Analyzing Methodologies and Tools for Specifying Variability in Software Processes*

Jocelyn Simmonds
Departamento de Informática
Universidad Técnica Federico Santa María
Valparaiso, Chile
jsimmond@inf.utfsm.cl

María Cecilia Bastarrica
Computer Science Department
Universidad de Chile
Santiago, Chile
cecilia@dcc.uchile.cl

Luis Silvestre
Computer Science Department
Universidad de Chile
Santiago, Chile
lsilvest@dcc.uchile.cl

Alcides Quispe
Computer Science Department
Universidad de Chile
Santiago, Chile
aquispe@dcc.uchile.cl

ABSTRACT

Software process lines (SPrL) are families of highly related processes that are built from a set of core process assets in a prestablished fashion. Software companies may take advantage of SPrL in order to deal with different kinds of projects –development or maintenance, large or small, complex or simple– just defining a general process and modeling variability so that the general process can be adapted accordingly in each case. Formally specifying variability enables automatic tailoring. However, and provided that SPrL is quite a recent research area, there is no established methodology or notation for modeling process variability. In this paper we present the kinds of variability we have found to be relevant for processes, and we investigate the appropriateness of different approaches for modeling process variability such as SPEM 2.0 primitives and vSPEM, and other general notations for modeling variability such as OVM and feature models. We make an analysis based on the expresiveness of each notation for dealing with the required variability, as well as the understandability of the specification, adherence to standard formats and the tool support availability. We illustrate each option specifying the variability of the process of a medium size Chilean software company.

1. INTRODUCTION

Software processes are recognized as valuable for achieving productivity and quality in software development. However, defining a unique process is hard and expensive, and it is not necessarily adequate for all kinds of projects, e.g., a large and complex project probably requires a more sophisticated process than a simple maintenance project. Software process lines seem to be a good option for this problem. Therefore, the company defines an organizational process including the potential variability, and this process is tailored according to the characteristics of each project in order to achieve a project adapted process.

Formal software process specification enables different kinds of automatic processing and analysis, such as tailoring, scheduling and planning, among others. There are some languages specifically created for process specification. In this work we use SPEM 2.0, the OMG standard for software process specification. However, there is no standard notation for defining variability in SPrL. Notations that have proved to be powerfull for software product lines may not be necessarily appropriate for this context. Moreover, there is a large variety of notations that have been proposed, ranging from native constructs of SPEM 2.0, to extensions to this language such as vSPEM, to more general variability modeling languages such as OVM or feature models. Different formalisms require different methodologies for variability specification within the software process domain engineering.

We survey different ways of specifying variability in SPrL, and we analyze them from a methodological point of view, as well as the availabity of appropriate and mature supporting tools so that they can be incorporated within a tool chain for automating software process tailoring.

The paper is organized as follows. Section 2 provides some background on software process modeling both, from a structural and from a behavioral point of view. In Sect. 3 we describe the ways a software process may vary and their methodological implications. Available notations for modeling process variability are presented in Sect. 4, and they are analyzed in Sect. 5. Section 6 discusses some related work. Finally, Sect. 7 draws some conclusions and presents some further work.

*This work has been partly funded by project Fondef D09I1171, Chile

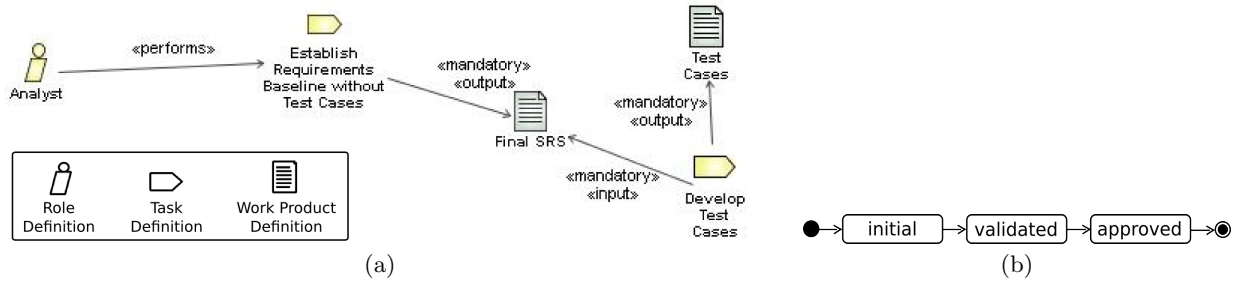

Figure 1: (a) SPEM 2.0 task example: "Establish Requirements Baseline without Test Cases", and (b) state machine associated to "Final SRS" work product.

2. SOFTWARE PROCESS MODELING

The Software and Systems Process Engineering Meta-Model (SPEM 2.0) [34] is the OMG standard notation for modeling software and systems development processes and their components. SPEM 2.0 is defined as a UML 2.0 profile, and takes an object-oriented approach to process modeling. SPEM diagrams are used to model two different views of a process: a static view, where process components (tasks, work products and roles) are defined; and a dynamic view, where interaction diagrams (like UML activity diagrams) are used to model how process components interact in order to accomplish the goals of the modeled process.

We have been working in supporting Rhiscom, a medium size software company in Chile. This company develops software for the retail industry. It has grown steadily in the last five years counting these days on around 70 employees and also having affiliate offices in Peru, Bolivia and Ecuador. Rhiscom has defined a software process that has been applied ad improved for a couple of years, and only recently it has been formalized in SPEM 2.0 using the EPF Composer[1].

In this section, we give a brief overview of the SPEM 2.0 concepts and notation, using Rhiscom's development process as a running example.

2.1 Process Components

We use SPEM 2.0 to specify processes because this notation gives process engineers mechanisms for managing families of processes. In order to encourage process maintenance and reuse, SPEM 2.0 makes a difference between the definition of process building blocks and their later use in (possibly more than one) processes. Process components, like tasks, roles and work products, are defined and stored in a Method Library. These definitions can later be reused in multiple process definitions. A process is a collection of activities, where an activity is a "big-step" grouping of role, work product and task uses. Roles perform activity tasks, and work products serve as input/output artifacts for tasks.

Roles are used to define the expected behavior and responsibilities of the team members involved in a process. For example, a role definition like "Analyst" is used to represent team members that gather input from stakeholders and define requirements. Note that roles do not represent individual team members, and that one team member may take on several roles. Also, a single role may be responsible for more than one work product, as well as modify multiple work products.

Work products represent anything produced, consumed, or modified by a process. Tangible work products are usually called "artifacts", while intangible products are called "outcomes". Work products that will be handed off to internal or external parties are also classified as "deliverables". Documents, models, repositories, source code and binaries are examples of artifacts and deliverables, while an event like notifying a party that an activity has concluded is an outcome. Formally, roles and work products are stereotyped UML classes, «role» and «work product», respectively. As such, generalization, aggregation and composition relationships can be used to define more complex roles and work products.

A task is a set of subtasks/steps that are performed by possibly multiple roles, and involve the creation or modification of one or more work products. Ideally, only one role is responsible for the task. As such, a task definition is a stereotyped class diagram, where the roles that are responsible for it and those that will perform the associated work are identified, as well as the input and output work products (which can be tagged as mandatory or optional).

For example, the class diagram in Figure 1a shows the task definition of the "Establish Requirements Baseline without Test Cases" task, which occurs during the RE phase of the Rhiscom development process. This task definition is associated to one role and one work product, "Analyst" («performs») and "Final SRS" («mandatory» and «output»), respectively. This means that a team member with an analyst profile should be in charge of this task, and that this task has one mandatory output work product, the final version of the Software Requirements Specification. Task definitions can also include tool definitions and guidance, but we have omitted these elements from this paper in order to simplify presentation.

2.2 Process Behavior

After having defined the basic process components, we can now specify how work products change state, how tasks are grouped into activities, and finally, how previously defined process components are used to define processes.

A work product may go through different states during its

[1]http://www.eclipse.org/epf/

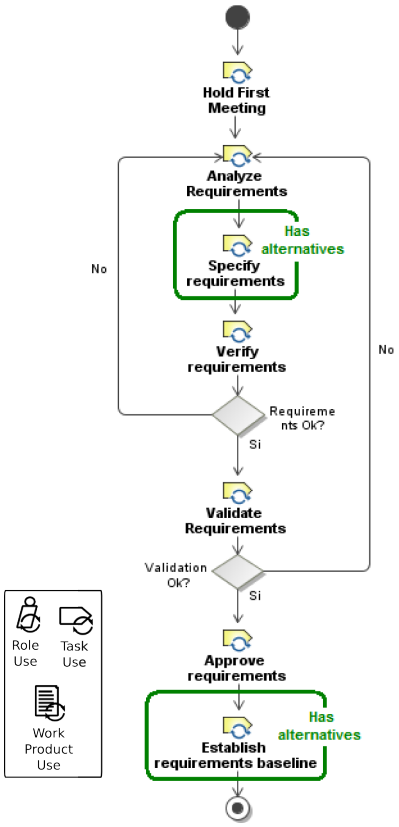

Figure 2: Detailed specification of Rhiscom's "Requirements" activity.

lifetime. As such, work products can have an associated state model. Process engineers can use this model to specify a work product's states, as well as the permitted transitions between these states. Ideally, such a model can be used to determine how complete a work product is. For example, the state machine for the "Final SRS" document mentioned in the previous section can be seen in Figure 1b. This state machine has three states: "initial", "validated" and "approved". The "initial" state represents an empty document, whereas the "validated" state indicates that the contents of the document have been validated by the Analyst, but the document is not necessarily complete. The final state, "approved", is used to indicate that the document is complete and that the client has signed off on the document. These work product states can also be used as guards in task definitions.

An activity is a logical grouping of task, role and work product uses. Activity diagrams are used to model the workflow between activity tasks. For example, Figure 2 shows the detailed workflow of the "Requirements" activity at Rhiscom. Each task can be associated to various predefined roles and work products (not shown in Figure 2 to increase legibility). The activity has a simple workflow, built using existing tasks. The outcome of two tasks ("Verify requirements" and "Validate requirements") affect control, since negative results return the activity to earlier tasks. Two tasks ("Specify requirements" and "Establish requirements baseline") are marked as "Has alternatives". This is not SPEM 2.0 notation, but it simply means that there are various ways of realizing a generic task. For example, the "Establish Requirements Baseline without Test Cases" task described before is one way of realizing the "Establish Requirements Baseline" task that appears in Figure 2. We will continue to discuss this variability example in the next section.

Finally, we can specify a process by using the element definitions we have previously described. A process is a set of activities, where the relationship between these activities is also specified using an activity diagram. Task, role, work product and activity definitions are recommendations made by a process engineer, so they can be overridden when creating a new process, e.g., by adding/removing a work product from a task. For example, Figure 3 shows the general development process used at Rhiscom, which consists of five major activities (where the "Requirements" activity is the one detailed in Figure 2), carried out in sequence. We have marked two activities are optional: "Requirements" and "Design". SPEM 2.0 does not have a stereotype to visually indicate optionality, so the notation in Figure 3 is not standard.

3. SOFTWARE PROCESS VARIABILITY MODELING

In this section, we first discuss variability in the context of software process lines, as well as identify various examples of variability in our running example. We then discuss a general methodology for identifying and specifying software process variability.

Companies tend to use similar processes to develop different types of projects (e.g., new development, maintenance, extension, etc.). Given a set of similar processes, we can create a family of processes by identifying the common aspects of these processes, as well as how these vary according to the type of project being developed. Thus, a software process line (or family) consists of two things: a general model of the process, as well as a specification of what process elements vary, and how. Individual process definitions are created by removing variability from the general model, allowing the generation of process definitions that can be specifically tailored to each new project.

Now we can continue explaining the "Requirements" activity example presented Figure 2. This activity is a general workflow of RE tasks that must be carried out as part of the RE phase of Rhiscom's development process. The diagram in Figure 2 includes two types of tasks: concrete (non-variable) tasks, like "Hold First Meeting", and generic (variable) tasks, like "Specify Requirements". In this example, "Specify Requirements in plain text" and "Specify Requirements in Use Cases" are valid alternatives to the "Specify Requirements" task, while the "Establish Requirements Baseline" task can be replaced by the "Establish Requirements Baseline without Test Cases" or "Establish Requirements Baseline & Test Cases" tasks. All configurations of the "Requirements" activity share the same concrete tasks, only varying in their instatiation of the generic tasks. Note that all sources of variability must be removed when configuring the activity.

The general development process shown in Figure 3 exhibits another form of variability. In this case, two activities ("Requirements" and "Design") have been marked as optional. Since Rhiscom develops for a niche market, it does not al-

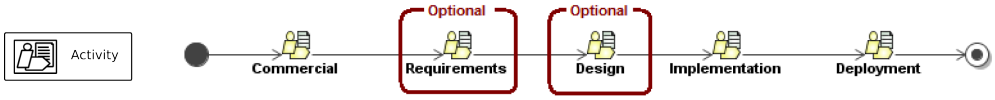

Figure 3: Rhiscom's general development process

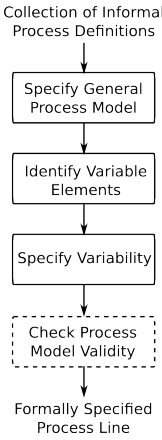

Figure 4: Overview of our methodology for specifying software process lines.

ways carry out the "Design" phase because new products are similar to already-developed products. In the case of larger projects, or those where there is some uncertainty, Rhiscom carries out both phases. Maintenance projects usually skip both phases, since the client has already specified the required improvements in the "Commercial" phase. Again, the variable elements of the model must be removed through a configuration step before the process can be enacted and applied to a project.

In Table 1, we have listed the different instances of variability that we encountered when formalizing Rhiscom's development process. Eight modeling elements (column one) were identified as variation elements in this process; the type of each modeling element is described in the "Type" column; the place in the process where the modeling element is located is indicated in the "Location" column; whether the modeling element is optional or has some alternatives is specified in the last column. As seen in Table 1, we identified several types of variable elements: two activities, four tasks, one work product and one role; and variable elements were either optional, or had alternatives (these are listed in the table).

During the process of formalizing Rhiscom's development process, we identified a set of necessary steps for extracting a software process line from a collection of existing processes. The overview of this methodology is given in Figure 3. First, Rhiscom's employees informally described the development processes they use in practice. Given that these processes had common elements and workflows, we started by formalizing the shared parts of these processes (corresponding to the "Specify General Process Model" step). We then identified necessary variation points of the general process, i.e., process elements that vary amongst the input processes (the "Identify Variable Elements" step). After identifying the variable elements of the process, we specified how these elements vary (the "Specify Variability" step).

The final step of our methodology, "Check Process Model Validity", is optional. Whether or not this step is required depends on the type of formalism used to specify variability. As discussed in Section 1, there are two families of variability specification formalisms: domain-specific and general variability modeling formalisms. SPEM 2.0 and vSPEM are examples of domain-specific variability modeling formalisms, while feature models and OVM are examples of general variability modeling formalisms. When using the second class of formalisms, we must be careful to ensure that the resulting specification actually corresponds to a valid process model, and thus the extra step is required in this case. The end result is a formally specified software process line, which can now be configured to each new project.

4. FORMALISMS FOR MODELING VARIABILITY IN SOFTWARE PROCESSES

In this section, we give an overview of various formalisms that can be used to specify software process variability. In Sections 4.1 and 4.2, we discuss two domain-specific approaches that have been specifically designed for modeling process line variability, and in Sections 4.3 and 4.4 we discuss how more general variability modeling formalisms can also be used to specify software process variability.

4.1 SPEM Primitives

The SPEM 2.0 standard defines four indirect variability relationships, which must be specified between two variability elements[2] of the same type (e.g., between two work products, between two roles, etc.):

1. Contributes: a source variability element «contributes» its properties to the target variability element without directly altering any of the target element's properties. The target element takes on any extra attributes and associations defined by the source element, except for those already defined by the target element.

2. Replaces: a source variability element «replaces» its target variability element. In this case, only the incoming associations of the target element are preserved, both the target's attributes and outgoing associations are replaced by the source element's. The target of multiple «replaces» relations can only be replaced by one source element in a configuration.

3. Extends: a source variability element «extends» the definition of its target variability element, possibly overriding the target's attributes and associations.

[2]In this work, we restrict ourselves to the following variability elements: tasks, roles and work products.

Table 1: Types of variability encountered in practice.

Modeling Element	Type	Location	Optional or Alternative
Requirements	Activity	General process	Optional
Design	Activity	General process	Optional
Specify Requirements	Task	Requirements Workflow	• Alt1: Specify Requirements in plain text • Alt2: Specify Requirements in Use Cases
Establish Requirements Baseline	Task	Requirements Workflow	• Alt1: Establish Requirements Baseline without Test Cases • Alt2: Establish Requirements Baseline Test Cases
Meet for integration agreements	Task	Design Workflow	Optional
Execute Test Cases	Task	Construction Workflow	Optional
SRS Baseline	Work product	Establish Requirements Baseline Task	• Alt1: Final SRS • Alt2: SRS & Test Cases
Analyst	Role	Requirements Workflow	• Alt1: Analyst Tester • Alt2: Analyst with Use Case skills

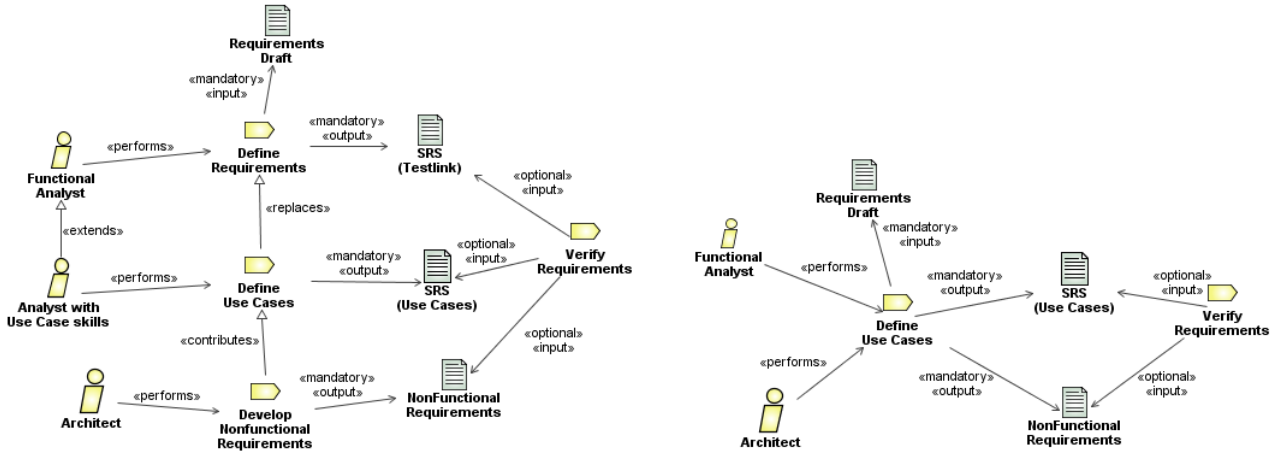

Figure 5: (a) SPEM 2.0 specification of the variability associated to the "Specify requirements" task, and (b) a valid configuration of this model

4. Extends-Replaces: in this relationship, a source variability element first «extends» its target variability element, and then «replaces» it.

Instances of these relations may override each other, so variability relations must be resolved in a predetermined order (first «contributes», then «replaces», then «extends» and finally «extends-replaces»). The process tailoring step is successful if the process engineer can resolve away all variability (in a non-conflicting manner). A priori, it is hard to predict how variability relations interact with each other, which is why the SPEM 2.0 variability mechanism is classified as an indirect variability specification mechanism.

In Figure 5a, we used the SPEM 2.0 primitives to model the variability associated to the "Specify requirements" task (which is part of the "Requirements" activity). The figure shows three variability relations: the task "Develop Nonfunctional Requirements" contributes to task "Define Use Cases", the task "Define Use Cases" replaces "Define Requirements" and the role "Analyst with Use Case skills" extends "Functional Analyst". In order to remove variability from this model, we must first resolve the «contributes» relation between the two tasks: task "Define Use Cases" acquires two associations, an incoming one from "Architect" and an outgoing one to "NonFunctional Requirements". The next step is to resolve the «replaces» relation between the two tasks: "Define Use Cases" replaces "Define Requirements", while keeping its own incoming and outgoing associations. Finally, "Functional Analyst" inherits "Analyst with Use Case skills"'s link to "Define Use Cases". The resulting SPEM model (without variability) is shown in Figure 5b.

Even in this small model, it was difficult to foresee the results of the tailoring step. Regular process models can get much larger, involving dozens of tasks and even hundreds of work products, so the overall effect of the modeled variability is not always clear. Also, deciding how and where to include variability is a time-consuming, non-replicable and people-dependent process.

4.2 vSPEM

Taking into account the limitations of the existing SPEM 2.0 variability mechanisms mainly from the understandability point of view, Martinez et al. [31] have proposed vSPEM, a SPEM 2.0 extension that allows the direct specification of process variability. In this proposal, the process engineer defines process variation points (called *VarPoints*), as well as variants that can fill the variation points. In other words, the relationship between a variation point and its variants

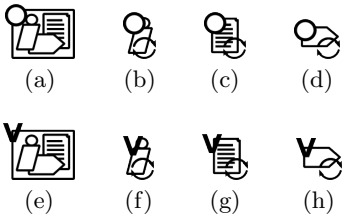

Figure 6: (a) – (d) *VarPoint* icons, and (e) – (h) Variant icons.

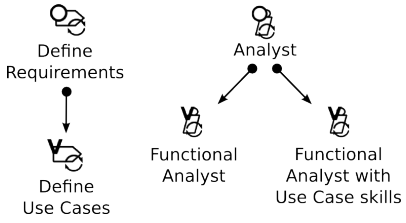

Figure 7: vSPEM specification of two variation points associated to the "Specify Requirements" task.

is a SPEM 2.0 «replaces» relation, and during the process tailoring step, each variation point is replaced by exactly one variant (which must be of the same type). Also, variation is now specified at the "use" level instead of the process component "definition" level, i.e., activities, role uses, work product uses and task uses are valid variation points.

vSPEM introduces new icons to represent variation points and variants: the *VarPoint* icons are shown in Figures 6a – 6d, and the variant icons are shown in Figures 6e – 6h. In Figure 7, we show a partial vSPEM specification of the variability modeled in Figure 5a. This model has two variation points, "Define Requirements" and "Analyst". The "Define Requirements" variation point models the «replaces» variability relation in Figure 5a, and as such, has only one variant, "Define Use Cases". On the other hand, SPEM 2.0 indirect variability relations («contributes», «extends» and «extends-replaces») must be modeled by introducing new variants that already include extra associations required by the definition of the corresponding variability relation. For example, the "Analyst" variation point has two variants, "Functional Analyst" and "Functional Analyst with Use Case skills", which model the two possible instantiations of the «extends» relation in Figure 5a. The first variant represents the base role, a functional analyst, and the second variant is the extended role, which is a functional analyst that can also specify use cases.

The tailoring step is now much simpler: since each variation point is associated to one or more variants, a valid configuration includes exactly one variant for each variation point in the process model. Also, since the indirect variability relations have been made explicit by including new variants, it is also much clearer as to which associations are involved in the final model. For example, if the configuration of the process model includes the "Define Use Cases" task, then the roles and work products associated to this task must be included in the configured model, information that may be obfuscated in a larger SPEM model.

According to the empirical studies carried out by the authors in [31], the vSPEM notation was much more intuitive and easy to use than the SPEM 2.0 variability mechanism. Users also found that the results of the process tailoring step were more akin to their idea of the modeled variability when using the new notation.

4.3 Feature Models

Basic Feature Models [24] (BFM) are frequently used to model variability in product line engineering [13]. In a BFM, features are organized hierarchically, with edges representing parent-child relationships between features. A set of cross-tree constraints is also used to indicate relationships between non-directly related features.

BFMs allow the following parent-child relationships:

- Mandatory (*man*): the child feature must be included in all configurations in which the parent feature appears.
- Optional (*opt*): the child feature can be included in any configuration in which the parent feature appears.
- Alternative (*alt*): exactly one child feature can be included in any configuration in which the parent feature appears.
- Or (*or*): one or more child features can be included in any configuration in which the parent feature appears.

Cross-tree constraints are simple boolean formulas between nodes of the BFM. In this work, we consider the following two types of cross-tree constraints:

- Requires: $m \rightarrow \bigwedge_{i=1\ldots k} n_i$, i.e., feature m requires the inclusion of features $n_1, n_2 \ldots n_k$.
- Excludes: $m \rightarrow \bigwedge_{i=1\ldots k} \neg n_i$, i.e., feature m requires the exclusion of features $n_1, n_2 \ldots n_k$.

For example, the feature model shown in Figure 8 models both the common and variable elements of the "Requirements" activity (see Figure 2). This activity has seven mandatory tasks, which are modeled as mandatory features (which are preceeded by the • symbol in Figure 8). The two tasks that can be realized in more than one way (labeled "Has alternatives" in Figure 2) are decomposed into additional subfeatures using an alternative relationship (indicated by the ⩚ symbol in Figure 8), where alternatives are preceeded by the ▫ symbol. This model does not have any cross-tree constraints.

A configuration of a BFM is a subset of its nodes. A valid configuration of a feature model satisfies the specified parent-child relationships, as well as the model's cross-tree constraints. For example, a valid configuration of the feature model in Figure 8 includes all the mandatory features listed under "Requirements Workflow", including exactly one alternative for each feature involved in an alternative relation

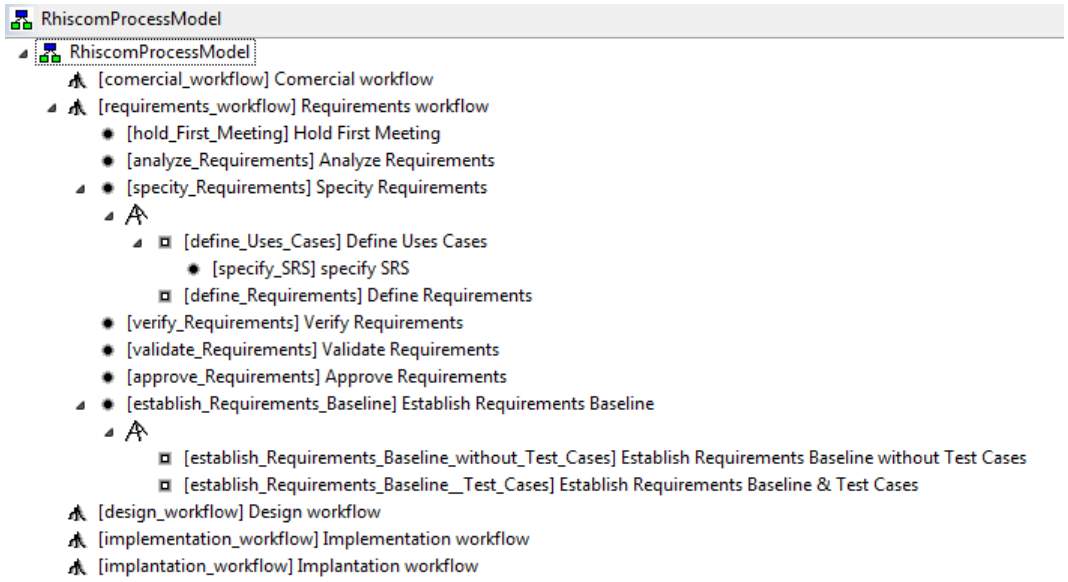

Figure 8: Screenshot of the fmp tool.

(e.g., "Define Use Cases" and "Establish Requirements Baseline & Test Cases").

Cardinality-based Feature Models [15] (CFM) are an extension to BFMs, where UML-like multiplicities (so-called *cardinalities*) have been added to certain elements of the model:

• Feature cardinality ($[m..n]$): specify the number of instances of the feature that can be part of the final configuration, where m and n are the lower and upper bounds, respectively.

• Group cardinality ($< m..n >$): specify the number of child features that can be selected when its parent feature is selected, where m and n are the lower and upper bounds, respectively.

Feature cardinalities can be used to indicate how many team members that meet a given profile are needed to carry out the tailored process. For example, by adding the feature cardinality [1..2] to the feature representing "Functional Analyst", we are indicating that one to two team members that meet the "Functional Analyst" profile are needed to carry out the "Define Requirements" task in the "Specify Requirements" activity. Group cardinalities affect how many variants can replace a variation point. By definition, variation points must be replaced by one variant, so we have not used this notation in our modeling experiments.

Finally, Extended Feature Models [24] (EFM) are an extension to BFMs, where additional information about features is available. This is usually accomplished by adding feature attributes [8, 9, 25]. There is no consensus on a notation to define attributes. However, most proposals agree that an attribute should at least have a name, domain and value. For example, we have annotated the features in Figure 8 with their SPEM type, in this case, all features are tasks. Note that feature models specifying process variability can also include roles and work products, but we have not included them in this example.

4.4 Orthogonal Variability Modeling

Orthogonal Variability Models [36, 37] (OVM) is another notation used to model product line variability. The main difference between OVMs and feature models is that OVMs only document the variabilities present in a product line, whereas feature models model both the common aspects of a product line and its variability. Thus, OVM elements are either variation points or variants: variation points indicate elements that may vary, while variants represent different possible realizations of a variation point.

Each variation point must be related to at least one variant, which is called a "dependency" in OVM (and corresponds to the BFM concept of relations). OVM defines three types of dependencies: mandatory, optional, and alternative, which are interpreted just like their BFM counterparts. Group cardinalities can be defined for alternative dependencies. Additionally, each variant must be related to one and only one variation point. Finally, OVMs also allow requires and excludes constraints between modeled elements, these can be defined between two variants or two variation points, or between a variation point and a non-dependent variant. Another difference with feature models is that OVMs elements are not hierarchical, since each variation point is orthogonal to the rest (hence the name of the modeling notation); however, this hierarchy can be modeled using constraints.

Figure 9 shows how we modeled the variability in the "Requirements" activity using VEdit, an OVM graphical modeling tool. Since OVM focuses on variability, we only included the variable elements of Figure 2 in this model (unlike the feature model in Figure 8, which included both common and variable elements). The "Requirements" activity has two variation points, "Specify Requirements" and "Establish Re-

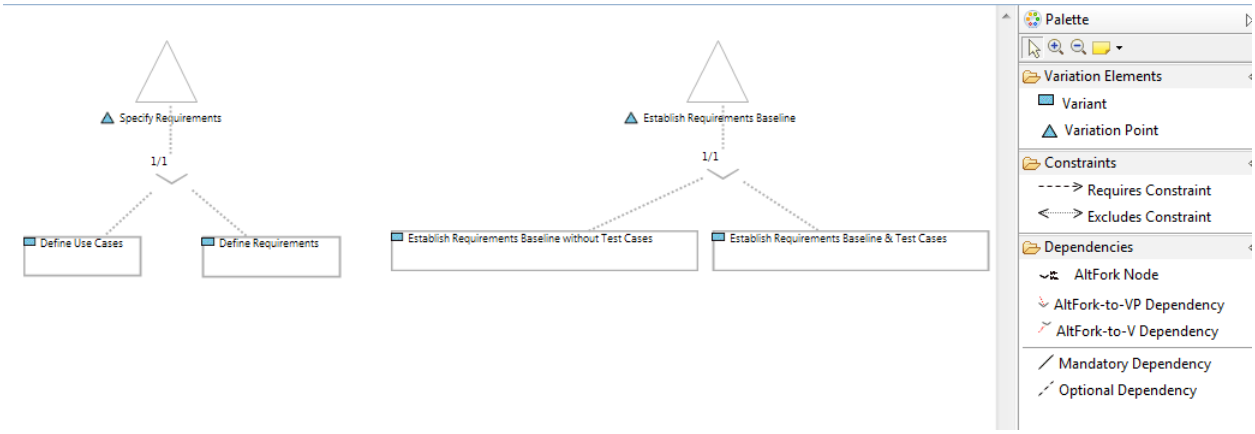

Figure 9: Screenshot of the VEdit tool.

quirements Baseline", each of which has two variants. This model does not have any additional contraints. Just like feature models, we can talk about valid OVM configurations. A valid configuration of an OVM is a subset of its variants such that its dependencies and constraints are satisfied.

4.5 Discussion

The SPEM 2.0 standard advocates the specification of a single process model that includes variability, where the mechanisms for specifying variability are quite expressive. Ideally, such a general process specification language would allow the definition of highly reusable processes; however, in practice, too much generality has its disadvantages. In this case, since both the process and its variability are defined together, and variability is specified indirectly, it is hard to determine whether process variability was modeled correctly. The vSPEM proposal tries to address these concerns by doing two things: 1) specifying variability independently from the process, and 2) simplifying the variability specification mechanisms. The authors of the proposal carried out some preliminary usability studies [31], which indicate that users effectively found it easier to work with vSPEM rather than SPEM 2.0. Finally, we also studied how process variability can be specified using popular product line formalisms: different types of feature models (Section 4.3) and orthogonal variability models (Section 4.4). Since these are general variability modeling languages, we must always check whether models specified in these languages correspond to valid SPEM 2.0 models. In our (limited) experience, we did not need the full expressiveness of these languages to formalize process variability, as processes seem to vary less than products. Thus, the decision about which formalism to use comes down to a question of available tool support, which we discuss in the next section.

5. TOOL SUPPORT EVALUATION

One of the main advantages of formalizing variability models is that we can analyze these models using automated techniques. For example, by mapping a feature model into a propositional formula [32], an off-the-shelf SAT solver can be used to determine whether a feature model is consistent or not (i.e., it has at least one valid configuration), or whether a given configuration is valid. By formalizing variability, we can apply these same analyses to process variability models. For example, we can check that the modeled variability is consistent, and also determine valid configurations that can be used to automatically tailor the corresponding process.

Table 2 summarizes some of the main features of eight tools that support process variability modeling. In this table, column "Supported Formats" lists which standard input/output file formats are supported by the tool; "Underlying Formalism" indicates which formalism from Section 4 is used by the tool; "Supported Analyses" lists the relevant analysis tasks offered by the tool; "Interface" lists the types of user interfaces the tool has; "Availability" indicates the state of the tool and how it is made available. We used these tools to model the variability identified in Table 1, and we indicate the degree of usability of each tool in column "Usability" (values explained below).

Clafer. Clafer (**cla**ss, **fe**ature, **r**eference) [7] is a concept modeling language for specification and analysis of software product lines. Clafer provides first-class support for feature modeling, including feature modeling extensions like cardinality-based feature modeling. The authors have developed a first version of a Clafer to Alloy [23], which enables model analysis through the use of the Alloy Analyzer. The Alloy Analyzer supports various analyses, like checking model consistency and generating valid configurations. The authors also provide a SPLOT to Clafer translator, as well as a Clafer parser in Java (available at [27]). We assigned this language a Low usability score because it does not have its own IDE.

EPF Composer. The Eclipse Process Framework (EPF) Composer [19] aims at producing a customizable software process enginering framework. The Process Framework Project has two goals: to provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process, and to provide exemplary and extensible process content for a range of software development and management processes supporting iterative, agile, and incremental development, and applicable to a broad set of development platforms and applications. Its usability was

Table 2: Available tool support.

Tool	Supported Formats	Underlying Formalism	Supported Analyses	Interface	Availability	Usability
Clafer [7]	xml and ecore	FM	Check model consistency, and generate valid configurations	Text-based	Clafer to Alloy parser available	Low
EPF Composer [19]	xmi	SPEM 2.0	Check metamodel consistency	Form-based	Open source, available online	Medium
FaMa-OVM [44]	xml	OVM	Check metamodel and model consistency, and generate all valid configurations	Text-based	Preliminar prototype available, new version under construction	Low
fmp [28]	xml and uml	FM	Check model consistency, generate valid configurations, and check partial configurations	Form-based	Eclipse plugin, no longer supported	Medium
Hydra [17]	xml	FM	Check model consistency	Graphical Editor	Prototype tool, research thesis	Medium
SPLOT [26]	xmi and xsml	FM	Check model consistency, generate valid configurations, and find the number of common and dead features	Form-based	Available online	Medium
VEdit [33]	xml	OVM	Syntax check (based on OVM metamodel)	Graphical Editor and Form-based	Prototype research, available Eclipse plugin	Medium
XFeatures [41]	xml and xmi	FM	Check metamodel consistency and validate feature model	Graphical Editor	Open source, available Eclipse plugin	Medium

evaluated to Medium because it provides a form-based user interface, but its underlying variability concepts are complex. Nevertheless, and considering that it is a domain-specific environment, the "Chech Process Model Validity" stage of the methodology is not required because it only allows syntactically correct process variability specification.

FaMa-OVM. FaMa-OVM [44] provides automated analyses for orthogonal variability models. This tool offers basic analyses, like checking consistency and generating valid configurations, but also focuses on verifying user-specified quality conditions, allowing the generation of an optimal configuration as well as the most representative one. This tool is still in the prototype stage, and is available at [43]. We have assigned this tool a Low usability score because its input format has not yet been documented, and it only offers a limited text-based console to carry out model analysis.

fmp. The Feature Modeling Plug-in (fmp) [28] is an Eclipse plug-in for editing and configuring cardinality-based feature models. Fmp can be used standalone in Eclipse or together with fmp2rsm plug-in in Rational Software Modeler (RSM) or Rational Software Architect (RSA). fmp2rsm integrates fmp with RSM and enables product line modeling in UML. This tool is mainly used to check feature model consistency, as well as to generate valid configurations and to check the validity of partial configurations. The project has been completed, so the tool is no longer maintained by its original developers; however, the project is now open-source, and the code is available on SourceForge. A screenshot of the tool, showing the feature model from Section 4.3, is shown in Figure 8. We assigned this tool a Medium usability score because it only works on older versions of Eclipse, and cross-tree constraints are not shown explicitly.

Hydra. Hydra [17] offers a full graphical interface (copy, paste, zoom, etc.) for specifying feature models. This tool also allows the specification of cross-tree constraints, using the CSP solver Choco [14] to check these constraints. Hydra is implemented as a plug-in for Eclipse and is based on de facto standards within the modeling community, such as Ecore [49] and GMF [40], which promotes interoperability with other tools. A screenshot of the tool, showing the feature model from Section 4.3, is shown in Figure 10. We assigned this tool a Medium usability score because cross-tree constraints must be expressed directly in the syntax that the Choco tool takes as input.

SPLOT. SPLOT (Software Product Lines Online Tools) [26] is a collection of interactive online tools for editing, configuring, and analyzing feature models. SPLOT is also a feature model repository. SPLOT supports basic feature models (i.e., no cardinality) and offers basic model analyses, like those described at the beginning of this section, as well as some more advanced analyses like finding the number of common and dead features (common features appear in every model configuration, while dead features do not appear in any configuration). A screenshot of the tool, showing the feature model from Section 4.3, is shown in Figure 11. We assigned this tool a Medium usability score because models are shown as simple trees, and cross-tree constraints must be entered directly as boolean formulas.

VEdit. The VEdit [33] (VARMOD-EDITOR) is part of the VARMOD-Tool-Environment. VEdit supports the specification of product line variability models using Orthogonal Variability Models. This tool offers a graphical model editor, supporting the definition of variation points and variants, variability constraints and constraint dependencies (requires, excludes). A screenshot of the tool, showing the

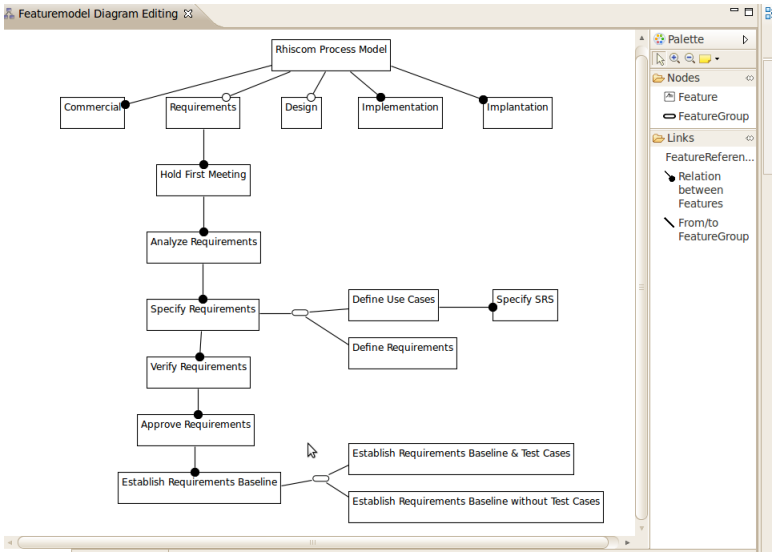

Figure 10: Screenshot of the Hydra tool.

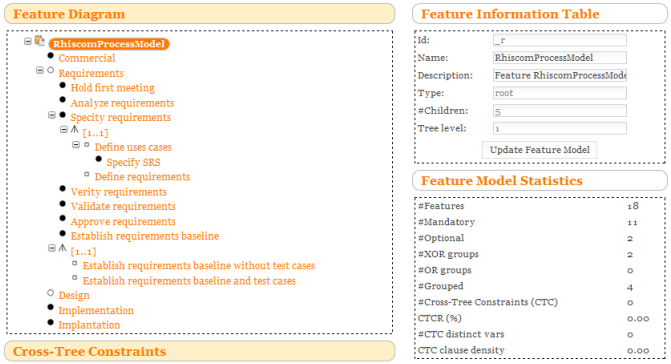

Figure 11: Screenshot of the SPLOT tool.

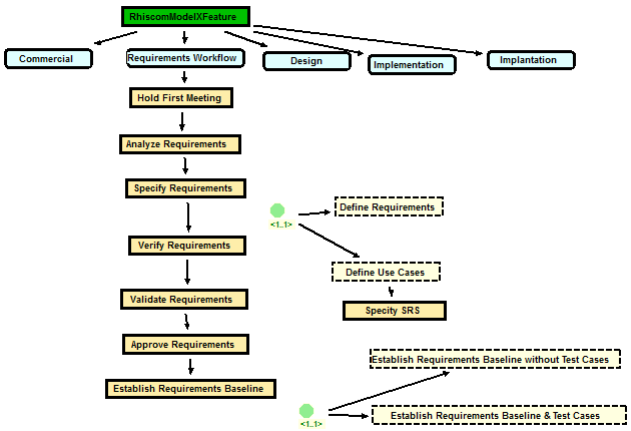

Figure 12: Screenshot of the XFeatures tool.

OVM from Section 4.4, is shown in Figure 9. We assigned this tool a Medium usability score because, even though models can be edited both graphically and in text-mode, this tool does not offer any of the model analyses made available by other OVM-based tools like FaMa-OVM.

XFeatures. XFeatures [41] is a graphical feature model editing tool. This tool checks for dependency and cross-tree constraint consistency, but does not generate valid configurations. XFeatures is an active, open source project (although the latest version of the tool dates from 2005), and is available as a plug-in for Eclipse. A screenshot of the tool, showing the feature model from Section 4.3, is shown in Figure 12. We assigned this tool a Medium usability score because, even though the feature models are shown graphically, it uses an unusual color convention and its cross-tree constraint editor is quite poor.

6. RELATED WORK

Providing a detailed comparative analysis to show the relative advantages and disadvantages of different variability modeling approaches would provide guidance for selecting a particular approach in a specific context and would offer practitioners a qualified portfolio of available techniques [12]. This idea was widely developed in the Software Product Line (SPL) and Business Process Management (BPM) communities; however it has been poorly addressed by the SPrL community yet.

6.1 Software Product Lines (SPL)

Several authors within the SPL community have reported comparisons among different existing variability modeling methods. Sinnema et al. [47] discuss the commonalities and differences between six variability modeling techniques: VSL [10], ConIPF [22], CBMF [15], COVAMOF [48], Koalish [5], and Pure::Variants [39]. The authors use an example product family in the domain of license plate recognition on handhelds, such as PDAs, to illustrate the different approaches. The methods are also evaluated using a framework focused on two major aspects: how variability information

is represented (representing choices and products, using abstractions to manage complexity) and tool support (views, active specification, configuration guidance, inference, effectuation).

Djebbi et al. [18] evaluated some Product Lines Management Tools: XFeature [41], Pure::Variant [39], and RequiLine [52] to determine to what extent such tools address the set of thirteen criteria whose definition is based on technical and scientific industrial expectations. After the evaluation, Pure::Variant and RequiLine were the tools that best satisfied the defined criteria.

Asikainen [4] presents a technical report that discusses SPL variability. The core part of his discussion consists of an analysis and comparison of methods for modeling variability. The evaluated methods fall in three categories: feature-based, architecture-based, and other methods. In order to give each modeling method a uniform treatment, the author used the following criteria for the evaluation: a) What is being tried to achieve?; b) What is the knowledge being represented and reasoned about?; c) What is the semantics given to the knowledge?; d) What kind of tool support is provided or suggested to supportting approach?; and e) How has the method been evaluated?. The evaluated methods were: CONSUL [11], RequiLine [52], Mannion [30], Koala [51], and COVAMOF [48].

6.2 Business Process Management (BPM)

In the world of BPM, there are also some authors that have reported evaluations of existing tools and techniques for managing variability in process models. Aiello et al. [2] reported the evaluation of existing tools and frameworks for variability management. Such evaluation was made considering five requirements: a) those that deal with the expressive power for specifying variability; b) techniques supported to derive variability expressions; c) requirements for service-based processes with variability; d) run-time requirements connected with consistency and fault handling; and e) requirements stemming from the need of managing evolution of processes with variability. The tools and frameworks involved in the evaluation were: the Process Variants by Options (PROVOP) [21], the Variability extension to Business Process Execution Language (VxBPEL) [1]; ADEPT [16]; the configurable workflow models (CWM) [20]; the DECLARE framework [35]; the Business Process Constraint Network (BPCN) [29]; and the Process Variant Repository (PVR) [46]. The authors conclude that none of the existing frameworks addresses all or even most of the five requirements. Therefore, there is space for extensive research and development in the area of frameworks for the explicit management of variability.

Ayora et al. [6] reported the results of the application of Configurable Event-driven Process Chain (C-EPC) [45], Variant-Rich Process Models (within the PESOA) [38] and PROVOP [21] to manage the process variability in three case studies: a) a vehicle repair process; b) a healthcare process; and c) an e-business shop. In order to evaluate how well such approaches deal with business process variability modeling, the authors defined twenty-two criteria to measure if the approaches provide the mechanisms that allow the specification dealing with variability concepts and if they have any other desirable quality factors to ensure their successful adoption. After analyzing the selected approaches, the PROVOP approach achieved almost all of the twenty-two requirements needed when dealing with variability.

6.3 Software Process Variability

Currently, there are several proposed techniques and tools for addressing variability modeling in SPL and BPM; however few techniques and tools have been proposed to specifically address variability modeling in software processes. To the best of our knowledge, SPEM 2.0 [34], vSPEM [31] and the German V-Modell XT (an existing example for a software process line) [42] seem to be the only systematic efforts toward addressing variability modeling in software processes. However, beyond having a set of techniques, it provides significantly better benefits to have a comparative analysis of such techniques. As was described above, there are several efforts that try to identify the most suitable techniques and tools for addressing variability modeling in SPL and BPM; however it seems that the software process community has not addressed this topic extensively yet. Since most of the concepts of SPL can be transferred to software processes [42][3][50] and that there is an analogy between software processes and business processes, it seems reasonable to expect that the existing tools and techniques for modeling variability in SPL and BPM could be also useful for variability modeling in software processes. The question that arises is now, what tools and techniques available for variability modeling in SPL or BPM could be appropriate for modeling variability in software processes? This research work is a first effort to analyze SPEM 2.0, vSPEM and some SPL tools and techniques for variability modeling in order to investigate the suitability of such approaches for variability modeling in software process.

7. CONCLUSIONS

All four notations that have been analyzed for specifying software process model variability have the expressive power for capturing all kinds of relevant variability. However, those general purpose languages -feature models and OVM- are more powerful, so they may allow to express certain conditions that are meaningless for software processes, e.g. a variant work product that may be realized by two different variant roles, because all process model elements are indistinguishable. This situation makes it error prone to use general purpose notations for modeling variability in software processes.

SPEM 2.0 counts on a series of native constructs for modeling variability. Moreover, the EPF Composer supports the specification of all of them providing an integrated environment for modeling the process and its variability. However, the understandability of the specification has been proved to be very low. Building and evolving these process models in highly difficult and almost unpredictable.

vSPEM improves understandability of SPEM 2.0 constructs allowing the specification of process variability. This notation is highly promising because it includes variability forms captured in feature models but restricted to what is relevant for processes. Nevertheless, there are still no supporting tools for vSPEM, and thus we cannot incorporate variability

specification in vSPEM in our tool chain for implementing automatic process tailoring, that was our initial goal.

Feature models are the most mature and widespread notation for specifying variability. As such, there are several good supporting tools, but there is still there the difficulty of having a notation that allows specifying situations that are meaningless.

OVM is a very promising notation because it has all the power of feature models but it is much more compact since it is not necessary to specify commonalities and these are most of the modeling elements in the process model domain. Supporting tools for OVM are still immature: they provide a nice simple visualization, but the output of the specification does not comply with any standard, and this makes it very unconvenient to incorporate these tools in the tool chain.

As a conclusion, we will be using feature models for the moment being careful about what we specify. The SPLOT tool will be our best option for the moment because it provides a robust user interface, it generates standard xml output, and it provides a rich set of analyses. In a near future we will still need to reevaluate the possibility of using either vSPEM or OVM as their tools get more mature.

8. REFERENCES

[1] Chang ai Sun and Marco Aiello. Towards Variable Service Compositions Using VxBPEL. In Hong Mei, editor, *High Confidence Software Reuse in Large Systems, 10th International Conference on Software Reuse, ICSR 2008*, volume 5030 of *Lecture Notes in Computer Science*, pages 257–261. Springer, 2008.

[2] M. Aiello, P. Bulanov, and H. Groefsema. Requirements and Tools for Variability Management. In *34th Annual IEEE Computer Software and Applications Conference Workshops, COMPSACW*, pages 245–250, 2010.

[3] Ove Armbrust, Masafumi Katahira, Yuko Miyamoto, Jürgen Münch, Haruka Nakao, and Alexis Ocampo. Scoping software process lines. *Software Process: Improvement and Practice*, 14(3):181–197, 2009.

[4] Timo Asikainen. *Modelling Methods for Managing Variability of Configurable Software Product Families.* PhD thesis, Helsinki University of Technology, Department of Computer Science and Engineering, 2004.

[5] Timo Asikainen, Timo Soininen, and Tomi Männistö. A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families. In Frank van der Linden, editor, *5th International Workshop on Software Product-Family Engineering, PFE 2003*, volume 3014 of *Lecture Notes in Computer Science*, pages 225–249. Springer, 2003.

[6] Clara Ayora, Victoria Torres, and Vicente Pelechano. Dealing with Variability in Business Process Models: An Evaluation Framework. Technical Report PROS-TR-2011-05, Universitat Politécnica de Valéncia, Centro de Investigación en Métodos de Producción de Software, 2011.

[7] Kacper Bąk, Krzysztof Czarnecki, and Andrzej Wąsowski. Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled. In *3rd International Conference on Software Language Engineering*, Eindhoven, The Netherlands, 10/2010 2010.

[8] Don S. Batory. Feature Models, Grammars, and Propositional Formulas. In *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, pages 7–20, 2005.

[9] Don S. Batory, David Benavides, and Antonio Ruiz Cortés. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 49(12):45–47, 2006.

[10] Martin Becker. Towards a general model of variability in product families. In Jan Bosch Jilles van Gurp, editor, *Workshop on Software Variability Management*, pages 19–27, 2003.

[11] Danilo Beuche, Holger Papajewski, and Wolfgang Schröder-Preikschat. Variability Management with Feature Models. *Science of Computer Programming*, 53(3):333–352, 2004.

[12] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In Dirk Muthig and John D. McGregor, editors, *13th International Conference on Software Product Lines, SPLC 2009*, volume 446 of *ACM International Conference Proceeding Series*, pages 81–90. ACM, 2009.

[13] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley Professional, third edition, August 2001.

[14] EMN Constraint. Choco Solver. `http://http://www.emn.fr/z-info/choco-solver/`, Accessed September 2011.

[15] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[16] Peter Dadam and Manfred Reichert. The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support. *Computer Science-Research and Development*, 23(2):81–97, 2009.

[17] Salazar Jose R. / Universidad de Málaga. Herramienta para el modelado y configuración de modelos de características, in Spanish. `http://caosd.lcc.uma.es/spl/hydra/`, Accessed October 2011.

[18] Olfa Djebbi, Camille Salinesi, and Gauthier Fanmuy. Industry Survey of Product Lines Management Tools: Requirements, Qualities and Open Issues. In *15th IEEE International Requirements Engineering Conference, RE 2007*, pages 301–306. IEEE, 2007.

[19] Eclipse Foundation. Eclipse Process Framework Project. `http://www.eclipse.org/epf/`, Accessed October 2011.

[20] Florian Gottschalk, Wil M. P. van der Aalst, Monique H. Jansen-Vullers, and Marcello La Rosa. Configurable Workflow Models. *International Journal of Cooperative Information Systems*, 17(2):177–221, 2008.

[21] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Managing Process Variants in the Process Life Cycle. In José Cordeiro and Joaquim Filipe, editors, *ICEIS 2008 - Proceedings of the Tenth*

International Conference on Enterprise Information Systems, Volume ISAS-2, pages 154–161, 2008.

[22] Lothar Hotz, Katharina Wolter, Thorsten Krebs, Sybren Deelstra, Marco Sinnema, Jos Nijhuis, and John MacGregor. *Configuration in Industrial Product Families: The ConIPF Methodology*. IOS Press, 2006.

[23] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[24] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University, November 1990.

[25] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, January 1998.

[26] Computer Systems Group / Generative Software Development Lab. SPLOT - Software Product Line Online Tools. `http://www.splot-research.org/`, Accessed September 2011.

[27] Generative Software Development Lab. Clafer. `http://gsd.uwaterloo.ca/clafer`, Accessed September 2011.

[28] Generative Software Development Lab. Feature Modeling and Model Templates. `http://gsd.uwaterloo.ca/featureModelingAndModelTemplates`, Accessed September 2011.

[29] Ruopeng Lu, Shazia Wasim Sadiq, and Guido Governatori. On managing business processes variants. *Data & Knowledge Engineering*, 68(7):642–664, 2009.

[30] Mike Mannion. Using First-Order Logic for Product Line Model Validation. In Gary J. Chastek, editor, *Software Product Lines, Second International Conference, SPLC*, volume 2379 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2002.

[31] Tomás Martínez-Ruiz, Félix García, Mario Piattini, and Jürgen Münch. Modelling software process variability: an empirical study. *IET Software*, 5(2):172–187, 2011.

[32] Marcílio Mendonça, Andrzej Wasowski, and Krzysztof Czarnecki. SAT-based Analysis of Feature Models is Easy. In *Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24-28, 2009, Proceedings*, pages 231–240, 2009.

[33] Software Systems Engineering Research Group/ University of Duisburg-Essen. VARMOD-PRIME Tool-Environment. `http://www.sse.uni-due.de/wms/de/?go=256`, Accessed October 2011.

[34] OMG. Software and Systems Process Engineering Metamodel specification (SPEM) Version 2.0. `http://www.omg.org/spec/SPEM/2.0`, Accessed June 2011.

[35] Maja Pesic, M. H. Schonenberg, Natalia Sidorova, and Wil M. P. van der Aalst. Constraint-Based Workflow Models: Change Made Easy. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer, 2007.

[36] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, November 2010.

[37] Klaus Pohl, Frank van der Linden, and Andreas Metzger. Software Product Line Variability Management. In *Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings*, page 219, 2006.

[38] Frank Puhlmann, Arnd Schnieders, Jens Weiland, and Mathias Weske. Variability mechanisms for process models. Technical Report TR 17/2005, PESOA: Process Engineering in Service-Oriented Applications, 2005.

[39] PV. Pure systems, 2011. `http://www.pure-systems.com/`.

[40] C.G. Richard. Eclipse modeling project: A domain-specific language (dsl) toolkit, 2009.

[41] O. Rohlik and A. Pasetti. XFeature Modeling Tool. Automatic Control Laboratory, ETH Zürich, Accessed October 2011. `http://www.pnp-software.com/XFeature/Home.html`.

[42] H. Dieter Rombach. Integrated software process and product lines. In Mingshu Li, Barry W. Boehm, and Leon J. Osterweil, editors, *International Software Process Workshop, Unifying the Software Process Spectrum, SPW 2005*, volume 3840 of *Lecture Notes in Computer Science*, pages 83–90. Springer, 2005.

[43] Fabricia Roos-Frantz, David Benavides, A. Ruiz-Cortés, André Heuer, and Kim Lauenroth. Complementary material. `http://www.lsi.us.es/~dbc/material/SofQualJ11/`, Accessed September 2011.

[44] Fabricia Roos-Frantz, David Benavides, A. Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal Special Issue on Quality Engineering for Software Product Lines*, 2011.

[45] Michael Rosemann and Wil M. P. van der Aalst. A configurable reference modelling language. *Information Systems*, 32(1):1–23, 2007.

[46] Shazia Wasim Sadiq, Maria E. Orlowska, and Wasim Sadiq. Specification and validation of process constraints for flexible workflows* 1. *Information Systems*, 30(5):349–378, 2005.

[47] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 49(7):717–739, 2007.

[48] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A Framework for Modeling Variability in Software Product Families. In Robert L. Nord, editor, *Third International Conference on Software Product Lines, SPLC 2004*, volume 3154 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2004.

[49] D. Steinberg, F. Budinsky, M. Paternostro, and

E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2008.

[50] Thomas Ternité. Process Lines: A Product Line Approach Designed for Process Model Development. In *35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009*, pages 173–180. IEEE Computer Society, 2009.

[51] Rob C. van Ommering. Building product populations with sofware components. In *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002*, pages 255–265. ACM, 2002.

[52] Thomas von der Maßen and Horst Lichter. RequiLine: A Requirements Engineering Tool for Software Product Lines. In Frank van der Linden, editor, *5th International Workshop on Software Product-Family Engineering, PFE 2003*, volume 3014 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2003.