Configuration of Multi Product Lines by Bridging Heterogeneous Variability Modeling Approaches

Deepak Dhungana
Siemens AG Österreich
Corporate Technology, Vienna, Austria
deepak.dhungana@siemens.com

Dominik Seichter and Goetz Botterweck
Lero–The Irish Software Engineering Research Center
University of Limerick, Limerick, Ireland
goetz.botterweck@lero.ie

Rick Rabiser and Paul Grünbacher
Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler University, Linz, Austria
rick.rabiser@jku.at

David Benavides and José A. Galindo
Department of Computer Languages and Systems
University of Seville, Seville, Spain
benavides@us.es

***Abstract*—In industrial settings, products are rarely built by one organization alone. Software vendors and suppliers typically maintain their own product lines, which can contribute to a larger (multi) product line. The teams involved often use different approaches and tools to manage variability of their systems. It is unrealistic to assume that all participating units can use a standardized and prescribed variability modeling technique. The configuration of products based on several models in different notations and with different semantics is not well supported by existing approaches. In this paper we present an integrative approach that provides a unified perspective to users configuring products in multi product line environments, regardless of the different modeling methods and tools used internally. We also present a technical infrastructure and a prototypic implementation based on Web Services. We show the feasibility of the approach and its implementation by using it with two different variability modeling approaches (i.e., one feature-based and one decision-oriented approach) on an example derived from industrial experience.**

I. Introduction and Motivation

Software product lines (SPL) are increasingly developed beyond the boundaries of a single organization [1]. Distributed teams create software products in a collaborative effort. Variability management and product configuration in such contexts need to reconcile the different modeling approaches, notations and tools in use. Due to significant differences in practices in different domains it is unlikely that there will ever be just *one* standardized variability modeling approach. The increasing number of "island solutions" to variability modeling and product configuration restricts communication and hinders collaboration between distributed product line engineers. Hence, there is a strong need for an integrative infrastructure enabling the collaboration between different organizations developing multi product lines. In particular, the approach must support different variability modeling languages, notations, and tools.

This paper focuses on the product configuration aspects of such an infrastructure. We propose an approach to facilitate the integration of variability models[1] created with different modeling approaches and potentially by different teams. The specific tools or data formats (cf. [2], [3], [4] for an overview) used when creating the variability models are not relevant for an end-user who only cares about the available choices and their implications.

Therefore, we make the internal technical aspects of using variability models for configuration transparent to the stakeholders performing the configuration. We unify configuration operations on variability models and give the freedom of data representation to the modeler by allowing variability models to be accessed through Web Services. We do *not* force organizations to overly *integrate* their configuration tools (in the sense of deep integration where internals of the tools have to be adapted). Instead, we allow them to *compose* their configuration mechanisms using wrappers and interface definitions.

The remainder of this paper is structured as follows: In Section II we present an example of multi product lines. Section III describes our I*nvar* approach that facilitates the use of heterogeneous variability models during product configuration. In Section IV we present our I*nvar* prototype for two different variability modeling tools. In Section V we present a preliminary validation of our approach using different scenarios of applying the approach. We then present related work in Section VI and conclude the paper with a summary and discussion of future work in Section VII.

II. Multi Product Lines: A Motivating Example

We illustrate the challenges related to software variability management and product configuration in multi product line environments using an example of an Enterprise Resource Planning (ERP) system. Though fictitious, the example is based on a real-world product line of an industry partner – a medium-sized vendor from the ERP domain [5]. The company

[1]Throughout this paper, we use the term "variability model" to refer to product line models regardless of the specific approach and notation used, e.g., feature models, decision models, OVM models, COVAMOF models, etc.

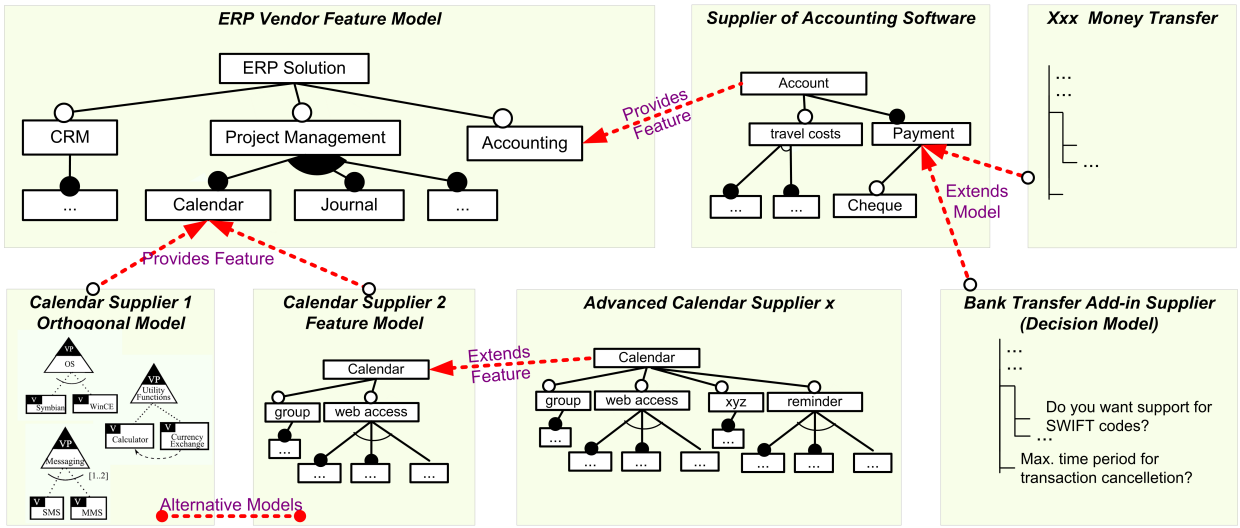

Fig. 1. Example of interrelated multi product lines in the ERP domain. Vendors and suppliers depend on each others' products, hence the variability models are related to each other in different ways. In our approach, the relationships between the models are expressed as IMDI links (see Table II). We have included an OVM model in this example for illustrative purposes only. OVM is not yet supported by the proposed approach.

offers enterprise software products to about 20.000 customers and 50.000 active users in central Europe. Software products include applications for customer relationship management, accounting, payroll, ERP, as well as production planning and control. Customized products are an essential part of the company's marketing strategy.

In the example depicted in Figure 1, the main vendor of the ERP application integrates several suppliers providing specific encapsulated functionality. The vendor uses a feature model to present the set of available choices and to communicate extension and integration possibilities for other systems. The suppliers use different approaches and tools to deal with variability. For example, some suppliers use feature modeling tools, while others apply orthogonal variability modeling or decision modeling. Nevertheless, the models are related to each other or depend on each other. Such relationships may be the result of technical dependencies among the software products the variability models define. Dependencies among models may also occur because of the role taken by the product, e.g., the position in a supply chain or the specific purpose of the model (technical configuration, marketing, or documentation). These relationships play an important role when the models are used together during end-user product configuration of the ERP application.

The example model presents a common scenario: The ERP vendor defines in her feature model that a Calendar can be selected to support project management. Two suppliers provide different alternatives for the Calendar feature with diverse, more detailed configuration choices. One supplier uses an OVM model to describe the variability of her calendar, the other supplier uses a feature model (possibly in a different notation than the vendor feature model).

For the configuration of an ERP solution in such a context, an integrative infrastructure is needed, which works on shared concepts among the different modeling notations. This imposes several research and practical challenges: When attempting to integrate different variability models, one also has to consider interfaces between the models. For instance, there should be mechanisms for defining dependencies across models. It is important to allow unrelated models to change independently and minimize coupling between related models. Our solution to these problems is to allow the variability model providers to manage and evolve the models themselves. A configuration front-end for end-users makes use of the available models in a repository.

III. The *Invar* Approach

In this section, we present the *Invar*[2] framework which allows to "plug-and-play" variability models. "Plugging" refers to simply adding new variability models to a shared repository. "Playing" refers to presenting the options to the end-user allowing her to configure the required product. For this purpose, a variability model is seen as an autonomous entity, which can be plugged into the configuration space to provide configuration options. Autonomous however does not necessarily mean independent, because variability models may be related to each other as we have shown in the motivating example in Section II. Our approach allows using variability models distributed across multiple repositories by accessing them through Web Services providing configuration choices. An end-user works with a front-end for product configuration

[2]Invar is a nickel-steel alloy notable for its uniquely low coefficient of thermal expansion. In metallurgy, it is a good example of how one can profit by combining different metals, to achieve special desired properties. We chose the name *Invar* for our approach to reflect on the need to combine/integrate/compose different variability modeling approaches, notations and tools in the context of multi product lines. Alternatively, *Invar* stands for "**in**tegrated view on **var**iability".

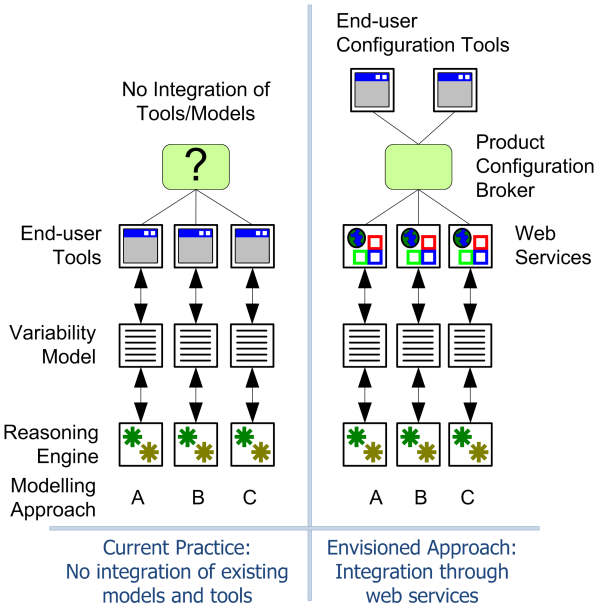

Fig. 2. A highly simplified view of model-based product configuration: current state of practice (left) and I*nvar* approach (right).

and can use the configuration services without knowing details about the concrete variability models "behind" the services.

Figure 2 depicts the I*nvar* approach compared with the current state of practice. Currently (left side) multiple heterogeneous variability modeling approaches are used by different organizations. Different reasoning and analysis engines (e.g., SAT solvers [6], rule engines[7]) are adopted for interpreting the models' semantics. There is no integration of the diverse tools supporting different notations. In some organizations variability may also be managed "manually" using textual descriptions or spreadsheets; these "tools" are typically not integrated with other variability modeling tools due to the lack of formal semantics of their approach or simply because they are not dedicated tools for configuration.

With I*nvar* (cf. right side in Figure 2), stakeholders create variability models using an approach of their choice. I*nvar* defines key operations and queries (*configuration primitives*) on variability models to allow the integration of heterogeneous configuration approaches. These configuration primitives are implemented as Web Services to allow uniform access to the models. Participating units using different modeling approaches are able to reuse the variability models from other units. I*nvar* provides a single and transparent configuration tool to end-users. This ensures interoperability and allows reusing variability models in different contexts (e.g., one model may be shared between several companies and each one may use it to create different products). It would also be possible to integrate manual approaches using Excel or Text Files into I*nvar* by defining new web services. However that would require specification of semantics for each approach, which is why we have not yet integrated any such approach.

A. Assumptions

Regardless of the variability modeling approach used, these approaches share some common characteristics. A Variability Model consists of Variables and Constraints over these variables. Each variable has a Type (e.g., Boolean, Integer, String). Depending on the particular approach there are different Types of Constraints, e.g., "Optional Sub-features" and "Alternative Groups". A modeler creates a list of variables and a list of constraints. An Assignment of values to the variables corresponds to a Configuration of the model. For a given configuration, we can decide whether it satisfies the constraints defined in the model. Hence, a model defines a set of configurations complying with it. The user implicitly or explicitly adds more constraints, as she takes the configuration decisions. Then, we can determine the possible values for each variable. Adding more constraints eventually leads to a model which has exactly one valid assignment and each variable has exactly one value. This represents the configured product, for instance in terms of selected and deselected functionality. If no valid assignment is possible the model is unsatisfiable.

Based on these assumptions, we have defined a set of Web Services for accessing variability models. The set of primitive operations and queries on the models required for developing such Web Services are based on our experience of developing product configuration applications [8], [7], [6], [9], [10], [11].

B. Configuration Primitives

There are typical operations needed by end-user product configuration tools that "execute" or evaluate variability models. These operations are provided as methods by most of the existing APIs [7], [10], [8], [11], [6] allowing model-based product configuration. For example, the end-user may query the set of all available options at a time, or send a request to select one of the options. The basic concepts in most of these APIs revolve around *options* or *choices* for the model consumer. Typical operations on the models are:

Loading and initializing models: load() gets models from their persistent storage to memory, while reload() loads the model again. The operation init() is used to start a new configuration session based on a model. save() persists the session for future use.

Querying the model for available options: For instance, nextQuestion() gets the next available question to be answered by the user (regardless if this is a feature, decision, or variation point). Analogously, previousQuestion() allows to obtain the previous question. peek() allows previewing the next available question, without having to answer it.

Operating on the available options: The operation setValue() sets the value of feature attributes or defines the answer to a decision question. The primitives select() and deselect() allow choosing or eliminating (i.e., deciding not to buy) elements in a list of presented options. undo() and redo() allow canceling the last action or replaying the last selection. addOption() adds new options to available questions.

Notifications: The success() method shows if an operation was carried out successfully while error() indicates problems.

The primitive contradiction() shows whether the choices of the user are consistent with the model's semantics. The operations selected() and deselected() are used to inform other tools or users about user actions.

Obviously, this set of primitive operations is not complete or fixed. For instance, in some cases, several questions can be proposed at the same time, i.e., depending on the end-user application there may be other operations that are useful in building complex user interfaces. Our approach can easily be extended to include new operations or queries on the models. The list of currently supported operations is presented in Table I.

C. Inter-model Dependencies

Whenever a variability model is plugged into the configuration environment, it needs to explicitly define its relationships to the other models. This is done by adding an *inter-model dependency information (IMDI) packet* together with the model. Dependencies between models are defined using if *condition* then *action* clauses. These can be compared to conventional cross–tree constraints within one model. IMDI packets do not affect the internal semantics of the models in use. An IMDI action is executed when its condition evaluates to TRUE.

A summary of conditions and actions, currently supported by the Invar framework is listed in Table I. In the following we describe the basic types:

Inter-model constraints: If selecting or deselecting an option in one model has implications on other models, an inter-model constraint needs to be defined. To specify such constraints, the conditions isSelected() and isDeSelected() are used. The corresponding actions are doSelect() and doDeSelect(). For example, in our example in Figure 1, one could specify the link between Vendor.Accounting and Supplier.Account as if Vendor.Accounting.isSelected() then Supplier.Account.doSelect().

Informative actions: One can also define actions that do not change the models. Some actions like inform(), recommend() or warn() are used to simply present information such as recommendations or warnings to the user.

Conditional inclusion of models: If several variability models are used for product configuration the order of presenting the models to the end-user has to be defined. We define the action includeModel() which changes the presentation order. This influences the model navigation strategy (i.e., which model is configured in which order). For example, in Figure 1 the link between Calendar Supplier1 and ERP Vendor can be specified as if Vendor.Calendar.isSelected() then Supplier1.includeModel().

D. Invar Architecture

An overview of the architecture of our integrative infrastructure is presented in Figure 3, which depicts five main components (numbered in the figure).

(1) Vendor model repositories: Product vendors or suppliers add their variability models to model repositories. The models may or may not contribute to the same product and are not necessarily dependent on each other.

(2) Configuration Web Services: The different models residing in (possibly distributed) repositories are accessed by configuration Web Services. The Web Service provides a standard interface for different configuration front-ends (websites, mobile devices, stand-alone applications, etc.). For each type of model, designated configuration services are developed (by implementing an interface) that can read the data formats, interpret the content, and perform operations on the models.

(3) Invar repository: The Invar repository defines aggregations of different models from the vendor repositories by logically grouping them. For instance, one particular model may be part of multiple product lines, as it may contribute to more than one product.

(4) Configuration broker: The configuration broker enables the communication between the Web Services. It reads the inter-model dependency information to determine which Web Services are affected when products are configured. The configuration broker also translates events from the end-user configuring and passes them on to the Web Services that need to react to the end-user's interactions.

(5) End-user product configuration front-ends: The configuration choices defined in the variability models are presented to the end-user in a product configuration front-end. This can be a website or a stand-alone application. We provide an example user-interface through the Invar framework website at http://invar.lero.ie, which is shown in Figure 4.

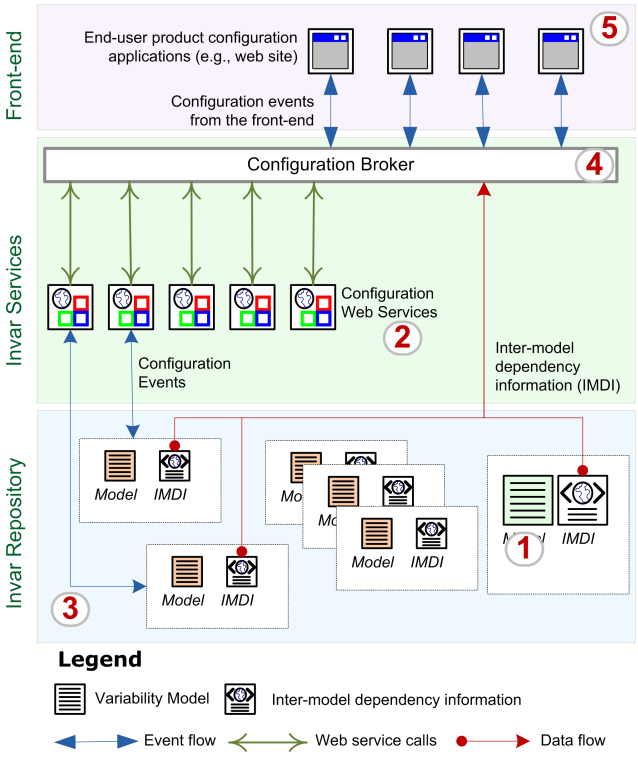

Fig. 3. Architecture of the integrative Invar infrastructure.

TABLE I

A SUMMARY OF CURRENTLY SUPPORTED CONDITIONS AND ACTIONS, WHICH CAN BE USED TO CREATE IMDI FOR THE I*nvar* APPROACH TO MANAGE INTER-MODEL-DEPENDENCIES. THIS LIST MAY BE EXTENDED AS REQUIRED TO BUILD MORE COMPLEX RELATIONSHIPS BETWEEN MODELS.

Keyword	**Type**	**Description**
isInit()	condition	This condition is only true when a user starts configuring a model for the first time. The actions connected with isInit() conditions are executed immediately after each model has been initialized on its Web Service.
isSelected()	condition	This conditions evaluates to true whenever a specified option of a given question in a specific model is in the state selected. The condition is evaluated after each change of a user to a model, i.e., after the user answers a question related to the link.
isDeSelected()	condition	This conditions evaluates to true whenever a specified option of a given question in a specific model is in the state deselected. The condition is evaluated after each change of a user to a model, i.e., after a user answers a question.
doSelect()	action	Set a specified option, of a given question in a specific model to the state selected.
doDeSelect()	action	Set a specified option, of a given question in a specific model to the state deselected.
includeModel()	action	Add a model to the list of included models of the current navigation strategy, if it was not yet initially included or included by another action. Each model can only be included once.
addOption(...)	action	Adds an option to one model as a child of an existing feature. This is usually required, when a model extends another model.
inform()	action	Display a specified message to the user, which can have the type *information*, *recommendation* or *warning*.

TABLE II

EXAMPLES OF IMDI LINKS, THAT COULD BE USE TO MODEL THE INTER MODEL DEPENDENCIES DEPICTED IN FIGURE 1. THIS IS NOT AN EXHAUSTIVE LIST, AS THE IMDI LINKS CAN BE USED TO MODEL MORE COMPLEX RELATIONSHIPS AS DEPICTED HERE.

Intermodel dependency	**Example of IMDI link in** I*nvar*
Model_x and Model_y are alternative models, the choice of the model depends on Feature_a in Model_{parent}	if Model_{parent}.Feature_a.isSelected() then Model_x.includeModel()
Model_x *extends model* Model_y by adding Feature_{a1} as a child of Feature_a	if Model_x.isInit() Model_y.Feature_a.addOption(Feature_{a1})
Model_x *provides feature* Model_y.Feature_a	if Model_y.Feature_a.isSelected() then Model_x.Feature_a.doSelect()

E. I*nvar* *Configuration Service*

Central to the implementation of I*nvar* is the generic configuration interface defined for accessing the diverse variability models. The configuration service definition has to be implemented *once* for each modeling notation. The Web Service is designed such that the configuration options are presented as questions to the end-user. Questions are only a means to render the variation point to present it to the user. This means the user is asked questions about a certain "feature" (in the wider sense) or a property of the system which she configures. The possible answers to the question (the available alternatives) are presented to the user such that she may choose one or many of them depending on the type of variability. The notion of "questions" and possible answers as options is therefore key to the I*nvar* configuration service.

The interface consists of two parts: A *variability model query part* providing basic information about models (e.g., the set of available questions and the possible answers) and an *operational part* directly interacting with models to assign answers to specific questions (e.g., when selecting a particular feature).

The configuration service also defines a set of predefined question types. The types have been defined based on how the end-user is supposed to answer them. For example, the question type Alternative refers to questions where the user can select exactly one option (rendered using radio buttons or comboboxes in the UI); for Optional the user can pick multiple items (rendered using checkboxes in the UI) and MoreThanOne refers to cardinality (1..*) (rendered using multi-selection checkboxes in the UI).

IV. IMPLEMENTATION

Our current implementation of I*nvar* allows creating and maintaining repositories for sharing variability models and supports end-user configuration based on these models. The infrastructure relies on the Web Services for accessing variability models and on the configuration front-end. Any Web Service API can be used to generate Web Services, which can be plugged into the I*nvar* framework based on a provided WSDL-description.

We have implemented the I*nvar* service configuration interface for two different modeling approaches, i.e., the feature-oriented FaMa tool suite [8] and the decision-based DOPLER [7] tool suite. We chose these approaches as we have gained several years of experience of applying them in academic and industrial settings including large-scale product lines and they cover diverse "flavors" of variability modeling [8], [7], [6], [9], [10], [11].

A. Plugging in FaMa Feature Models to I*nvar*

1) FaMa Background: A feature model describes a set of products of a SPL in terms of features and relationships among them. A feature model is represented as a hierarchically arranged set of features composed by relationships between a parent feature and its child features as well as cross–tree

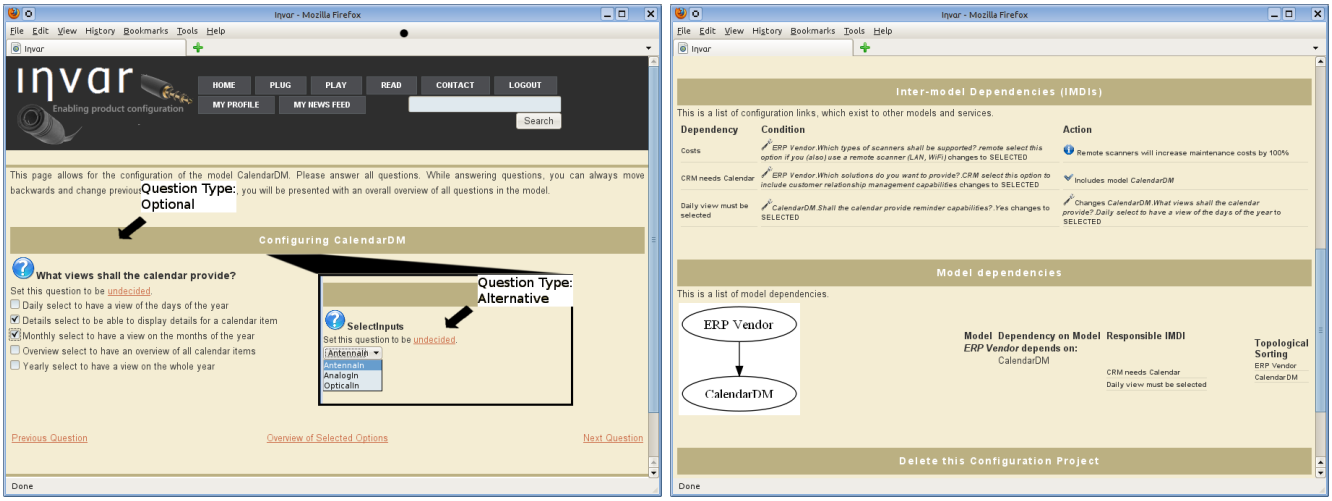

Fig. 4. The *Invar* prototype: Presentation of questions from heterogenous models (left), definition and visualization of inter-model dependencies (right).

constraints that are typically inclusion or exclusion statements in the form: *if feature F is included, then features X and Y must also be included (or must be excluded).*

The FaMa approach [8] allows using different solvers in the back-end to perform analysis operations on feature models. Currently it implements analysis using constraint programming, SAT and BDD solvers. Other solvers can easily be plugged–in. FaMa also provides capabilities to automatically test new implementations [12].

The implementation of the FaMa modeling approach to provide *Invar* configuration services had to consider several issues: (*i*) the FaMa approach itself was not designed to be used for questionnaire-based product configuration, (*ii*) FaMa was primarily implemented as a framework to perform automated analysis of feature models, i.e., automated extraction of information from feature models and not concretely for product configuration. Nevertheless, the adaptation to *Invar* Web Service interfaces was almost straightforward. Some of the key mappings between the *Invar* configuration steps and FaMa can be summarized as follows:

Question Types: FaMa supports cardinality-based feature models [13] where group relationships can include UML–like cardinalities. For the sake of simplicity, here we only considered feature models with four kinds of relationships, which were mapped to *Invar* question types as follows: A mandatory relationship in a feature model is a relationship between a parent and a child where the child has to be selected whenever the parent is selected. Hence, in this case, no question is asked to the user. An optional relationship between a parent and a child means that the child can be selected or deselected whenever the parent feature is selected. We mapped an optional relationship to an Alternative question type in *Invar* with only one option, this is, a single check box. An or-relationship between a parent and a set of children determines that at least one child has to be selected whenever the parent is selected. Any combination of children is also allowed. We mapped the or-relationship to a MoreThanOne question type in *Invar* with multiple check boxes. An alternative relationship between a parent and a set of children determines that one and only one child has to be selected whenever the parent is selected. An alternative relationship maps to the Alternative question type in *Invar*.

Order of Questions: Feature models are not conceived for question–based configuration. Hence, there is no predefined order of presenting questions to the end–user. In our implementation we decided to traverse the tree in a pre–order–like style. Other options can be contemplated like adding an attribute to the relationships assigning priorities for configuration purposes. This would allow the modeler to introduce question orders.

Feedback: In FaMa, at any time a configuration can give some feedback to the *Invar* configuration service. The feedback supported includes: (*i*) inform whether a given feature (component) is selected or deselected, (*ii*) determine whether the current configuration is valid, i.e., it is possible to extend the configuration to a valid product, (*iii*) calculate the total number of potential configurations of the model, (*iv*) inform about the number of questions that have not been decided yet, (*v*) calculate the number of potential configurations available according to the current selection/deselection of features (components), and (*vi*) determine whether the current configuration is valid as a final product.

B. *Plugging in* DOPLER *Decision Models to* Invar

1) DOPLER *Background:* A decision model describes a set of available configuration variables, i.e., a *decision* arises whenever for a given goal there exist two or more options for achieving it. A decision model is represented as multiple hierarchies of decisions that need to be taken by the user when configuring a set of reusable artifacts. A number of decision modeling approaches have been proposed over the years, see [14] for a comparative analysis.

The DOPLER approach [7] allows defining the reusable assets of a product line (e.g., the components in the product line) and mapping them to the available decisions describing their variability. A domain-specific meta-model defines the possible types of assets, their attributes and dependencies. In addition to hierarchical dependencies among decisions, other dependencies (comparable to cross–tree constraints) are modeled using rules of the form *if condition then action* [7]. The DOPLER tool suite [7] uses a Java-based Rule Language (JRL) and execution engine as a back-end for evaluating the rules defined in models. For a description of different application examples refer to [7], [15].

The implementation of the DOPLER decision modeling approach to provide *Invar* configuration services was rather straightforward, as the DOPLER approach itself was designed to be used for questionnaire-based product configuration [10]. The mapping from *Invar* to DOPLER in many cases only required calling the respective method in the DOPLER API. Some of the key mappings between the *Invar* configuration steps and DOPLER can be summarized as follows:

Question Types: We had to map the *Invar* question types to DOPLER decision types. DOPLER decision types are Boolean, String, Number and Enumeration. Enumeration decisions can be defined with a cardinality defining the subset of the set of possible answers to the decision that might be selected (e.g., 1:1, 1:n, 2:6). For the sake of simplicity, we have only implemented the mapping for Boolean and Enumeration decision types. This is sufficient as String and Number decisions can also be presented as an Enumeration decision with one option (being a string or a number). More specifically, we mapped the DOPLER Boolean decisions to the Alternative question type with the options yes or no, the DOPLER enumeration decisions with cardinality 1:1 or 0:1 were also mapped to the question type Alternative (with the enumeration literals as options), and the DOPLER enumeration decisions with all other possible cardinalities were mapped to the *Invar* question type Optional.

Order of Questions: In DOPLER, the order of taking decisions is defined by the decisions' dependencies. Top level decisions (which are not dependent on other decisions) are presented and answered first. Decisions which directly depend on top level decisions can be answered next, and so forth. In addition to these hierarchical dependencies, DOPLER allows defining logical dependencies that cross–cut the hierarchical tree structure. For example, answering a particular decision might require changing the value of another decision located somewhere else in the hierarchy. In the *Invar* configuration service interface, the methods getFirstQuestion(), getNextQuestion() and getPreviousQuestion() implement the navigation strategy. When initializing a DOPLER decision model with *Invar*, first a sorted list is built based on the decision hierarchy. This list is frequently updated when new decisions are added or the order of decisions is changed due to some rule defined in the DOPLER decision model. The order of decisions on one level (e.g., top level) is randomly defined. Logical dependencies are currently not considered by the first/next/previous methods because they would require "jumping" within the model. Whenever taking a decision has an effect on another decision, this effect has to be presented separately, e.g., by informing the user that the other decision was changed and asking her whether she wants to jump to that decision in the list.

Feedback: In DOPLER, making decisions leads to the inclusion and/or parametrization of assets related to the decisions. For example, selecting the document management solution by answering a decision question might lead to the inclusion of the respective software component(s) implementing document management. Answering a lower level decision (e.g., on which type of scanner is required) might parameterize the document management software component. Feedback to the user of the *Invar* service implementation for DOPLER is given by presenting her with the assets that are required for the product currently being configured.

V. PRELIMINARY VALIDATION

We have tested our approach by creating the variability models for the ERP example with FaMa and DOPLER. We composed these product line models and configured products using the *Invar* product configuration Web Service infrastructure. More specifically we have tested three different settings representing different typical scenarios of how vendors and suppliers may interact, collaborative, a competitive, and complementary settings. These examples are similar but not the same as in Figure 1.

A. Collaborative Setting

Figure 5 depicts a collaborative setting, where the main ERP vendor relies on two suppliers for calendar and journal components. The configuration application needs to integrate the three models. The end-user configuring the ERP application would not be aware of the three models in the background. She would also not care about the modeling notation used in the models because the configuration options are presented as questions and possible answers.

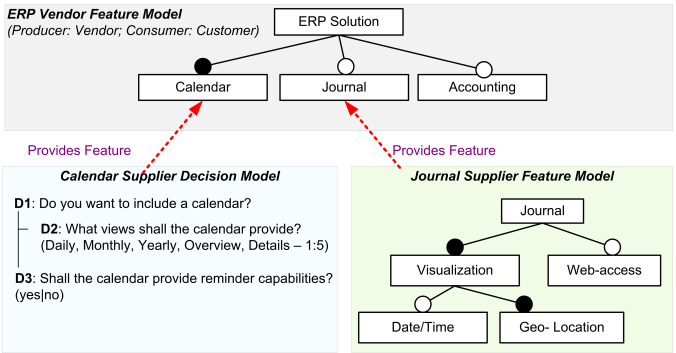

Fig. 5. Example application of a collaborative setting: One main model from the vendor and individual models for the sub-systems in different modeling notations.

We conducted the following steps to set up the configuration application for this scenario in *Invar*.

1) Web Services capable of reading the two feature models and the decision model are registered to *Invar*.
2) A configuration application is defined by referencing the three models. All three models are included in the configuration process by default. The start model is defined to be the ERP vendor feature model.
3) Inter-model dependency links are defined. In the example models, there are two such links if vendor.calender.isSelected() then calendarSupplier.D1.doSelect() and if vendor.journa.isSelected() then journalSupplier.journal.doSelect().

During product configuration, firstly the questions from the feature model of the vendor are presented to the user (cf. Section III-E for how these questions are created). Inter-model dependency links are resolved by the configuration broker as soon as the calendar or the journal options are selected. The corresponding Web Services are informed about the selection of the feature (as the user answers the questions). The output of the configuration process is the list of selected features and assets delivered by the corresponding Web Services.

B. Competitive Setting

Figure 6 presents a competitive setting, where two suppliers provide functionality related to archiving documents. One of these needs to be chosen for the end-product. The selection of the supplier can be based on complex metrics and criteria as described in [16], [17]. In *Invar*, we are currently not considering such metrics. We provide mechanisms to select different suppliers based on the decisions taken by the end-user during product configuration.

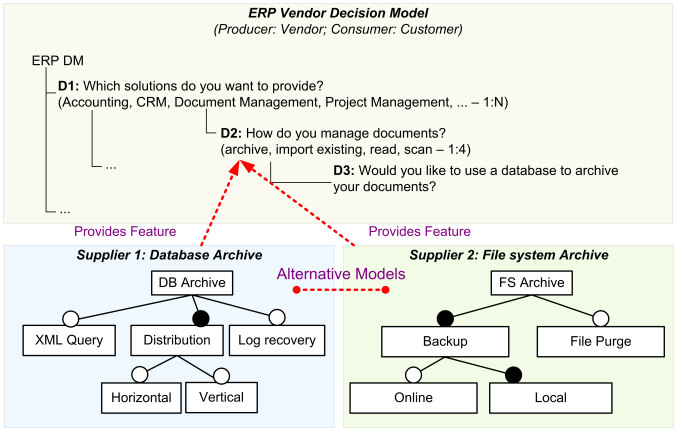

Fig. 6. Example application of a competitive setting: One main model from the vendor and individual alternative models for the same sub-system.

We conducted the following steps to set up such a configuration application in *Invar* for this scenario.

1) As in the previous scenario, Web Services capable of reading the two feature models and the decision model need to be registered.
2) The configuration application is defined by referencing the three models. However, not all three models are included in the configuration process by default. The supplier models are embedded into the configuration process using *conditional inclusion actions* defined in IMDI packets. The start model is defined to be the ERP vendor decision model.
3) Inter-model dependency links are defined using conditional inclusion actions if vendor.D3.isSelected() then DatabaseArchive.includeModel() and if vendor.D3.isDeSelected() then FileSystemArchive.includeModel().

During product configuration, firstly the questions from the feature model of the vendor are presented to the user. As soon as question *D3* is answered, the configuration broker determines which other models should be included in the configuration process by evaluating the IMDI packets. The correct supplier is chosen "behind the scenes" and the end-user is presented with a seemingly tailored configuration interface (showing only the relevant questions). The output of the configuration process, just like in the previous scenario is the list of selected features.

C. Complementary Setting

In order to provide support for complementary relationships (if one model extends other models) special dependency links must be defined to provide an integrated view to the configuring end-user. For example in Figure 7, the bank transfer add-in extends the available payment methods in the accounting package.

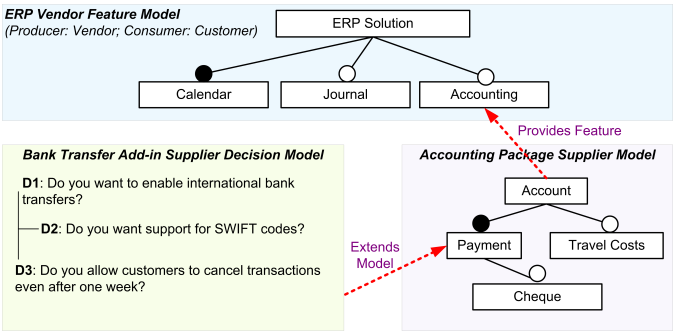

Fig. 7. Example application of a complementary setting: One main model is extended by another model, resulting in an additional option.

We conducted the following steps to set up such a configuration application in *Invar* for the complementary setting.

1) Again, Web Services capable of reading the two feature models and the decision model are registered in *Invar*.
2) A configuration application is defined by referencing the three models. Two models (Vendor and Accounting) are included in the configuration process by default. The start model is defined to be the ERP vendor feature model.
3) The option payment in the accounting package is extended by one option "bank transfer" in the beginning. This is done by adding two links initialized() Accounting.Payment.addOption(BankTransfer) and

if Accounting.BankTransfer.isSelected() then BankTransfer.includeModel().

During the configuration process, the user has to select between two options of the question related to *payment type*. Upon selection of *BankTransfer*, the *Bank Transfer model* is added to the configuration process. The output of the configuration process is the list of selected features just like in the previous scenario.

VI. Related Work

We structure our discussion of related work in the areas of variability in large-scale software development, coordination during product derivation, software integration, software ecosystems, and standardization efforts.

a) Variability in large-scale software development: Deelstra *et al.* [18] classify different types of approaches along the dimensions of scope of reuse and domain scope. Hierarchical product lines [19] can be applied when there is a broad family with a number of focused product categories. Bühne *et al.* [20] extend an existing meta-model for variability modeling to structure product lines into a hierarchy. Van Ommering [21] introduced the concept of *Product Populations*. Dhungana *et al.* [22] explore how to structure the modeling space for large product lines on multiple levels of abstraction. Reiser and Weber [23] suggest to model complex product lines with a hierarchy of feature models, which they call multi-level feature trees. *In contrast to all these approaches,* I*nvar focuses explicitly on enabling the integration of different variability modeling approaches to support and coordinate end-user configuration.* Schmid [24] presents some examples of existing distributed variability modeling mechanisms in real-world, large-scale projects. This paper however, does not provide a solution to integrating heterogenous modeling approaches.

b) Coordination during product derivation: Czarnecki *et al.* [25] present an approach allowing the staged configuration of features models. Reiser *et al.* [26] address the problem of modeling and supporting feature selections in large-scale embedded systems and propose product sets and configuration links to define dependencies between different feature models at the time of their selection. Hubaux *et al.* [27] propose feature configuration workflows as a new formalism for supporting configuration of large-scale systems. They also propose a technique to specify concerns in feature diagrams and to build automatically concern-specific configuration views [28]. The issue of multi-level configuration is also addressed by [5] who demonstrate how decision models can be used to support the configuration of a complex system across multiple levels of software vendor, customers and end-users. Czarnecki *et al.* [29] discuss the customization in application engineering over multiple levels. Metzger *et al.* [30] have presented an approach to integrate feature models and orthogonal variability models, by differentiating between product variability and software variability. *The* I*nvar approach is different, as we focus on variability of different subsystems, rather than different abstractions of the same subsystem.*

c) Software integration: Our work is also related to integrating complex software solutions and tools. In the area of EAI (enterprise application integration) Hohpe and Woolf [31] present fundamental patterns for integrating heterogeneous and asynchronous systems. More recently, this issue has also been addressed in the area of software and systems engineering. For instance, Moser *et al.* [32] propose an engineering service bus to integrate arbitrary tools in the engineering life cycle.

d) Software ecosystems: Bosch [1] discusses the transition from product lines to software ecosystems. He identifies three types of ecosystems (operating system-centric, application-centric, and end-user programming software ecosystems) and their characteristics, success factors, and challenges. One could argue that our approach is related to the intention of end-user programming ecosystems, in the sense that the ultimate goal is to allow an end-user to create the required application himself. *In contrast to end-user programming, however, we focus on the* configuration *of a solution and abstract from the programming or composition of its implementation.* Similar to the "formalization of interoperability" discussed by Bosch, our approach ensures interoperability through the definition of an integrative architecture and the interfaces between system components.

e) Standardization efforts: There is an ongoing effort to standardize a variability modeling approach based on the common variability modeling language (CVL) [33] which is a generic language to define variability. The goal of CVL standardization is to create a new standard notation. *In contrast, the goal of* I*nvar is to integrate existing notations, rather than defining a new one.* This is the main difference between CVL and I*nvar*. Nevertheless, some CVL concepts like *configurable units*(consisting of a variability interface) and their composition with other *configurable units* can be compared to the composition of different configuration services in I*nvar*.

VII. Conclusions and Future Work

In this paper, we presented an approach to facilitate exchange of variability models during product configuration (regardless of the techniques, notations and tools in use in an organization) and to make these models available across organizational boundaries. Based on an illustrative example, we defined the basic user interactions required for configuration in general and mapped these interactions to the concrete semantics of individual modeling approaches, i.e., an approach for feature modeling and an approach for decision modeling. We provide a technical infrastructure and a prototypic implementation of an integrative approach based on Web Services. We have shown the feasibility of the approach and its implementation by applying it for two different variability modeling approaches and showing that we are able to integrate them in the context of an example ERP system.

In the future we plan to formalize the dependencies specified using the IMDI links, improve and extend the I*nvar* approach to support more complex scenarios: (i) The participating teams may work at rather different levels of abstraction and may consider variability at different levels of granularity. It

should be possible to aggregate the different models, e.g., by merging or splitting decision points. (ii) Organizations may not be willing to share their models, e.g., to protect their intellectual property. Hence, there should be mechanisms allowing the protection or limited visibility of models. (iii) It is equally important to identify and establish ownership of the models and governing model maintenance by establishing change management processes. There might be situations where variability that appears to be the same at the surface (e.g., when configuration options are presented to the end-user) differs when analyzing its implementation. Such semantic dissonances can be very difficult to detect and reconcile. We also plan to integrate more modeling approaches to $Invar$. This will allow us to fully validate the approach and the platform.

ACKNOWLEDGEMENTS

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero; by the Christian Doppler Forschungsgesellschaft, Austria and Siemens VAI Metals Technologies; by the European Commission (FEDER) and Spanish Government under project SETI (TIN2009-07366); by the Andalusian Government under ISABEL and THEOS projects (TIC-2533, TIC-5906) and by Siemens Corporate Technology CT T CEE.

REFERENCES

[1] J. Bosch, "From software product lines to software ecosystems," in *SPLC 2009*. San Francisco, CA, USA: ACM ICPS Vol. 446, Carnegie Mellon University, 2009, pp. 111–119.

[2] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," *Information & Software Technology*, vol. 49, no. 7, pp. 717–739, 2007.

[3] L. Chen, M. Babar, and N. Ali, "Variability management in software product lines: A systematic review," in *SPLC 2009*. San Francisco, CA, USA: ACM ICPS Vol. 446, Carnegie Mellon University, 2009, pp. 81–90.

[4] L. B. Lisboa, V. C. Garcia, D. L. dio, E. S. de Almeida, S. R. de Lemos Meira, and R. P. de Mattos Fortes, "A systematic review of domain analysis tools," *Information and Software Technology*, vol. 52, no. 1, pp. 1 – 13, 2010.

[5] R. Rabiser, R. Wolfinger, and P. Grünbacher, "Three-level customization of software products using a product line approach," in *42nd Annual Hawaii International Conference on System Sciences*. Waikoloa, Big Island, HI, USA: IEEE CS, 2009, p. 10.

[6] D. Benavides, S. Segura, and A. Ruiz-Corts, "Automated analysis of feature models 20 years later," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[7] D. Dhungana, P. Grünbacher, and R. Rabiser, "The DOPLER metatool for decision-oriented variability modeling: A multiple case study," *Automated Software Engineering*, vol. 18, no. 1, pp. 77–114, 2011.

[8] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez, "FaMa framework, http://www.isa.us.es/fama," in *SPLC 2008*, 2008, p. 359.

[9] R. Rabiser, P. Grünbacher, and D. Dhungana, "Requirements for product derivation support: Results from a systematic literature review and an expert survey," *Information and Software Technology*, vol. 52, no. 3, pp. 324–346, 2010.

[10] R. Rabiser, P. Grünbacher, , and D. Dhungana, "Supporting product derivation by adapting and augmenting variability models," in *SPLC 2007*, 2007, pp. 141–150.

[11] G. Botterweck, M. Janota, and D. Schneeweiss, "A design of a configurable feature model configurator," in *VaMoS*, ser. ICB Research Report, D. Benavides, A. Metzger, and U. W. Eisenecker, Eds., vol. 29. Universität Duisburg-Essen, 2009, pp. 165–168.

[12] S. Segura, R. Hierons, D. Benavides, and A. Ruiz-Cortés, "Automated metamorphic testing on the analyses of feature models," *Information and Software Technology*, vol. In Press, Corrected Proof, pp. –, 2011.

[13] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[14] K. Schmid, R. Rabiser, and P. Grünbacher, "A comparison of decision modeling approaches in product lines," in *5th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2011)*. Namur, Belgium: ACM Press, 2011, pp. 119–126.

[15] R. Rabiser, D. Dhungana, W. Heider, and P. Grünbacher, "Flexibility and end-user support in model-based product line tools," in *EUROMICRO-SEAA*. IEEE Computer Society, 2009, pp. 508–511.

[16] H. Hartmann and T. Trew, "Using feature diagrams with context variability to model multiple product lines for software supply chains," in *SPLC 2008*. IEEE Computer Society, 2008, pp. 12–21.

[17] H. Hartmann, T. Trew, and A. Matsinger, "Supplier independent feature modelling," in *SPLC 2009*. San Francisco, CA, USA: ACM ICPS Vol. 446, Carnegie Mellon University, 2009, pp. 191–200.

[18] S. Deelstra, M. Sinnema, and J. Bosch, "Product derivation in software product families: a case study," *Journal of Systems and Software*, vol. 74, no. 2, pp. 173–194, 2005.

[19] J. Bosch, "The challenges of broadening the scope of software product families," *Commun. ACM*, vol. 49, no. 12, pp. 41–44, 2006.

[20] S. Bühne and K. Lauenroth, "Modelling requirements variability across product lines," in *13th International Conference on Requirements Engineering*. IEEE CS, 2005, pp. 41– 50.

[21] R. C. van Ommering, "Software reuse in product populations," *IEEE Trans. Software Eng.*, vol. 31, no. 7, pp. 537–550, 2005.

[22] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer, "Structuring the modeling space and supporting evolution in software product line engineering," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1108–1122, 2010.

[23] M.-O. Reiser and M. Weber, "Managing highly complex product families with multi-level feature trees," in *14th IEEE International Requirements Engineering Conference (RE'06)*. Minneapolis, MN, USA: IEEE CS, 2006, pp. 149–158.

[24] K. Schmid, "Variability modeling for distributed development - a comparison with established practice," in *SPLC 2010*, 2010, pp. 151–165.

[25] K. Czarnecki, S. Helson, and U. Eisenecker, "Staged configuration using feature models," in *SPLC 2004*, R. Nord, Ed. Springer-Verlag, 2004, vol. LNCS 3154, pp. 266–283.

[26] M.-O. Reiser, R. T. Kolagari, and M. Weber, "Compositional variabilityconcepts and patterns," in *42nd Hawaii International Conference on System Sciences*. Waikoloa, Hawaii, USA: IEEE Computer Society, 2009, pp. 1–10.

[27] A. Hubaux, A. Classen, and P. Heymans, "Formal modelling of feature configuration workflows," in *SPLC 2009*. Pittsburgh, PA, USA: ACM ICPS Vol. 446, Carnegie Mellon University, 2009, pp. 221–230.

[28] A. Hubaux, P. Heymans, and D. D. Pierre-Yves Schobbens, "Towards multi-view feature-based configuration," in *16th International Working Conference on Requirements Engineering, REFSQ 2010)*. Essen, Germany: Springer, 2010, pp. 106–112.

[29] K. Czarnecki, M. Antkiewicz, and C. H. P. Kim, "Multi-level customization in application engineering," *Commun. ACM*, vol. 49, no. 12, pp. 60–65, 2006.

[30] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, and G. Saval, "Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis," in *15th IEEE International Requirements Engineering Conference (RE'07)*. New Delhi, India: IEEE CS, 2007, pp. 243–253.

[31] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[32] T. Moser, F. Waltersdorfer, A. Zoitl, and S. Biffl., "Version management and conflict detection across heterogeneous engineering data models," in *Proc. 8th IEEE International Conference on Industrial Informatics (INDIN 2010), Osaka, Japan*, 2010, pp. 928–935.

[33] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen, "Developing a software product line for train control: A case study of CVL," in *SPLC*, 2010, pp. 106–120.