Supporting Distributed Product Configuration by Integrating Heterogeneous Variability Modeling Approaches

José Angel Galindo Duarte, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, Paul Grünbacher

Supporting Distributed Product Configuration by Integrating Heterogeneous Variability Modeling Approaches

José A. Galindo[1,5], Deepak Dhungana[2], Rick Rabiser[4], David Benavides[1], Goetz Botterweck[3] and Paul Grünbacher[4]

[1]Department of Computer Languages and Systems, University of Seville, Spain, {jagalindo,benavides}@us.es
[2]Siemens AG Österreich, Corporate Technology, Vienna, Austria, deepak.dhungana@siemens.com
[3]Lero–The Irish Software Engineering Research Center, University of Limerick, Ireland, goetz.botterweck@lero.ie
[4]Christian Doppler Laboratory MEVSS, Johannes Kepler University, Linz, Austria, rick.rabiser@jku.at
[5]Diverse Team, INRIA Rennes, France, jagalindo@inria.fr

November 9, 2015

Abstract

Context: In industrial settings products are developed by more than one organization. Software vendors and suppliers commonly typically maintain their own product lines, which contribute to a larger (multi) product line or software ecosystem. It is unrealistic to assume that the participating organizations will agree on using a specific variability modeling technique—they will rather use different approaches and tools to manage the variability of their systems. **Objective:** We aim to support product configuration in software ecosystems based on several variability models with different semantics that have been created using different notations. **Method:** 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 prototype implementation based on web services. **Results:** We show the feasibility of the approach and its implementation by using it with the three most widespread types of variability modeling approaches in the product line community, i.e., feature-based, OVM-style, and decision-oriented modeling. To demonstrate the feasibility and flexibility of our approach, we present an example derived from industrial experience in enterprise resource planning. We further applied the approach to support the configuration of privacy settings in the Android ecosystem based on multiple variability models. We also evaluated the performance of different model enactment strategies used in our approach. **Conclusions:** Tools and techniques allowing stakeholders to handle variability in a uniform manner can considerably foster the initiation and growth of software ecosystems from the perspective of software reuse and configuration.

1 Introduction and Motivation

Software product lines (SPL) are increasingly developed beyond the boundaries of single organizations [1]. For instance, in software ecosystems distributed organizations and 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 diverse practices in different domains it is unrealistic to assume the use of a single and standardized variability modeling approach, despite ongoing standardization efforts[1]. However, the increasing number of "island solutions" to variability modeling and product configuration hinders communication and collaboration between product line engineers. Especially in software ecosystems [1] it is infeasible to assume one kind of modeling approach for all units of the ecosystem. The required coordination between the participating organizations (e.g., along a supply chain) further complicates this issue. Hence, there is a strong need for an integrative infrastructure enabling the collaboration between different organizations developing product lines. The approach needs to support different variability modeling languages, notations, and tools. It also needs to support variability at different levels of granularity to model, e.g., customer-facing features, architectural elements, or configuration decisions.

We propose the I*nvar* approach, which facilitates the integration of heterogeneous variability models[2] potentially created by different teams. In this paper we focus on the product configuration aspects of our integrative infrastructure. I*nvar* deliberately hides the internal technical aspects of using different variability models for configuration from the stakeholders performing the configuration. The specific tools or data formats (see [3, 4, 5]) used for defining the variability models are not relevant for the end users who primarily focus on the available configuration choices and their implications. I*nvar* unifies configuration operations on variability models and allows modelers to freely choose a data representation by accessing variability models through web services. Our approach does *not* force organizations to *integrate* their configuration tools by adapting the internals of the tools. Instead, we allow them to *compose* their configuration mechanisms using wrappers and interface definitions. We validate our approach by integrating three different variability modeling "dialects", i.e., feature modeling, orthogonal variability modeling (OVM), and decision modeling. We also show how typical scenarios in software ecosystems can be supported with I*nvar* and assess the performance of I*nvar* regarding the different model enactment strategies of the approach, e.g., different orderings and settings of choices during product configuration.

An earlier version of this work appeared in [6] and the tool prototype was presented in a short tool demonstration paper [7]. In this article *we provide the following extensions* to this earlier work: (i) we discuss requirements for configuration based on multiple variability models (in different notations) in a software ecosystem context and relate the requirements to the literature; (ii) we describe the integration of a third variability modeling technique based on OVM [8], thus broadening the scope of our work; (iii) we extend I*nvar* with different model enactment strategies allowing different orders in a configuration process based on multiple models; (iv) we present an experiment assessing the performance of the enactment strategies; (v) we demonstrate the feasibility and flexibility of our I*nvar* approach by applying it to a realistic scenario, i.e., the configuration of the permission system in the Android ecosystem[3] based on multiple variability models; and (vi) we discuss the benefits of I*nvar*, e.g., by comparing it to manual configuration based on multiple variability models.

The main contributions of this paper are:

- a set of *configuration* primitives for integrating arbitrary variability modeling approaches to support product configuration;
- a *method for defining dependencies between multiple variability models*;
- support for *composing and integrating heterogeneous variability modes* such as feature models, decision models, and orthogonal variability models;

[1] OMG Common Variability Language [2], `http://www.omgwiki.org/variability/doku.php`

[2] 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

[3] `android.google.com`

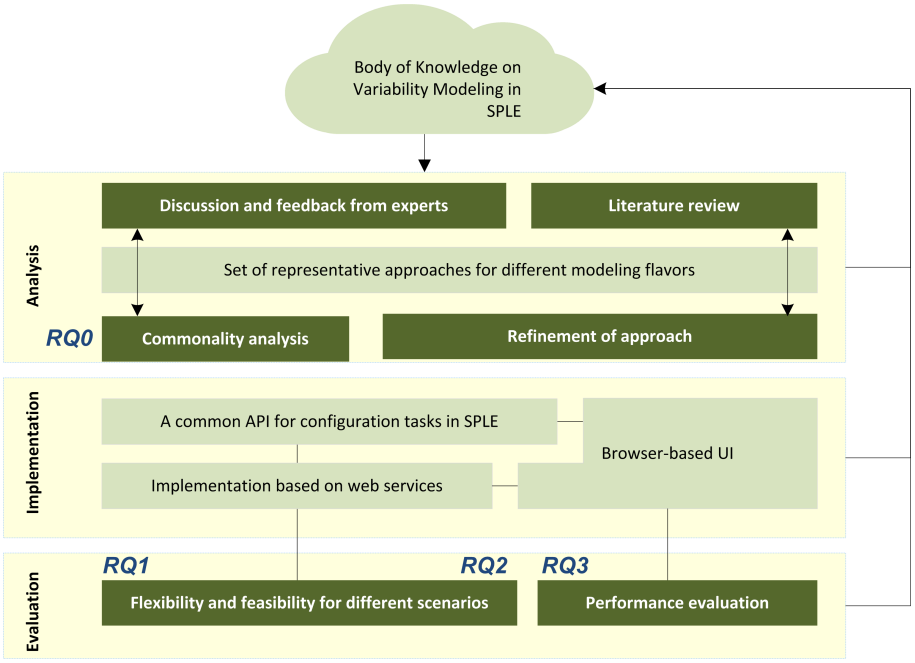

Figure 1: Overview of research approach.

- support for *different enactment strategies* during product configuration with multiple variability models to customize configuration orderings;
- *evidence regarding the feasibility of* Invar for different configuration scenarios by using it in an enterprise resource planning (ERP) context and for the Android ecosystem; and
- a *discussion of the benefits of* Invar compared to manual configuration based on multiple variability models.

The remainder of this paper is structured as follows: In Section 2 we outline our research questions and approach. In Section 3 we discuss the background of this work, i.e., software product lines and the three different variability modeling approaches we later integrate with our approach. In Section 4 we present an example of a multi product line as a further motivation for our work. Section 5 describes how Invar enables the use of heterogeneous variability models during product configuration. Section 6 presents the Invar prototype for three different variability modeling tools. Section 7 presents a validation of the flexibility of our approach using different scenarios of applying the approach derived from industrial experience in the ERP domain and from the Android ecosystem. We also present a study of the performance of our approach regarding its model enactment strategies. We discuss the benefits of Invar (e.g., when compared to manual configuration based on multiple models) and explicate the threats to validity of our research results. We discuss related work in Section 8 and conclude the paper with a summary and discussion of future work in Section 9.

2 Research Approach

We explore the following four research questions:

RQ1: How can different variability modeling approaches be integrated to support product configuration in the context of software ecosystems?

RQ2: Is *Invar* sufficiently extensible and flexible to allow the integration of different variability modeling approaches?

RQ3: Does *Invar* support realistic configuration scenarios in software ecosystems?

RQ4: What is the impact of the *Invar* model enactment strategies on configuration performance?

In order to address these questions we followed the research process shown in Figure 1, comprising the stages of analysis, implementation, and evaluation.

Analysis (RQ1). Our main hypothesis is that different existing variability modeling flavors can be integrated to support product configuration in a software ecosystem context. To validate this idea, we first discussed it with experts from the SPL community and performed a literature review. Later, we extracted the commonality and variability among the different product line modeling approaches. There exist several families of variability modeling approaches that have been designed for different research scenarios [9], resulting in a large number of tools, languages, and operations. The variety of variability model flavors leads to obstacles when different organizations collaborate to configure products. For instance, the relationships among product lines and their variability must be defined, configuration front-ends must be integrated, collaborative and distributed configuration must be supported, and different configuration scenarios must be taken into account. We analyzed these issues further to understand the key challenges and defined requirements related to product lines across organizational boundaries. We discuss these challenges and requirements based on a motivating example in Section 4.

Implementation. We implemented a solution enabling the joint use of different variability modeling languages by providing a usable front-end for users configuring products based on multiple models in different notations. Our implementation is based on a set of configuration primitives for arbitrary variability modeling approaches and a method for defining dependencies between multiple variability models. We developed a common configuration application programming interface (the *Invar* API) to support different enactment strategies for variability models, e.g., different orderings and settings of choices during product configuration.

Evaluation (RQ2, RQ3, RQ4). Finally, we implemented the *Invar* API and an integrative infrastructure based on web services. We applied the *Invar* approach and integrated three different types of variability models (RQ2) – i.e., feature models, orthogonal variability models (OVM), and decision models. To demonstrate the feasibility of the approach for realistic configuration scenarios (RQ3), we describe its use in two different cases, one derived from industrial experience in the ERP domain and one from the Android ecosystem. In both scenarios, configuration is based on multiple variability models created with different modeling approaches and integrated with *Invar*. We then analyzed *Invar* regarding the performance of its different model enactment strategies (RQ4). Performance is essential as it is important for users to avoid lags between the different configuration steps [10], in particular when combining large and complex models.

3 Background: Software Product Lines and Variability Models

We briefly describe basic concepts of software product lines and variability modeling. We highlight three flavors of variability modeling approaches, i.e., feature modeling, orthogonal variability modeling (OVM), and decision modeling. These approaches share the goal of providing a systematic method for describing reusable software artifacts such that practitioners can configure and adapt the artifacts to match their requirements. We use parts of the example shown in Figure 3 to illustrate the concepts. The larger

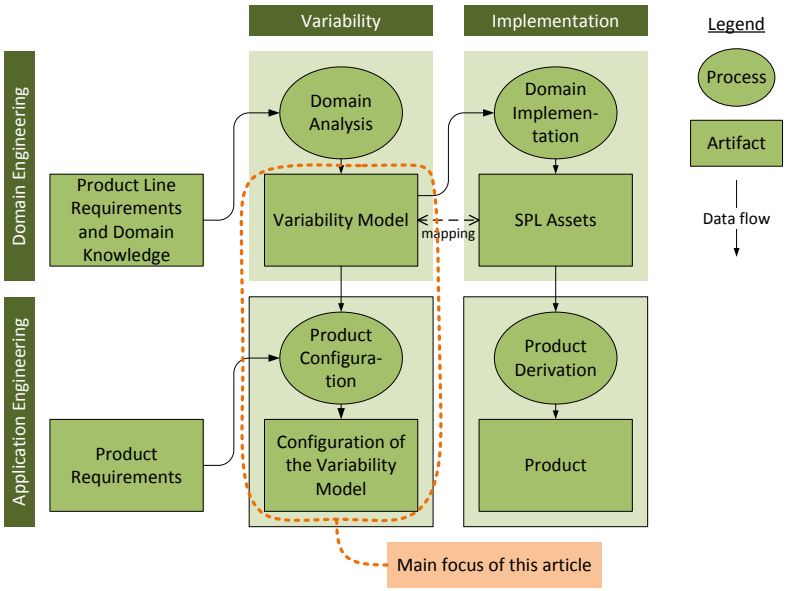

Figure 2: Schematic overview of Software Product Line Engineering (SPLE)

context and details of how these parts fit together will be discussed later in Section 4.

Software Product Lines. Software Product Line Engineering (SPLE) often distinguishes two life cycles, i.e., *Domain Engineering* to establish the product line and *Application Engineering* to derive and configure products based on the models and assets provided by the product line [8]. Other frameworks suggest a similar distinction but speak, for instance, of *Core Asset Development* and *Product Development* [11]. The overview in Figure 2 shows the key activities in software product line engineering. In this article we focus mainly on the left part, i.e., variability modeling and product configuration. The right part concerned with the implementation of a product line's core assets and derivation of concrete (implementation) artifacts is out of scope of this article.

In domain engineering the configuration options supported by the software product line are typically defined using a *Variability Model*. We use this term as a generalization for three flavors of models, i.e., feature models, the orthogonal variability model, and decision models, which will be introduced in the following subsections. Please note that many different variants exist even in each of these families of modeling approaches as shown by Czarnecki *et al.* [9]. Here, we focus on the key concepts common to all approaches: we assume that a variability model describes the configuration choices for deriving products from a product line, i.e., it defines the set of all products that can be derived. The process of *Product Configuration* uses a variability model to pick one product out of this set by making configuration decisions that narrow down the set of possible products in multiple stages. The resulting configuration of the variability model can then be used to derive a concrete implementation of the chosen particular product.

Feature Models. Feature modeling is currently the most widely used approach for modeling variability [12, 9]. A feature model is typically designed to capture stakeholder-visible characteristics and aspects of a system, such as functional and non-functional features of individual products. Starting with FODA (Feature Oriented Domain Analysis [13]) in 1990, numerous variants of feature-based variability modeling tools and techniques have been developed (see [14] for an extensive list). 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 (or compound) feature and its child features (or subfeatures) as well as cross-tree (or cross-hierarchy) 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 excluded).* For instance, the ERP Vendor Feature Model shown in Figure 3 defines the features CRM, Project Management and Accounting as optional subfeatures of the feature ERP Solution.

The Orthogonal Variability Model (OVM). OVM is commonly used for documenting software product line variability [8]. In an OVM model a variation point (VP) defines a variable item that can vary from one product to another. Variants (V) document how the variation points can vary. For instance, in the OVM model Calendar Supplier 1 shown in Figure 3 Messaging is a VP and SMS and MMS are Vs. All VPs are related to at least one V and each V is related to exactly one VP. Both VPs and Vs can be either optional or mandatory. A mandatory VP must be bound in all products of the product line which is not the case for optional VPs. Binding a VP means making a decision about its variants. A mandatory variant has to be always selected when its parent VP is bound. An optional variant can be optionally selected when its parent VP is bound. In OVM, optional variants may be grouped into alternative choices, i.e., SMS, MMS, or both can be selected as Vs of VP Messaging. This group is associated to a cardinality [min...max] prescribing constraints on how many Vs may be chosen in an alternative choice, i.e., at least *min* and at most *max* Vs of the group. As in feature models cross-type relationships may be defined in OVM between Vs or VPs [8, 15].

Decisions Models. The family of Decision Modeling approaches [16] exists nearly as long as feature-oriented modeling. Similar to the role which FODA [13] plays in the context of feature-based variability management most if not all decision modeling approaches have been influenced by the Synthesis method [17]. A decision model describes the differences between products in a product line. It uses a set of multiple related decisions (represented as configuration variables) that need to be made by the user when configuring a product. Decisions are often represented as questions with a defined set of possible answers. Products are derived from a decision model by setting values to the decisions, e.g., by answering questions and following the sequence defined by the decisions' dependencies. The set of values possible for a decision is defined by its data type, e.g., Boolean (to select or unselect an option), Enumeration/Set (to select from a set of possible answers), Number (to enter a numerical value), or String (to set a text as an answer). For instance, in the Decision Model in Figure 3 the Bank Transfer Add-In Supplier defines a Boolean decision 'Do you want support for SWIFT codes?' and a number decision 'Max. time period for transaction cancellation?'.

The three modeling approaches share **common characteristics**. A *Variability Model* consists of a set of *Variables* and *Constraints* over these variables. Each variable has a *Type*, for instance, Boolean, Integer, or String. Depending on the approach there are different *Types of Constraints*, e.g., "Optional Subfeatures" (e.g., feature Accounting is an optional sub-feature of ERP Solution) and "Alternative Groups" (e.g., different alternative web access options). A modeler creates a set of variables and 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 compliant configurations. Users implicitly or explicitly adds more constraints as they makes 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 capabilities. If no valid assignment is possible the model is unsatisfiable.

4 Motivating Example and Discussion of Requirements

We illustrate the challenges of software variability management and product configuration in multi product line environments using an example of an Enterprise Resource Planning (ERP) system. Even though

the concrete example as given here is fictitious, it is based on a real-world product line of an industry partner – a medium-sized vendor from the ERP domain [18]. The company 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. While for the sake of simplicity we only use this example throughout the paper to motivate and explain our approach, we also demonstrate its feasibility and flexibility when using it to support the configuration of the Android permission system based on multiple variability models (cf. Section 7.3).

In the example depicted in Figure 3, 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 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 configured software products. Dependencies among models may also be the result of the role taken by the product, e.g., the position in a supply chain or the specific purpose of the model such as technical configuration, marketing, or documentation. The relationships play an important role when the models are used together during end-user product configuration of the ERP system, e.g., they influence which configuration choices are valid.

The example model presents a common scenario: the ERP vendor defines in a feature model that a Calendar feature is available 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 its calendar; the other supplier uses a feature model (in a different notation than the vendor feature model). The calendar of Supplier 2 can further provide extended capabilities based on another supplier. Similarly, the vendor's accounting feature is provided by a supplier who again relies on other suppliers using decision models to describe the variability of their systems.

As demonstrated by the ERP example and confirmed by the literature [19, 20, 21, 19], companies currently are employing diverse solutions for modeling variability and supporting product configuration. For the configuration of a common product in such a context, an integrative infrastructure is needed, which works on concepts shared among the different modeling notations. This imposes several research issues and practical challenges: when attempting to integrate different variability models, one has to consider interfaces between the models. For instance, there should be mechanisms for defining dependencies across models (cf. Figure 4). However, it is important to allow unrelated models to change independently and minimize coupling between related models, to ease their management and evolution. Moreover, it is mandatory to provide a configuration front-end for end users that provides an integrated view on the configuration choices defined in different models. More specifically, based on our motivating ERP example and our research questions, we state five requirements for a potential solution. The definition of these requirements mainly grounds in the industrial experience of the authors [22, 23, 24, 25]. We followed a goal-oriented requirements engineering approach [26], i.e., we iteratively refined initially defined requirements in several workshops among the authors of this article. We also related the requirements to the literature.

R1–Inter-model relationships (cf. RQ1 and RQ2). In a multi-model configuration scenario, end users need support for making configuration choices independently of the model encoding these choices. Modelers thus need support for defining dependencies between model elements (i.e., features, variants, decisions) and dependencies between models (e.g., it may be only required to configure the accounting software when enabling the feature accounting in the ERP solution). Existing work such as Reiser *et al.*'s work on compositional variability [27] and Seibel *et al.*'s work on mega models [28] confirms this requirement. The requirement is related to RQ1 and RQ2.

R2–Application programming interface (cf. RQ1 and RQ2). It is important to decouple vari-

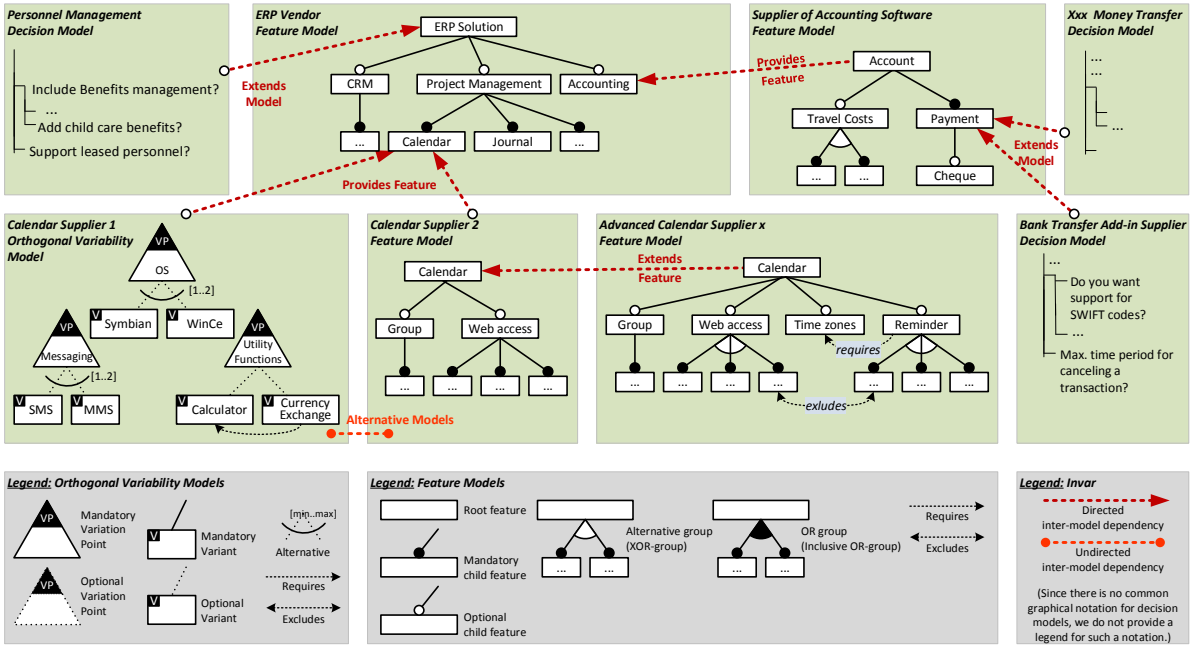

Figure 3: Example of interrelated product lines in the ERP domain. Vendors and suppliers depend on each other's products, hence the variability models are related to each other in different ways. In *Invar* the relationships between the models are expressed as IMDI links (see Table 2). The red lines demonstrate the different possible relationships.

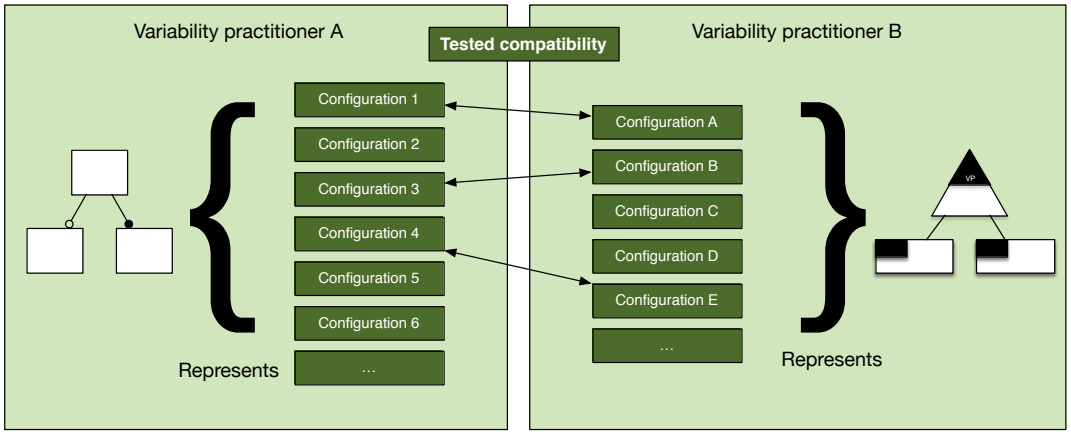

Figure 4: When configuring based on multiple variability models relations between configuration choices need to be defined.

ability model implementation and variability model usage. The detailed semantics of model elements and models should be hidden from end users. An API thus is needed through which configuration front-ends can transparently access the configuration functionality. This allows abstracting from the specifics of the different variability modeling approaches used in the system. For instance, there are different tools for

analyzing and configuring each type of variability model, e.g., FaMa [29] or FAMILIAR [30] are available for feature models. The API should provide a common entry point to integrate these different tools. This requirement is related to RQ1 and RQ2.

R3–Integration of multiple distributed variability model repositories (cf. RQ1 and RQ2). The organizations contributing to software ecosystem with multiple product lines use different model repositories [31, 32]. For example, the companies providing the two calendar software applications will typically host the calendar variability models in different places. Moreover, some providers may want to hide their model implementation and architecture from others. Thus, it is important to provide mechanisms that enable this multi-tenant configuration process while leaving the configuration back-ends isolated. This requirement is related to RQ1 and RQ2.

R4–Collaborative and distributed configuration sessions (cf. RQ3 and RQ4). Depending on the model size the configuration process may take significant time for users. In large systems configuration will thus typically be carried out in multiple sessions and in a staged manner [33], i.e., intermediate results are persisted and picked-up again at a later time. For example, in the described example 50 individual configuration choices have to be made based on multiple models. Also, is it important that the users of the platform do not experience any lag[4] in between the different configuration steps [10]. This requirement is related to RQ3 and RQ4.

R5–Support for different configuration settings (cf. RQ3). A spectrum of different configuration processes is necessary when involving multiple organizations [34]. For example, in a *collaborative setting* a vendor of a product aggregates the final product by integrating contributions from multiple suppliers. In a *competitive setting* multiple alternative suppliers are available for the concrete realization of a certain feature. In *complementary settings* third-party suppliers reuse and extend the functionality of a common platform.

5 The Invar Approach

Invar[5] addresses requirements R1-R5 through a "plug-and-play" approach for managing the integration of variability models: "plugging" means adding new variability models to a shared repository while "playing" means presenting the configuration options from multiple variability models to end users during configuration. In Invar a variability model is treated as an autonomous entity, which can be plugged into the configuration space to provide configuration options. Autonomy however does not necessarily mean independence as variability models may be related to each other (cf. the ERP example). 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 and can use the services without knowing details about the concrete variability models made available by the services. Note that while invar supports multiple and diverse variability models, it is not required to have a semantic 1:1 mapping. Instead, developers should determine how to configure their models. Moreover, there are circumstances where more than one mapping can be acceptable. For example, when coping with feature models having attributes, some developers may want to configure their value [29] while others will assign a concrete value in the model representation [35].

Based on the common characteristics of variability models described in Section 3, we defined a set of web services for accessing and using variability models. The operations and queries on the models, which are required for developing such web services, are based on our experience of developing product configuration applications [14, 36, 37, 29, 38, 39]. Figure 5 depicts the Invar approach compared to the

[4] `http://www.nngroup.com/articles/response-times-3-important-limits/`

[5] 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 benefit from 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".

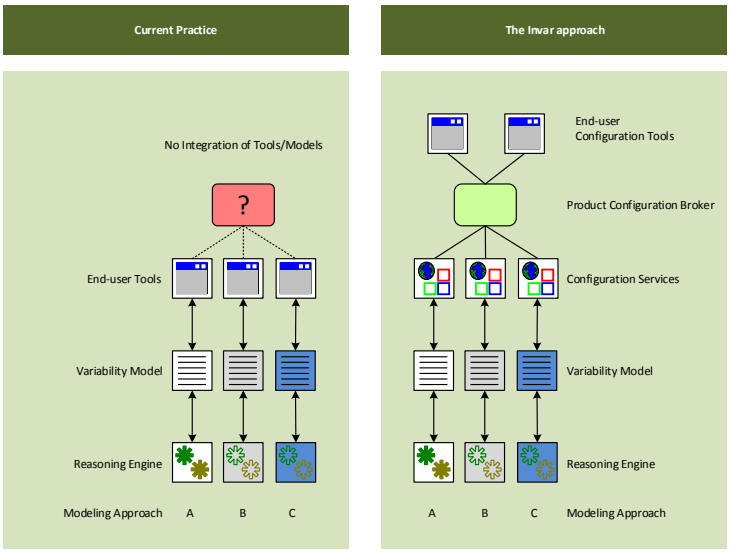

Figure 5: A simplified view of model-based product configuration: current state of practice (left) and the Invar approach (right).

current state of practice. In the current state of practice multiple heterogeneous variability modeling approaches are used by different organizations (left side). Different reasoning and analysis engines – for instance, SAT solvers [14] or rule engines [36] – are adopted for interpreting the models' semantics. There is no integration of the diverse tools supporting different notations. In practice organizations frequently define variability in configuration files or spreadsheets, which are typically not integrated with other variability modeling tools.

Using Invar, stakeholders create variability models using an approach of their choice (see right side of Figure 5). Invar defines key operations and queries (*configuration primitives*) on variability models to allow the integration of heterogeneous approaches. These configuration primitives are realized as operations of a configuration service to allow uniform access to the models and are managed by the *configuration broker* component. The participating organizational units can reuse variability models from other units even if they use different modeling approaches. Invar provides a single and transparent configuration tool to end users. This ensures interoperability and allows reusing variability models in different contexts. For example, one model may be shared between several companies and each one may use it to create different products. Moreover, Invar relies on the concept of *inter-model relationships* to allow the description of restrictions between multiple variability models. The participating organizational units could also create their own configuration tools and access diverse models via the Invar *configuration services* without having to know all the details about the actual modeling approaches underneath. It is even possible to integrate textual configuration files or spreadsheets into Invar by defining new configuration services that interpret these representations. Finally, Invar relies on end-user configuration applications to interact with the users, thereby allowing users to use the more appropriate user interface for each case (e.g., web sites, internet bots).

5.1 Question Types and 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 [36, 29, 39] allowing model-based product configuration. For example, the end user may query the set of all available options at a time, or 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:

Question Types: Invar relies on the concept of question types to allow the proper selection of elements within a configuration process. A question type represent the way a decision to select the elements present in a configuration: i) an optional question presents a set of optional items to be selected by the user without restrictions in the number of selections the user has to make; ii) an alternative question represents a decision where the user have to choose one item from a set of them; iii) a more-than-one question represents a decision where the user must select at least one item from a set of choices. Internally, these question types are mapped into the different variability modeling relationships and thus the user decisions are propagated to the back-end configuration tools.

Loading and initializing models: load() transfers models from their persistent storage to memory, while reload() loads the model again from memory. 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. The approach only assumes that a question about an option or choice can be asked to a user. Analogously, previousQuestion() allows to obtain the previous question. peek() allows previewing the next available question without having to answer it. Walking through options step by step requires defining a certain order. For instance, for a tree-based structure this could be done in a breadth-first or depth-first manner. In our ERP example, such a walk-through would start with a question about what key features the ERP solution should have (CRM, Project Management, Accounting) and could then continue with detailed questions on CRM, Project Management, or Accounting, depending on the answer to the first question and the related models. It could however also start with a question: "Do you need CRM support?", continue with detailed questions on CRM, and only later ask "Do you need Project Management support?".

Operating on the available options: The operation setValue() allows assigning a value to an option (e.g., it sets feature attributes or sets answers to decisions). For instance, setting the value (CRM;Project Management) on question "What key features shall your ERP solution have?" would constrain the following questions to CRM-related and Project Management-related questions while excluding Accounting (since that optional feature was not selected in this case). The primitives select() and deselect() allow to explicitly decide about including or excluding a feature. This distinction is necessary as not selecting a feature is not the same as excluding it from the configuration. undo() and redo() allow canceling the last action or replaying the last selection. addOption() adds new options to available questions. Note that the validity of a configuration relies on the implementation of each specific tools. This is, when the user selects and option is checked with the tool providing support for that concrete model flavor. For example, if an user selects an option representing a feature from a feature model, FaMa is in charge of checking the validity of the selection. By definition, we do not allow conflicting inter-model relationships.

Notifications: The success() operation 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 users or other tools about user actions.

Obviously, this set of primitive operations is neither complete nor fixed. Depending on the end-user application there may be other operations that are useful in building complex user interfaces. For instance, in some cases, several options can be proposed at the same time. Our approach thus can easily be extended to include new operations or queries on the models.

Table 1: A summary of currently supported operations (conditions and actions), which can be used to create IMDI links for the *Invar* approach to manage inter-model-dependencies. This list may be extended as required to build more complex relationships between models.

Operation	Type	Description
isInit()	condition	This condition only evaluates to 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 configuration service.
isSelected()	condition	This condition 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 condition 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	Add an option to one model as a child of an existing option. This is usually required when a model extends another model.
inform(...), warn(...), recommend(...)	action	Display a specified message to the user, which can have the type *information*, *recommendation* or *warning*.

5.2 Inter-model Dependencies

Whenever a variability model is plugged into the *Invar* configuration environment, it needs to explicitly define its relationships to the other models (cf. the example in Section 4). 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 which can be compared to conventional cross-tree constraints regarding 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 1. 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. The conditions isSelected() and isDeselected() are used to specify such constraints. The corresponding actions are doSelect() and doDeSelect(). For instance, in our example in Figure 3, one could specify the link between Vendor.Accounting and Supplier.Account as "if Vendor.Accounting.isSelected() then Supplier.Account.doSelect()".

Informative actions: Modelers can also define actions such as inform(), recommend() or warn() that are intended to inform the users without changing the models.

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 3 the link between Calendar Supplier1 and ERP Vendor can be specified as "if Vendor.Calendar.isSelected() then Supplier1.includeModel()".

Table 2: Examples of IMDI Links for the inter-model dependencies depicted in Figure 3.

Inter-model dependency	**Example of IMDI link in *Invar***
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()

5.3 Configuration Service and Enactment Support

Central to the *Invar* approach 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 configuration service is designed such that the configuration options are presented to the end user as questions. Questions are only a means to present the variation point to the user. This means the user is asked questions about a certain "feature" (in the wider sense) or a property of the system 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 *Invar* configuration service.

The interface consists of two parts: the *variability model query part* provides basic information about models (e.g., the set of available questions and the possible answers). The *operational part* directly interacts with the 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 combo boxes 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).

When configuring a model in *Invar* the user is prompted to answer questions comprising a set of options a user has to select or deselect. The order in which such questions are presented to the user can vary depending on different criteria. *Invar* provides two mechanisms to help product line engineers define in which order the questions need to be presented to the user.

The first mechanism uses **predefined orders** to present configuration options to the user. Three orders depending on the way a tree can be traversed, i.e., *in-order*, *pre-order* and *post-order*, are available for feature models. *Invar* also offers two ordering styles based on the number of options the questions have *"more to less"* and *"less to more"* available for feature models and orthogonal variability models. For decision-oriented models, the order of questions either is given through their visibility conditions [36] or can be defined similar to the definition used for feature models (tree traversal). Finally, *Invar* also supports an alphabetical order and a random order for all types of models. More details about the offered orders in each plugin are presented in Section 6.

Alternatively, *Invar* provides a second mechanism, i.e., an interface that an engineer can use to define a **custom order**. This interface defines two methods the engineer has to implement, i.e., getOrders() that returns the names of the orders and getQuestions(String order) that returns the list of sorted questions given a particular order.

The order in which questions are answered can be important from an end-user point of view but also can have an impact on the performance of the back-end configuration engines [40]. We present experiments to evaluate *Invar*'s performance regarding the different enactment strategies in Section 7.4.

5.4 Integrating Variability Models: The *Invar* Architecture

The five main components of the architecture of our infrastructure are shown in Figure 6 (numbered in the figure).

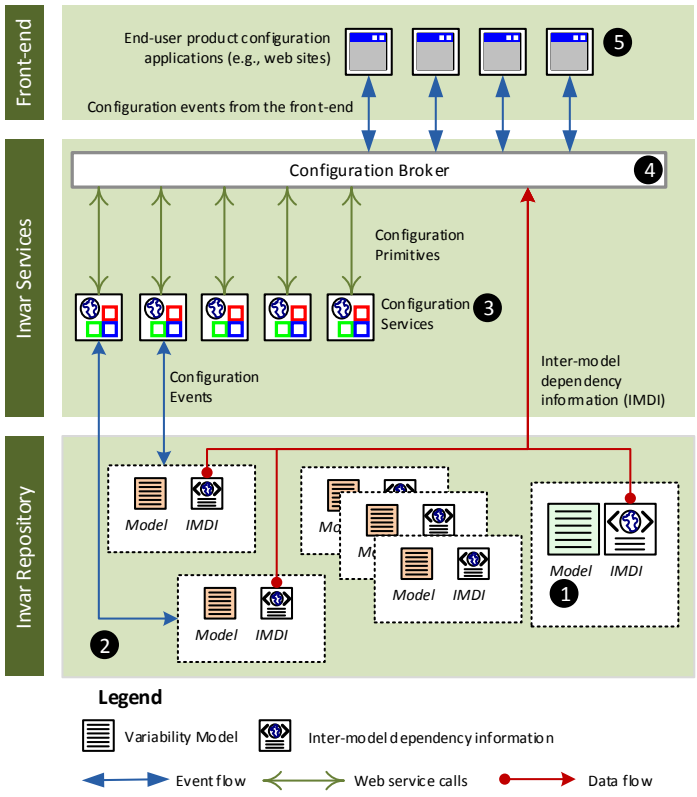

Figure 6: Architecture of the *Invar* infrastructure.

201 + 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.

201 + 2 *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.

201 + 3 *Configuration services:* The different models residing in (possibly distributed) repositories are accessed by configuration services. A configuration service provides a standard interface for different configuration front-ends such as websites, mobile devices, or stand-alone applications. 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. An easy implementation of configuration web services can be achieved through the use of web services.

201 + 4 *Configuration broker:* The configuration broker enables the communication between the configuration services. It reads the inter-model dependency information to determine which configuration services are affected when products are configured. The configuration broker also translates events from the end users and passes them on to the configuration services that need to react to the end user's interactions.

201 + 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 also shown in Figure 7.

6 Implementation

Our current implementation of *Invar* allows creating and maintaining repositories for sharing variability models. It further supports end-user configuration based on these models. The infrastructure relies on 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 *Invar* framework based on a provided WSDL description. For the front-end, we use a layered Java/J2EE application platform based on the Spring Framework for providing centralized, automated configuration and wiring of the application objects. Please note that while we use web services others approaches such as network sockets are also valid.

We chose web services as a base technology for the prototype, because we believe that capabilities of web services are well suited for addressing our research problem, i.e., composition of heterogeneous variability models. For instance, web services are inherently *heterogeneous*, which helps us to connect different variability models. The *flexible composition* of web services allows us to configure products dynamically and include different variability models from different web services into the configuration environment when they are needed [41]. In a typical architecture for serviced-based applications one can find a *provider*, a *consumer*, and a *registry*. All these three main components can also be found in the *Invar* framework. The different web services are providers of variability models and the *Invar* configuration site is the service registry and the consumer of the model information at the same time [42].

Figure 7 shows a screenshot of using the *Invar* application for our example from in Section 4. It shows the different features we are willing to offer, different inter-model relationships, as well as dependencies between multiple calendars suppliers.

7 Evaluation

We have discussed how different variability modeling approaches can be integrated (see Sections 4 and 5) and thereby addressed research question RQ1. Regarding RQ2 we demonstrate in Section 7.1 that *Invar* is flexible enough to allow the integration of different variability modeling approaches. Regarding RQ3 we evaluate the feasibility of our approach by applying it to three different realistic configuration scenarios in the ERP example and by using it to configure privacy settings in the Android ecosystem (Sections 7.2 and 7.3). Finally, regarding RQ4 we present in Section 7.4 a performance assessment of different *Invar* model enactment strategies.

7.1 RQ2: Integrating Three Different Variability Modeling Approaches

Regarding RQ2 – is *Invar* sufficiently extensible and flexible to allow the integration of different variability modeling approaches? – we have implemented the *Invar* service configuration interface for the feature-oriented FaMa tool suite [29], the OVM-oriented FaMa tool [43, 44] and the decision-based DOPLER [36]

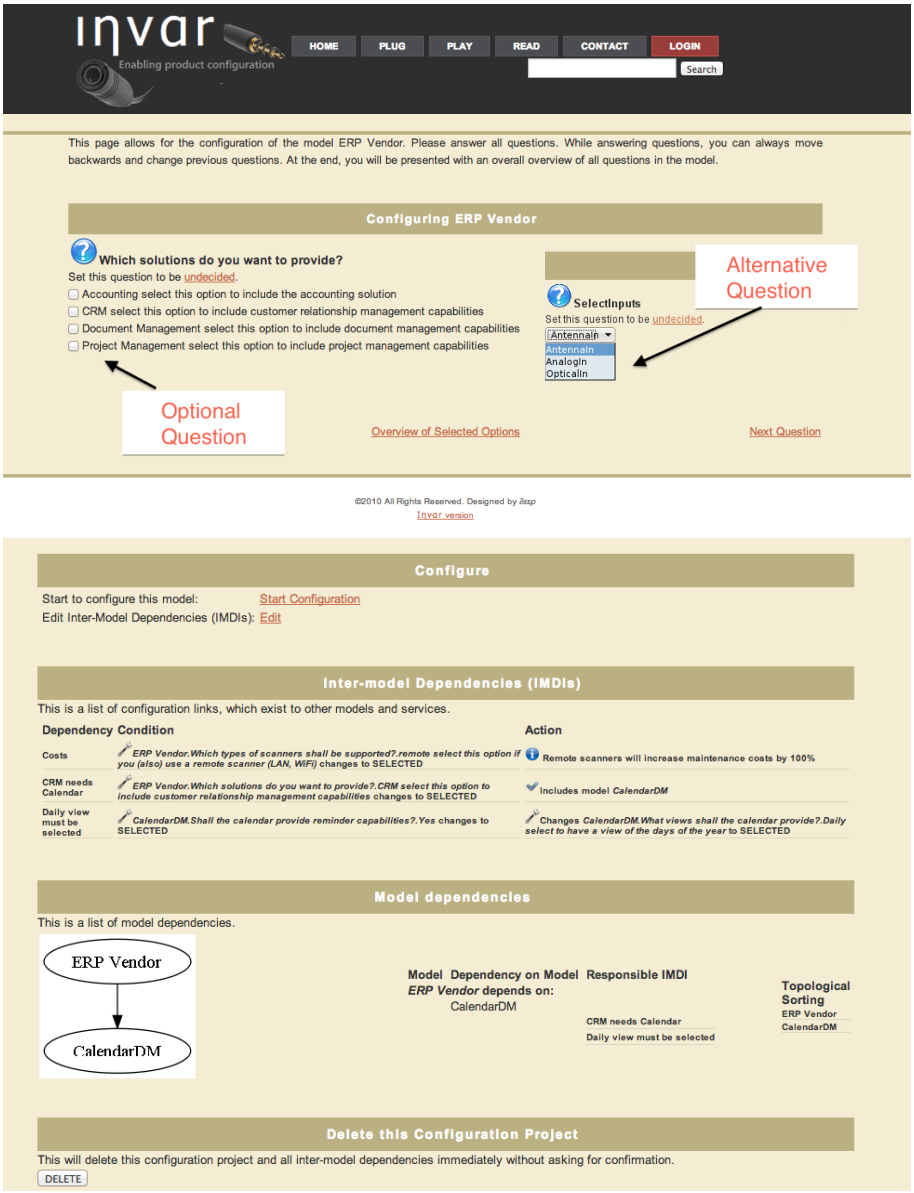

Figure 7: The *Invar* prototype: Presentation of questions from heterogeneous variability models (left), definition and visualization of inter-model dependencies (right).

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 [29, 36, 14, 38, 45, 39]. Furthermore, these three approaches represent three distinct classes of modeling techniques in SPLE.

7.1.1 Feature Models

The FaMa approach [29] supports different kinds of feature model dialects and 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 [46].

The *implementation* of *Invar* configuration services for FaMa faced several issues, as FaMa was not designed to be used for questionnaire-based product configuration. Also, FaMa was primarily implemented as a framework to perform automated analysis of feature models, i.e., it eases the automated extraction of information from feature models and not configuration based on feature models. Nevertheless, the adaptation to the *Invar* Web Service interfaces was almost straightforward – a good example demonstrating the flexibility of *Invar*. The key mappings between the *Invar* configuration steps and FaMa can be summarized as follows:

Question Types: 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, as there is no choice anyway. 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*. In Figure 7 we can see one of those questions for selecting the input of the solutions.

Order of Questions: Feature models are not designed for workflow-oriented configuration. Hence, there is no predefined order of presenting questions to the end user. In our implementation we decided to default to a pre-order traversal. The FaMa FM implementation provides (*i*) three traversal-like orders (in-order, pre-order and post-order); (*ii*) two orders based on the variability in the models, i.e., the first one starting with the questions with most options and ending with the questions with least options, and the second one in the opposite order, from least options to the most; and (*iii*) an out of the box way to traverse the questions by name, using an alphabetical order (see Section 5.3). The user can also define a custom order by manually sorting the list of questions on features.

Feedback: At any time a configuration can give feedback to the *Invar* configuration service. The feedback supported includes: (*i*) informing whether a given feature is selected or deselected, (*ii*) determining whether the current configuration is valid, i.e., it is possible to complete the configuration to a valid product without taking back choices, (*iii*) calculating the total number of potential configurations of the model, (*iv*) informing about the number of questions that have not been decided yet, (*v*) calculating the number of potential configurations available according to the current selection/deselection of features, and (*vi*) determining whether the current configuration is valid as a final product. A configuration is valid if all the features involved are either selected or deselected and if there are no conflicts between the existing constraints in the models and the actual state of the features, meaning that for each feature it is clear whether via the feature state whether it is part of the configuration or not.

7.1.2 Orthogonal Variability Models

FaMa-OVM [44] is an instance of the FaMa framework [29] that uses the OVM notation [8]. The same analysis capabilities and extensions can be supported as for feature models (see Section 7.1.1). The OVM modeling approach was also not designed to be used in a questionnaire-based configuration process so we

Table 3: Mapping of feature modeling elements to Invar configuration primitives.

Feature Modeling	**Invar**
Feature	*Option*
Mandatory subfeature	(ignored, feature will be selected in any case)
Optional subfeature	*Alternative* with only one option (one checkbox)
Or group	*MoreThanOne* (multiple checkboxes)
Alternative group	*Alternative* with multiple options

again faced some issues when *implementing* the FaMa-OVM configuration service for Invar. However we could solve the problems as follows:

Question Types: Internally, FaMa-OVM supports the usage of OVM as described by Pohl *et al.* [8]. In Invar we take into account *mandatory*, *optional*, *alternative* and *or* relationships as they offer a straight way to map OVM to Invar. A mapping of a mandatory relationship between a mandatory VP and a V is not necessary, since this does not define any variability to be configured. If the mapping of the mandatory relationship is between an optional VP and a V, then we map that relationship as an Alternative question type in Invar. Every optional relationship is mapped as Alternative question type in Invar with only one option. Every or-relationship is mapped as a MoreThanOne question type in Invar. An alternative relationship maps to the Alternative question type in Invar.

Order of Questions: In our implementation we use a random order as a default option to traverse every VP in the model. However, as Invar offers the ordering mechanisms presented in Section 5.3, we also allow the FaMa-OVM Invar plugin to offer an alphabetical order and two orders based on the amount of variants that every variation point has, one from more-to-less and another one from less-to-more variants.

Feedback: FaMa-OVM allows analyzing OVM making it possible to give feedback to the user. More specifically, FaMa-OVM returns to Invar the same feedback as FaMa FM, i.e., (*i*) when a VP or a V is selected/deselected; (*ii*) when the configuration is valid; (*iii*) if the actual configuration is valid as a final product; (*iv*) how many valid configurations exist in the model; and (*v*) how many questions are remaining to finish the configuration.

Table 4: Mapping of OVM modeling elements to Invar configuration primitives.

Orthogonal Variability Modeling	**Invar**
Variant	Option
Variation point	Option
Mandatory relationship	(ignored, variant will be selected in any case)
Optional relationship	*Alternative* with only one option (checkbox)
Or relationship	*MoreThanOne*
Alternative choice	*Alternative*

7.1.3 Decision Models

The Dopler approach [36] allows defining decision models together with the reusable assets of a product line (e.g., reusable software components) and mappings between the assets and the decisions. 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* [36]. The Dopler tool suite [36] 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 diverse application examples refer to [36, 47].

The *implementation* of the DOPLER decision modeling approach to provide Invar configuration services was straightforward, as the DOPLER approach itself was designed to be used for questionnaire-based product configuration [37]. 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 mapped 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. For example, in Figure 7 we can see the question *"Which solutions do you want to provide"* which in the background is a DOPLER enumeration.

Order of Questions: In DOPLER, the order of making decisions is defined by the decisions' dependencies. Top-level decisions (which are not dependent on other decisions) are presented and answered first. Decisions that 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 whenever 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, which might confuse the end user. Whenever making a decision that 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 navigate to that decision. Invar allows to define custom non-predefined orders. For DOPLER we do not delegate any order options to the Invar ordering component, as DOPLER models have been created keeping in mind a configuration process and has its own mechanisms [36] to define the order as described above.

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 leads 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) parametrizes the document management software component. Feedback to the user of the Invar service implementation for DOPLER is thus given by presenting her with the assets that are required for the product currently being configured.

Table 5: Mapping of decision modeling elements (DOPLER) to Invar configuration primitives.

Decision Modeling (DOPLER)	**Invar**
Boolean decisions	*Alternative* with choices "yes" and "no"
Enumerations with cardinality 0:1 or 1:1	*Alternative*
Enumerations with other cardinality	*Option*

7.1.4 Summary

After implementing *Invar* for three different modeling approaches we can assert that *Invar* is flexible enough to allow integrating the most common variability languages.

As a summary, Figure 8 depicts the relations between the different configuration choices users can make using *Invar* and the key concepts of the different variability model types supported by the current version of *Invar*, i.e., FaMa feature models, FaMa OVM, and DOPLER decision models.

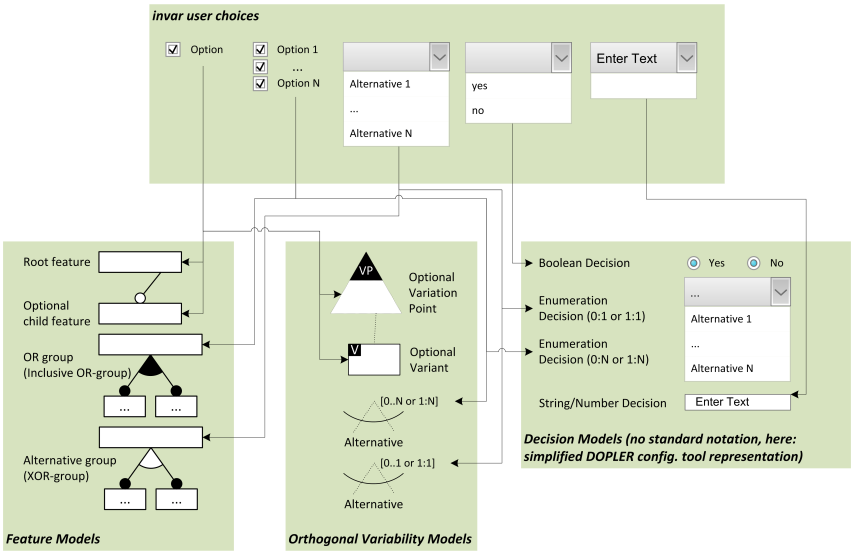

Figure 8: Relations between the configuration choices for the user offered by *Invar* and the key concepts of the three different variability modeling approaches so far integrated by *Invar*.

7.2 RQ3: Applying *Invar* in Realistic Configuration Settings – ERP example

Regarding RQ3 – does *Invar* support realistic configuration scenarios in software ecosystems? – we evaluated the feasibility of the *Invar* approach by creating the variability models for the ERP example, combining decision models based on DOPLER and FaMa feature models and OVM models. We composed these product line models and configured products using the *Invar* product configuration Web Service infrastructure. More specifically, we tested three different settings representing different typical scenarios of how vendors and suppliers may interact: a collaborative, a competitive, and a complementary setting.

7.2.1 Collaborative Setting

Collaborative settings are common in industry, where a vendor of a product aggregates the final product by integrating the required parts from its sub-contractors. Through the use of *Invar*, supply chain management can be supported, independent of the modeling notations used by the sub-contractors. Not only the vendor but also the suppliers benefit from such an approach, as they can refer to the vendor variability model as part of their "specification" and produce parts complying to the features in the base model, or model constraints based on the vendor-specific features.

Figure 9 depicts a collaborative setting, where the main ERP vendor relies on two suppliers, one for a calendar component and one for journal and accounting components. The configuration application

needs to integrate the models. The end user configuring the ERP application would be unaware of the 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.

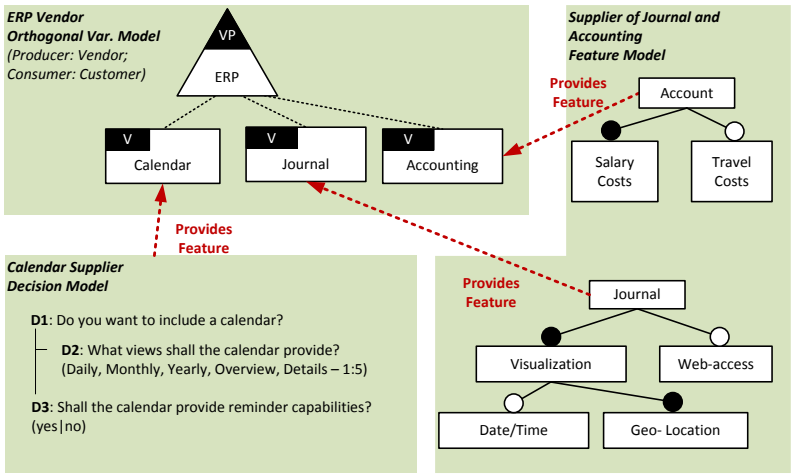

Figure 9: Example application of a collaborative setting: One main model from the vendor and individual models for the subsystems in different modeling notations.

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

1. Web services capable of reading feature models, decision models, and OVM models were registered to *Invar*.

2. A configuration application was 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 OVM model.

3. Inter-model dependency links were defined. In the example models, there are three such links (i) if vendor.calender.isSelected() then calendarSupplier.D1.doSelect(), (ii) if vendor.journal.isSelected() then journalAccountingSupplier.journal.doSelect() and (iii) if vendor.accounting.isSelected() then journalAccountingSupplier.
account.doSelect().

During product configuration, first the questions from the OVM model of the vendor are presented to the user (see Section 5.3 for how these questions are created). Inter-model dependency links are resolved by the configuration broker as soon as the calendar or the journal or the accounting options are selected. The corresponding web services are informed about the selection of the variant (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.

7.2.2 Competitive Setting

A competitive setting is formed whenever more than one supplier is available for a certain feature and the end product may contain only one of those. Such scenarios are quite common in industry, as typically there are a range of variant components, which can be assembled to create a final product. For *Invar*, this

means we need to use constructs that allow conditional inclusion of variability models in the configuration space.

Figure 10 presents a competitive setting, where two suppliers provide functionality related to archiving documents. One of these needs to be chosen for the final product. The selection of the supplier could for example be based on complex metrics and criteria as described in [48]. In *Invar*, we are currently not considering such metrics. We provide mechanisms to select different suppliers based on the decisions made by the end user during product configuration.

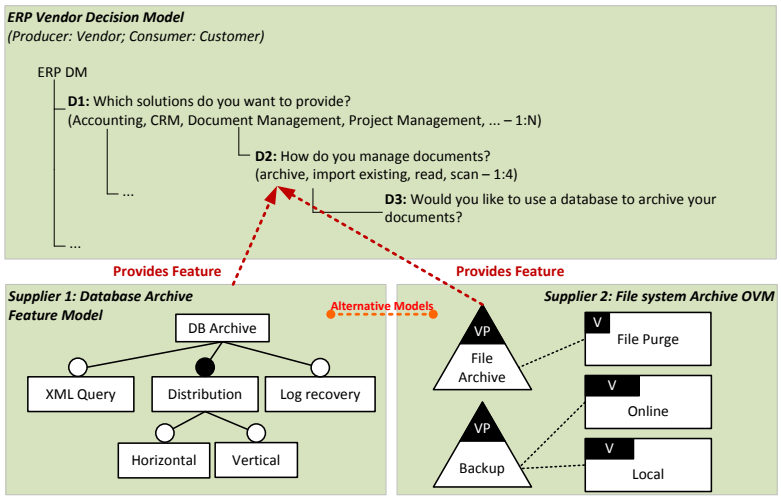

Figure 10: Example application of a competitive setting: One main model from the vendor and individual alternative models for the same subsystem.

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

1. As in the previous scenario, web services capable of reading feature models, decision models, and OVM models were registered to *Invar*.

2. The configuration application was defined by referencing 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 were defined using conditional inclusion actions: `if vendor.D3.isSelected() then DatabaseArchive.includeModel()` and `if vendor.D3.isDeselected() then FileSystemArchive.includeModel()`.

During product configuration, first the questions from the decision 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.

7.2.3 Complementary Setting

Complementary settings typically occur in software ecosystems, where third-party suppliers reuse the functionality of a common base platform to provide additional features to the customers. This leads to

an extension of the decision space for the users during the configuration process and in the background it means the underlying models are enriched/extended by providing additional features.

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 end user performing configuration. For example, in Figure 11, the bank transfer add-in (decision model) extends the available payment types in the accounting package (OVM).

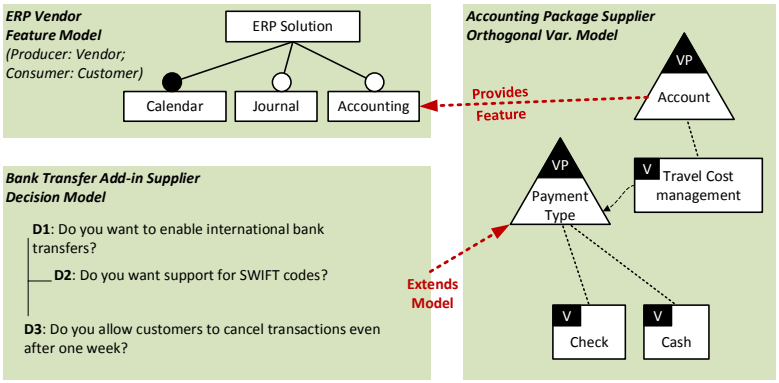

Figure 11: Example application of a complementary setting: One main model is extended by another model, resulting in an additional option.

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

1. Again, web services capable of reading the feature model, the OVM model, and the decision model were registered in *Invar*.

2. A configuration application was 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 variation point payment type in the accounting package is extended by one variant "bank transfer" in the beginning. This was done by adding two links initialized() Accounting.PaymentType.addOption(BankTransfer) and if Accounting.BankTransfer.isSelected() then BankTransfer.includeModel().

During the configuration process, the user has to select between three options (Bank Transfer, Check, Cash) of the question related to *Payment Type*. Upon selection of *Bank Transfer*, the *Bank Transfer model* is added to the configuration process. The output of the configuration process is the list of selected features.

7.3 RQ3: Applying *Invar* in Realistic Configuration Settings – Android privacy settings

To further evaluate the flexibility and feasibility of *Invar* in different multi-model configuration scenarios, we used *Invar* to support configuring the variability of the Android permission system, which has been already proven difficult to configure [49]. The privacy of mobile phone users has been the focus of an ongoing discussion within the security community [50, 51]. Researchers have modeled the variability existing in the Android emulator to optimize the selection of the tests to execute when developing Android apps [52, 53]. Furthermore, they have modeled the different traceability relationships in between the

Android application structure, the emulator options, and the permissions system in Android. Both, OVM and DOPLER have been used by researchers to model this variability. The different models are to be developed and configured by different entities. For example, the applications are to be configured by the app developers while the security requirements are to be configured by the app users and the mobile platform configurations to execute the app are to be configured by different mobile phone manufacturers (or the android emulator designers).

7.3.1 The Android permissions system

Android is based on Linux, thus it provides a privilege-separated access system[6]. This is, each application runs with a different identity (user/group ID). Different parts are also separated, in doing so separating applications from each other and from the operating system itself.

Table 6: Relations between the hardware features used and Android permissions required

Hardware feature	Permission name	Implies This Feature Requirement
Bluetooth	BLUETOOTH	android.hardware.bluetooth
	BLUETOOTH_ADMIN	android.hardware.bluetooth
Camera	CAMERA	android.hardware.camera
		android.hardware.camera.autofocus
Location	ACCESS_MOCK_LOCATION	android.hardware.location
	ACCESS_LOCATION_EXTRA_COMMANDS	android.hardware.location
	INSTALL_LOCATION_PROVIDER	android.hardware.location
	ACCESS_COARSE_LOCATION	android.hardware.location.network
		android.hardware.location
	ACCESS_FINE_LOCATION	android.hardware.location.gps
		android.hardware.location
Microphone	RECORD_AUDIO	android.hardware.microphone
Telephony	CALL_PHONE	android.hardware.telephony
	CALL_PRIVILEGED	android.hardware.telephony
	MODIFY_PHONE_STATE	android.hardware.telephony
	PROCESS_OUTGOING_CALLS	android.hardware.telephony
	READ_SMS	android.hardware.telephony
	RECEIVE_SMS	android.hardware.telephony
	RECEIVE_MMS	android.hardware.telephony
	RECEIVE_WAP_PUSH	android.hardware.telephony
	SEND_SMS	android.hardware.telephony
	WRITE_APN_SETTINGS	android.hardware.telephony
	WRITE_SMS	android.hardware.telephony
Wi-Fi	ACCESS_WIFI_STATE	android.hardware.wifi
	CHANGE_WIFI_STATE	android.hardware.wifi
	CHANGE_WIFI_MULTICAST_STATE	android.hardware.wifi

A finer-grained security access is also provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform. Table 6 shows the set of Android permissions related to the hardware features it requires. We will be using these relations in between features and permissions to enable the configuration of valid products according to certain user-selected security aspects.

7.3.2 Modeling the Android apps structure and feature requirements using OVM

Android has been built to dynamically adapt the applications developed for the operating system to different mobile platform configurations with different screen resolutions, and connectivity capabilities. However, as each application is developed by a different developer (or a set of developers), the structure

[6] `http://developer.android.com/guide/topics/manifest/uses-feature-element.html`

itself can differ in between different apps. We here present an OVM model that encodes the different variation points existing in an Android app.

Manifest declaration. Each Android application contains a file called MANIFEST.xml where all the different permissions required by the app are specified. Also, in this file, the actions, activities, services, and settings are specified.[7]

From the set of variation points described in the manifest.xml we considered the following ones:

- **Compatible-screens.** This section describes the different screens supported by an Android application. For defining a compatible screen we need to provide the resolution and dots-per-inch of all bitmap artifacts used in our app.
- **Uses-sdk.** This section presents the minimum Android SDK required for the application to work. This refers to the minimum Android version to execute the application in.
- **Uses-feature.** Android apps may use different and diverse features present in mobile platform configurations. Those features should be included as references in the manifest.xml. An exhaustive list of all existing features in the ecosystem can be found at `http://developer.android.com/guide/topics/manifest/uses-feature-element.html`.
- **Uses-permissions.** Android apps may use different permissions. Those permissions need to be declared explicitly in the manifest. We will use I*nvar* to define different IMDI links between uses-permissions and hardware features so we can reason and configure products that satisfy a user's security requirements.

To model this variability existing in the manifest.xml file we chose OVM as a notation because there is no commonality to care about. Figure 12 shows the different variation points existing in the Android manifest.xml. Please note that not all the variability information existing in the manifest.xml is presented in the Figure but only the information related to the permissions system.

It is important to note that the different intra-model relationships existing in Figure 12 (dashed blue lines) have been obtained from the permission descriptions. This is, if in the manifest we declare that we use the feature *android.hardware.bluetooth* in our application it is mandatory to ask for the BLUETOOTH permission.

7.3.3 Modeling Security related user preferences using Dopler

The third thing to model when configuring the Android privacy settings are the user preferences. In this case, the most convenient and natural way of encoding this is by using decision models. In Dopler, we can develop a variability model that poses the different security questions to the user, for example, to determine if the configured product may expose data through the Wi-Fi interface. Figure 13 shows the Dopler model that allows a company to define its security concerns when configuring the valid apps and mobile platform configurations existing in the Android ecosystem for the company users of Android devices.

7.3.4 Modeling I*nvar* inter-model relationships by using the Android features and permissions as a proxy

Finally, what is still missing is the definition of the required inter-model relationships. These are the links between the permissions, the application characteristics, and the mobile platform configurations properties. Definition of appropriate inter-model relationships enables I*nvar* to configure applications

[7] `http://developer.android.com/guide/topics/manifest/manifest-intro.html`

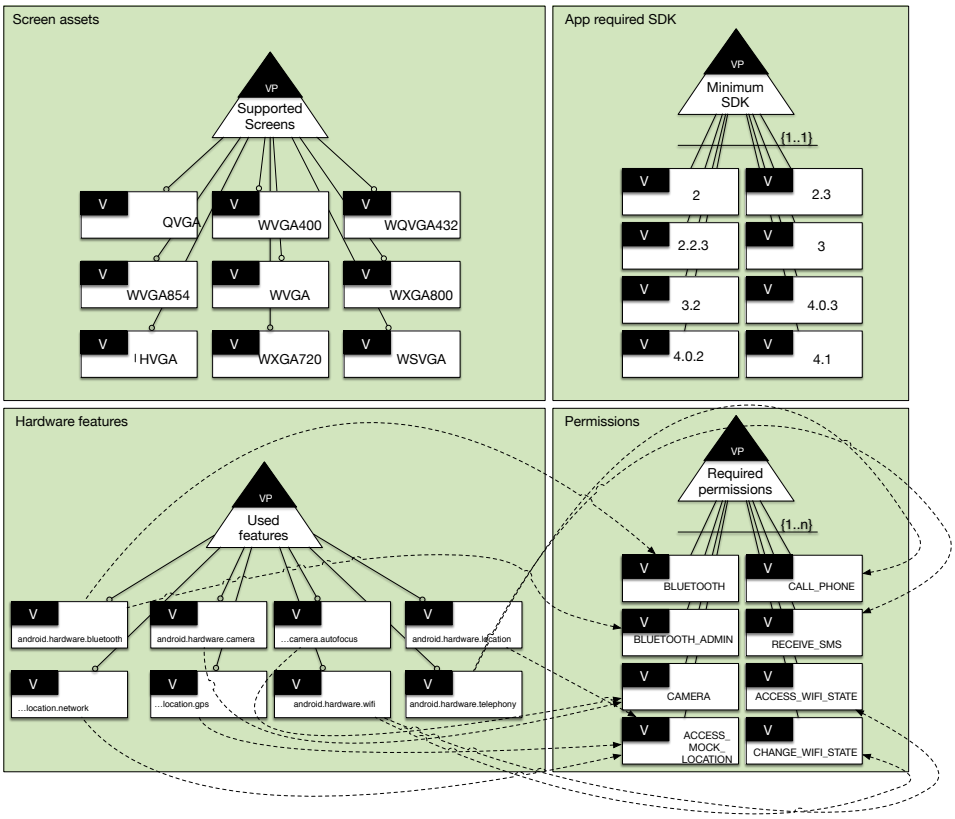

Figure 12: OVM model of the Android application variability related to the permissions system.

1. Would you need to use Bluetooth in your company?
   (a) Would you allow users to add/remove Bluetooth devices?
2. Would you need to allow GSM networking capabilities?
3. Would you need to allow third party apps to know your location?
4. Would Wi-Fi be an option for you employees?
5. Will your app users need access to the camera?

Figure 13: DOPLER questions used by a company to define the security criteria of an Android app

that meet the mobile platform configurations and privacy settings. We linked the app supported screens and the screens available in the mobile platform configurations. This is marked with a circle in Figure 14. Also, we linked the application-used API with the emulator-supported Android versions. Finally, we linked the permissions with the different security concerns defined in the DOPLER model.

We conclude that I*nvar* is able to cope with ecosystem topologies such as the one existing in the Android ecosystem.

7.4 RQ4: Evaluating the Performance of Different Enactment Strategies

Regarding RQ4 – what is the impact of the I*nvar* model enactment strategies on configuration performance? – we also assessed the performance of our approach. In particular, it is important that users do not experience lags between the different configuration steps [10]. In this section we thus show the time required to configure a set of different random models with I*nvar* in order to determine which order (which enactment strategy) provides better results. Executing this experiment, we determined the time required for each variable ordering, showing the average time for automatically configuring a model depending on the number of features and constraints. Further, in this section we demonstrate the "stress" our current implementation is able to withstand, by determining the maximum number of features and cross-tree constraints that it can handle without exceeding an affordable amount of time.

In I*nvar*, the configuration process of a multi product line is carried out by selecting or deselecting options from a set of questions. I*nvar* allows defining the order to present those questions to the user in different ways. Mendonça *et al.* [40] demonstrated that when analyzing a feature model, the way the solver assigns values to the variables has an effect on the time required to analyze the model. Our hypothesis is that *the order used to present the options in the configuration process to the user also has an effect on the time needed by the back-end tool to reason over the model.*

To assess whether our hypothesis was correct we designed a set of tests. To execute those tests we selected the FaMa feature model I*nvar* plugin. We selected it because it can be used with SAT or CSP solvers and offers the most orders that can be used in an I*nvar* configuration process.

First, we conducted tests with the FaMa I*nvar* plugin using a CSP-based solver. We noticed that our CSP implementation for the feature models was not as responsive as it should and rarely can reason over models with more than 200 features. Usually, when analyzing a feature model, SAT solvers are more sufficient to reason over bigger models than CSP solvers [54]. Therefore, we created a new version of the FaMa I*nvar* plugin, this time using a SAT solver. We conjecture that our results will be similar if we used other solvers than SAT but we wanted to test the approach with larger feature models in order to increase the number of different scenarios where I*nvar* can provide configuration support based on feature models.

To verify our hypothesis we created a set of random models to be configured by the FaMa I*nvar* plugin. Those models vary in numbers of features and percentages of cross-tree constraints. Its numbers of features go from twenty features to five hundred and from zero percent of cross-tree constraints to twenty percent.

To automate the testing procedure we created a tool that acted as the I*nvar* core and the *user* in the configuration process. First an I*nvar* order was selected to perform the testing. Later, this tool created random models using BeTTy [55] and uploaded them to I*nvar*. To generate random models, BeTTy provides a well-tested implementation for the algorithm presented by Thüm *et al.* [56]. After uploading the models, the tool acted as the I*nvar* core and iterated over the questions while randomly selecting or deselecting options. This process was executed in one hundred iterations, yielding the average of the time required. In Table 7 we describe the variables used to run our experiments.

Results are presented in Figure 15. This figure shows the time in milliseconds (vertical axis) needed to configure a model with a given number of features (horizontal axis) and ratio of constraints. Please note that the time is shown on a logarithmic scale.

We can highlight that *Inorder* offers better time consumption results making I*nvar* more responsive. For example, as shown in Figure 15, for models with two hundred features *Inorder* only requires half of the time of all other orderings.

Our conclusion is that the order we select to configure an I*nvar* model affects the time required by the back-end tool to analyze it. Those results show that the *Inorder* strategy requires less computational resources than the other strategies. Therefore, it should be used by default in the scenarios where no specific order is required.

Hypotheses			
Null Hypothesis (H_0)	The order used in the *Invar* configuration process has an effect on the performance of the back-end tool in an *Invar* configuration process		
Alt. Hypothesis (H_1)	There is not a significant difference in the average time required by the back-end tools in an *Invar* configuration process		
Design			
Dependent Variable	Time required by the back-end tool to configure a model		
Independent Variable	Technique used for random feature model generation	**Levels:**	Random
Blocking Variables	Number of Features	**Levels:**	20, 50, 100, 200, 500
	Percentage of Cross-Tree Constraints (with respect to the number of features)	**Levels:**	0%, 5%, 10%, 20%
	Order used when configuring a model	**Levels:**	preorder, inorder, postorder, from more options to less, from less options to more, alphabetically
Constants	SAT solver	**Value:**	Sat4j
	Heuristic for variable selection in the SAT solver	**Value:**	*Default*

Table 7: Table with experiment variables

7.5 Discussion

Without *Invar*, considerable manual efforts would have been required to integrate the different models to support product configuration in a multi product line environment. More specifically, instead of just linking existing models using *Invar* IMDI links, modelers would need to either create one large model using one particular approach that integrates all the other models or configure options provided by each model and check manually which dependencies between the different models need to be taken into account. While the first strategy would be sufficient for smaller examples, for large multi product line or software ecosystem settings, it will not be feasible. The second strategy would require the configurers to understand all the different modeling approaches, which is also not feasible. Furthermore, configuration errors would be more likely as relevant configuration information would have to be passed to different teams and tools manually. Moreover, there are some scenarios, such as in the Android ecosystem example described above, where the communication between the different practitioners (e.g., Android developers and app developers) in the configuration process is not feasible without a solution like *Invar*.

For example, when manually configuring the Android permissions system, practitioners would need to perform the following tasks:

1. Configure the permissions required for each of their applications.
   (a) Send it to the security staff to check if the permissions used in the app are ok with the configured privacy policies.
   (b) Reconfigure the application until it fits the requirements defined by the policies.
2. Configure the hardware capabilities required by each of their applications.
   (a) Test the application in front of devices capable of offering the security hardware required to satisfy the constraints in the configuration.
   (b) Reconfigure the application until it fits the requirements.

3. Repeat the process until the three layers involved in the configuration agree.

Thus, while we do not provide any quantitative evidence that configuration based on multiple models using *Invar* is faster than manual configuration based on multiple models without *Invar*, we still are confident that using *Invar* is beneficial in the described configuration scenarios and will outperform manual configuration. We plan to conduct experiments with practitioners (manual configuration based on multiple models vs. configuration based on multiple models automated by *Invar*) as part of our future work.

The successful application of *Invar* in the different settings suggests that the approach is flexible enough and feasible to address realistic scenarios in industry (RQ2 and RQ3). Of course, these scenarios do not cover all possible use cases in industry, which is why we are still working on the application of *Invar* in other scenarios.

The performance evaluation suggests that the integrative infrastructure does not add much overhead to the configuration process, meaning the performance of the individual tools and solvers is not weakened by our approach (RQ4). Of course, the existing bottlenecks in the integrated tools (if any), are still present in *Invar*.

Regarding the requirements identified in Section 4, we can claim that these are addressed by the current implementation of *Invar*.

- The IMDI links described in Section 5.2 allow us to express interfaces between models, both at the level of elements (between features, variants, decisions) and at the level of models (e.g., when an optional model should be considered whenever a particular feature is selected). This covers requirement **R1**.
- The described configuration primitives implemented by the *Invar* Web Service provide an application programming interface (API) through which configuration front-ends can access the configuration functionality (cf. requirement **R2**).
- The service-oriented architecture also allows the integration of multiple repositories (cf. requirement **R3**).
- The web-based access combined with persistence functions, support collaborative, distributed configuration sessions. This addresses requirement **R4**. However, we do not cover parallel configurations as explained in Section 9.
- The exploration of the three settings described in the preceding sections demonstrates that *Invar* is able to handle various configuration settings, which occur in larger product line projects involving multiple organizations. This covers requirement **R5**.

Using the *Invar* approach companies can continue using their favorite modeling approach and still integrate the product configuration front-ends to participate in a multi product line or ecosystem. Also, variability models may be reused by third parties, without imposing any change to the current practices of the involved organizations. The *Invar* approach makes the use of variability models transparent to the end users. Finally, our research provides a starting point for creating a *marketplace for variability models* and fosters healthy competition between the participants. A full validation of our approach can only be done only after an "ecosystem community" has actually started using *Invar* and exploiting its capabilities by composing ecosystems using the provided infrastructure. It will be equally important to extend the scope of the framework and include more modeling approaches by adding new configuration services.

We conclude that integrating different variability modeling techniques for configuration purposes in a software ecosystem is feasible and our approach is flexible enough to support different configuration scenarios. We even think that it is probably unavoidable to enable automated product configuration in the context of software ecosystems with multiple variability models created using different approaches.

7.6 Threats to validity

Even though the examples and validations presented in this paper provide evidence that our proposed solution is feasible and flexible, there are some assumptions that we made that may affect the validity of our results.

The requirements R1-R5 have been collected by the same researchers that have implemented the discussed approach. Hence, the validity of the requirements can be questioned and it is not surprising that all the requirements are fulfilled by the approach. To mitigate this threat, we discussed requirements with other researchers in the community and iteratively refined requirements after discussions at various conferences and workshops. Furthermore, the requirements reflect the researchers' industry experience over many years.

A similar threat to validity is that the authors have been involved in the evaluation of the results. However, different authors were responsible for the implementation of the approach and for creating and using models based on the approach. Some authors only provided feedback on evaluation results and did not interfere with the implementation or modeling. Still, conducting experiments with practitioners remains as an important next step, which is part of our future work.

Similarly, the set of selected variability modeling approaches for initial analysis and integration may have been biased by the authors' expertise in the related methodologies. Therefore, during the selection step of our research, there is a threat to internal validity. However, the three selected approaches represent flavors of the three most commonly used approaches reported in the literature and have been applied in several projects with different partners over several years.

Another threat is that the list of identified challenges/requirements might not be exhaustive and missing aspects that are essential for configuration in large-scale industrial projects. However, the authors considered real scenarios from the industry such as the Android permissions system and the ERP examples derived from industrial experience to mitigate this threat.

The applicability of the *Invar* approach in three selected modeling paradigms suggests that it is flexible enough to be applied in different settings. However, we cannot generalize the results to claim that it is useful for all kinds of variability modeling approaches and tools.

We do not provide any quantitative evidence that configuration based on multiple models using *Invar* is faster than manual configuration based on multiple models without *Invar*. While we are confident that using *Invar* is beneficial in the described configuration scenarios and will outperform manual configuration, conducting experiments with practitioners is an important part of our future work to compare a manual configuration based on multiple models with a configuration process automated by *Invar*.

We evaluated the performance of the tools (i.e., that is of different model enactment strategies) using randomly generated models. The validity of the results therefore also depends on the random model generators in use. However, we used BeTTy [55] to generate random models. BeTTy provides a well-tested implementation for the algorithm presented and evaluated by Thüm *et al.* [56].

We conclude that while several threats exist, our results are promising considering that we have been able to encode variability of ecosystems in different contexts such as the Android permissions system or an ERP system.

7.6.1 Limitations of the approach

While we have shown the validity of our approach in different scenarios and contexts, it is still unclear how difficult would be to apply *Invar* in a real context casestudy. Then, it is important to admit the potential weaknesses and limitations of our approach. Further evaluations should be done in order to determine these limits. Nevertheless, we envision the following main limitations in *Invar*:

– Cost related issues. Currently there is no evidence that the use of *Invar* would reduce the costs for configuring software ecosystems. Is it true, that after the first implementation, the coding

efforts were reduced and we were able to encode the variability existing in Android with no major issues. However, companies should decide where it is worth applying our techniques in a commercial perspective. We have the intuition that this would depend on the size of the project and the number of ecosystems to configure, however, this will require future research.

- Time to be ready. Time is a variable that goes side by side with cost in software projects. Applying *Invar* in a commercial product would require to teach practitioners either to define a new variability model previously known by them or to teach them how to configure a new one. Again, we have the intuition that this would depend in the number and size of the projects to develop.

- Scalability. Currently, the *Invar* architecture is developed using rest services, thus, it can scale easily in terms of CPU and power to configure (e.g. using cloud providers). However, communication between different practitioners are required to first orchestrate the multi-model configuration in *Invar*. This is also a parameter to think when choosing to use our techniques and should be evaluated prior to apply them in an industrial context.

8 Related Work

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

Variability in large-scale software development. Deelstra *et al.* [57] classify different types of approaches along the dimensions scope of reuse and domain scope. Hierarchical product lines [58] are an example of an approach that can be applied when there is a family with a broad domain scope focusing on a number of product categories. Bühne *et al.* [59] extend an existing meta-model for variability modeling to structure product lines into a hierarchy. Reiser and Weber [60] suggest to model complex product lines with a hierarchy of feature models called multi-level feature trees. Van Ommering [61] introduces the concept of *Product Populations*. Dhungana *et al.* [62] explore how to structure the modeling space for large product lines at multiple levels of abstraction. Holl *et al.* [63] describe the concept of product line bundles to package variability models and product line tool extensions to support sharing and deploying models and tools in multi product lines. *In contrast to these approaches, Invar focuses on enabling the integration of different variability modeling approaches to support and coordinate end-user configuration.* Schmid [64] 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 heterogeneous modeling approaches. Metzger *et al.* [15] have presented an approach to integrate feature models and orthogonal variability models, by differentiating between product variability and software variability. *The Invar approach is different as we focus on variability of different subsystems, rather than different abstractions of the same subsystem.*

Coordination during product derivation. Czarnecki *et al.* [33] present an approach supporting the staged configuration of features models. Focusing on large-scale embedded systems Reiser *et al.* [65] propose product sets and configuration links to define dependencies between different feature models at the time of their selection. Hubaux *et al.* [66] 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 automatically build concern-specific configuration views [67]. The issue of multi-level configuration is also addressed by Rabiser *et al.* [18] 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.* [68] discuss the customization in application engineering over multiple levels. *Invar focuses on presenting the configuration options provided by multiple models created with different heterogeneous modeling approaches to the end user in an integrated fashion.*

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 [69] 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.* [70] propose an engineering service bus to integrate arbitrary tools in the engineering life cycle. Similarly, Blanc *et al.* [71] propose the use of a model bus to facilitate the integration of tools in model-based environments. The issue of integrating and relating arbitrary models has also been targeted by mega-modeling approaches, which propose the use of a registry for models and meta-models [72].

Software ecosystems. Bosch and Bosch-Sijtsema [73] discuss the impact of three trends in large-scale software development, i.e., software product lines, global development and software ecosystems and related problems observed in industry. They describe five different approaches, which potentially address these problems, organized from integration-centric to composition-centric approaches (integration-centric development, release groupings, release trains, independent deployment, open ecosystem). Bosch [1] also discussed 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, however, ensures interoperability through the definition of an integrative architecture and the interfaces between system components.

Standardization efforts. There is an ongoing effort to standardize variability modeling based on the common variability modeling language (CVL) [2], a generic language to define variability. The goal of CVL standardization is to create a new standard notation. *In contrast, the goal of Invar is to integrate existing notations, rather than defining a new one.* Invar could be extended to use CVL and other approaches side-by-side. 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 *Invar*.

9 Conclusions, Summary and Future Work

Nowadays, the variability modeling techniques are being applied to diverse and varying scenarios such as the Android ecosystem and the Linux kernel that go far beyond the boundaries of software product lines. In such contexts, it is equally important to integrate data, models, and tools across organizations and inter-organizational boundaries – together with the alignment of the business strategy among the cooperating units. From the perspective of software reuse and configuration, the initiation and growth of a software ecosystem can be fostered considerably if the involved stakeholders are supported with tools and techniques for dealing with variability in a uniform manner. Moreover, the scalability of the variability approaches, not only in terms of CPU power but also from the point of view of integration and collaboration is a main concern in software product line and software ecosystems research.

In this paper, we presented an approach to facilitate the integration of variability models during product configuration regardless of the modeling techniques, notations, and tools used. Based on an illustrative example, we defined the basic user interactions required for configuration in general (configuration primitives) and mapped these interactions to the concrete semantics of individual modeling approaches, i.e., an approach for feature modeling, and OVM-based approach, and an approach for decision modeling. Also, we evaluated this approach in terms of scalability when referring to the time required to configure a variability model and more important, checking the adaptability of the approach by modeling different scenarios. For doing so, we provide a technical infrastructure and a prototypical

implementation of an integrative approach based on web services that supports defining dependencies between multiple variability models as well as different strategies for model enactment during product configuration. We showed the flexibility and feasibility of the approach and its implementation by applying it to three different variability modeling approaches and by showing that we are able to integrate them in the context of an example ERP system and in the context of the Android permissions system. We also provided initial evidence on the performance of the approach, more specifically, its different model enactment strategies, during configuration.

We can conclude after this research that, i) is important to enable the support for different enactment strategies to scale in terms of CPU related costs; ii) the use of IMDI is a mandatory aspect when configuring software ecosystems such as the Android ecosystem; iii) is important to offer a common facade to configure software ecosystems as complex as Android where we see a misuse of the permissions system; iv) different users use to speak different languages, this is, we need to support a diversity of modeling languages to then better communicate and integrate the work of different practitioners. Nevertheless, there are several issues where future efforts are required, e.g., to define more scalable ways of defining inter-model dependencies, thus, to achieve industrial-scale application of this approach by modeling the configuration projects in *Invar*. Currently, each model has to explicitly define its relationships to the other models and, if options in one model have implications for other models, those inter-model constraints (dependency links) need to be defined. Another example is the uncertainty at this point whether practitioners will rely in variability modeling techniques to configure and manage the configurations in a distributed ecosystem. Indeed, this is an important aspect that requires further validation and efforts.

To ease of some of those difficulties, in future work we plan to formalize the dependencies specified using the IMDI links and improve and extend the *Invar* 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 new modeling approaches in *Invar*. This will allow us to fully validate the approach and the platform. Eventually, we also want to provide guidance for product line engineers for defining the order of models and creating the inter-model links.

Material

A version of *Invar* is hosted in `http://invar.lero.ie`. Please note, that the software is distributed with a closed license that does not allow redistribution. However, some of the plug-ins linked in *Invar* are open-source and can be found in `http://github.com/invar-fama-pluggins`.

Acknowledgements

This work was supported, in part, by Science Foundation Ireland grants 03/CE2/I303_1 and 10/CE/I1855 to Lero; by the Christian Doppler Forschungsgesellschaft (Austria), Siemens VAI Metals Technologies, and Siemens Corporate Technology; by the European Commission (FEDER), by the Spanish government under TAPAS (TIN2012-32273) project and the Andalusian government under Talentia program and the

COPAS (TIC-1867) project. Also, we would like to express special thanks to Dominik Seichter for his previous contributions to this research.

References

[1] J. Bosch, From software product lines to software ecosystems, in: SPLC 2009, ACM ICPS, San Francisco, CA, USA, 2009, pp. 111–119.

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

[3] M. Sinnema, S. Deelstra, Classifying variability modeling techniques, Information and Software Technology 49 (7) (2007) 717–739.

[4] L. Chen, M. Babar, N. Ali, Variability management in software product lines: A systematic review, in: SPLC 2009, ACM ICPS, San Francisco, CA, USA, 2009, pp. 81–90.

[5] L. B. Lisboa, V. C. Garcia, D. Lucrédio, E. S. de Almeida, S. R. de Lemos Meira, R. P. de Mattos Fortes, A systematic review of domain analysis tools, Information and Software Technology 52 (1) (2010) 1–13.

[6] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, J. Galindo, Configuration of multi product lines by bridging heterogeneous variability modeling approaches, in: 15th International Software Product Line Conference, 2011, pp. 120–129.

[7] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, J. Galindo, Integrating heterogeneous variability modeling approaches with invar, tool demonstration, in: 7th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2013), ACM, Pisa, Italy, 2013.

[8] K. Pohl, G. Böckle, F. Van Der Linden, Software product line engineering: foundations, principles, and techniques, Springer, 2005.

[9] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, A. Wasowski, Cool features and tough decisions: a comparison of variability modeling approaches, in: VaMoS, 2012, pp. 173–182.

[10] S. MacKenzie, C. Ware, Lag as a determinant of human performance in interactive systems, in: INTERACT'93 and CHI'93 conference on Human factors in computing systems, ACM, 1993, pp. 488–493.

[11] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.

[12] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, K. Villela, Software diversity: state of the art and perspectives, International Journal on Software Tools for Technology Transfer, Special Section on Software Diversity 14 (5) (2012) 477–495.

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

[14] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later, Information Systems 35 (6) (2010) 615–636.

[15] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, 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), IEEE CS, New Delhi, India, 2007, pp. 243–253.

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

[17] S. P. Consortium, Synthesis guidebook, Tech. rep., SPC-91122-MC. Herndon, Virginia: Software Productivity Consortium (1991).

[18] R. Rabiser, R. Wolfinger, P. Grünbacher, Three-level customization of software products using a product line approach, in: 42nd Annual Hawaii International Conference on System Sciences, IEEE CS, Waikoloa, Big Island, HI, USA, 2009, pp. 1–10.

[19] T. Berger, R. Rublack, D. Nair, J. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: 7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), VaMoS '13, 2013.

[20] D. Lettner, M. Petruzelka, R. Rabiser, F. Angerer, H. Prähofer, P. Grünbacher, Custom-developed vs. model-based configuration tools: Experiences from an industrial automation ecosystem, in: MAPLE/SCALE 2013, 17th International Software Product Line Conference, SPLC '13 Workshops, ACM, 2013, pp. 52–58.

[21] M. Vierhauser, R. Rabiser, P. Grünbacher, A case study on testing, commissioning, and operation of very-large-scale software systems, in: 36th International Conference on Software Engineering, ICSE Companion 2014, ACM, 2014, pp. 125–134.

[22] R. Rabiser, D. Dhungana, P. Grünbacher, K. Lehner, C. Federspiel, Product configuration support for nontechnicians: Customer-centered software product-line engineering., IEEE Intelligent Systems 22 (3).

[23] R. Wolfinger, S. Reiter, D. Dhungana, P. Grünbacher, H. Prähofer, Supporting runtime system adaptation through product line engineering and plug-in techniques, in: Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on, IEEE, 2008, pp. 21–30.

[24] R. Clotet, D. Dhungana, X. Franch, P. Grünbacher, L. López, J. Marco, N. Seyff, Dealing with changes in service-oriented computing through integrated goal and variability modeling, in: 2nd International Workshop on Variability Modelling of Software-intensive Systems (VAMOS 2008), ICB-Research Report No. 22, University of Duisburg Essen, Essen, Germany, 2008, pp. 43–52.

[25] D. Benavides, J. A. Galindo, Variability management in an unaware software product line company: an experience report, in: The Eighth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '14, Sophia Antipolis, France, January 22-24, 2014, 2014, p. 5.

[26] A. van Lamsweerde, Goal-oriented requirements engineering: a guided tour, in: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, 2001, pp. 249–262.

[27] M.-O. Reiser, R. Kolagari, M. Weber, Compositional variability - concepts and patterns, in: HICSS '09. 42nd Hawaii International Conference on System Sciences, IEEE, 2009, pp. 1–10.

[28] A. Seibel, S. Neumann, H. Giese, Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance, Software and Systems Modeling 9 (4) (2010) 493–528.

[29] D. Benavides, P. Trinidad, A. Ruiz-Cortés, S. Segura, Fama, in: Systems and Software Variability Management, Springer, 2013, pp. 163–171.

[30] M. Acher, P. Collet, P. Lahire, R. B. France, FAMILIAR: A domain-specific language for large scale management of feature models, Science of Computer Programming (SCP) 78 (6) (2013) 657–681.

[31] J. Bosch, P. Bosch-Sijtsema, From integration to composition: On the impact of software product lines, global development and ecosystems, Journal of Systems and Software 83 (1) (2010) 67–76.

[32] G. Holl, P. Grünbacher, R. Rabiser, A systematic review and an expert survey on capabilities supporting multi product lines, Information and Software Technology 54 (8) (2012) 828–852.

[33] K. Czarnecki, S. Helson, U. Eisenecker, Staged configuration using feature models, in: SPLC 2004, Springer, Boston, MA, USA, 2004, pp. 266–283.

[34] M. Mendonça, D. Cowan, Decision-making coordination and efficient reasoning techniques for feature-based configuration, Science of Computer Programming 75 (5) (2010) 311–332, coordination Models, Languages and Applications (SAC08).

[35] R. Olaechea, S. Stewart, K. Czarnecki, D. Rayside, Modelling and multi-objective optimization of quality attributes in variability-rich software, in: Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages, ACM, 2012, p. 2.

[36] D. Dhungana, P. Grünbacher, R. Rabiser, The DOPLER meta-tool for decision-oriented variability modeling: A multiple case study, Automated Software Engineering 18 (1) (2011) 77–114.

[37] R. Rabiser, P. Grnbacher, M. Lehofer, A qualitative study on user guidance capabilities in product configuration tools, in: 27th IEEE/ACM International Conference Automated Software Engineering (ASE'12), ACM, Essen, Germany, 2012, pp. 110–119.

[38] R. Rabiser, P. Grünbacher, D. Dhungana, Requirements for product derivation support: Results from a systematic literature review and an expert survey, Information and Software Technology 52 (3) (2010) 324–346.

[39] G. Botterweck, M. Janota, D. Schneeweiss, A design of a configurable feature model configurator, in: 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2009), ICB Research Report vol. 29, Sevilla, Spain, 2009, pp. 165–168.

[40] M. Mendonça, A. Wasowski, K. Czarnecki, D. Cowan, Efficient compilation techniques for large scale feature models, in: Proceedings of the 7th international conference on Generative programming and component engineering, ACM, 2008, pp. 13–22.

[41] M. N. Huhns, M. P. Singh, Service-oriented computing: Key concepts and principles, IEEE Internet Computing 9 (1) (2005) 75–81.

[42] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Services: Concepts, Architecture and Applications, Springer Verlag, 2004.

[43] F. Roos-Frantz, J. Galindo, D. Benavides, A. Ruiz-Cortés, FaMa-OVM: a tool for the automated analysis of OVMs, in: 16th International Software Product Line Conference-Volume 2, ACM, 2012, pp. 250–254.

[44] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, K. Lauenroth, Quality-aware analysis in product line engineering with the orthogonal variability model, Software Quality Journal (2011) 1–47.

[45] R. Rabiser, P. Grünbacher, D. Dhungana, Supporting product derivation by adapting and augmenting variability models, in: SPLC 2007, IEEE CS, Kyoto, Japan, 2007, pp. 141–150.

[46] S. Segura, R. Hierons, D. Benavides, A. Ruiz-Cortés, Automated metamorphic testing on the analyses of feature models, Information and Software Technology 53 (3) (2011) 245–258.

[47] R. Rabiser, D. Dhungana, W. Heider, P. Grünbacher, Flexibility and end-user support in model-based product line tools, in: Euromicro Conference on Software Engineering and Advanced Applications, IEEE CS, Patras, Greece, 2009, pp. 508–511.

[48] H. Hartmann, T. Trew, A. Matsinger, Supplier independent feature modelling, in: SPLC 2009, ACM ICPS, San Francisco, CA, USA, 2009, pp. 191–200.

[49] J. Burns, Developing secure mobile applications for Android (2008).

[50] W. Enck, M. Ongtang, P. McDaniel, Understanding android security., IEEE security & privacy 7 (1) (2009) 50–57.

[51] W. Enck, D. Octeau, P. McDaniel, S. Chaudhuri, A study of android application security., in: USENIX security symposium, Vol. 2, 2011, p. 2.

[52] H. Turner, J. White, J. Reed, J. A. Galindo, A. Porter, M. Marathe, A. Vullikanti, A. Gokhale, Building a Cloud-based Mobile Application Testbed, IGI Global, 2012.

[53] J. A. Galindo, H. Turner, D. Benavides, J. White, Testing variability intensive systems using automated analysis. an application in android., Tech. Rep. 01, ETSII. Avda. de la Reina Mercedes s/n. Sevilla España (Apr 2014).

[54] S. Segura, D. Benavides, A. Ruiz-Cortés, Functional testing of feature model analysis tools: a test suite, Software, IET 5 (1) (2011) 70–82.

[55] S. Segura, J. Galindo, D. Benavides, J. Parejo, A. Ruiz-Cortés, Betty: Benchmarking and testing on the automated analysis of feature models, in: U. Eisenecker, S. Apel, S. Gnesi (Eds.), Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12), ACM, Leipzig, Germany, 2012, pp. 63–71.

[56] T. Thüm, D. Batory, C. Kästner, Reasoning about edits to feature models, in: 31st International Conference on Software Engineering, ICSE '09, IEEE, 2009, pp. 254–264.

[57] S. Deelstra, M. Sinnema, J. Bosch, Product derivation in software product families: a case study, Journal of Systems and Software 74 (2) (2005) 173–194.

[58] J. Bosch, The challenges of broadening the scope of software product families, Communications of the ACM 49 (12) (2006) 41–44.

[59] S. Bühne, K. Lauenroth, Modelling requirements variability across product lines, in: 13th International Conference on Requirements Engineering (RE'05, IEEE CS, Paris, France, 2005, pp. 41–50.

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

[61] R. C. van Ommering, Software reuse in product populations, IEEE Trans. Software Eng. 31 (7) (2005) 537–550.

[62] D. Dhungana, P. Grünbacher, R. Rabiser, T. Neumayer, Structuring the modeling space and supporting evolution in software product line engineering, Journal of Systems and Software 83 (7) (2010) 1108–1122.

[63] G. Holl, M. Vierhauser, W. Heider, P. Grünbacher, R. Rabiser, Product line bundles for tool support in multi product lines, in: 5th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2011), ACM ICPS, Namur, Belgium, 2011, pp. 21–28.

[64] K. Schmid, Variability modeling for distributed development – a comparison with established practice, in: SPLC 2010, Springer, Jeju Island, Korea, 2010, pp. 151–165.

[65] M.-O. Reiser, R. T. Kolagari, M. Weber, Compositional variability–concepts and patterns, in: 42nd Hawaii International Conference on System Sciences, IEEE CS, Waikoloa, HI, USA, 2009, pp. 1–10.

[66] A. Hubaux, A. Classen, P. Heymans, Formal modelling of feature configuration workflows, in: SPLC 2009, ACM ICPS, San Francisco, CA, USA, 2009, pp. 221–230.

[67] A. Hubaux, P. Heymans, P.-Y. Schobbens, D. Deridder, Towards multi-view feature-based configuration, in: 16th International Working Conference on Requirements Engineering, Springer, Essen, Germany, 2010, pp. 106–112.

[68] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, Multi-level customization in application engineering, Commun. ACM 49 (12) (2006) 60–65.

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

[70] T. Moser, S. Biffl, Semantic tool interoperability for engineering manufacturing systems, in: ETFA, 2010, pp. 1–8.

[71] X. Blanc, M.-P. Gervais, P. Sriplakich, Model bus: Towards the interoperability of modelling tools., in: U. Amann, M. Aksit, A. Rensink (Eds.), MDAFA, Vol. 3599 of Lecture Notes in Computer Science, Springer, 2004, pp. 17–32.

[72] J. Bézivin, F. Jouault, P. Rosenthal, P. Valduriez, Modeling in the large and modeling in the small, in: U. Amann, M. Aksit, A. Rensink (Eds.), Model Driven Architecture, Vol. 3599 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 33–46.

[73] J. Bosch, P. Bosch-Sijtsema, From integration to composition: On the impact of software product lines, global development and ecosystems, Journal of Systems and Software 83 (1) (2010) 67–76.

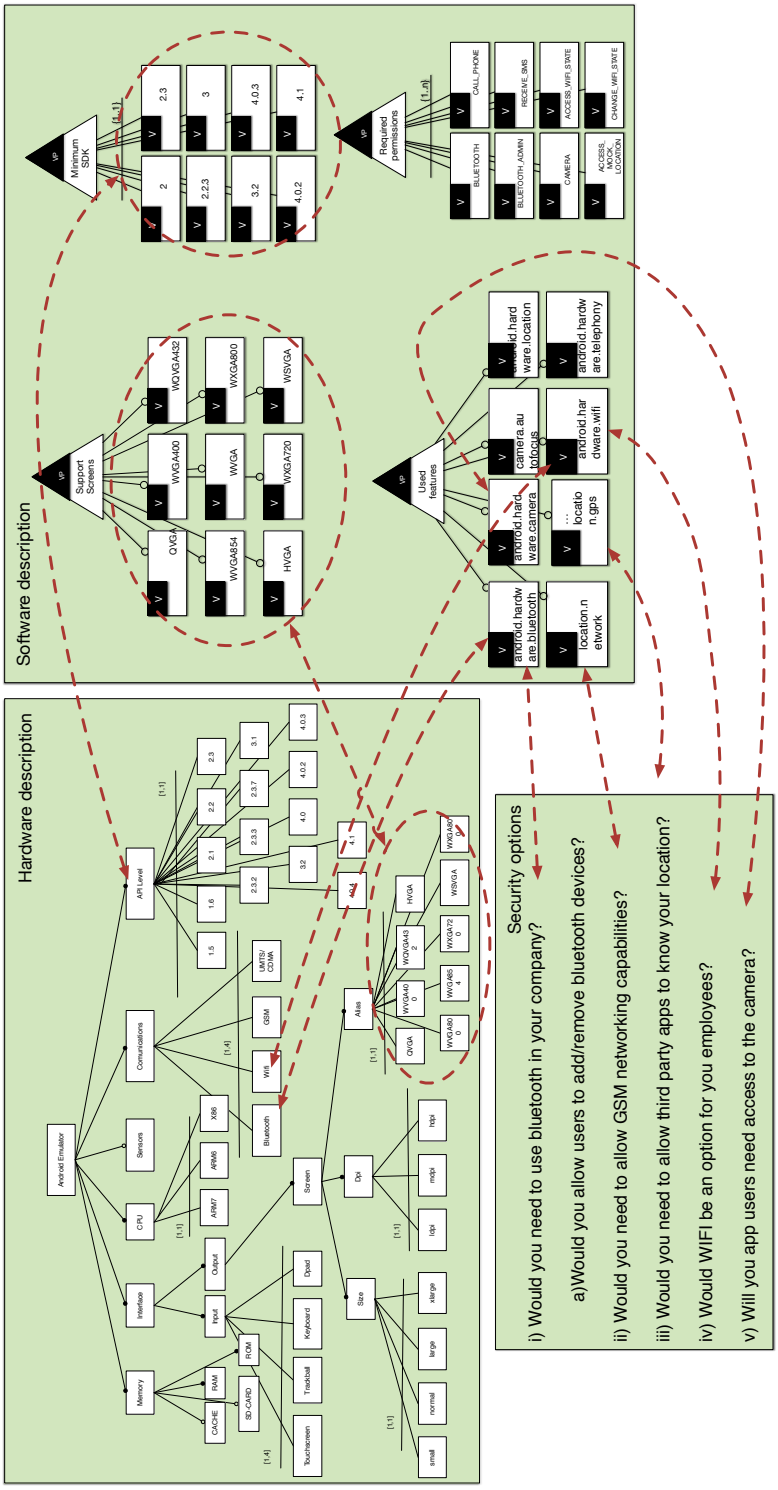

Figure 14: Integrating the OVM and the DOPLER models of the Android permissions system using *Invar*.

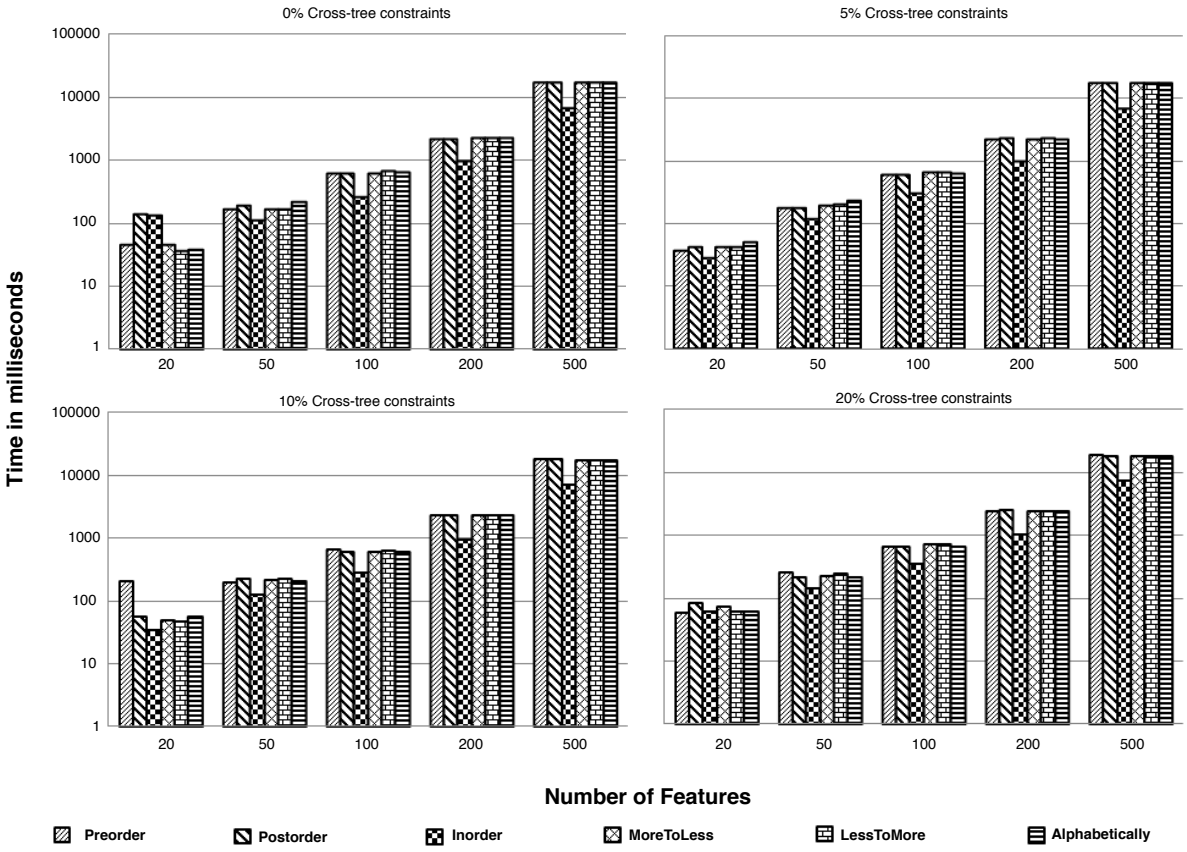

Figure 15: Time in milliseconds needed to configure a model with a given number of features and percentage of cross-tree constraints.