2011 15th International Software Product Line Conference

metadoc Feature Modeler

A Plug-in for IBM Rational DOORS

Anil Kumar Thurimella
Harman International & TU Munich
Karlsbad, Germany
Anil.Thurimella@gmail.com

Dirk Janzen
metadoc GmbH
Birkenfeld, Germany
dirk.janzen@metadoc.de

Abstract—**Extending commercial tools for variability management is the problem addressed in this paper. IBM Rational DOORS [1] is a well accepted tool in the area of requirements engineering and supports an Application Programming Interface (API) called DOORS eXtension Language (DXL). In this contribution we present a way of extending DOORS for product line requirements engineering based on a DXL plug-in called *metadoc feature modeler.* Established concepts in the community such as feature diagrams, instantiation of feature models, automatic deficit analysis and issue-based variability are introduced into DOORS. These concepts are abstracted using the DOORS meta-model. As DOORS is directly extended using new components, no special synchronizers are required between feature modeling environment and DOORS (which is a difference with respect to traditional feature modeling tools). We share our experience in using the feature modeler at a medium-sized company.**

Keywords-feature modeling;requirements engineering;IBM Rational DOORS;requirements reuse

I. INTRODUCTION

Clements and Northop define a *Software Product Line* (SPL) as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. *Software Product Line Engineering* (SPLE) [2] is the scientific discipline that deals with SPLs.

SPLE includes two higher-level processes: domain engineering and application engineering. Both processes perform same activities (e.g. requirements engineering, design and testing) but for different purposes. *Domain engineering* focuses on the creation and maintenance of assets. *Application engineering* focuses on development products for individual market-segments by instantiating the assets.

Variability is introduced in SPLE as an abstraction to deal customization and reuse of artifacts during development of SPLs. Here, documentation and communication of variability is done based on variability models.

In product line requirements engineering, higher-level properties of a SPL are represented and communicated to various stakeholders using features. Here variability is represented using feature models [3], which provide a hierarchical tree-based representation of features in terms of variants and invariants of a SPL. Stakeholders perform feature modeling based on computer-supported tools. Sound tool support is required to enable feature modeling in an industrial environment. Therefore, professional tool support for feature modeling is the higher-level problem addressed in this contribution.

IBM Rational DOORS is one of the well accepted (with more than 100,000[2] end-users worldwide) commercial tools within the requirements engineering industrial community. There is no support for variability management (e.g. feature modeling and instantiation of feature models) within the commercial DOORS. Therefore, to support variability extensions for DOORS are required. Using DOORS eXtension Language (DXL) [6], the tool offers a possibility to develop plug-ins internally. This contribution presents a way of introducing well established feature modeling concepts from the product line engineering community into DOORS based on a DXL plug-in. Previously, Schmid et al. [7] presented a similar view of extending professional requirements engineering tools (e.g. DOORS) for SPLs.

1.1 Pros and cons of DOORS integrations

There are two fundamentally different approaches to support feature modeling for DOORS. The first approach is to extend DOORS functionality (e.g. by using DXL). Another, approach is to support DOORS integrations with existing feature modeling tools. In the following we present disadvantages of integrating existing feature modeling tools with DOORS. We also present advantages of DOORS integrations.

There exists feature modeling tools from the scientific community (e.g. Requiline [4], EMF feature modeling [8]) as well as from tool vendors (e.g. pure::variants [5]). These conventional tools should be integrated within tool infrastructures of companies. Let us consider a case in which a

[1] **D**ynamic **O**bject-**O**riented **R**equirements **S**ystem (DOORS) is currently developed, supported and marketed by IBM [20]. DOORS product line supports a family of solutions for requirements management and is a higher-level tool in the IBM Rational tool suite.

[2] The number of users is based on information communicated from the tool vendor in a conference. Hull et al. also describes DOORS as a prominent requirements management tool used by tens of thousands of users [21].

978-0-7695-4487-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SPLC.2011.17

company maintains its requirements in a DOORS infrastructure. Usage of a conventional feature modeling tool developed in a platform other than DOORS, creates a tool integration problem between DOORS and the corresponding feature modeling tool. In this case, synchronizers are used (e.g. pure::variants DOORS synchronizer [11]) to support integration between DOORS and the feature modeling tool. The synchronizers frequently import and export information (regarding feature models, instantiated features and requirements) between the feature modeling tool and DOORS. Here, information is stored and maintained redundantly (within DOORS database and the database of the feature modeling tool). Information in the feature modeling tool should be maintained consistently with information in DOORS database. In addition, the company needs to invest additional efforts for support and server maintenance of the feature modeling tool.

There are also advantages of integrating existing feature modeling tools into DOORS. Feature modeling tools can also support functionality that is not within the scope of DOORS. In this case, DOORS integration is helpful to reuse the available functionality. For example, let us consider the case of pure::variants, which supports both feature models and family models. The family model concept enables definition of components as well as code generation. Both component modeling and code generation are not within the scope of DOORS. Here, integrating DOORS and pure::variants allows companies to couple requirements to family models of pure::variants. Another advantage is that existing professional feature modeling tools could be used as generic solutions (are not limited to DOORS) that could be integrated with several requirements engineering tools (e.g. RequisitePro [26]).

Considering the disadvantages of DOORS integrations for feature modeling and our intent to focus on DOORS, we contribute a DXL plug-in to support feature modeling.

1.2 Requirements for the feature modeler

Within the scope of this paper, our contribution addresses the following requirements.

R1: Support for feature trees. As a part of this requirement, we plan to support feature trees within DOORS. From a requirements modeling prospective, DOORS supports text-based requirements engineering using concepts such as structured/formalized documents, templates and tabular representations. No explicit graphical requirements modeling is supported within DOORS. Feature modeling approaches (e.g. FODA [3], FORM [22] and extensions from Rebisch et al. [23]) use graphical notations. Therefore, graphical modeling concepts should be introduced in DOORS to construct and maintain feature trees. The concept should work in conjunction traditional text-based requirements engineering concepts of DOORS. There exist approaches based on textual templates (Schmid et al. [7], Eriksson et al. [18] and Buhne et al. [14]) to model variability modeling in DOORS. We differ from these researchers by focusing on graphical trees for feature models.

R2: Instantiation of feature models. In the context of application requirements engineering, instantiation of feature models should be enabled within DOORS. In particular, end-users should be able to select features for individual products.

R3: Automatic deficit analysis. In domain engineering, during the creation of feature diagrams there is a possibility that the end-user of a feature model creates invalid relationships between features. Similarly in application engineering, an end-user could select an invalid subset of features for a new product. Therefore, automatic deficit analysis should be supported within DOORS for feature models (constructed as a part of R1) in domain engineering and for the instantiated features (obtained by addressing R2) in application engineering. In particular, the following sub-requirements should be implemented using DXL:

- R3.1: Feature models should be checked for redundancies (same concept modeled twice in the model), anomalies (possible instantiations lost due to improper relationships between features) and inconsistencies (conflicting relationships in the model).
- R3.2: Instantiated features should be checked for validity of the obtained subset with respect to missing features and wrongly selected features.

For this purpose, we use the deficits specified by von der Massen and Lichter [19] as well as von der Massen [16]. Researchers such as Padmanabhan and Lutz [28] and White et al. [27] also emphasize automated checks on feature models.

R4: Issues modeling for features. Product line requirements engineering involves collaboration between stakeholders from various backgrounds. For example, during the construction of feature models domain experts perform decisions collaboratively on the identification of variation points. Similarly, during instantiation of feature models domain and application engineering stakeholders collaboratively make decisions on the selection of features for individual products of a product line. Therefore, there is a need to support collaboration concept within the feature modeler which is useful both in domain engineering and application engineering. Issue-based variability [17, 29] is a recent advancement in the product line engineering community which supports stakeholder collaboration using issue modeling [30]. Based on the methodological guidelines of using issues for variability [17, 29], issue modeling should be supported in the DOORS feature modeler.

The remainder of this paper is organized as follows. Section 2 gives an overview of DOORS. Here relevant functionalities of DOORS that are useful for SPLs are presented. To address the requirements described above (R1-R4), we developed a plug-in using DXL to contribute the *metadoc feature modeler*[3]. The higher-level design concepts and functionalities relevant for end-users are covered in Section 3. We share our experience in using the feature modeler at a company that produce home appliances for global markets. A summarized experience is presented in section 4. Related work is discussed in section 5. Furthermore, the paper is concluded in section 6.

[3] metadoc feature modeler is a commercial plug-in for DOORS, which is currently being developed and supported by metadoc GmbH.. The plug-in is compatible and available for DOORS client versions 9.1 and 9.2.

This paper includes an appendix, in which feature modeling concepts are explained.

II. IBM RATIONAL DOORS FOR SPLS

DOORS database is hierarchically organized based on folders and projects. A *folder* is similar to a folder used in a Windows operating system. A *project* is a folder with a higher-level significance. For example, a customer project could be a DOORS project.

The project concept of DOORS is useful to realize domain requirements engineering and application requirements engineering. For example, in the context of domain engineering, feature models and corresponding requirements specifications are maintained within a platform project. The instantiated features and requirements are documented in customer projects specific to application engineering teams.

The requirements are documented in a formal module, which is similar to a structured document. A *formal module* supports tabular representation for requirements with rows representing objects and columns representing attributes. An *object* contains textual description and attributes values that are useful to specify a requirement. A requirement is usually specified using one or more instances of an object.

Objects within a formal module are hierarchically structured based on sections and sub-sections. The hierarchical structure of objects within a module is useful to model hierarchy within feature modeler.

Traceability is supported using link modules. A *link module* contains several links. A link is useful to connect two objects. A typical DOORS database consists of several formal and link modules. The modules are contained within folders and projects.

Views and filters are supported at the level of a formal module. These concepts are useful to create different views on feature models for different stakeholders. For example, a marketing manager would not be interested in fine grained features/requirements. In this case, a special view on a feature model could be defined for the marketing manager.

Snapshots of a project, folder or a module are captured to form a *baseline*. In a requirements engineering practice, baselines are created regularly. Baselines are also created as a part of contracts between various companies. For example, requirements that are agreed between a manufacturer and a supplier are captured within a specific baseline. To support systematic exchange of requirements between various companies (e.g. between manufacturer and supplier), exchange interfaces are used.

In DOORS discussions are supported on objects as well as on formal modules. From the scientific community, issues modeling approaches have been used to support structured discussions between stakeholders. For example, Issue-Based Information Systems (IBIS) [15] supports discussions between stakeholders based on issues. In particular, stakeholders add proposals and arguments to the issues to perform decision collaboratively. In this contribution, we tailor DOORS discussions to support IBIS scheme on the feature models (see requirement R4).

DXL provides APIs to programmatically access the database and to extend DOORS functionalities. The metadoc feature modeler presented in this contribution was developed as a DOORS plug-in using DXL. The feature modeling plug-in tailors the DOORS functionalities that are required in real projects such as views, filters, baselines, exchange and discussions for SPLs.

DOORS uses the client-server architecture. Here, each of the distributed stakeholders uses a client, which is connected to the server. The metadoc feature modeler is installable and accessible from a normal DOORS client. Therefore, no special modifications are needed on the DOORS server.

III. THE METADOC FEATURE MODELER

To enable feature modeling within DOORS, we implemented a user interface using DXL. A screenshot for the user interface is presented in Figure 1. A feature model is visualized as a tree (see R1), which is expandable and collapsible vertically. Feature relationships and constraints are adjustable on the windows available on the right hand side of the user interface.

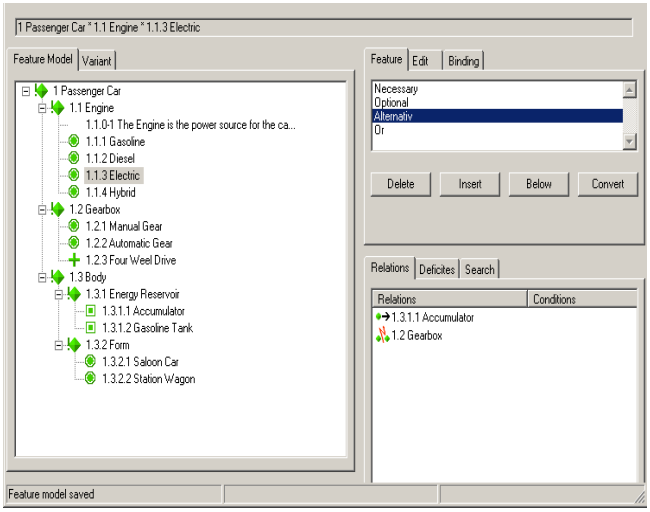

Figure 1. A screenshot of the metadoc feature modeler, with the passenger car feature model represented in the form of a tree.

We present usage scenarios of the feature modeler based on an example from the automotive domain. A Passenger Car product line is presented in the screenshot. In the feature tree, feature relationships and constraints are distinguished based on a set of graphical notations (see Table 1 for details). The first eight notations in the table are used during construction of models. The last two icons of the table are additionally used during instantiation of feature models. The passenger car feature model is also presented as a conventional feature diagram in Figure 2.

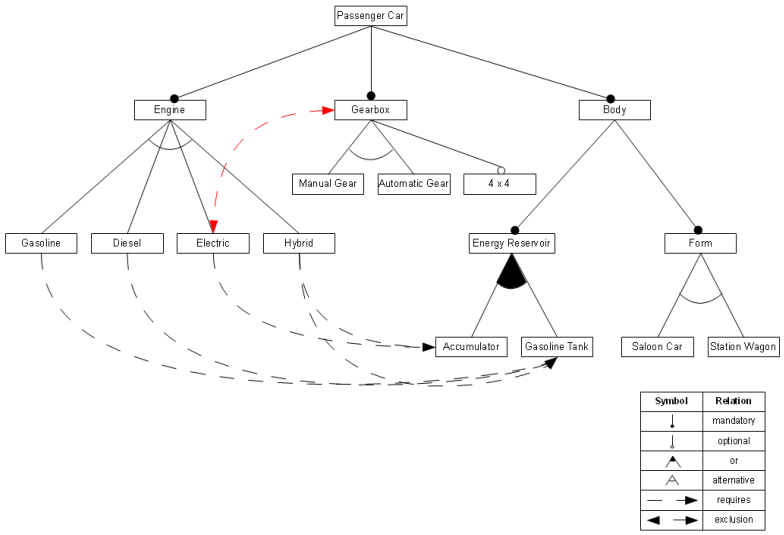

Figure 2. Feature diagram from the passenger car product line. Note that the model contains deficits.

TABLE 1. NOTATIONS USED IN THE METADOC FEATURE MODELER.

Notations used in the feature modeler	Modeling concept
	Mandatory
	Core feature. The exclamatory mark signifies that it is mandatory for all possible instantiations.
	Optional
	Alternative
	Or
	Exclude
	Requires. The direction of arrow shows the direction of implication.
	Alternative selected. The red plus sign represents that the feature is selected in the product instantiation.
	Alternative deselected. The blue minus sign represents that the feature is excluded in the product instantiation.

The Passenger Car (see figures 1 and 2) is decomposed into three mandatory features: Engine, Gear box and Body. The three features are mandatory for all product instantiations (i.e. core features). The Engine is composed of the four alternatives Gasoline, Diesel, Electric and Hybrid. The Gear box has also two alternatives, Manual or Automatic and has an optional feature 4x4. The body is based on Energy Reservoir and Form. The Energy Reservoir contains Accumulator and Gasoline Tank that are modeled using Or. Form is modeled using the two alternatives Saloon Car and Station Wagon. Constraints are also represented within the feature model. For example, Electric and Hybrid requires Accumulator. Another constraint, Electric excludes Gearbox is also modeled within the diagram.

Feature modeler is based on a meta-model, which is elaborated in section 3.1. The domain engineering activities supported by the modeler are presented in section 3.2 and application engineering activities are covered in section 3.3. Limitations of the feature modeler are described in section 3.4.

3.1 The meta-model behind the feature modeler

To present a vision for product line requirements engineering within DOORS a meta-model is proposed (see Figure 3). The meta-model has two layers. The concepts shown as a part of Standard DOORS package are traditional concepts supported by DOORS. The classes represented as a part of metadoc-FM package are new concepts introduced within DOORS.

Tree view. The concepts represented using the stereotype *<<Feature modeler>>* are the special concepts introduced by the plug-in. The Tree View supported by the feature modeler is an abstraction for graphical feature tree in DOORS. For example, the screenshot presented in Figure 1 is an instance for the Tree View, which is generated from the combination of a Formal module (with the stereotype <<Feature modeler>>) and a Link module (the stereotype <<Feature modeler>>).

DOORS support views on each formal module, which is an inbuilt functionality within DOORS. Our tree view concept combines information from a formal module and a link module and is not supported within standard DOORS. Therefore, a tree view in a feature modeler is different when compared with a standard DOORS view.

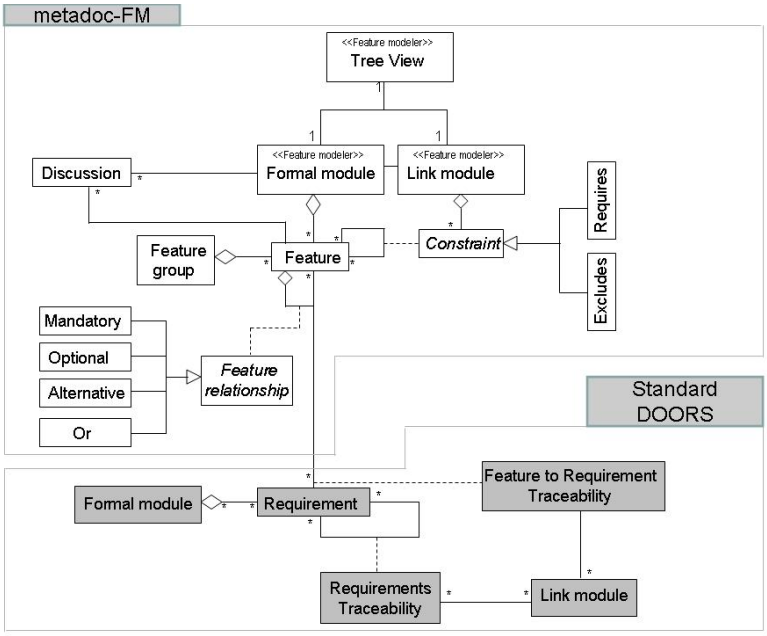

Figure 3. The DOORS meta-model behind the metadoc feature modeler is represented as a UML class diagram. Concepts introduced for feature modeling and discussions for variability are contained in the metadoc FM package. Remaining classes with filling that are contained in the Standard DOORS package are conventional requirements concepts**.**

A project can have several instances of a feature trees (tree views). Instead of having one big feature model, the tool offers a possibility to have several small and interconnected feature models with each small feature model stored in a different metadoc formal and link modules. We call this as a *multi-model environment*.

Features and relationships. On the user interface of the feature model, features are created hierarchically. Instances of Feature are contained in the Formal module of feature modeler. The hierarchical decomposition of features is supported and is modeled using *Feature relationship*, which could have instances Mandatory, Optional, Alternative and Or. Features having a same parent and the same feature relationship are grouped using a Feature Group.

Each feature within the diagram has a section ID. For example, Manual Gear belongs to the section 1.2.1. Based on this concept, hierarchy of a feature model is mapped to hierarchy within a DOORS formal module. A feature can be enriched several documentation points within its subsection. For example, the engine feature in section 1.1 (see Figure 1) is enriched with a description in its subsection.

Constraints. Interdependencies between features are expressed using Constraint concept, which could be Requires and Excludes. These relationships are supported from tree view and the link information is stored within a Link module of feature modeler.

Discussions. Traditional discussions supported by DOORS are also enabled on feature diagrams. This is modeled between Discussion and Feature as well as Discussion and Formal module with stereotype <<Feature modeler>>. This provides a basis for distributed stakeholders to collaborate during for decision-making.

Conventional requirements specifications/models. The meta-model also abstracts conventional requirements specifications/models within DOORS. For this purpose, we use a concept within the meta-model called Requirement[4]. Instances of Requirement are contained in a normal Formal module. A project contains several instances of a formal module. Requirements Traceability (traceability within requirements in a single system requirements engineering process) is supported using several instances of Link module.

Traceability between requirements and features. Features within the modeler should be mapped to requirements specifications. Feature to Requirement Traceability (many-to-many relationship between classes Requirement and Feature) expresses the traceability between traditional DOORS requirements and features supported by the feature modeler.

The application of feature to requirement traceability concept in practice could be different across different companies. In some industrial setting features are considered as parts of requirements. Here a simple solution is to document corresponding requirements as sub-sections of features, assuming no many-to-many mapping between features and requirements. In case of many-to-many mapping,

[4] There is no concept within DOORS called a Requirement. We introduce this concept to create a generalized abstraction. There could a significant difference how a requirement is realized across several companies.

requirements should be documented in separate sections of the same formal module or in separate modules within the same folder. DOORS traceability links (as proposed by Feature to Requirement Traceability should be used) to map requirements and features. In some companies, features are treated separately with respect to requirements (also see industrial experience described in this paper). In this case, requirements should be documented in separate formal modules (as well as in separate folders). Similar to the above case, feature to requirement traceability should be used to map features and corresponding requirements.

3.2 Domain engineering

Construction of feature models. After starting the user interface of the feature modeler, an und-user should create features. The creation of features is done at the same hierarchy or one hierarchy below the current level. Later, the end-user should assign feature relationship to the feature (by selecting a feature and adding a relationship as shown in Figure 1). Similarly, constraints are defined within the modeling environment. In the running example, Electric requires Accumulator and excludes Gearbox are modeled within the tree view (see Figure 1).

Automatic deficit checks. The feature modeler supports automatic checks (see R3) for redundancies, anomalies and inconsistencies, which are explained below.

A feature model contains *redundancy*, if at least one concept is semantically modeled in multiple ways. Let use consider an example. In an alternative relationship, all alternative child features are mutually exclusive. Only one of the children may be selected for a product instantiation. Here, an exclusion constraint between two alternative children leads to a redundancy. If requires constraint is modeled between two core features or between two mandatory features with same parent, the feature model has a redundancy.

A feature model contains *anomalies* if potential instantiations are lost though they should be possible. If a feature model contains anomalies it is likely that the domain has been captured wrongly. For example, connecting a core feature and an optional feature with requires constraint is an example of anomaly. In this case, optional feature is not optional any more.

In case of an *inconsistency*, the model contains information that is in conflict with other information in the same model. In most cases it is not possible to create consistent product instantiations from inconsistent feature models. Inconsistencies are therefore considered as a major deficit. For example, exclusion between two core features can never be fulfilled and no product may be instantiated from that feature model. Another example for an inconsistency is an exclusion constraint between two features that have been connected with requires constraint.

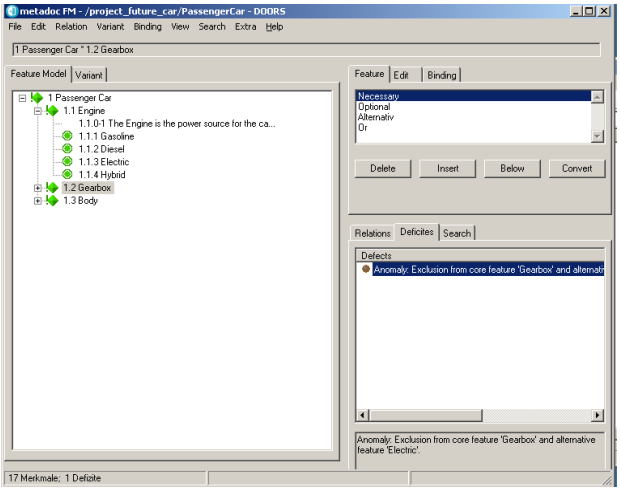

Figure 4. Automatic checks on the feature diagrams. The screenshot showing anomaly within the model.

The automatic checks for deficits were implemented by programmatically navigating the tree structure of a feature model using DXL and performing checks for the defects. The result of the deficit checks are viewed in the Deficits window. In the running example, the feature model contains an anomaly (see Figure 4), that is, a core features Gearbox is connected to Electric using the exclusion. Based on the recommendations given by feature modeler, end-user iteratively improves the feature diagrams to develop high quality feature models.

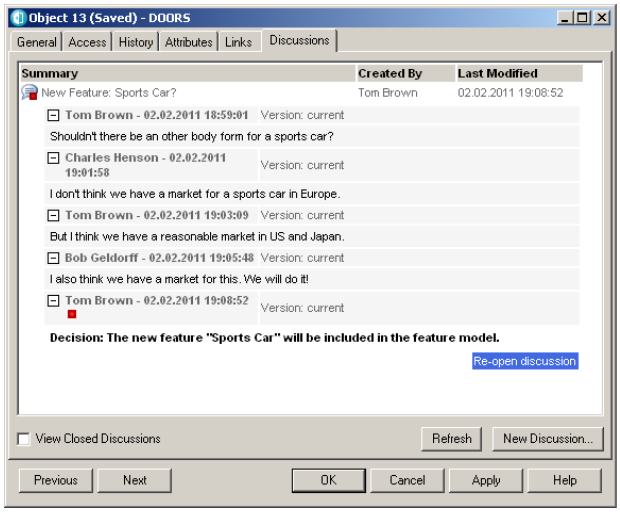

Figure 5. Usage of issues between distributed participants to perform decisions on the evolution of features.

Issues on features. Distributed stakeholders create issues on the features. For example, Tom Brown creates an issue (Shouldn't there be another body form for a sports car?) regarding changes on the feature Form. Relevant participants propose options and arguments on the issue. Based on the

argumentation obtained a decision to include the feature in the model is made by the project manager. Figure 5 presents the captured rationale history during decision-making. The change rationale captured is stored within the feature model.

3.3 Application engineering

The end-user should trigger the product instantiation by starting a new variant on the feature modeler. The core features can be automatically selected. For the rest of the features, end-users should instantiate them by selecting the features (see Figure 6). If an optional feature is instantiated, the mandatory features within the optional feature are automatically instantiated.

To aid product instantiation, we implemented automatic checks for two types of deficits, namely missing features and wrongly selected features. These two types of deficits are detailed below.

Missing features. A feature is not selected although it should be included.

- From a group of features with an or-relationship to one parent, no feature is selected although the parent feature is selected.
- Similar to the above case, from a group of features with an alternative-relationship to one parent no feature is selected although the parent feature is selected.
- One or more features at the destination of requires constraint are not selected although the feature at the source of this constraint is selected.
- One or more parent features are not selected although one or more of their child features are selected.

Wrongly selected features. A feature is selected for the product although it should be excluded

- From a group of features with an alternative-relationship to one parent more than one feature is selected.
- Both features connected with an excludes-constraint are selected.

In the running example two deficits (missing features) were identified based on the instantiated features (see Figure 6). Using the hints provided by the feature modeler, selecting/deselecting features (see graphical notations in Table 1) is done iteratively to obtain a high quality instantiation.

During product instantiation process, discussions (similar to Figure 5) could be created to resolve issues that occur during instantiation of feature models. In the running example, an issue is “How to instantiate alternatives of Forum?”. The alternative features are proposals. Distributed stakeholders add their arguments. Based on the arguments, a decision is made on the selection of a feature (e.g. Station Wagon).

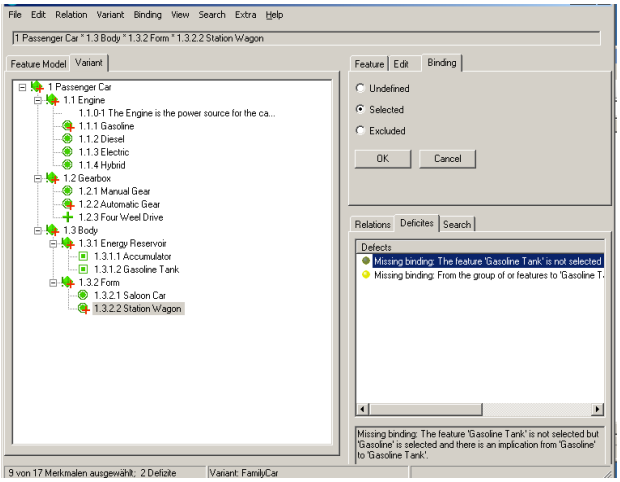

Figure 6. Selection of features during instantiation of the feature diagram.

By default, the instantiated features for a product are stored in the formal module related to the feature model. However, this could be configured based on the needs of the projects. For example, the instantiated features could be reported into a new formal module. Another possibility is to generate documents or excel sheets based on the instantiated features.

3.4 Limitations

This contribution focus only on requirements engineering phase and integration to other lifecycle models (e.g. coupling between features and components from Bak et al. [13]) is not considered.

An instance of the tree view could be opened for edit by a single end-user. However, using a multi-model environment and assigning different stakeholders to different feature models, we enable several stakeholders to work on different parts of the model at the same time.

IV. INDUSTRIAL EXPERIENCE

In the following, we share our experience in deploying and using metadoc feature modeler in a medium sized company. The experience described is based on a collaboration work with the company performed over a period of two years.

The case. The company is a medium sized manufacturer (about 1700 employees) of electric household appliances. Approximately 900,000 units are shipped per year. High class appliances are developed and manufactured for global market-segments. Number of product variants produced by a company in this domain is relatively high. For example, the company under consideration releases few hundred product variants per year.

One of the major departments of the company was consulted by the authors. The department deals with the electronic control units (ECUs), which are used to control the appliances. An appliance is usually equipped with one or more

ECUs and a user interface consisting of a display, keys and switches. Due to cost reduction one goal is to use only a small set of different ECUs for all of appliances.

The development and production of the ECUs are performed by various suppliers. The department drafts and sends requirements specifications to suppliers. The ECUs obtained are verified and integrated. Delay of ECU deliveries from their suppliers or low quality deliveries from their suppliers lead to a delay in the release of the company products. Therefore, the department needs to send high quality specifications to the suppliers.

Before usage of the metadoc feature modeler, the department used a single system requirements engineering. Specifications for each product are maintained in spread sheets and documents. Using these spreadsheets and documents, specifications for suppliers were also maintained. Here, the requirements specifications were redundantly developed and maintained for various product variants as well as for various ECU suppliers. There was no explicit variability management.

Within this domain, parameterization is used to characterize various components of the products. Requirements specifications are huge and include information about several parameters. Specifications (including documentation of all possible parameters involved on requirements) are big and often exceed maximum size of spread sheets. One more characteristic of the domain is that the parameters have several interdependencies. Therefore, requirements that are attributed with these parameters have interdependencies as well.

Relevant employees of the company have experience in DOORS. The company was in the process of improving their requirements engineering infrastructure.

SPL issues. From an SPL prospective, the development faced two major issues:

- The number of product variants developed by the company is increasing with time (few hundred variants are being produced currently). Managing specifications for the increasing number of product variants is time consuming and error prone. Big spreadsheets have been created but could hardly be overviewed and kept consistent.
- Due to the increasing number of product variants and constraints used in the domain, creating the requirements specifications for the ECU suppliers is highly time consuming and error prone.

The department needed a solution to manage the increasing number of variants. At least semi-automated generation of requirements specifications for various suppliers was desired.

Solution. Introduction of explicit variability management based on feature models was planned. metadoc feature modeler was chosen as it allowed the developers to model the features of the product line of appliances and the interdependencies of these features.

Before rollout was performed, the feature modeler was tested for scalability. Approximately 10,000 features were created within a feature diagram. At this level of complexity, the feature tree on the user interface is usable (that is, scrollable, collapsible and expandable). Furthermore, automatic deficit checks on feature diagrams, instantiation of feature as well as deficit analysis on instantiated features worked fine (with reasonable performance). Therefore, the tree view and the DXL routines behind the feature modeler are feasible to manage large feature models without any special treatment/approaches.

During rollout, four workshops where held together with the department. Domain experts and the department management participated in the workshops. The first workshop was one-day workshop. Within the workshop, feature modeling and the metadoc feature modeler was taught. In the first day, an initial feature model with around 300 features was created. In the following workshops this model has been redesigned and refined. During workshops, domain experts were asked a set of questions in order to elicit variability. These questions were asked during creation of features.

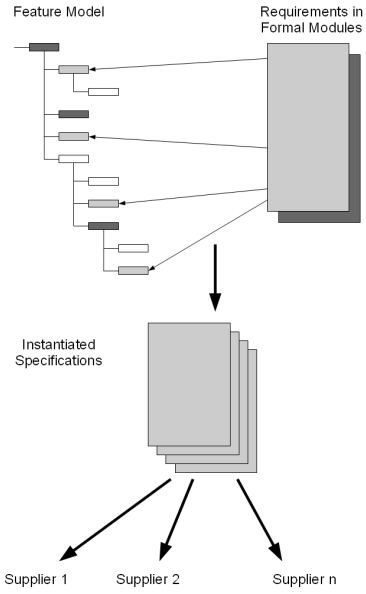

Figure 7. Requirements engineering workflow in DOORS after the introduction of feature modeler.

As ECUs are developed by suppliers, no special restructuring of the department was required in the process of transitioning from a single system requirements engineering to a SPL based requirements engineering. The new requirements engineering workflow within DOORS is shown in Figure 7.

As an output from the four workshops a realistic feature diagram was developed. All the existing requirements specifications were put into DOORS. During this process a redesign of the office-based document structure took place. As the next step, requirements specifications were linked to the respective features of the feature diagram. This ensured traceability between features in the modeler and requirements

modules in DOORS (see Feature to Requirement Traceability in Figure 3).

In application requirements engineering, feature modeler support functionalities such as selection of features, discussions and automated deficit analysis to instantiate features. Additionally, a requirements generator (see Figure 7) was developed to automatically instantiate specifications for individual product variants as well as to instantiate requirements specifications for suppliers. Here, concrete requirements for the instantiated features are derived using the existing traceability links between instantiated features and requirements (Feature to Requirement Traceability in the meta-model, see Figure 3) and traceability links within requirements (see Requirements Traceability). The generated specifications are sent to suppliers.

Lessons Learned. Within one day, the developers get used to the feature modeling technique. Furthermore, they were able to capture an initial feature model of their product line. We interpret that the participants learned and used feature modeler quickly.

During creation of variability, usage of a set of questions was helpful to elicit variability. A dedicated stakeholder or a role (e.g. variability manager) should take care of asking and tracking answers to these questions.

A difference between requirements and features was observed. Higher-level characteristics and dependencies were modeled as a part of feature diagram. Requirements specifications included requirements that are attributed with several parameters specific to the domain of home appliances.

The feature models together with automatic deficit analysis provided the possibility to create consistent specifications within the company. This solved the problem of inconsistent spreadsheets with redundantly documented requirements. After the company started using the feature modeler, time taken to create requirements specifications was significantly improved. Furthermore, the resulting specifications contained fewer errors than before using the feature modeler.

One more benefit is that the reuse of requirements improved significantly. Furthermore, by reusing requirements from the feature model, the company is able to generate specifications for the suppliers. This is an experience of using feature models in the context of manufacturer to supplier relationship, where a manufacturer generates specifications for his suppliers.

Within the organization, feature models laid a good foundation to facilitate the communication between different stakeholder groups (e.g. marketing, development and suppliers).

From our experience we suggest that, feature modeling is an easy to comprehend methods to capture, manage and understand variability in a product line development. However, there is a need for reasonable tool support to be used in an industrial context. Extending well established requirements management tools brings feature modeling to real live product line development.

V. Related Work

Variability can be represented using textual templates (e.g. textual use cases from Bertolino et al. [25] and XML based representations from Cechticky et al. [9]). Traditional DOORS extensions for variability focus on introducing variability by extending textual concepts. Schmid et al. [7] define a template for decision tables within a formal module to document variability. For the instantiation and evolution of variability models, a Java based tool (that has interface with DOORS) is used. In another contribution, Eriksson et al. [18], supports feature modeling by building textual templates within DOORS formal modules. The use case diagrams from Rose are exported and are inserted into textual templates of DOORS as graphical attachments. Orthogonal Variability Modeling (OVM) [2] is another related approach. Buhne et al. [14] introduce concepts of OVM meta-model within formal modules by defining textual templates (similar to Schmid et al. and Eriksson et al.). We differ form the above described contributions by introducing DXL components for graphical feature modeling within DOORS. Additionally, our contribution introduces automatic deficit analysis and issue-based variability management within DOORS.

BigLever Software Gears is one more related professional tool [12]. The tool is being developed and supported by BigLever. Gears focuses on producing a product line portfolio using product configuration and feature profiles. We differ from Gears by focusing on feature modeling. Gears supports an integration with DOORS. Based on this integration, there is a possibility to use metadoc feature modeler together with Gears for the generation of a product line portfolio.

MOSKKITT [10] is an Eclipse based feature modeling tool to support complexity. Particular features supported are various views on feature models and navigation between related model elements. Our contribution focuses on using DOORS inbuilt functions such as modules, views, filters and hierarchical structures to handle complex systems.

VI. Conclusion

IBM Rational DOORS is one of the well accepted tools in the requirements engineering practice. This paper focuses on supporting concepts such as feature diagrams, automatic analysis of feature models and issue-based discussions in DOORS. For this purpose, we developed a light-weight DXL plug-in called metadoc feature modeler, which is installable in an arbitrary DOORS infrastructure. We add graphical modeling concepts for feature modeling to supplement traditional text-based requirements engineering supported by DOORS. Using similar extensions to DOORS, other modeling approaches (e.g. orthogonal variability modeling) could be supported. We describe experience in using the feature model in a medium size company that produces home appliances. Extending established tools provide opportunity to bring variability modeling into live projects.

ACKNOWLEDGEMENTS

Many thanks to our industrial partners for their co-operation on the feature modeler. Furthermore, we would like to thank SPLC 2011 program committee members for their feedback on our paper. Special thanks to Deepak Dhungana and Gerald Heller for their comments on our work.

REFERENCES

[1] P. Clements and L. Northrop, "A Framework for Software Product Line Practice-Version 4.2 [online]". *Carnegie Mellon, Software Engineering Institute* URL: http://www.sei.cmu.edu/prodvolnuctlines/framework.html, Pittsburgh, USA, 2006.

[2] K. Pohl, G. Böckle and F. van der Linder, Software Product Line Engineering Foundations, Principles, and Techniques, Springer, 2005.

[3] K. Kang, S. Cohen, J. Hess, W. Nowak and S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Technical Report, *CMU/SEI-90-TR-21,* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

[4] T. Maßen and H. Lichter, "RequiLine: A Requirements Engineering Tool for Software Product Lines", *5th International Workshop Software Product-Family Engineering, PFE 2003*, LNCS vol 3014, pp. 168 – 180.

[5] D. Beuche, Pure variants technical white paper, Pure systems, URL: http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf, 2006.

[6] IBM Rational DOORS, DXL Reference Manual, Version 9.2.

[7] K. Schmid, K. Krennrich and M. Eisenbarth, "Requirements Management for Product Lines: Extending Professional Tools," 10th International Software Product Line Conference (SPLC'06), 2006, pp. 113-122.

[8] EMF feature modeling, http://www.eclipse.org/proposals/feature-model/

[9] V. Cechticky , A. Pasetti , O. Rohlik , W. Schaufelberger, XML-based Feature Modeling, Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004

[10] Moskkitt feature modeler, http://oomethod.dsic.upv.es/labs/index.php?option=com_content&view=frontpage&Itemid=1

[11] pure::varinats for IBM Rational DOORS, http://www.pure-systems.com/pure_variants_for_Doors.166.0.html

[12] Biglever's Gears, http://www.biglever.com/solution/framework.html

[13] Bąk, K., K. Czarnecki, and A. Wąsowski, "Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled", *3rd International Conference on Software Language Engineering*, Eindhoven, The Netherlands, 10/2010.

[14] S. Buhne, K. Lauenroth, K. Pohl, "Modelling Requirements Variability across Product Lines," 13th IEEE International Requirements Engineering Conference (RE'05), 2005, pp. 41-52.

[15] W. Kunz, and H. Rittel, *Issues* as *elements of information systems* (Working Paper No. 131). Berkeley: University of California at Berkeley, Institute of Urban and Regional Development, 1970.

[16] Thomas von der Maßen, Feature-basierte Modellierung und Analyse von Variabilität in Produktlinienanforderungen (Dissertation), Technischen Hochschule Aachen, 2007.

[17] A.K. Thurimella, Issue-based Variability Modeling, VDM Dr. Müller, 2009.

[18] M. Eriksson, H. Morast, J. Börstler, and K. Borg. 2005. The PLUSS toolkit - Extending telelogic DOORS and IBM-rational rose to support product line use case modeling. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (ASE '05). ACM, New York, NY, USA, 2005, 300-304.

[19] T. von der Massen and H. Lichter. Deficiencies in feature models. In Tomi Mannisto and Jan Bosch, editors, Workshop on Software Variability Management for Product Derivation - Towards Tool Support, 2004.

[20] IBM Rational DOORS, http://www-01.ibm.com/software/awdtools/doors/productline/

[21] E. Hull, K. Jackson and J. Dick, *Requirements Engineering*, SpringerVerlag, 2004.

[22] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", *Annals of Software Engineering 5*, 1998, pp. 143-168, .

[23] M. Riebisch, K. Böllert, D. Streitferdt and I. Philippow, "Extending Feature Diagrams with UML Multiplicities", In Proceedings of the *Sixth Conference on Integrated Design and Process Technology (IDPT 2002)*, Pasadena, CA June 2002.

[24] D. Griss, R. Allen, and M. d´Alessandro "Integrating Feature Modelling with the RSEB". In Proceedings of the *5th International Conference of Software Reuse (ICSR-5),*1998.

[25] A. Bertolino, A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Use Case Description of Requirements for Product Lines", In Proceedings of the *International Workshop on Requirements Engineering for Product Lines (REPL'02)*, 2002, pp. 12–18.

[26] Rational RequisitePro, http://www-01.ibm.com/software/awdtools/reqpro/.

[27] J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortes. Automated Diagnosis of Product-Line Configuration Errors in Feature Models. In Proceedings SPLC '08. IEEE, 2008, 225-234.

[28] P. Padmanabhan and R. Lutz. Tool-Supported Verification of Product Line Requirements. *Automated Software Engg.* 12, 4, 2005, 447-465.

[29] A. Thurimella and B. Bruegge, "Evolution in Product Line Requirements Engineering: A Rationale Management Approach," In Proceedings IEEE RE'07, 2007.

[30] A. Dutoit, R. McCall, I. Mistrik and B. Paech, Rationale Management in Software Engineering, Springer, 2006.

APPENDIX: FEATURE MODELING

In this appendix, feature modeling concepts used in this paper are presented. *Feature relationships* are used between parent feature and child features of a feature model. We use the following feature relationships:

- *Mandatory* implies that child feature is selected if the patent is selected
- *Optional* implies that the child feature could be selected
- *Alternative* one of the child feature must be selected
- *Or* at least one of the child features must be selected

Selection of a feature could have an affect on the selection (independent of feature relationships) of other features. This is represented based on a *constraint*. A constraint is modeled using the following concepts

- *Requires* – selection of the a feature implies the selection of another feature
- *Excludes* – selection of a feature hinders the selection of another feature.