Knowledge-based Configuration for Configuring Combined Hardware/Software Systems

Thorsten Krebs[1] and **Lothar Hotz**[2] and **Andreas Günter**[3]

Abstract. In this paper, we present a survey on research topics we consider with in the EU project *ConIPF* (*Configuration of Industrial Product Families*). The application domain which is considered in this project contains combined hardware/software systems. Those systems share a lot of commonalities and are configured in a rich variety of outcoming products. Besides others, we discuss issues concerning features and evolution. Feature trees describing the functionality in terms of customer requirements are used to efficiently select, combine and configure components from the product catalogue. Mappings between these feature models and the product descriptions have to be developed and modeled sufficiently for the configuration task. A further problem area arises in this application domain with evolution in both the products themselves and the models describing the product catalogue.

1 Introduction

The project *ConIPF* (*Configuration of Industrial Product Families*) is a three year project that is carried out in the European Union (EU) with four partners (two industrial and two university partners). These partners are the University of Hamburg, the University of Groningen, Robert Bosch GmbH and Thales Naval Netherlands.

Software is an important basis of most technical systems. Growing complexity and variability of technical systems replace the development of single software products with the development of product families. Furthermore, the increasing use of embedded systems combining hardware and software make the process of software development to a difficult task. To get started, we will analyse software product development processes of industrial companies, restricting us to industrial product lines with hardware and software components. As industrial products, we will focus on the car periphery supervision domain, which includes services like pre-crash detection, and parking assistance. Besides considering software and hardware components, also requirement templates, feature models, and intermediate representations will be examined.

Currently one approach for handling software systems is the product line approach [11]. In this research field distinct kinds of variability of software and their realization with appropriate architectures and implementations are examined. In the project, we will consider how configuration methodologies known in Artificial Intelligence can be applied to the product line approach. Already discovered problem areas in the current situation are pointed out in the following. Possible solutions to these problems are given as far as the project status allows to consider this.

The paper is structured as follows. First we give an overview about some application domains. Section 3 introduces aspects about Configuration known in Artificial Intelligence (CAI). In Section 4 related work is discussed. In Section 5 we describe the problem domain and aspects of the possible solution domain. Section 6 presents future work that will be done in ConIPF.

2 Application Domain

As a relatively new domain for applying knowledge-based configuration methods in the EU project ConIPF combined hardware/software systems consisting of hardware components like sensors, electronic control units, displays and software components like assembler programs, DLLs, functions are explored.

2.1 Application Areas

The following are examples for application areas in the above mentioned domain:

- The **Car Periphery Supervision** pursues detection of the car environment, recognition of hazardous situations and handling of difficult traffic situations. Main goals are avoiding stress, accidents and thereby damage to the driver and the car itself. Typical applications are parking assistance, pre-crash detection, automatic cruise control, side obstacle detection, changes in geometry (e.g. trailer operation) and diagnosis.
- **Gasoline** and **Diesel Systems** are complex combinations of electronic control units (ECUs) like engine control systems, fuel support systems and on-board diagnosis systems. This includes hard- and software. These systems are developed for low-cost, midrange and high-end market segments and for different country-specific requirements like emission rules in Europe and the US.

2.2 Variability

In domains like the above mentioned, combined hard- and software systems are developed. The combination of these components is determined by customer requirements that are mapped to features which affect the possible selection of hardware components and software modules. Thus, variability in the development process are considered by the following aspects:

- **Customer requirements** can be of very different nature. Thus, not all possible combinations of a system are described in feature models. Sometimes the customer is not interested in all the details

[1] LKI, Fachbereich Informatik, Universität Hamburg, email: krebs@informatik.uni-hamburg.de
[2] HITeC, c/o Fachbereich Informatik, Universität Hamburg, email: hotz@informatik.uni-hamburg.de
[3] dito, email: guenter@informatik.uni-hamburg.de

of possible functionality. This makes the system not fully configurable with traditional knowledge-based configuration methods. Instead, components that are needed for a specific customer might have to be newly developed.

- **Features** are hierarchically modeled in a tree structure describing the system functionality. There can be hundreds of nodes in such feature trees. Features themselves can be related to each other by means of alternatives, existence or exclusion. Furthermore, features taking part in such relations can be mandatory or optional. In [6, 13] feature models that are under development in the area of combined hard- and software systems are described.
- **Software and hardware components** exist in distinct versions and variations, with sometimes explicitly documented or implicitly known dependencies between each others.

2.3 Current Configuration Approach

Currently there is no automatic configuration of standardized products in the companies that we considered. The focus is placed on functions for establishing a bottom-up composition of multi-functional systems. Variability usually evolves over long time periods (often over years). In order to establish a suitable product line approach, the need of computerized configuration support arises.

The development takes place in an evolutionary cycle such that future products benefit from the development experience. Development of products is an incremental procedure, which is customer specific. First a concept is elaborated which then is used for describing the product functionality. Subsequently, software components have to be coded, linked, packaged, tested and calibrated - in that order. When a new product is created the development starts often from similar products that has been developed before. Then existing models are copied and modified to meet the desired functionality. When the end of line is reached the product is delivered to the customer (e.g. a car manufacturer).

To handle different versions of software modules and changings Software Configuration Management (SCM) tools are used. SCM systems focus on controlling the whole evolutionary process of software system development. This process is seen as a continuous process which does not stop while the software is in use. To support this constantly changing process, SCM systems have been developed. There are a number of SCM systems which support the main concepts of SCM systems more or less (for surveys in SCM see e.g. [3, 4, 7, 21]). These concepts are: The *representation* of software objects as *atoms* by giving them a rudimentary name and a not further specified content ”description” (e.g. ”program code”, ”documentation”, ”test result” etc.), or as *configurations* by enumerating independently changing atoms or other configurations. *Version control* examines how interim products are produced in the course of product development and how development of different aspects of the product can proceed in parallel. *Change control* examines how changes to the software objects are more or less formally described by giving information about the change like the objective of the change, the state of the change (e.g. open, rejected) etc. *Process control* supports the whole software development process, by describing the process in terms of ”completion, acceptance, integration & test, take over”. *Distribution* is related to distributed development of software by locally distributed developers. Further issues on the relation between SCM and CAI are discussed in [14].

2.4 Some Requirements for Future Systems

There are a few goals for future development of products. All product-specific documentation should be collected in one place to allow usage in different stages (and departments) during the development process. Customer offers then could be made based on feature models. Furthermore system features could be used for a better understanding with the customer. For example opposing features could be detected. Also evaluation of customer requirements could be supported. Further issues are:

- For the development an aim is to distinguish between changes of existing products, which are already included in a **product line** and new product specific functionality, which is not part of a product line. By integrating new products into a product line approach a larger group of developers can be supported.
- Also **feature trees are evolving** over time. Features are integrated into already existing trees depending on whether they are relevant for the product line. New features can be used in future product developments.
- A further **goal of a configuration methodology** is the possibility of calculating time necessary for adopting the software and cost for the necessary hardware components for a customer specification. Further, a technical judgment through the sales department leading to some kind of filtering is aspired.

3 Knowledge-based Configuration known in Artificial Intelligence (CAI)

The configuration of technical systems is one of the most successful application areas of knowledge-based systems. [10] made a general analysis of configuration problems in which four central aspects (or knowledge types) concerned with configuration tasks were identified:

- A set of domain objects (*concepts*) in the application domain and their *properties* (parameters). By instantiating a domain object *concept instances* are created. Thus, a domain object describes its instances.
- A set of relations between domain objects. Taxonomical and compositional relations with alternatives, number restrictions, and optionality are of particular importance for configuration. But further relations can be expressed between arbitrary domain objects.
- A *task specification* (configuration objective) that specifies the demands, which a created configuration must fulfill.
- *Control knowledge* for specifying the configuration process.

For all those kinds of knowledge not only a model can be declaratively defined but also an inference machinery is given, which interprets the knowledge. Thus, CAI provides not only a modeling facility like STEP or UML but operational, processable models. In the so-called structure-based approach a compositional, hierarchical structure of the domain objects serves as a guideline for controlling the solution process, and as a logical basis for configuring. That means, that the constructs used for modeling can be mapped to a logical language where the semantics are well-defined [19]. The constraint-based approach consists of representing restrictions between objects or their properties by means of constraints, and evaluating these by constraint propagation. This approach is not in conflict with the structure-based approach but is frequently combined with it. Other approaches are resource-based and case-based configuration.

Concepts used in the area of configuration have well-defined, system independent semantics which are manifested in implementations termed *configuration systems* (CS) like *EngCon* [1]. Such systems provide a more formal notion of consistency and completion than Software Configuration Management systems [18]. For instance, in CS the consistency of the hierarchy is well-defined (namely a strict specialization hierarchy) and will be checked by a specific module of a CS. Constraint solving uses methods that have been proven correct, and property values of components can be inferred. The control mechanism determines that all open issues (e.g. properties, parts) of the configuration are handled by the configuration process [9]. All these modules of a CS are general and thus, domain independent. A domain specific configuration system can be obtained by implementing a domain specific application user interface over the domain specific configuration model. Such a user interface maps those kinds of knowledge and inference processes to interface components that are suitable for the domain specific configuration process being supported. For example, in the case of software development a user interface may consist of presentation tools for software objects using the concepts mentioned, presenting the evolutionary process etc. CS map the configuration process to an operational computer supported process. As such, CAI gives a kernel technology for distinct application domains, but which always needs an appropriate surrounding.

In CAI, this has been done traditionally for domains where hardware components are configured. For software, the main challenge is to understand the specific concepts of the software development process in terms of the general concepts of the logic-based configuration terminology.

Note that most configuration tasks from the principle-, variant- and adaptive construction areas known in the field of mechanical engineering can be equated with configuration.

For software product lines which are the objects of our attention, the non-static aspect of running software is important [18]. The main challenge is to understand the specific concepts of the software development process in terms of the general concepts of the logic-based configuration terminology.

4 Related Work

The Finnish research group around T. Männistö explores the usage of tool support for the configuration of software product families. In [22] a Linux Familiar operating system distribution is modeled with a configuration modeling language aimed at representing the structure of physical products. The used language is largely suitable for software product configuration. But it is also stated that functions, features or resources and optimality criteria, as well as versioning and reconfiguration would be useful.

In [2] the possibility of applying techniques developed for configuring mechanical and electronic products for configuring software is studied. A way of representing much of the architectural knowledge using the configuration modeling concepts is defined. This indicates that it is relatively easy to provide software configuration support using existing techniques. However, it is required to extend the current conceptualization of configuration knowledge to capture software products adequately.

5 Problem Areas and Possible Solutions

In this section we present problems mentioned in the Software Configuration Management (SCM) community and their potential solutions coming from Configuration Methodologies known in AI (CAI). We focus on *software product representation*, *versioning*, and *representing distinct kinds of knowledge*. Further on, we present issues on feature modeling and evolution (for the last topic see also [17] and [5]).

5.1 Software Configuration Management

In the following some issues concerning Software Configuration Management are presented.

- In current commercial SCM systems *files* and file handling are the basic components, thus, operations like merging, revision etc. are based on files not on objects [4]. *Models* and software products (*instances*) are roughly seen as the same kind of entities namely software programs [5]. These facts demonstrate a main problem: there is no abstract, declarative model of the source code being configured [17]. Each task is done directly on source code and files. In CAI a configuration language is given which provides the means for defining the central aspects of configuration tasks (see Section 3). With such a model, software products could be defined on a concept level, and the instances (e.g. files, source code), which realize a specific application, can be computed by the configuration system.
- In [21] it is mentioned that versioning is mainly based on a directed acyclic graph by using the `is-version-of` relation. In CAI one can represent this aspect in the framework of specialization hierarchies. Here, versions can be described in terms of subconcepts with specializing properties or relations. Thus, also version management of structures, relationships and interfaces can be modeled, which is a further requirement mentioned in [7]. As described in Section 3, not only a model but also an inference machinery is given by CAI, hence, consistent configurations selected from multiple versions can be computed. Thus, the selection of appropriate components is dynamically supervised by the configuration system.
- The CAI methodology is general and generic. Hence, distinct aspects like features, requirements, designs, test cases, task agenda, standard code components and their relations, etc. could be represented (see [21]). These diverse aspects of a domain can be modeled in distinct knowledge bases and can be combined, e.g. by using *strategies* [9], in an integrated system. As an example, in [13] a feature model is described, which includes common and variable aspects of the capabilities of software products. Such a feature model could be used by a configuration system for identifying suitable software and hardware components.

Further open issues which do not have an obvious solution and are still research aspects in both fields, CAI and SCM are:

Combination of CAI methods and SCM techniques The features of SCM, that are well understood are e.g. buildability, versioning on file level, and (partly) workspaces. Their integration with the kernel methods of CAI must be taken into account. Further integration issues for SCM are discussed in [7].

Representing functionality of software modules When assembling software components the problem is to identify suitable components, specify their interfaces, and infer the overall state description of the aggregate. For this task the *functionality of software modules* in terms of states and their dependencies are important. Methods like state charts and automaton can possibly be used for modeling [16]. Further on, techniques like *simulating* totally or partly configured components should be considered.

Product variability In [7] distinctions between permanent variants: variability at construction time and product variability: variability that must be handled by the product are mentioned. The first can be handled by domain modeling. For considering the variability at run time, variable components and their dependencies should be part of the resulting configuration. In this case methods from CAI like *reconfiguration*, and knowledge base evolution should be examined.

Distributed, concurrent work In [7] further on, problems with distributed work on same products are discussed, e.g. connectivity between group members, multi-site, multi-organization. Approaches in CAI which work in this area are named *distributed configuration*. Decisions about distributing artefacts, parallel development, merging of configured components, and propagating of dependencies are studies in this field.

Aspects, which are more related to SCM, are deeply discussed in [4, 7, 21]: Automated change integration and merging of changes; interoperability among configuration management systems; relationship between software architecture and configuration management systems; dealing with building of executable object modules, representing data flow among modules, representing control flow among modules; deployment issues and post deployment phase: distributing software into the field and the maintaining if once there. How these problems can be handled with CAI as a kernel technology is still open.

5.2 Feature Models

Features in terms of customer requirements and system functionality are used to specify products to be developed. Features are hierarchically modeled in tree structures to allow specializations and decompositions. These feature trees are then used to support the configuration task.

A set of hard- and software components has to be combined to fulfill specific customer requirements. The desired functionality within a product line can differ so much that a non-expert might not be able of building a valid composition of given components. Further, the components each have a specific functionality which together build the functionality of the whole system. Speaking of functionality in terms of customer requirements the major difficulty is to choose the components of a system so that they work together and provide the desired system functionality.

A main problem is to map system requirements as a set of features to the hard- and software components actually combined in the developed system. Different components may work together well or may be not. In the latter case they should not be connected to construct a stable system architecture. Further, the union of the components functionality has to provide the desired functionality corresponding to the customer requirements.

The wishes of the end user are evolving with new improvements in technical research areas. Thus, the models describing customer requirements have be able to deal with this. In rather short evaluation cycles new products are introduced into the product catalogue and thus technical requirements and dependencies are getting more complex. Because multiple software components are already present and used for new product development, the integration of a case-based approach seems promising. A *case* would describe an already developed software components, which should be integrated into the configuration process.

The configuration tools *KONWERK* [8] and *EngCon* [1] are examples for structure-oriented configuration tools. They are able to guide a user through the configuration process in order to create a solution tailored to suffice the user specific configuration task. A flow of configuration steps can be modeled as process knowledge and used to enhance the quality of possible solutions. A methodology for using ontologies as domain knowledge input is also provided. This methodology is able of dealing with taxonomic relations, decompositions, artifact specific parameters and constraints between components and their parameters.

Both tools currently do not support feature trees in the form described above. It is in the scope of the *ConIPF* project to evaluate if feature trees can be incorporated into them. Further, a mapping between features and components has to be defined to allow an efficient composition of the desired system.

5.3 Evolution

Reuse of existing components can be enhanced when models of components are present and are used for configuring new products (configurations). But, mostly not only existing components have to be incorporated in new products but some components have to be modified to supply customer requirements. The inclusion of those new components in the model would be an evolution of the model and give the possibility of future reuse of those new components.

Some examples of how evolution can influence the system development task are:

- A new kind of short range radar is developed. This one can observe more of the surrounding environment. Other software modules (drivers) may be needed to make this work properly.
- A CPS system may consist of a set of applications. Usually the significant differences in these systems are the application versions and the composition of sensors and software modules.
- One customer would like to have a combination of 4 yellow and 2 red bar graphs for displaying the range detected by a park pilot system and a short interval tone for making the critical distance audible. Another car manufacturer may wants to have a different display but the same speaker settings. And both are not yet included in the configuration model.

Following entities can be subject of evolution inside the model:

- Inserting new components and/or new features
- Handling distinct dependencies between different versions of components
- Evolving relations between features and components
- Expanding decomposition and taxonomical relations

A further example where evolution can occur is given by changing user requirements during development time. Thus, those user requirements are not fixed in the beginning of system development. For requirements that are foreseen in a configuration model the knowledge-based configuration approach is suitable, because changing requirements could be handled by subsequent configuration sessions. But, if there are requirements which are not part of a model (e.g. intuitively discovered during testing a prototype of the product), this situation should be handled with further methods. One notion is to develop some kind of open, innovative, expandable model that covers most (probably unknown) requirements in a generic way and supports the refinement of the model during the configuration session (compare [15]).

A more economical aspect of evolution is given by a small amount of personal and financial resources which are used for evolution. Evolution of a model is mostly not planned and included in the product development process for a specific customer. For a customer more specific needs are important, like quick specific samples, short development time, etc.

Currently a copy-and-modify-approach exists often 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. While 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.

Requirements for the methodology in respect to evolution are:

Combining of modeling, and configuration: 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 or intrinsic approach, where modeling and configuring are interchanged.

Combining of modeling, configuration and even implementation: 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, for new customer requirements a suitable product is configured 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 - by evolving it. How this can be supported during the implementation phase is an open issue.

Ease of modeling: Because of the combination modeling, configuring, and implementation the modeling task is not done by a specific kind of knowledge engineer, who knows all about the modeling facilities, but the modeling will be done by software developers. Thus, the modeling has to be included in their daily work, and known tooling.

Distributed development: Software development and modeling is done by multiple developers in a distributed fashion. Thus, distributed configuration, distributed modeling and distributed implementation has to be taken into account.

Version management for models: To handle the evolving model some kind of version management for the model (not only for some software modules) has to be realized.

Evolution of existing products: A further question is how new versions are included in existing products which are already in use. In this case methods from CAI like *reconfiguration*, and knowledge base evolution are to be examined.

6 Future Work

Future work will be evaluating the suitability of existing configuration methods for usage in combined hardware/software systems. Therefore a possible integration of feature trees into structure-oriented configuration has to be examined. A mapping between feature trees and product catalogues has to be modeled. Here existing representations e.g. of the *KONWERK* tool [8] can possibly be used since they already provide the hierarchical concepts of specializations and decompositions. Non-hierarchical relations (i.e. requires, excludes etc.) as needed by feature models should be integrated.

Thus, the scope of the *ConIPF* project is to examine the following issues:

- In order to support realistic industrial applications, guidelines will be described for facilitating domain modeling (see [18, 20] for similar approaches).
- A further focus is set to products that can be developed quickly and with little effort. On the one hand an early result can be produced and on the other hand, for those components libraries can be developed for reusing generic software components.
- For modeling, known configuration description languages will be used and possibly further developed for describing specific aspects of software modules, like functionality and state descriptions.
- A technology used for modeling will be Description Logics (DL), which provides a well-defined semantics for basic concept and role (=relation) definitions (for a description of a DL system see [12]). The combination of a DL with CAI will be examined to support inferencing on the concept level, e.g. for automatically classifying new component models in a specialization hierarchy.

REFERENCES

[1] V. Arlt, A. Günter, O. Hollmann, T. Wagner, and L. Hotz, 'Engcon - engineering & configuration', in *Proc. of AAAI-99 Workshop on Configuration*, Orlando, Florida, (1999).

[2] T. Asikainen, T. Soinen, and T. Männistö, 'Representing software product family architectures using a configuration ontology', in *Proc. of 15th European Conference on Artificial Intelligence (Configuration Workshop)*, Lyon, France, (July 21-26 2002).

[3] S. Dart, 'Concepts in configuration management systems', in *Proc. of the 3rd. Intl. Workshop on Software Configuration Management*, Trondheim, Norway, (1991).

[4] J. Estublier, 'Software configuration management: a roadmap', in *ICSE - Future of SE Track*, pp. 279–289, (2000).

[5] J. Estublier, J.M. Favre, and Morat P., 'Toward PDM / SCM: integration?', in *Proc. of the 8. Intl. Workshop on Software Configuration Management*, LNCS 1439, pp. 75–95, Bruxelles, Belgium, (July 1998). Springer Verlag.

[6] 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, San Diego, CA, USA, (August 19-23 2002). Springer Verlag.

[7] K. Frühauf and A. Zeller, 'Software configuration management: State of the art, state of the practice', in *9th International Symposium on System Configuration Management (SCM-9)*, Toulouse, France, (1999).

[8] A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995. in german.

[9] A. Günter and R. Cunis, 'Flexible control in expert systems for construction tasks', *Journal Applied Intelligence*, **2(4)**, 369–385, (1992).

[10] 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, (1999).

[11] J. van Gurp, J. Bosch, and M. Svahnberg, 'On the notion of variability in software product lines', in *Proceedings of the 2001 Working IFIP/IEEE Conference on Software Architecture*, (2001).

[12] V. Haarslev and R. Möller, 'Consistency testing: The race experience', in *Proceedings TABLEAUX'2000*. Springer-Verlag, (2000).

[13] A. Hein, M. Schlick, and R. Vinga-Marting, 'Applying feature models in industrial settings', in *Software product lines - Experience and research directions*, ed., Donohoe P., pp. 47–70. Kluwer Academic Publishers, (2000).

[14] L. Hotz and A. Günter, 'Using knowledge-based configuration for configuring software?', in *Proc. of the Configuration Workshop on ECAI 2002*, Lyon, France, (2002).

[15] L. Hotz and T. Vietze, 'Innovatives Konfigurieren in technischen Domänen', in *S. Biundo (Hrsg.), 9. Workshop Planen und Konfigurieren*, Kaiserslautern, Germany, (1995). DFKI Saarbrücken.

[16] 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).

[17] T. Männistö, *Towards Management of Evolution in Product Configuration Data Models*, Ph.D. dissertation, University Helsinki, 1998.

[18] T. Männistö, T. Soininen, and R. Sulonen, ‘Product configuration view to software product families’, in *Software Configuration Workshop (SCM-19)*, Toronto, Canada, (2001).

[19] 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*, (1997).

[20] 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).

[21] 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.

[22] K. Ylinen, T. Männistö, and T. Soinen, ‘Configuring software products with traditional methods - case linux familiar’, in *Proc. of 15th European Conference on Artificial Intelligence (Configuration Workshop)*, Lyon, France, (July 21-26 2002).