e.g. video card chips, processor speed or front side bus rate (FSB) and is hardly aware of the consequences. Thus, the quality of generated solutions can differ substantially in comparison to expert solutions. The knowledge base itself offers only little help since the domain knowledge usually is defined in terms of the solution: objects and restrictions between objects and their properties are defined by a domain expert who has detailed knowledge about the underlying domain theory. Furthermore, the knowledge base is built with respect to different technological aspects like updating and maintaining the knowledge, the configuration method used (i.e. the expressibility of the knowledge representation language) and optimality aspects. As a result, the knowledge base rarely reflects the problem-oriented view of amateur customers.

In order to support users in a custom fashion, configuration methods should be used in combination with methods that help to compensate the lack of expert knowledge. The three major aspects are:

Generating Explanations: The generation of explanations (e.g. *reasoning maintenance systems: TMS, ATMS, JMTS)* can provide a user with useful information about consequences of decisions during the configuration process.

Incremental Configuration: An incremental configuration approach supports the user with direct feedback about each decision made during the configuration process. This way the user is enabled to browse through the space of possible solutions and has a better basis for the next decisions. This approach also simplifies the choice of configuration decisions to retract (e.g. *undo*).

Plan Recognition: Another promising approach is the use of plan recognition. This method can be used to infer user intentions and to use this information to guide the configuration process, i.e. the extended use of appropriate (automatic) calculation methods and defaults to disburden the user from difficult decisions.

In this paper we focus on the use of plan recognition in an incremental configuration process. We show that recent approaches can be successfully integrated in order to guide non-expert users through the configuration process. We use the configuration of personal computers as an example. This

[1]Business To Business
[2]Business To Costumer

is a simple but well understood domain concerning the possible dependencies between hardware components and their properties.

The remainder of this paper is organized as follows. Section 2 gives a brief survey on incremental configuration with focus on the configuration engine EngCon[3]. Section 3 gives a short overview about plan recognition methods with focus on applicability in incremental configuration scenarios and in Section 4.2 we describe the application of plan recognition in an incremental configuration approach. Section 5 gives a survey about related work and Section 6 succeeds with a final discussion.

2 Incremental Configuration

In an incremental configuration process the user enters data to influence the process and thereby to adapt the solution to his specific problem description. Almost all recent configuration systems integrate user interaction in the decision process and thus try to generate a solution that fits best the customer requirements. The used methods of some product configuration engines are quite simple: *CAS Configurator* and *Cameleon EPOS*[4] (formerly *ET-EPOS*) for example use rules and decision trees to manage the configuration process. While the knowledge base in the CAS Configurator has to be modeled in the programming language C++, the decision trees in Cameleon EPOS can be modeled without any programming skills using the spread sheet of a standard office suite.

The *camos.Configurator*[5] (formerly *SECON*) assists the interactive configuration process by bidirectional evaluated rules, called *passive constraints*. By request of the user, it is possible to break such a restriction. This results in an inconsistent configuration but it also increases the acceptance of the system for sales and distribution.

Due to the fact that these tools use simple mechanisms, they provide a good consumer acceptance. They are suitable for domains with a manageable amount of possible solutions. But the use of rules and/or decision trees makes it impossible to represent complex relations and dependencies.

The intention of our approach is to support engines which are able to handle large scenarios with complex relations and dependencies. Examples for this class of configuration engines are *ILOG JConfigurator*, *Selectica ACE Enterprise*, *Tacton Configurator*[6] (formerly *SICS Obelics*) and *EngCon*. Except for EngCon all of these configurators are focused on constraint propagation techniques.

Modern constraint-based configuration engines mostly use a form of dynamic extension of the traditional (static) constraint paradigm: *generative* constraint satisfaction [Stumptner *et al.*, 1998]. A generative constraint satisfaction problem (CSP) is an extension of dynamic CSPs [Mittal and Falkenhainer, 1990] which complies with the characteristics of configuration tasks. During each configuration step the configuration is potentially expanded by additional components. Generative CSP is an elegant way to handle problems where the number of involved components is not known from the beginning.

In contrast to most configuration engines EngCon does not focus on a single approach (rules, decision trees, constraints, etc.). It combines different approaches for the configuration of complex scenarios. The outcome of an incremental configuration process in EngCon is exactly one solution, or none if the customer requirements are not compliant with the components' dependencies.

2.1 Configuration in EngCon

Configuration describes composing a complex product from individual components, out of a vast amount of possible product variants with respect to customer requirements and dependencies between and within these components. For achieving this, the configuration steps *parameterization* (i.e. setting attribute values), *decomposition* (i.e. instantiating parts of an aggregate) and *specialization* (i.e. "casting" an object to a more specific concept) are used.

The configuration knowledge is separated into object descriptions and their attributes (*domain knowledge*), the process describing a sequence of configuration steps (*procedural knowledge*) and a *task specification* [Ranze *et al.*, 2002]. This is further elaborated in the following subsections.

Domain Knowledge

Objects in the application domain (*domain objects*) are described by means of *concepts*. Concepts are represented through a name, their parameters (e.g. numbers, string, intervals or sets of numbers) and their relations to other concepts. There exist taxonomic (*is-a*) and compositional (*has-parts*) relations and restrictions (i.e. constraints) between domain objects and / or their properties.

Procedural Knowledge

While configuration steps affect the configuration (e.g. specifying object types or their attributes), procedural decisions have a strong influence on dynamic configuration flow of the configuration process (i.e. the order in which the configuration steps are evaluated). Thus, declarative procedural knowledge defines knowledge for the explicit navigation in the solution space.

The mechanisms used to control the configuration process are called *strategies*. Strategies are executed in a predefined order, based on assigned priorities. Different knowledge types for controlling the configuration process are covered. These include *focal knowledge*, describing which components are important, *agenda selection criteria* for selecting which configuration steps in the agenda should be executed and *calculation methods* stating how a configuration step is executed.

The configuration procedure in EngCon is cyclic. The following steps recur for each configuration step:

1. The current partial configuration is examined for possible actions that can further develop this configuration. These configuration steps are collected in an agenda.
2. Definable agenda selection criteria are evaluated to generate a ranking for the possible actions. These criteria

[3]www.encoway.de

[4]www.solyp.de, www.access-commerce.com

[5]www.camos.de

[6]www.ilog.com, www.selectica.com, www.tacton.com

are based on the concepts defined in the knowledge base and their attributes.

3. The selected configuration step is executed. Different calculation methods like user interaction or using default values can be applied.

4. The constraint net is propagated. Inconsistencies in the current partial configuration can be detected.

Task Specification

The task specification describes the configuration objective. This specifies the demands a created configuration has to accomplish. An object out of the knowledge base is declared to be *goal concept* and gets specified in its attributes during the configuration process.

3 Plan Recognition

Plan recognition has been applied to many fields of AI resulting in a wide range of AI-applications ranging from medical and technical diagnoses, natural language understanding to story interpretation and planning (e.g. [Ng and Mooney, 1991; Hobbs *et al.*, 1993]). Approaches to plan recognition usually focus on one of the following objectives:

1. prediction of future actions of a system or an agent, and

2. generation of explanations of observed actions based on some background theory.

The latter objective is usually directly associated with abductive inference. Abduction was introduced C.S. Pierce in his theory of science but has not gained much attention in AI until the early 90's (perhaps except for [Pople, 1973]). Beginning with [E. Charniak, 1985], plan recognition and especially abductive inferences have been promoted in AI-research. Abductive inference can be described by the following inference rule:

$$\frac{\alpha \rightarrow \beta, \beta}{\alpha}$$

Given some observation β, an abductive inference infers from β to the cause α. Given a logical background theory, Δ can be described by two key properties (a) $\alpha \cup \Delta \vdash \beta$ and (b) $\alpha \cup \Delta \not\vdash \perp$[7]. Therefore, abductive inference can be viewed as an inversion of *modus ponens* with some important differences. Abduction is a non-monotonic inference with a much less strict interpretation of logical implication (material implication). In order to generate reasonable results on abduction logical implication has to be interpreted as a relation from cause to effect (which is clearly not the case for *modus ponens*) (for detailed discussion on the precise semantics of abductive reasoning refer to [Kautz, 1987]).

Generating explanations by abductive inference can be divided into two phases:

1. hypothesis generation and

2. hypothesis selection.

[7]Inconsistency

In the first step all valid hypothesis are generated. Different methods have been proposed for this process depending on the underlying formal model of the represented knowledge. In the second step, an appropriate hypothesis has to be chosen among the candidates generated in the first phase[8]. The choice of the appropriate selection criteria is essential and strongly depends on the problem domain. In technical domains like diagnosis of technical devices syntactic selection criteria like global, syntactic minimality (*Occam's Razor*) can be used successfully since an expert is able to choose among different plausible candidate sets. On the other hand, in domains like natural language understanding and pragmatics, domain-dependent selection criteria seem to provide better results. They allow e.g. to define different levels of granularity.

But considering prediction, the abductive approach provides only a weak indirect support: the generated explanation can be used to build up a rough prediction. Furthermore, the representational restriction on horn clauses together with the results concerning the complexity of abduction makes it difficult to use in configuration approaches.

The abductive approach to plan recognition clearly focuses on generating explanations and is based on a *closed-world-assumption* (CWA) and is therefore not able to handle new knowledge/observations without extending the knowledge base. With the focus on prediction of future events, dynamic and static belief networks have been introduced (e.g. [Albrecht *et al.*, 1998], [Intille and Bobick, 1998]). Based on observations of previous actions, the aim is to predict precise future behavior in uncertain environments with incomplete knowledge. Dynamic belief networks (DBN) are characterized by an incremental growth over time. The nodes represent the domain variables at different points of time as an ordered sequence integrated in a belief network. DBN have been proved to be a successful approach in domains with a restricted set of relevant variables. However in configuration domains with up to thousands of domain variables, the complexity of this approach suffers from inefficient inference.

4 Extending the Goal Graph

4.1 Goal Graph

In order to support users with different background knowledge, both effective prediction of future actions and generating explanations is needed. A graph-based approach that captures these requirements was introduced by Hong [Hong, 2001]. A *goal graph* is used to recognize user intentions and therefore provides the configuration context. This can be used to generate explanations. In the context of knowledge-based configuration, goals represent configuration solutions – i.e. special characteristics of the selected goal concept from the configuration model. The PC for instance potentially can serve for hundreds of features. This wide range of functionality is not easy to overview for the user.

This approach is influenced by *planning graphs* as defined by [Blum and Furst, 1997] and *consistency graphs* by [Lesh

[8]Often these phases are tightly coupled in order to minimize the generation of redundant hypothesis i.e. the hypothesis selection criteria are already used during generation.

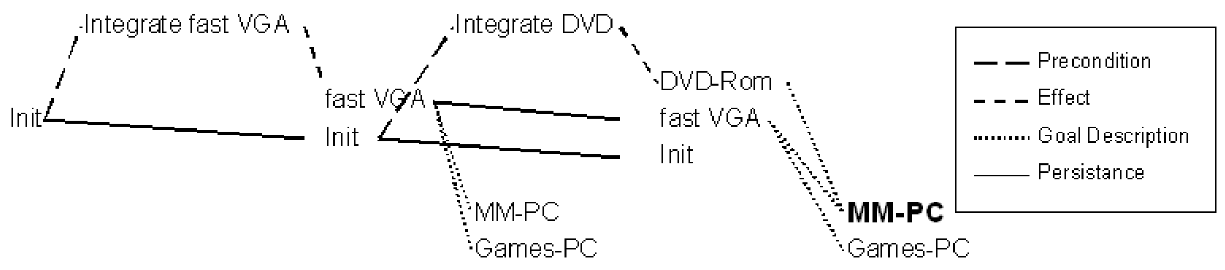

Figure 1: Goal Graph

and Etzioni, 1995]. Goal graphs are constructed and analyzed in two stages. First, the graph is constructed with the knowledge of observations. After that, the content of the graph is evaluated, concluding in goal assumptions.

Figure 1 depicts a goal graph that represents two time steps. The first recorded configuration step is the integration of a fast VGA card; this leads to two goal assumptions: a multimedia PC and a PC for game playing. In the second time step, the integration of a DVD-ROM was observed. This strengthens the assumption that the multimedia PC is the user intention and makes the goal assumption of a PC for games playing redundant, i.e. negligible.

A goal graph is growing incrementally with the amount of observed configuration steps and the states reached by these actions. There is always an initial node where the actions, states and goals are successively attached in chronological state levels. Different kinds of edges are used to combine nodes in the graph, depending on the types of nodes (see Figure 1). Goal nodes in the current state level describe potential user intentions.

4.2 Extending the Goal Graph for Incremental Configuration

The goal graph cannot directly be adopted for usage in EngCon. As we will further elaborate, a few adaptations are necessary in order to gain assumptions which are sufficiently accurate e.g. for dynamically changing the configuration procedure. Using goal graphs, a known problem is the potential recognition of equally plausible plans (as shown in Figure 1 where in the first time step no clear assumption can be given). The following two aspects are identified in order to improve the quality of goal assumptions in the domain of incremental configuration:

Emphasizing User Actions: In the original algorithm, the number of relevant actions is used to minimize this problem, but this way the content of actions remains hidden. Actions (i.e. configuration steps) are triggered by user interaction as well as system sided. But the user has no explicit impact on system sided decisions (e.g. taxonomic inferences or constraint propagation). Thus, emphasizing user actions in contrast to system decisions should yield to a more precise recognition of user intentions. When the user for example selects a specific motherboard for his PC, only one processor type might stay admissible. In this case the system would automatically specialize the processor without the user choosing such a configuration step.

Weighting of Goal Descriptions: Goals have a set of goal descriptions that make this goal a possible user intention. Such goal descriptions are not equally assessing the decision of an assumption and thus should not be seen equally significant. While some goal descriptions can be considered essential for a specific goal, others may only be casual hints. Defining a weighting for goal descriptions should also help advancing the quality of assumptions. While network cards are essential in servers, they may also exists in other PC types. Integrating a joystick for example gives a much stronger hint that the desired PC might be used for gaming.

The first criterion for computing the respective goals is based on the observed actions. All relevant actions A_{rel} (i.e. actions within a path leading from the initial state to the particular goal) are summed up and divided by the number of actions A in the whole graph. As actions are not seen as equally significant, their emphasized[9] values are used. A number between 0 (no action is relevant) and 1 (all actions are relevant) is computed:

$$Criterion_{action} = \frac{\sum A_{rel}}{\sum A} \mapsto [0 \mathinner{..} 1] \quad \text{with } A > \emptyset$$

The second criterion is the relation between valid goal descriptions and the descriptions of the particular goal. Like in the previous paragraph, here the emphasized values are used too, because goal descriptions are also not equally significant as well. The number of valid goal descriptions D_{graph} is divided by the number of descriptions that point to the particular goal D_{goal}. This also computes a number between 0 and 1.

$$Criterion_{description} = \frac{\sum D_{graph}}{\sum D_{goal}} \mapsto [0 \mathinner{..} 1]$$

In order to consider both of the above criteria, they have to be combined. Multiplying them computes a stochastically interpretable value inside the interval $[0 \mathinner{..} 1]$. This value is relatively small since empirically proved only a small amount of

[9]During the test phase, different values for multiplying the number of user interactions have been used (see also the evaluation in 4.4)

the observed actions (i.e. the configuration steps) is relevant for a particular goal. Optimization is gained by stretching the values for all potential goals so that the probabilities for n (partially) recognized goals $g \in G$ sum up to 1:

$$P_{sum} = \sum_{i=0}^{n} P(g_i) = 1$$

4.3 Algorithms

Goal Graph Construction The initial state is represented by a first state node. Actions are attached to states which describe a precondition for this action, and then states again are attached to actions as effects of these actions. Goals are attached to states which are valid goal descriptions for this goal. Over consecutive time steps, a structure alternating between states, actions and goals is constructed. Goal nodes in the current time step are potential user intentions.

In the following we present simplified pseudo-code algorithms for the action and goal expansion of the goal graph G in any time step i.

```
actionExpansion($A$)
  1. For all valid states $S$ in $G$ iterate all actions $a \in A$
  2. If any $s \in S$ is precondition for $a$ and $a \notin G$
     (a) Attach $Action(a)$ to $G$
     (b) Attach $PreconditionEdge(s, a)$ to $G$
     (c) For every effect $e$ of $a$
         i. Attach $state(e)$ to $G$
         ii. Attach effectEdge(a,e) to $G$
  3. For all valid $s \in S$ also valid in in the state level $i + 1$
     (a) Attach $s' = state(s)$ to $G$ in $i + 1$
     (b) Attach $persistenceEdge(s, s')$ to $G$
goalExpansion()
  1. For all valid states $S$ in $G$ iterate all goals $g \in G$
  2. If $\neg(g \in G)$ and if any $s \in S$ is goal description for $g$
     (a) Attach $goal(g)$ to $G$
     (b) Attach $descriptionEdge(s, g)$ to $G$
```

Goal Graph Analysis

Definition 4.1 (Valid Plan) *Given a goal g and a plan $p =< A, C >$ with A a set of recorded actions and C a set of temporal constraints ($a_i < a_j$) over A. Given the set of initial states I, p is a valid plan for g over I if and only if the actions in A can be executed in I in any order consistent with C, and g is fully achieved after that.*

Also, goals can be partially recognized, i.e. only a part of the goal descriptions of the particular goal are validated through states in the goal graph. In this case the particular characteristics of the configuration solution are not yet recognized but a subset of goal descriptions is.

In the following we also present a pseudo-code algorithm for the goal graph analysis.

```
goalGraphraphAnalysis()
  1. Set counters $i = 0$, $j = 0$ and the set of recognized goals $G_{rec} \leftarrow \emptyset$
  2. For all $g \in G$ in state level $i$ iterate all actions $a \in A$
  3. If there is a path from $a$ to $g$
     (a) Increment $i$ by the weighting of $a$
     (b) Increment $j$ by the weighting of the corresponding goal description $d_g$
  4. Set probability of $g$ to $\frac{i}{\sum a \in G} \times \frac{j}{\sum d_g \; inG}$
  5. Add $g$ with this probability to $G_{rec}$
```

Using goal graphs, one major problem is that concurrent goals may be recognized (in our example: multimedia PC and PC for game playing). This is the case when some of the goals have the same goal descriptions. A partial goal is *redundant* if the set of valid goal descriptions is a subset of another partial or fully recognized goal at the same time. A fully recognized goal is *redundant* if the set of valid goal descriptions is a subset of another fully recognized goal.

This mechanism consumes very little resources for plan recognition. It is proved that the algorithm can be executed in polynomial time. Furthermore, an efficiency growth could be seen in contrast to other graph-based methods like consistency graphs.

Theorem 4.1 (Goal Graph Construction is polynomial) *Given a goal graph with a observed actions, t time steps, i initial states and g goals. Given a_e the largest number of effects of any action and g_d the largest number of goal descriptions of any goal. In any time step the complexity of constructing the goal graph is polynomial in a, a_e, g, g_d, i and t.*

Proof 4.1 *The maximum number of state nodes in t is $i + a_e a$, the maximum number of action and goal nodes a and g respectively. Thus, in t there exist at most $n = i + a_e a + a + g$ nodes and in $t + 1$ there exist $n + a_e a + a + g$. The complexity for generating the nodes new in $t + 1$ is $O(a_e a + a + g)$ and is therefore polynomial.*

In $t + 1$ there exist at maximum $i + a_e a$ persistence, $(i + a_e a)a$ precondition, $a_e a$ effect and gg_d goal description edges. The complexity for generating all edges is $O(a_e a^2 + 2a_e a + ia + i + gg_d)$ and is thus also polynomial.

Theorem 4.2 (Goal Graph Analysis is polynomial) *Given a goal graph with t time steps and g complete or partial recognized goals at time step t. Given g_d the largest number of any goal description and a the number of observed actions. The number of possible paths between goals and their relevant actions and the time for recognizing all consistent goals is polynomial in a, g and g_d.*

Proof 4.2 *For any given goal in the graph, the maximum number of paths between relevant actions and the goal is the number of goal descriptions, i.e. g_d. At time step t there are at maximum g goals and a actions that are possibly relevant to this goal. Thus, the complexity for computing these paths is $O(g(g_d a))$ and therefore polynomial at any given time step.*

4.4 Evaluation

Note: The results presented in this paper can be seen as preliminary results. The tests have been carried out with a small number of students at the Center for Computing Technologies[10], University of Bremen.

The major issue about accuracy of an algorithm is the frequency of right assumptions. Moreover, the point in time during the configuration process, when these decisions are made can be important. This way they can have an explicit impact on the further configuration procedure.

The plan recognizer identifies user intentions in early configuration phases. This is not necessarily the case, since the generated probability values fluctuate and sometimes even lead to false assumptions. In general however, an increasing certainty in identifying the right goals can be noticed and very good assumptions are already made in early configuration stages. The generated results are very different concerning user-specific characteristics of the predefined goals and user skills (i.e. both, background knowledge in the current domain and the ability to use EngCon).

For general statements about the algorithm, results are evaluated in a combined fashion – i.e. probabilities for recognized goals have been averaged. In early stages of the configuration process, this averaged probability for correct assumptions starts with ca. 40%. But this value increases very fast over time with every observed action. In average, a probability of over 70% is gained for correctly predicting user intentions. This value may seem low at first sight, but one has to take into account that in an example with 5 partially recognized goals, the correct assumption reaches these 70% while the other 4 goals together make up the missing 30%.

Quantifier	Result
1	58.2%
2	61.6%
3	63.8%
4	63.8%
5	63.9%

	Result
with	63.8%
without	57.3%

Table 1: Enhancements through Algorithm Adaptations

The adaptations proposed in this paper (Section 4.2) improved the accuracy of the generated assumptions in the domain of incremental configuration. A probability growth of ca. 5% was reached for both, emphasizing on user interactions in contrast to system decisions (Table 1 on the left) and also for treating goal descriptions with various significance (Table 1 on the right). During the test phase, the quantifiers 1 to 5 have been used to emphasize user decisions. As shown in Table 1 on the left, accuracy of the predictions could be improved. Also, different quantifiers for goal descriptions have increased the accuracy of the algorithm. Attaching a laser printer e.g. is treated to point more likely to an office PC than integrating much memory points to a multimedia PC's as the latter may also yield to other PC types like data servers. Table 1 on the right shows the enhancement of differently significant goal descriptions (with) in contrast to treating all goal descriptions alike (without).

[10]www.tzi.org

5 Related Work

Plan recognition has been successfully applied to various domains [Carberry, 2001]. One example domain for the usage of goal graphs is the recognition of UNIX commands being the original field of Hong's algorithms. Also, several approaches for making the configuration process easier for non-expert users exist. Feature-oriented approaches [Kang *et al.*, 1990; Hein *et al.*, 2000] e.g. aim at hiding technical details from the user of configuration engines by abstracting the system functionality in terms of features. The integration of goal graphs in the domain of incremental configuration shown in this work is a novel approach.

6 Conclusion

In this paper we have shown how a graph-based plan recognition approach can be applied to an incremental configuration process in order to predict user intentions. The proposed algorithm recognizes configuration goals with high accuracy. Due to short processing times, good results are also expected for growing knowledge bases and plan libraries. A problem is the relatively high error rate at configuration start. This can be covered with methods from machine learning; user specific knowledge (e.g. knowledge about domain parts in which the user is expert or novice) can be collected. Every time a known user starts a configuration, his recorded preferences can be loaded and information about his behavior is present at early stages in the configuration process.

In the domain of knowledge-based configuration, goal recognition can be deployed for generating a better flow of configuration steps. Information about the context of configuration decisions can help in dynamically creating a configuration process that supports users with different background knowledge. For all application areas of the approach presented in this paper the major aspect is the classification of potential configuration goals.

Moreover, information gained from the goal recognizer can be deployed in other fields, e.g. in generating explanations. On basis of the current context, situations with unreachable goals can be recognized and explained to the user. Also, explanations are needed to justify system-sided choices (e.g. automatic specializations based on taxonomic inferences) [Bauer, 1996]. This way, the user's confidence in collaborating with such components can be advanced.

The configuration process can be divided into phases. Within a domain a user might have detailed background knowledge for some parts and less or even no knowledge about other parts. Such context sensitive coherence can be recognized within phases and e.g. the granularity of explanations or system decisions can be modified to fit the current situation.

References

[Albrecht *et al.*, 1998] D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Bayesian Models for Keyhole Plan

Recognition in an Adventure Game. *User Modeling and User-Adapted Interaction*, 8(1-2):5–47, 1998.

[Bauer, 1996] M. Bauer. Justification of Plan Recognition Results. In *Proceedings of 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 647–651, 1996.

[Blum and Furst, 1997] A. L. Blum and M. L. Furst. Fast Planning through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.

[Carberry, 2001] S. Carberry. Techniques for Plan Recognition. *User Modeling and User-adapted Interactions*, 11:31–38, 2001.

[E. Charniak, 1985] P. McDermott E. Charniak. Introduction to Artificial Intelligence. In *Addison Wesley, Meleno Park, California*, 1985.

[Hein *et al.*, 2000] A. Hein, M. Schlick, and R. Vinga-Martins. Applying Feature Models in Industrial Settings. In *Proc. of First Software Product Line Conference - Workshop on Generative Techniques in Product Lines*, Denver, USA, August, 29th 2000.

[Hobbs *et al.*, 1993] J.R. Hobbs, M.E. Stickel, D.E. Appelt, and P.Martin. Interptetation as Abduction. In *AI 63 / 69–142*, 1993.

[Hong, 2001] J. Hong. Goal Recognition Through Goal Graph Analysis. *Journal of Artificial Intelligence Research*, 15:1–30, 2001.

[Intille and Bobick, 1998] S. Intille and A. Bobick. Representation and Visual Recognition of Complex, Multi-Agent Actions using Belief Networks, 1998.

[Kang *et al.*, 1990] 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.

[Kautz, 1987] H. A. Kautz. *A Formal Theory of Plan Recognition*. PhD thesis, University of Rochester, 1987.

[Lesh and Etzioni, 1995] N. Lesh and O. Etzioni. A Sound and Fast Goal Recognizer. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1704–1710, 1995.

[Mittal and Falkenhainer, 1990] S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *In Proc. of 8th National Conf. on Artificial Intelligence (AAAI)*, pages 25–32, 1990.

[Ng and Mooney, 1991] H.T. Ng and R.J. Mooney. An Efficient First-Order Horn-Clause Abduction System based on the ATMS. In *AAAI–91 / 494–499*, 1991.

[Pople, 1973] H. Pople. On the Mechanisation of Abductive Logic. In *University of Pennsylvania, Theorem Proving and Logic II, Session 6*, 1973.

[Pulm, 2002] U. Pulm. Configuration Tools and Methods for Mass Customization of Mechatronical Products. In *Proc. of 15th European Conference on Artificial Intelligence (Configuration Workshop)*, Lyon, France, July 21-26 2002.

[Ranze *et al.*, 2002] 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.

[Stumptner *et al.*, 1998] M. Stumptner, G. Friedrich, and A. Haselböck. Generative Constraint-based Configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):307–320, 1998.

[Stumptner, 1997] M. Stumptner. An Overview of Knowledge-based Configuration. *AI Communications*, 10(2):111–126, 1997.