How to Configure a Configuration Management System – An Approach Based on Feature Modeling

Sebastian Wenzel
IBM Business Services GmbH
Wilhelm-Fay-Str. 30-34
Frankfurt, Germany
wenzel.sebastian@de.ibm.com

Thorsten Berger
University of Waterloo
200 University Ave West
Waterloo, ON, Canada
tberger@swen.uwaterloo.ca

Thomas Riechert
University of Leipzig
Johannisgasse 26
Leipzig, Germany,
riechert@informatik.uni-leipzig.de

***Abstract*—The accomplishment of an efficient IT service management is considered a significant success factor in large businesses. Configuration Management (CM) constitutes one of its core disciplines. Off-the-shelf CM systems support the maintenance of the IT by handling the lifecycle of so-called Configuration Items (CIs) and by establishing Change, Configuration and Release Management processes. However, due to the complexity of today's IT infrastructure in large companies, the tailoring of these systems based on concrete stakeholder requirements can become a laborious and error-prone task.**

We present an approach that enables the configuration of a CM system by leveraging variability management techniques stemming from product line engineering. The synthesis and configuration of a feature model is driven by the Common Data Model, a large domain-specific model that describes CIs and their relationships. We show how our feature-based approach can improve the tailoring of CM systems. Furthermore, we expand on its prototypical realization, elaborate on the integration into the requirements engineering process and discuss its applicability based on experiences obtained from a first evaluation.

IT service management; configuration management; feature modeling; requirements engineering

I. INTRODUCTION

During the past decades, the technological innovation of information technology has been the main driving force to achieve a higher level of efficiency and effectiveness within businesses [1]. However, the growing complexity of companies' IT environments has indicated a need for more comprehensive IT management support. One solution of tackling the growing complexity is the introduction of IT service management (ITSM) techniques. ITSM provides a process-centered view on the management of IT infrastructures and aims at assuring the quality of IT services.

One of the most important disciplines that ITSM comprises is Configuration Management (CM) which is responsible for keeping information about the managed IT infrastructure to be managed both up-to-date and accurate. According to Klosterboer [2], the implementation of CM is very difficult to accomplish. Many companies have problems with the realization of CM practices. Especially the tailoring and installation of the CM database and the establishment of change processes present some of the most complicated tasks. It is critical to design a concrete and accurate specification for the CM database that reflects all the data required for ITSM processes.

We were faced with the problem of configuring a CM database as part of an outsourcing project for a company that has to manage a large IT infrastructure with more than 2000 servers. The tailoring of the database, i.e. the creation of its concrete data model, was driven by requirements that had to be elicited from stakeholders. Additionally, the data model of the database had to conform to the *Common Data Model (CDM)*, a domain-specific model from IBM Tivoli that defines types of Configuration Items (CIs) and their relationships.

The manual and indirect tailoring of the database turned out to be very laborious and error-prone: First, the configuration knowledge is elicited indirectly via textual requirements from the customer. Second, the actual configuration has to be carried out by experts with significant knowledge about the database specification, the *CDM* elements and a considerable number of constraints.

In this context, we present a model-driven approach to creating a CM database specification that leverages Feature Modeling [3] techniques. It dynamically synthesizes a feature model that provides different levels of abstraction over the database specification, incorporates CI dependencies as constraints and supports a staged configuration process. In summary, it exposes the structure and configuration options of the database specification more explicitly and provides a more abstract view of it.

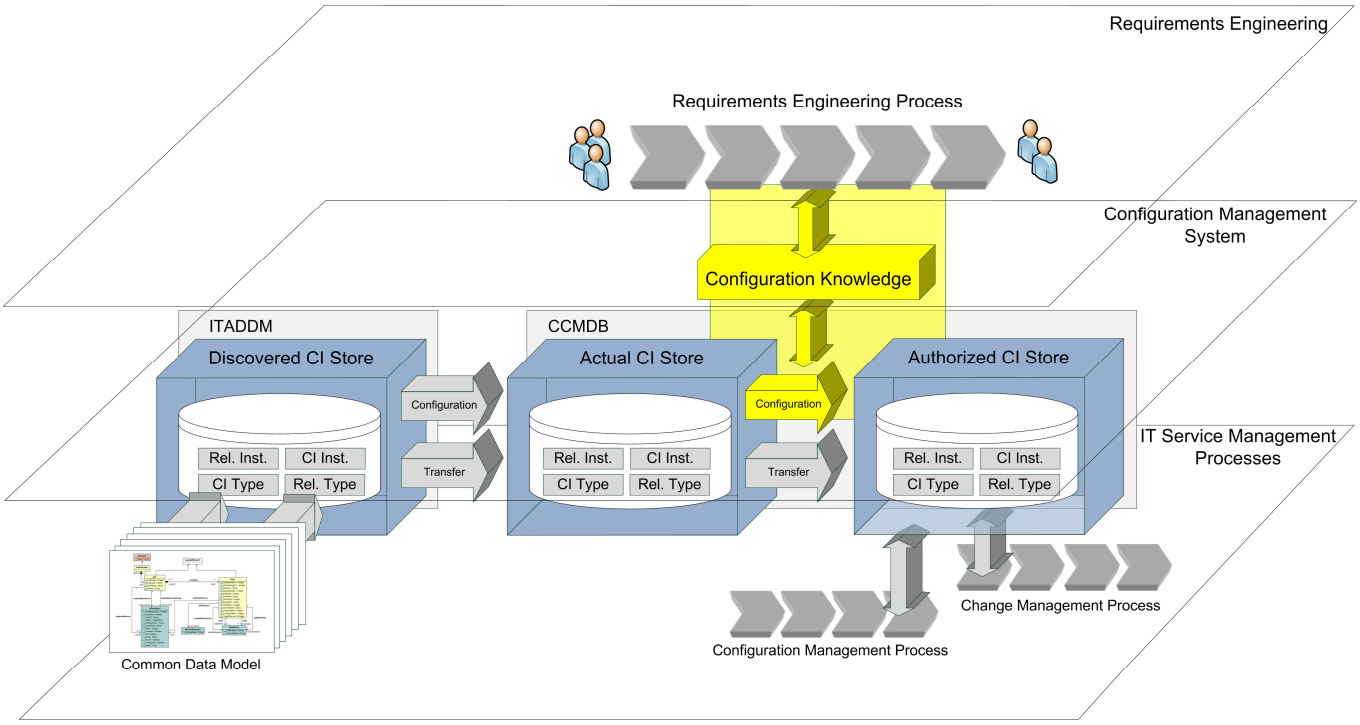

Figure 1. CM system and processes overview

The remainder of the paper is structured as follows: In section 2, we give an introduction to Configuration Management for IT services, describe the *Common Data Model* and portray our concrete problem context. Section 3 presents our approach that traces CM database tailoring back to a feature configuration problem. Section 4 expands on the prototypical realization of a tool relevant for our method and section 5 reports on the evaluation experiences gained. Finally, we discuss relevant conclusions and outline future work in section 6.

II. Configuration Management

CM basically denotes "the process responsible for maintaining information about Configuration Items required to deliver an IT Service, including their relationships. This information is managed throughout the lifecycle of the CI." [4]. For a more comprehensive introduction to CM, including a definition of Configuration Items, we refer to Alison et al. [5] and Lacy et al. [6].

A. System Architecture

Fig. 1 provides a high-level view on the realization of the CM system in our project context as well as the connection to the various service management processes. The system consists of two main parts: *ITADDM*[1] and *CCMDB*[2] [7]. *ITADDM* denotes the *Discovery System* responsible for discovering, collecting and storing information about the IT infrastructure. This information comprises CIs and their relationships and is saved in a database called *Discovered CI Store*.

However, not all the data that has been discovered by *ITADDM* is relevant for IT service management processes. Thus, the information is filtered and transferred into the *CCMDB*. The *CCMDB*, in turn, consists of two logical databases realized in one physical database. These logical databases are named *Actual CI Store* and *Authorized CI Store*. The former one just keeps a subset of the discovered data, but still contains sufficient information that is necessary for the CM system to operate correctly. This information is stored with a high level of detail and is necessary for root cause analysis, but not for the IT management itself. In contrast, the *Authorized CI Store* only keeps CIs and relationships that are subject to change and configuration management processes. This information is essential for a failure-free operation of the IT infrastructure.

Fig. 2 shows an example of how part of the data discovered by *ITADDM* is filtered for its usage in conjunction with IT service management processes. More precisely, the diagram shows parts of the *CI Stores'* specifications, which are sets of *CI Types* and their relationships. The *CI Types* themselves, their attributes and relationships are defined in IBM Tivoli's *Common Data Model*.

B. Common Data Model

The *Common Data Model*[3] is a domain-specific model that describes concepts in the CM domain. According to Tai et al. [8], *CDM* "provides consistent definitions for managed resources, business systems and processes, and other data, and

[1] IBM Tivoli Application Dependency Discovery Manager: http://www-01.ibm.com/software/tivoli/products/taddm/

[2] IBM Tivoli Change and Configuration Management Database: http://www-01.ibm.com/software/tivoli/products/ ccmdb/

[3] http://www.redbooks.ibm.com/redpapers/pdfs/redp4389.pdf

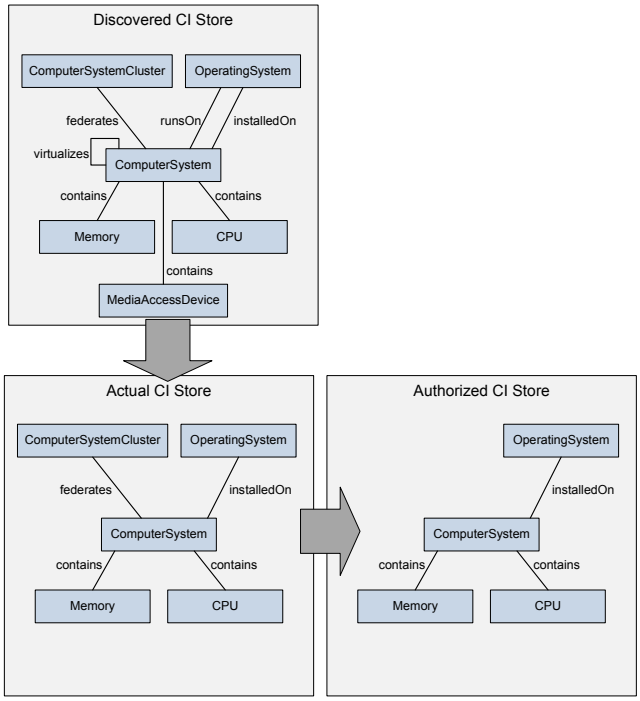

Figure 2. Filtering of CIs among the *CI Stores*

the relationships between those elements". Thus, it can be seen as a domain-specific language rather than just as a data model for the *CI Stores*. In fact, many CM tools from the IBM Tivoli family are built upon concepts as defined in the *CDM*.

Technically, it is modeled in UML2 and contains about 750 classes with attributes as well as 82 named association types (e.g. `contains`, `installedOn`, `virtualizes`). Three UML2 Profiles define stereotypes in order to specify technical tool and data mappings. *CDM* further introduces the notion of *Sections*, which categorize related classes. They are organized hierarchically and each of the 36 *Sections* corresponds to a concrete class diagram.

Classes that represent real-world CIs realize the interface `ConfigurationItem` and are subject to IT service management processes. Thus, they embody the main entities that are to be saved in the *CI Stores*. However, since administrative and meta-information also has to be stored in the *CI Stores*, all classes derived from `ModelObject` can be persisted in the databases. Furthermore, concrete relationships between classes are defined and named according to their corresponding association type. Altogether, almost 1600 unique relationships – defined as associations – exist in *CDM*.

C. CI Store Specification

In order to support the IT service management processes, the stores have to be tailored towards the stakeholders' requirements. Basically, this tailoring comprises the creation of a specification for the CM databases, i.e. for the *Actual* and the *Authorized CI Store*. A specification contains (1) a set of *CI Types* including meta/administrative information and (2) a set of relationships as defined by the *CDM*. Furthermore, a logical hierarchy is introduced, which is based on a specific relation between classes in *CDM*. This hierarchy is defined using a `Parent` attribute in classes, but each parent-child relation is further detailed by a corresponding association. Fig. 3 illustrates the mapping between a store specification and the *CDM*.

In this paper we focus, however, on the specification of the *Authorized CI Store*. Setting up the *Actual CI Store* is not addressed here since it is rather driven by technical aspects than by customer requirements. The mapping and transfer between *Discovered* and *Actual CI Store* is realized by predefined adaptors with the option to define the hierarchy depth.

D. Authorized CI Store

The current process of creating a specification for the *Authorized CI Store* can be characterized as follows:

Elicitation of requirements from the customer: Based on the current specification of the *Actual CI Store*, requirements reflecting the necessary *CI Types* and relations have to be elicited from the stakeholder. Our project, for example, comprised more than 700 requirements [9].

Analysis of requirements: CIs, meta/administrative information and relationships that are to be transferred from the *Actual* to the *Authorized CI Store* have to be identified on the basis of the elicited requirements and the *CDM*. In practice, requirements are currently mapped to *CDM* elements in Microsoft Excel spreadsheets.

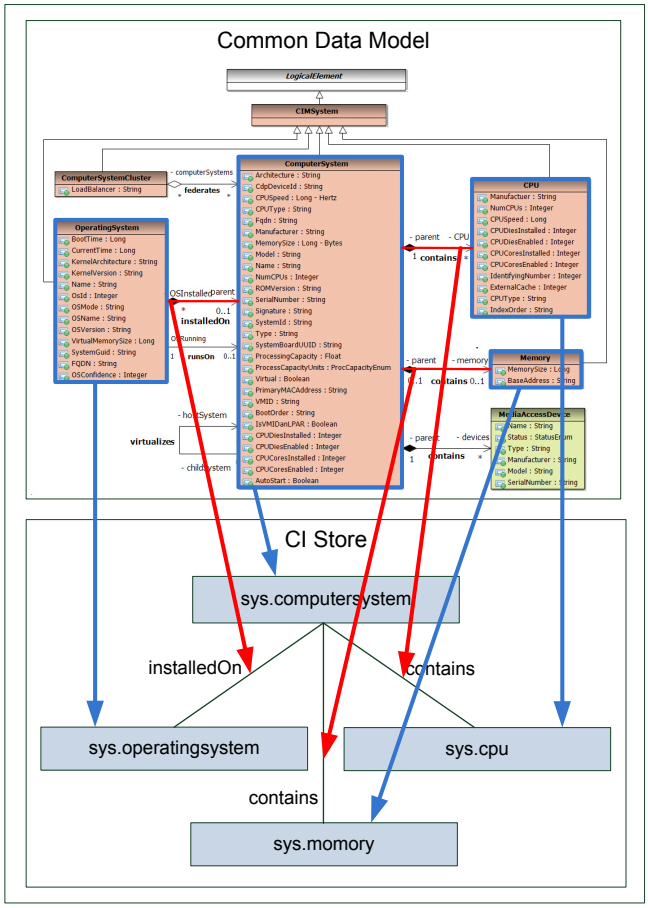

Figure 3. Mapping between *CDM* and *CI Store* specifications

Applying the specification: Finally, the specification has to be applied to the *Authorized CI Store* by entering all elements into a web configuration interface. Furthermore, the former hierarchy of the *Actual CI Store* has to be retained or recreated.

In its current form, the process turns out to be quite ineffective for the following reasons: First, profound knowledge about possible *CI Types* and relationships is expected from the stakeholders. Second, available elements are limited by the current specification of the *Actual CI Store*. Third, consistency between *CI Types* and relationships is difficult to maintain. Furthermore, terminology and translation issues concerning the textual requirements occur.

III. Feature Model Synthesis and Configuration

In order to bridge the gap between the (1) *Actual CI Store* specification, the definitions in the (2) *CDM* and the implicit (3) *configuration knowledge* of the stakeholders, we introduce an approach based on Feature Modeling and Feature Model Configuration [10,11] techniques as known from Software Product Line Engineering [12,13].

We try to reduce the disadvantage of the current method by providing a simplified and more coherent view on the *Actual CI Store specification* in form of a feature model. This model provides a higher level of abstraction for the selection of relevant *CI Types* and relations that are essential for the stakeholders. The goal is to obtain a specification for the *Authorized CI Store*.

Our approach (cf. Fig. 4) consists of three main steps:

- Feature Model Synthesis
- Feature Model Configuration
- *Authorized CI Store* Creation

The approach facilitates the configuration of the feature model on different levels of abstraction. On the highest level, the presented view is intended to be simpler and easier to understand for stakeholders without specific knowledge about the underlying *CDM*.

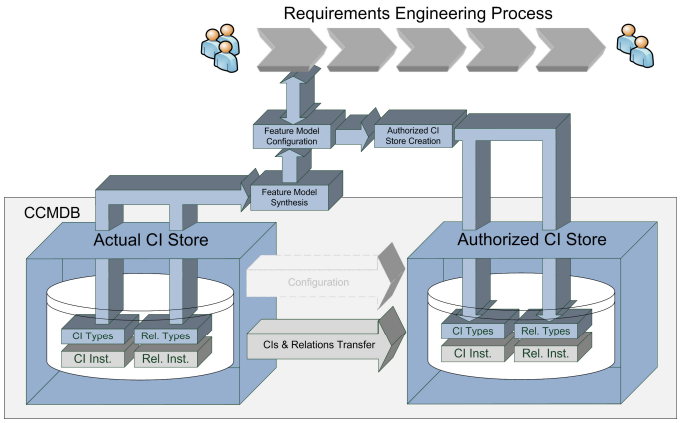

Figure 4. Feature model synthesis and configuration steps

A. Feature Model Synthesis

The first step of our approach deals with the dynamic creation of a feature model. The model is based on the current specification of the *Actual CI Store* and allows an adjustable representation of the prospective *Authorized CI Store*. We introduce four types of features:

- *Diagram Concept features*: root feature describing the underlying logical data model (i.e. the scope of the feature model).
- *CDM Section features*: describing the highest abstraction level of the *CDM - Sections*.
- *CI Type features*: representing *CI Types* contained in the logical data model.
- *CI Relation features*: representing relations between *CI Types*.

The feature model is built in three stages. In each stage features of different types are added to the model. All of them are optional, we didn't need to introduce mandatory features or mutual exclusions. An example of the feature model levels, created by the described procedure, is presented in Fig. 5. The synthesis stages are as follows:

The first stage consists of two steps: the creation of the *Diagram Concept feature* (e.g. *Actual CI Store*) and the creation of *CDM Section features* (e.g. `Administration Section` or `ComputerSystem Section`). These features are either child features of the *Diagram Concept feature* or of other *CDM Section features*. This stage of feature model synthesis is initially executed once for all projects.

The second stage of the synthesis comprises the creation of *CI Type features* corresponding to *CDM Sections*. The parent feature of these features is a *CDM Section feature*. This stage is automatically executed on the basis of the *CDM Section – CI Type* mapping and the *Actual CI Store* structure. For instance, the *CI Types* `CPU` and `ComputerSystem` belong to the *CDM* Section `ComputerSystem Section` and are part of the *Actual*

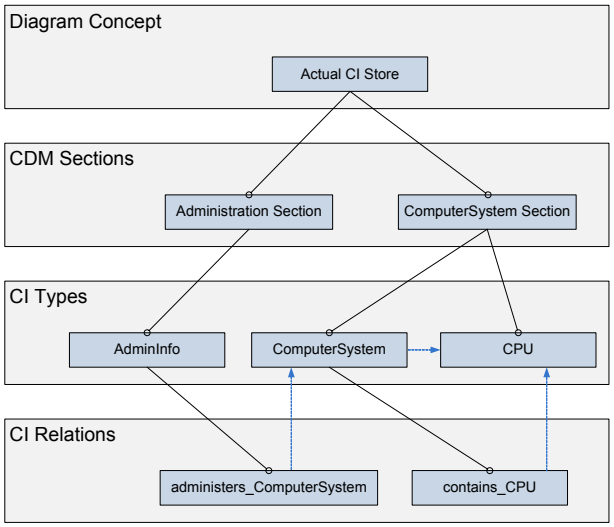

Figure 5. Levels of the feature model

CI Store; thus, they are added to the feature model as children of the `ComputerSystem Section` feature. If the added feature represents a *CI Type* which is not labeled in the *Actual CI Store* as top-level, a constraint pointing to the feature of its parent *CI Type* in the logical *Actual CI Store* hierarchy is added to the feature model.

The third stage of the synthesis creates *CI Relation features*. This stage is automatically executed on the basis of the *Actual CI Store* structure. Those CI relations are added to the feature model, for which the source *CI Type* and the target *CI Type* exist in the feature model. They are added to the model as children of the source *CI Type* feature. Furthermore, for each *CI Relation feature*, a constraint pointing to the target *CI Type feature* is added to the feature model.

B. Feature Model Configuration

The second step of the feature-based approach comprises a kind of staged configuration of the synthesized feature model. This configuration is performed by the stakeholders in order to select features directly and, thus, to omit or at least reduce the error-prone elicitation of requirements. We also leverage the choice propagation functionality in feature model tools for the purpose of assuring relationships, which have been added as extra constraints to the feature tree.

In summary, this step extends the current requirements engineering that is carried out for gaining configuration knowledge from stakeholders (cf. Fig. 1 and Fig. 4).

The configuration of the feature model is executed in three stages. The initial feature model is created in the first Feature Model Synthesis stage. In the **first configuration stage** the *CDM Sections* relevant for the stakeholder are selected. After that, the second feature synthesis stage is performed and the *CI Type features* corresponding to the selected *CDM Sections* are loaded. This allows the execution of the **second configuration stage** in which the stakeholders select the required *CI Types*. Based on the selected *CI Types*, the third feature model synthesis stage is executed and *CI Relation features* are added to the feature model. The **third configuration stage** is performed on the basis of the CI relations added in the third synthesis stage. The CI relations necessary for the stakeholder's IT infrastructure are selected, resulting in the final configuration of the feature model. This configuration is the basis for the specification of the *Authorized CI Store*.

C. Authorized CI Store Creation

The last step of our feature-based approach constitutes the creation of the *Authorized CI Store* specification. This specification is generated on the basis of the final configuration of the feature model (see Fig. 6). The *Authorized CI Store* specification is subdivided into two parts: a list of *CI Types* selected by the stakeholders and a list of selected CI relations between those selected *CI Types*. These specification lists are saved in database-specific XML format. Based on these XML files, the *Authorized CI Store* logical hierarchy is created in the CM system.

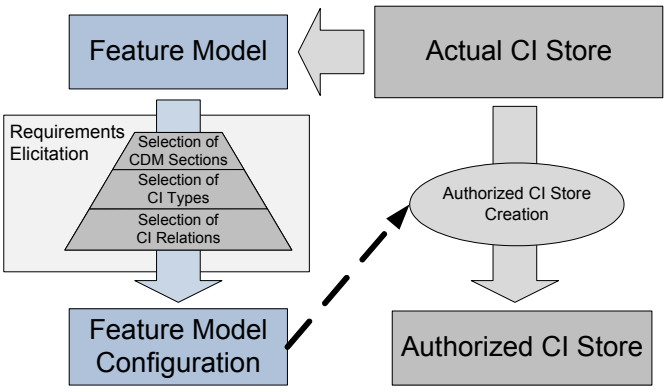

Figure 6. Requirements elicitation-based feature model configuration

IV. Prototypical Realization

We have realized our approach as an Eclipse plug-in, since we wanted to be able to embed it with other tools from IBM Tivoli and since we chose to integrate with FMP[4] [14] as a Feature Modeling tool. FMP turned out to be the most appropriate one for our purpose. It is available as Open Source software, supports basic Feature Modeling with extra constraints, staged configuration and choice propagation. Cardinalities are also supported in FMP, but were not necessary for our approach.

In summary, our plug-in extends FMP, realizes the feature model synthesis and staged configuration as well as it provides adapters for the *Actual CI Store* in order to obtain the current specification.

As described in section 3, the synthesis procedure creates a feature model in FMP by leveraging the structure of *CDM Sections* and loading the current *Actual CI Store* specification. We load subsections just on demand since we faced performance issues[5] when creating the whole feature model from a large *Actual CI Store* in one step. Our plug-in adds relationships as subfeatures and adds binary constraints in FMP in order to support choice propagation. Since there exists another logical hierarchy between CIs (cf. section 2.3), additional constraints representing it are introduced into the feature model. For further implementation details such as naming rules, feature ID definition for traceability reasons, or constraint realization, we refer to [15].

Fig. 7 illustrates the feature model view, especially with the `ComputerSystem` and `OperatingSystem` sections. Fig. 8 shows a list of constraints of the feature model presented on Fig. 7. For instance, constraints between the features `SYS.COMPUTERSYSTEM` and `SYS.OPERATINGSYSTEM` and between relations and whose target *CI Types*.

[4] Feature Modeling Plug-in: http://fmp.sf.net

[5] These are known issues owed to FMP's meta-modeling and just-in-time reasoning capabilities.

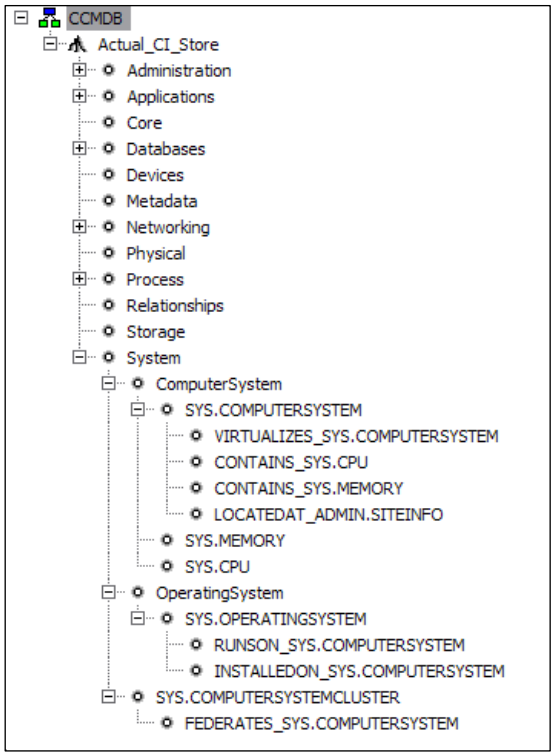

Figure 7. Feature model example

V. Evaluation

We evaluated our approach and the realized prototype in a small-scale setup with some colleagues. Although they did not represent stakeholders, they were familiar with customer projects. Their experience with *CDM* and the CM system ranged from deep to no experience at all with *CDM*.

Based on the goal of our work, we wanted to know (1) if the synthesized feature model provides a simplified view on the *CDM*-based *Actual CI Store* specification, (2) if our approach speeds up creating the specification and (3) how the tool would be accepted by stakeholders.

Accordingly, we gave a quick introduction into the approach and the tool. Thereafter, the participants performed a test scenario and created an *Authorized CI Store* specification. Finally, we asked them to fill out a questionnaire with nine questions.

We received very positive answers from the participants (for details cf. [15]): (1) The tree-based navigation and the support of constraints within the configured feature model were regarded as a significant advantage. (2) All participants also mentioned the time-saving potential. However, some of them also pointed out that time saving depends on the project size, i.e. the difference could be marginal for smaller projects. (3) Furthermore, participants agreed on the potential to increase customer acceptance, since less knowledge about *CDM* is necessary when using the tool. However, experts might miss some additional information that is intentionally omitted in the feature model.

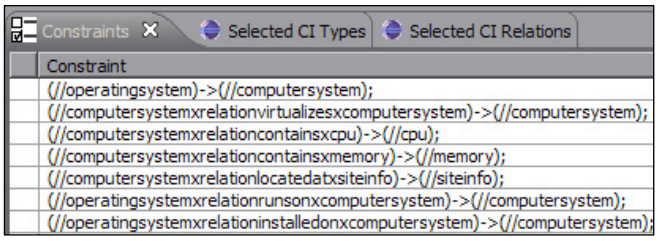

Figure 8. Feature constraints

In summary, the feature-based approach met with favor and appreciation participants of the evaluation. Especially the convenience and the focus on the stakeholder's interests and goals were emphasized very positively.

VI. Related Work

Although our approach is – to a certain degree – specific to the *CDM*, we depict some work that, in a broader sense, deals with variability in data models or data specifications by using Feature Modeling techniques.

Usually, feature models are used in various kinds of domain analysis. However, there is some work that uses feature models to provide a tree-oriented-view on fine-grained data with many relationships. Czarnecki et al. [16] elaborate on the expressiveness of feature models compared to rich ontology modeling techniques. In their work, they also provide a case study that synthesizes a feature model from a domain-specific ontology, i.e. they accomplish a more abstract view on domain data.

Barthold et al. [17] address the problem of variability in data models that appears in conjunction with software variability. They propose an approach to represent and manage data variability in entity models. Their approach is based on adapters that provide a specific view on the database, i.e. they, for example, omit entities or relations that are not relevant for a certain feature.

Some work that deals with mappings between UML diagrams and feature models comprises for example the following: Braganca and Macada [18] provide a mapping between features and the elements of Use Case diagrams. They establish a model-driven approach to deriving a concrete Use Case diagram that represents one product of a product line based on the feature configuration. Furthermore, Czarnecki and Antkiewicz [19] treat class and activity diagrams as templates containing variability in order to derive concrete model instances. They also deal with checking the consistency of derived UML diagrams.

VII. Conclusions and Future Work

In this work, we have developed a feature-based approach to creating a data specification for a *CI Store*. We dynamically synthesize a feature model that represents such specifications on a higher level of abstraction and provides a simplified view that is more stakeholder-oriented. This model is configured in three stages in order to obtain a concrete *CI Store* specification. The aim of our approach was to reduce the gap between

stakeholder's implicit configuration knowledge and the complex *Common Data Model's* definitions of CIs and relationships.

More precisely, we defined a mapping between features and *CDM* elements that exploits structural characteristics in order to obtain a hierarchical feature tree. Further *CDM* relationships are incorporated as extra constraints of the feature tree. We have realized the approach as a tool prototype and have performed a first, small-scaled evaluation.

However, there is definitely room for improvement in this field and several enhancements to the method are possible. The current focus was on providing a general view for all stakeholders on the complex *CDM*-based specifications. Stakeholder may be even more enabled to configure this complex data model by using hierarchically structured feature models that are tailored towards particular groups. View integration and derivation with feature models, as proposed in [16], could provide interesting opportunities. Concerning the actual configuration by the stakeholders, an increase of the number of stages would also be possible. Furthermore, the synthesized feature model could be extended with additional features that reflect supplementary meta information, as requested by some evaluation participants.

Another reason for extending the approach lies in the conceivable evolution of *CI Stores*. When IT service management processes change, the modifications have to be reflected in the *CI Store* specification as well.

ACKNOWLEDGMENT

We would like to thank our colleagues from the IBM Service Management Department, especially Claudia Dahl and Jürgen Pfaffenberger for supporting this work, as well as the evaluators of the prototype.

REFERENCES

[1] M. Khosrow-Pour and M. Khosrowpour, Cases on Information Technology And Business Process Reengineering, IGI Publishing, 2006.

[2] L. Klosterboer, Implementing ITIL Configuration Management, IBM Press, 2008.

[3] K. Czarnecki and U.W. Eisenecker, Generative Programming. Methods, Tools and Applications: Methods, Techniques and Applications, Addison-Wesley Longman, 2000.

[4] Office of Government Commerce, "ITIL V3 Glossary: Glossary of Terms, Definitions and Acronyms," 2009.

[5] C. Alison, A. Hanna, C. Rudd, I. Macfarlane, J. Windebank, and S. Rance, An Introductory Overview of ITIL V3, The UK Chapter of the itSMF, 2007.

[6] S. Lacy and I. Macfarlane, Service Transition, The Stationery Office Ltd, 2007.

[7] H. Madduri, S.S.B. Shi, R. Baker, N. Ayachitula, L. Shwartz, M. Surendra, C. Corley, M. Benantar, and S. Patel, "A configuration management database architecture in support of IBM service management," IBM Syst. J., vol. 46, 2007, pp. 441-457.

[8] L. Tai, R. Baker, E. Edmiston, and B. Jeffcoat, IBM Tivoli Common Data Model: Guide to Best Practices, IBM International Technical Support Organization: http://www.redbooks.ibm.com/abstracts/redp4389.html (2009.06.10), 2008.

[9] C. Dahl, Configuration Management: System Requirements Specification, IBM Corp., 2007.

[10] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged Configuration Using Feature Models," Software Product Lines, 2004, pp. 266-283.

[11] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, and C.U.P.P.S.E. INST, Feature-oriented domain analysis (FODA) feasibility study, The Institute, 1990.

[12] K. Pohl, G. Böckle, and F.V.D. Linden, Software Product Line Engineering. Foundations, Principles, and Techniques., Springer, Berlin, 2005.

[13] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," Inf. Softw. Technol., vol. 49, 2007, pp. 717-739.

[14] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature Modeling Plug-in for Eclipse," In proceedings of the Workshop on Eclipse Technology eXchange, 2004, pp. 67-72.

[15] S. Wenzel, "How to Configure a Configuration Management Database – An Approach Based on Feature Modeling," Diploma Thesis, University Leipzig, 2009.

[16] K. Czarnecki, C.H.P. Kim, and K.T. Kalleberg, "Feature Models are Views on Ontologies," Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society, 2006, pp. 41-51.

[17] J. Bartholdt, R. Oberhauser, and A. Rytina, "An Approach to Addressing Entity Model Variability within Software Product Lines," Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on, 2008, pp. 465-471.

[18] A. Braganca and R.J. Machado, "Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines," 11th International Software Product Line Conference, 2007. SPLC 2007, 2007, pp. 3-12.

[19] K. Czarnecki and M. Antkiewicz, "Mapping Features to Models: A Template Approach Based on Superimposed Variants," Lecture Notes in Computer Science, vol. 3676, 2005, p. 422.