Modeling Variability in Software Process Models[*]

Jocelyn Simmonds
Departamento de Informática
Universidad Técnica Federico Santa María
Santiago, 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. Software companies can use SPrLs to address the development of different types of projects – development or maintenance, large or small, complex or simple – and therefore reuse process knowledge in an organized way. This can be achieved by either defining a series of processes, one for each context, or by tailoring a general process to each context. Both approaches have their disadvantages, and currently, there are no conclusive proposals about how to manage process variability. In this paper, we propose a combination of notations and tools for formalizing software process models including their variability, which enables automated SPrL tailoring. We use the Eclipse Process Framework Composer for specifying the general process itself, and SPLOT, a feature modeling tool, for specifying process variability. We then use the Modisco/AMW tool to establish constraints between both models, in order to ensure that only reasonable variability is specified. Using an industrial case study, we show how these tools are used to specify and analyze software process models that include variability.

General Terms

Software processes, variability modeling

Keywords

software process lines, model-driven engineering, process asset reuse

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

1. INTRODUCTION

Software processes are recognized as valuable means for achieving productivity and quality in software development. However, defining a single 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 (SPrL) try to address this problem, advocating the specification of a series of processes, one for each possible project context, ensuring that there is an appropriate process for each context. However, it is hard and sometimes even impossible to anticipate all eventual contexts and thus the appropriate processes. An alternative is to use a software process line (SPrl) by defining a general process that includes potential variability, which is then tailored to the characteristics of each project, resulting in a project-specific process. However, this activity is only cost-effective if it can be automated, and automation is only possible if models are formally specified.

Formal software process specification enables different types of automated processing and analysis, such as consistency checking, evaluation, tailoring, project scheduling and planning, among others. In this context, we have defined the ADAPTE project[1] whose goal is to develop a technological and methodological framework that helps small and medium enterprises (SMEs) automatically tailor their software processes, allowing them to improve their productivity, as well as product quality. As part of this project, we have defined an initial model-based tailoring approach [9], where basic optionality was the only variability allowed. However, in practice we have realized that more sophisticated kinds of variation are necessary [24]. But there is still no standard technical solution proposed for modeling software processes along with their potential variability.

The ADAPTE framework is intended to be used by process engineers at SMEs, and companies of this type tend to use open source technology to reduce costs. Thus, a technical solution for these companies must satisfy the following quality requirements: 1) it must be built on existing open-source projects, so that there are little or no licensing costs;

[1]ADAPTE: Adaptable Domain And Process Transformation Engineering. `http://www.adapte.cl`

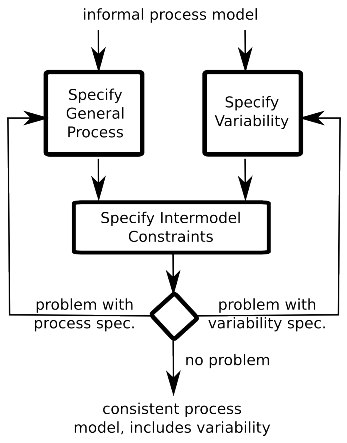

informal process model
Specify General Process
Specify Variability
Specify Intermodel Constraints
problem with process spec.
problem with variability spec.
no problem
consistent process model, includes variability

Figure 1: Overview of our approach.

2) it must be compliant with standard file formats so that our framework can interact with existing SME tool sets; and 3) it must be user-friendly, since there are little resources to be spent on training.

There are various languages specifically created for process specification. We propose to use SPEM 2.0 which is the OMG standard for software process specification [16]. Even though SPEM 2.0 includes primitives for specifying variability, they are hardly understandable by process engineers [14], and thus it is hard to validate processes specified in this way. Taking into account these limitations, Martinez et al. [13] have proposed vSPEM, a SPEM extension for managing variability that incorporates some ideas from the software product line community. However, the tool support for this notation is still immature. In [24], we studied how general variability modeling languages such as OVM and feature models can also be used to specify process variability. Notations that have proved to be powerful for specifying variability in software product lines may not be necessarily appropriate for process variability, mainly because not all variants they allow are valid in the context of processes, e.g., a process element such as a role may not be realized by variants of a different type as a task.

In this paper we propose the use of a series of notations and supporting tools for formally specifying software process models along with their planned variability. The overview of our approach is shown in Fig. 1: the general process and its variability are modeled separately, these models are then linked through integrity constraints so that it can be analyzed for consistency. The final result is the basis of a formally specified SPrL, which can be adapted to different project contexts. The general process model is specified using the Eclipse Process Framework Composer (EPFC), since this tool implements the SPEM 2.0 standard and it is freely available. We choose to specify process variability using feature models, since there is a vast amount of existing work on modeling variability with this formalism. Also, there are various free and user-friendly tools for this notation, like SPLOT, which also provides some built-in consistency checks. However, feature models are highly expressive, and since we are using an independent tool for specifying process models' variability, we could eventually allow the specification of meaningless models, e.g., we could specify features in SPLOT that are not part of the process, or fill a variation point with a process element of the wrong type. In order to deal with these potential problems, our proposal also includes a weaving model, which establishes a series of consistency constraints between the process model and its variability model; we specify this weaving model using the Modisco/AMW tool.

To demonstrate the feasibility of our approach, we describe the application of the complete tool chain to the process model of a medium-size Chilean software company. The tools supporting the activities in Fig. 1 meet the first two quality requirements of our framework: free tools and standard formats. EPFC and SPLOT both also satisfy the third requirement (user-friendliness). However, as the weaving model is manually created by the process engineer, this part of our approach is still cumbersome and not quite user-friendly. We believe that as part of our future work, the accidental complexity of the weaving model must be hidden under an additional interface.

The paper is organized as follows. Section 2 describes the SPEM 2.0 standard for formalizing software processes. Section 3 presents the kinds of variability found in software processes. The details about the modeling tools are presented in Sect. 4. The application example is presented in Sect. 5, desingcrib the general process in EPFC, its variability in SPLOT, and its constraints in AMW. Some related work is discussed in Sect. 6. Finally, we conclude in Sec. 7 with a summary of the paper and suggestions for future work.

2. SPECIFYING PROCESS MODELS WITH SPEM 2.0

SPEM 2.0 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. 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.

2.1 Process Components

SPEM 2.0 encourages process asset reuse by making 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, and these components are then used to define processes. 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 a 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).

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 can be used to define processes.

A work product may go through different states during its 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.

An activity is a logical grouping of task, role and work product uses. Activity diagrams are used to model the workflow between activity tasks. 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, and can be overridden when creating a new process, e.g., by adding/removing a work product from a task.

2.3 Variability

Variability is promoted in SPEM 2.0 by defining process elements that are stored in the Method Library and can be then combined in different ways to produce different processes. However, this mechanism by itself does not provide the process engineer with any guidance about recommended process patterns and process modeling practices. SPEM 2.0 provides four indirect mechanisms for defining how process elements vary: contributes, replaces, extends, and extends-replaces. Their usage rules can be found in [16] even though their practical usage is not widespread.

3. SOFTWARE PROCESS VARIABILITY

In this section, we first describe the types of process variability we have encountered in practice. We then present our methodological approach for modeling process variability including the consistency rules needed to make sure that we only specify correct variability models.

3.1 Software Process Lines

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, a family of processes can be defined by identifying the common aspects of these processes, as well as how they vary according to the type of project being developed. Thus, a software processes 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, corresponding to the analysis and design stages of the SEI's domain engineering, and individual process definitions that are created by resolving the variability in the general model, allowing the tailoring of specific process definitions that can be applied to each new project, as in the application (process) engineering.

Processes are formed by different types of process elements, and how these elements vary determines how the process varies.

3.1.1 Optionality

A task may be optional when it is not always required. That means that the process may still be valid if a certain task is not executed in certain contexts. For example, in any development, the "Architectural Design" document may be mandatory, since it guides the implementation, but the task "Approve Architectural Design" task may not be necessary for an in-house development. When a task is optional, all the work products that it generates should generally be optional as well. In the case of the former example, the "Architectural Design Approaval" would be an optional outcome that would only exist if the "Approve Architectural Design" is executed.

A work product may also be optional. There are two situations where this happens: (1) when the task that generates the work product is also optional, and (2) when the task that generates the work product has two alternatives, one that generates the work product and another that does not. Similarly, a role may be optional if the task it is associated to has two alternatives, one requiring the role and another one that does not require it.

3.1.2 Alternatives

A task may be realized by any one of a series of concrete, alternative tasks – when the process engineer defines the process, he/she needs to pick one of these alternatives. This is the case where different techniques may be applied for realizing a particular task, or, depending on the complexity of the project at hand, tasks of different complexity may be executed. A work product may also have alternatives, but these are generally related to alternative implementations of the tasks that generate them. Alternative roles are also related to task alternatives; e.g., a complex task may require a more competent person in charge than a simple task.

3.2 Our Approach

We propose a combined approach for specifying the analysis and design stages of the SPrL: first, the process engineer specifies the general process and its variability using separate models; we must then determine whether these two models are consistent with each other (by checking if a set of constraints holds on both models). This approach is shown in Fig. 2.

In order to start the tailoring process, i.e. generating particular processes for specific projects, the variability model must be consistent with the general process model. If the models are not consistent with each other, we should be able to give the user feedback about which rules were broken so that he/she can fix the input models. We have identified

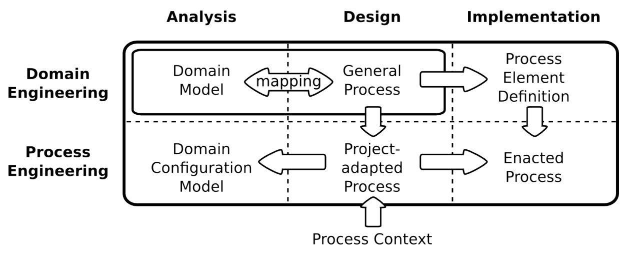


Figure 2: Elements of our approach as a process line.

an initial set of constraints that must hold for any process model specified along with its variability:

Rule 1: All process elements in the variability model must be defined in the general process's Method Library.

Rule 2: The variants of a process element must be the same type as the variation point.

Rule 3: Roles and work products which are only associated to optional tasks in the general model, must be marked as optional in the variability model.

For example, if the general process includes both the "Architectural Design" document and the "Approve Architectural Design" task, then both process elements must appear in the variability model and viceversa (Rule 1). If there are two alternative ways of specifying the "Architectural Design" document, e.g., "Informal Architectural Design" (box-and-line diagrams) and "Formal Architectural Design" (component diagrams), then both alternatives must also be defined as work products in the Method Library (Rule 2). If the "Approve Architectural Design" is an optional task and it is associated to the "Client Architect" role in the general model, and this role is not associated to any other process element, then "Client Architect" must be also marked as optional in the variability model (Rule 3).

4. TOOL SUPPORT

In this section, we give an overview of existing tool support for specifying the general process and its variability, and we present our tool proposal. Since we are modeling variability separate from the general process, we also give an overview and a recommendation of weaving tools, which are used to establish constraints between two models.

4.1 General Process Modeling

The Eclipse Process Framework Composer (EPFC) [6] is a tool that allows the specification of SPEM-like[2] software processes, supporting a broad variety of project types and development styles. In EPFC, the process is shown as a tree of process elements, where the connections between the elements are entered and shown using a form-based interface. Process engineers maintain a general library of process patterns, which can then be reused in different process definitions. Users of the tool can also "publish" a process, which generates a set of HTML pages that document the process, including graphical depictions of the process workflows. EPFC is an active project, it is freely available online.

[2]Internally, the EPFC uses its own metamodel, and not the SPEM 2.0 UML profile defined in the OMG standard.

4.2 Variability Modeling

In [24], we analyzed four different notations for specifying software process model variability: SPEM primitives, vSPEM [13], OVM [17], and Feature Models [10]. All four formalisms proved expressive enough to capture variability in process models, so the decision of which formalism to use rested on available tool support and usability, since process engineers at SMEs must be able to use the tool chain with little training.

We compared eight tools that supported these formalisms w.r.t. six criteria: supported file formats, underlying formalism, supported analyses, type of interface, availability and usability. The SPEM 2.0 variability modeling primitives are difficult to understand and use, and supporting tools like EPFC use basic form-based interfaces and provide limited analysis of the resulting models.

We also found that, while it was clear and compact to model process variability with vSPEM, its tool support is still immature [13]. The FaMa-OVM [21] tool provides automated analyses for orthogonal variability models, but uses non-standard input and output formats, which are not yet well documented. On the other hand, the VEdit [15] tool allows the graphical definition of OVMs, but does not offer any of the model analyses available in FaMa-OVM. Thus, for the time being, we have decided not to adopt either notation for modeling variability.

Feature models, on the other hand, have rich tool support; however, since feature models are a general purpose formalism, they can be used to express certain configurations that are meaningless for software processes. For example, we can express that a variant work product may be realized by two different variant roles, because the process model elements are indistinguishable as all features are of the same type. This situation makes it error-prone to use general purpose notations for modeling variability in software processes.

The Clafer (**cla**ss, **fe**ature, **r**eference) [3] language can be used to specify feature models, but lacks a native analysis component; currently, models must be translated into Alloy for analysis. The Feature Modeling Plug-in (fmp) [12], Hydra [23] and XFeatures [19] are all feature modeling tools available as Eclipse plug-ins, while SPLOT (Software Product Lines Online Tools) [11] is a web-based application. These tools rely on various external reasoning engines to analyze feature models; however, the fmp tool is no longer supported, and the interface for specifying cross-tree constraints in the Hydra and XFeatures tools can be difficult to use. On the other hand, the SPLOT tool has an easy-to-use interface, includes various built-in interactive analyses, and models can be exported as XML files. For these reasons, we have decided to use SPLOT.

4.3 Weaving

If we use a separate model to specify process variability as is the case of feature models, we need some way of checking whether the modeled variability is consistent with the general process model. One way of doing this is to use a general purpose weaving tool: we have two models, the general process model and the variability model, which when weaved together, must produce a consistent model.

The C-SAW [7] tool is a general-purpose model transformation engine, which takes as input models specified in ECL (a variant of OCL). Weaving is done in an automated manner: programmers write aspects that encapsulate cross-

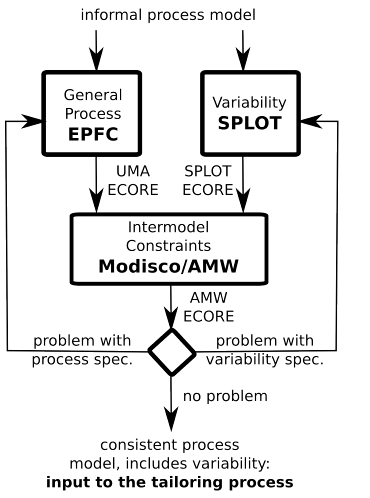

informal process model
General Process EPFC
Variability SPLOT
UMA ECORE
SPLOT ECORE
Intermodel Constraints Modisco/AMW
AMW ECORE
problem with process spec.
problem with variability spec.
no problem
consistent process model, includes variability: input to the tailoring process

Figure 3: Tool chain.

cutting constraints, and these are automatically applied at the appropriate point-cuts of the base program. The C-SAW tool does not maintain a weaving model, instead, it applies transversal changes consistently and efficiently in large models.

The XWeave [8] toolkit is a set of Eclipse-based tools for model weaving. This toolkit takes as input models that comply with the EMF Ecore [5] metamodel. XWeave can be used to create weaving models, where point-cut expressions and element names are used to define weaving rules. This toolkit is still in the prototype stage, and its authors are currently extending its variability support, including integration with pure variants [18], as well as support for symmetric weaving models.

The Atlas Model Weaver (AMW) [4] is a tool for establishing relationships between two or more input models. These relationships (or "links") are stored in a weaving model. Links can be established in a manual or semi-automated fashion, since the AMW tool includes an interactive interface for building the weaving model. The weaving model can then be used to generate model transformations that combine the input models. This tool is compatible with the Eclipse Modeling Tools framework, which makes it easy to integrate with tailoring transformations specified in ATL. For this reason, we choose to work with the AMW tool.

4.4 Our Tool Chain

Our tool chain is shown in Fig. 3. The general process is specified using the EPFC tool, while variability is modeled using using SPLOT. We then use the AMW tool to interactively establish consistency: this tool takes as input the general process model and the variability model, as well as a set of constraints according to the rules defined in Section 3.2. If both models are consistent, then AMW (optionally) produces a weaved model that takes into account the constraints; otherwise, the tool indicates which rules are not valid. The process engineer can define additional rules as needed. Since AMW has an interactive weaving model editor, the process engineer can check the effect of each rule as he/she adds it to the weaving model. The weaved model can then be visualized using the EPFC tool.

5. CASE STUDY: RHISCOM

We have been working in supporting Rhiscom, a medium-sized software company in Chile, in formalizing its software process [22]. This company develops software for the retail industry. It has steadily grown in the last five years, these days counting with about 70 employees, and also having affiliate offices in Peru, Bolivia and Ecuador. Rhiscom has defined a software process that has been applied and improved for a couple of years, and only recently it has been formalized using SPEM 2.0.

5.1 General Process

The Rhiscom general process model has been specified using the EPFC tool. In the corresponding figures, we have manually highlighted variation points, which we define in the next section. The EPFC tool saves process models in its own XMI format, which is conformant with the UMA metamodel. Since AMW takes as input ECORE-compliant models, we have defined an ECORE metamodel for the relevant parts of the UMA metamodel [3], which is automatically extracted from the files generated by the EPFC tool.

Figure 4 shows the general development process used at Rhiscom, which consists of five major activities, carried out in sequence. We have marked two activities as optional: "Requirements" and "Design". SPEM 2.0 does not have a stereotype to visually indicate optionality, so the notation in Fig. 4 is not standard.

Figure 5 shows the detailed workflow of the "Requirements" activity at Rhiscom. Each task is associated to various roles and work products, which are not shown in Fig. 5 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.

The class diagram in Fig. 6a shows the task definition of the "Establish Requirements Baseline without Test Cases" task, which is one of alternative ways of realizing the "Establish requirements baseline" task. 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".

The state machine for the "Final SRS" document previously mentioned can be seen in Fig. 6b. 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.

5.2 Variability

[3]Available at `http://www.adapte.cl/models/ecore/uma.ecore`

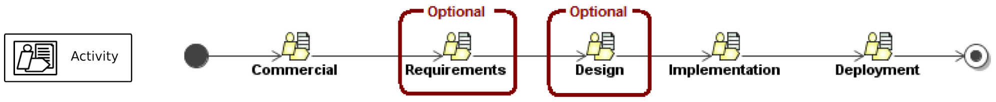


Figure 4: Rhiscom's general development process

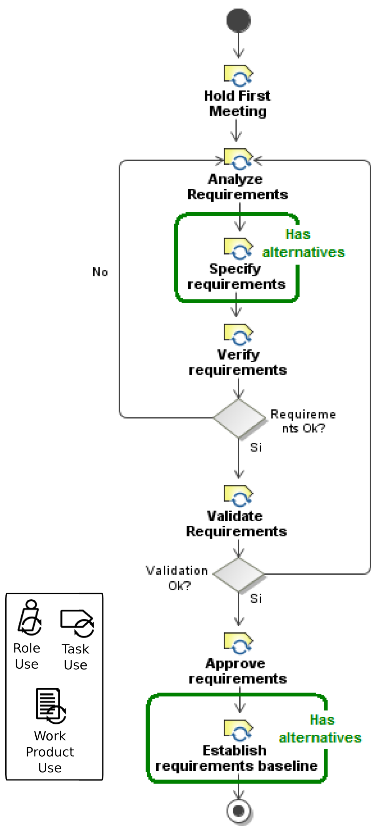


Figure 5: Detailed specification of Rhiscom's "Requirements" activity.

Currently, Rhiscom manually tailors the general process when starting new projects. Table 1 lists all the variability found in this process. Eight modeling elements (column one) were identified as variation points 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. We now explain a couple of examples.

Two activities in Fig. 4 are marked as optional – "Requirements" and "Design" – which are either kept or removed depending on the type of project. Since Rhiscom develops for a niche market, it does not always carry out the "Design" phase because new products are sometimes 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" activity.

Two tasks in Fig. 5 are marked as having alternatives: "Specify Requirements" and "Establish Requirements Baseline". The first task can be realized by one of two alternatives, "Specify Requirements in plain text" or "Specify Requirements in Use Cases"; the second one can also be realized by one of two alternatives, "Establish Requirements Baseline without Test Cases" or "Establish Requirements Baseline & Test Cases".

The feature model shown in Fig. 7 models the variability of this process. The root feature has three subfeatures: Activities, Work Products and Roles, used to group the different types of process elements (tasks appear under the Activities). For example, the "Requirements" activity has seven mandatory tasks, which are modeled as mandatory features (which are preceeded by the ● symbol in Fig. 7). The two tasks that can be realized in more than one way (labeled "Has alternatives" in Fig. 5) are decomposed into additional subfeatures using an alternative relationship (indicated by the ⩚ symbol in Fig. 7), where alternatives are preceeded by the ▫ symbol.

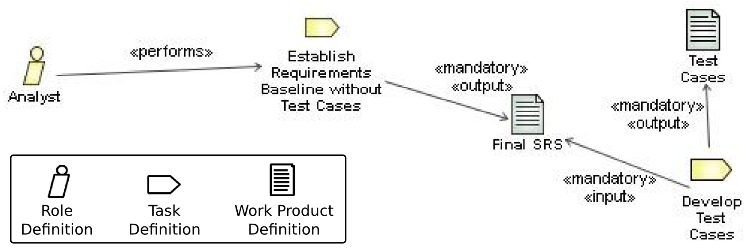
There are constraints between the variation points of the process. For example, the "Specify Requirements in plain text" task must be realized by the role "Analyst tester", while the "Specify Requirements in Use Cases" task must be realized by the "Analyst with Use Case skills". Addtionally, the "Final SRS" work product is generated by the "Establish Requirements Baseline without Test Cases" task, while the "Final SRS & test cases" work products are generated by the "Establish Requirements Baseline & Test Cases" task. These constraints are modeled as cross-tree constraints (shown in Fig. 7). Again, in order to later use the AMW tool, we created an ECORE metamodel of the SPLOT output format [4].

5.3 Weaving

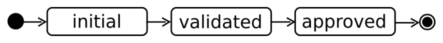
Figure 8 shows a screenshot of the AMW tool, where the feature model is the source model (far left) and the process model is the target model (far right). The center panel contains the weaving model created for these two models. In the Rhiscom example, this weaving model consists of five sets of constraints, one for each activity in Rhiscom's general development process (see Fig. 4). Selecting a constraint (or constraint set) in the weaving model highlights the relevant model elements in both the source and target models.

The following are some of the weaving constraints that Rhiscom's process engineer must add to their weaving model: 1) the variants "Specify Requirements in plain text" and "Specify Requirements in Use Cases" must be defined in Rhiscom's Method Library (Rule 1); 2) additionally, both of these elements must be defined as tasks if they are to replace the "Specify Requirements" task (Rule 2); 3) the "Analyst" role is only associated to optional tasks in the optional "Requirements" activity, so this role is also optional (Rule 3). For example, Fig. 8 shows the constraint set for the "Requirements" activity: the "One To One" constraints

[4]Available at `http://www.adapte.cl/models/ecore/splot_sfxm.ecore`



(a)



(b)

Figure 6: (a) SPEM 2.0 task example: "Establish Requirements Baseline without Test Cases", and (b) state machine associated to "Final SRS" work product.

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

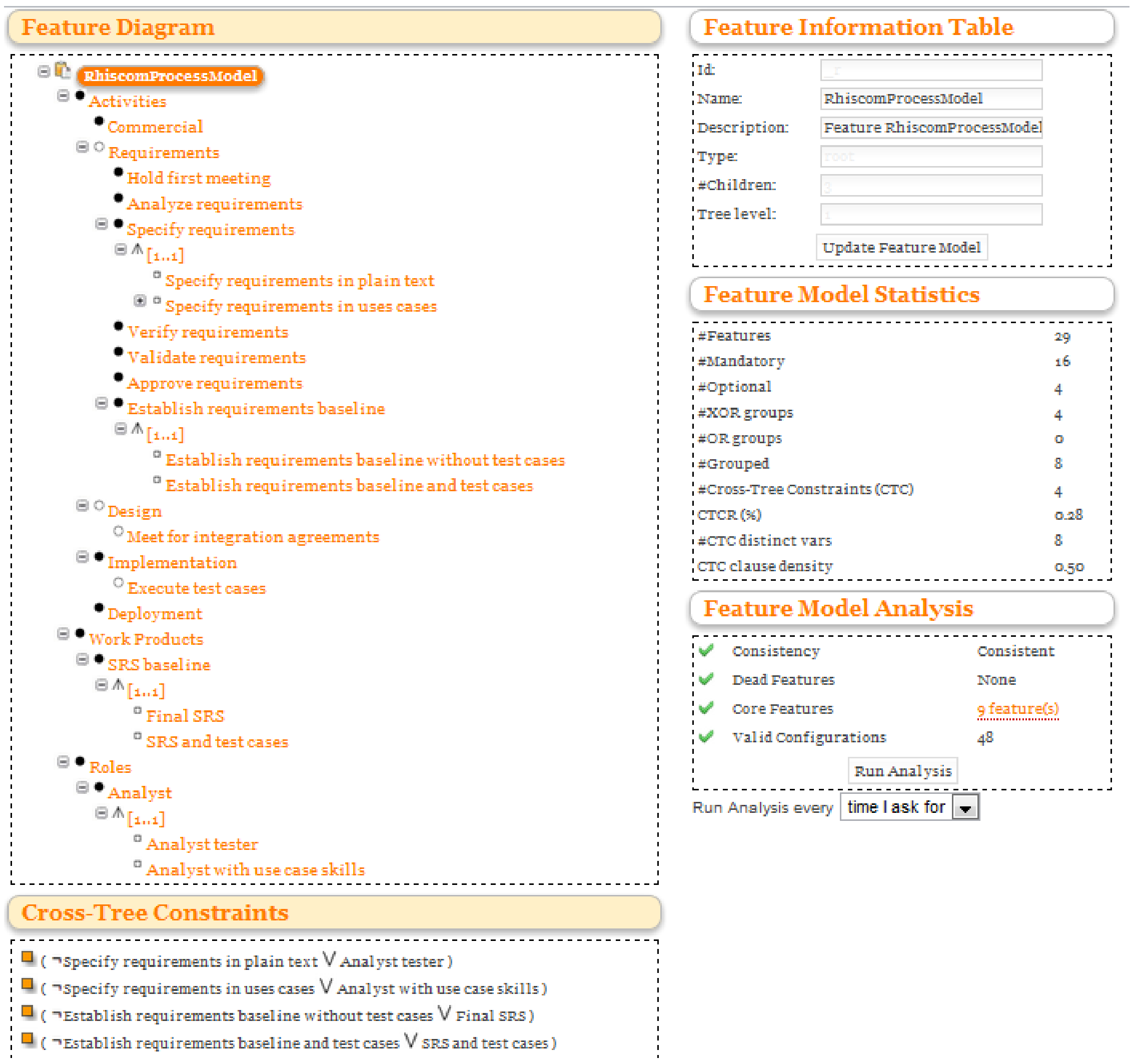
enforce Rule 1.

Since AMW has an interactive weaving model editor, the process engineer can check the effect of each rule as he/she adds it to the weaving model. In our example, the process engineer initially made a spelling mistake in the feature model, specifying a "Verity Requirements" task under the "Requirements" activity instead of "Verify Requirements". This resulted in a "One To One" constraint violation, since there is no "Verity Requirements" task in Rhiscom's Method Library. The AMW tool immediately notified the engineer about the violation, and he changed it to "Verify Requirements".

Another problem we detected using the AMW tool is that there were process elements of different types, but with the same name. For example, the "Design Architecture" task had two alternatives: "Informal Design Architecture" and "Formal Architectural Design", of differing complexity. The associated work product, "Architectural Design" had two alternatives: "Informal Architectural Design" (box and line diagrams) and "Formal Architectural Design" (component diagrams). The AMW tool warned the process engineer that Rule 2 was being violated (twice), as the name clash between the task and the work product definition meant that the "Design Architecture" task had a work product alternative, while the "Architectural Design" work product had a task alternative.

6. RELATED WORK

In this section, we give an overview of additional work on modeling software process variability. Currently, there are several proposed techniques and tools for addressing variability modeling in SPL; 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 [16] including its variability modeling primitives, vSPEM [13] and V-Modell XT [20] seem to be the only systematic efforts toward specifically addressing variability modeling in software processes. However, beyond the set of techniques they propose, 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; 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 [2][20][25] 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 could be also useful for variability modeling in software processes. The question that arises is now, what notations, tools and techniques available for variability modeling in SPL could be appropriate for modeling variability in software processes? This research work is an effort to propose a tool chain that allows modeling processes including their variability considering characteristics such as understandability, expressiveness, use of standards, tool availability, and the possibility of analyzing the specification for consistency.



- RhiscomProcessModel
  - Activities
    - Commercial
    - Requirements
      - Hold first meeting
      - Analyze requirements
      - Specify requirements
        - [1..1]
          - Specify requirements in plain text
          - Specify requirements in uses cases
      - Verify requirements
      - Validate requirements
      - Approve requirements
      - Establish requirements baseline
        - [1..1]
          - Establish requirements baseline without test cases
          - Establish requirements baseline and test cases
    - Design
      - Meet for integration agreements
    - Implementation
      - Execute test cases
    - Deployment
  - Work Products
    - SRS baseline
      - [1..1]
        - Final SRS
        - SRS and test cases
  - Roles
    - Analyst
      - [1..1]
        - Analyst tester
        - Analyst with use case skills

Id:	_r
Name:	RhiscomProcessModel
Description:	Feature RhiscomProcessModel
Type:	root
#Children:	3
Tree level:	1

#Features	29
#Mandatory	16
#Optional	4
#XOR groups	4
#OR groups	0
#Grouped	8
#Cross-Tree Constraints (CTC)	4
CTCR (%)	0.28
#CTC distinct vars	8
CTC clause density	0.50

✓	Consistency	Consistent
✓	Dead Features	None
✓	Core Features	9 feature(s)
✓	Valid Configurations	48

Run Analysis every time I ask for

- (¬Specify requirements in plain text ∨ Analyst tester)
- (¬Specify requirements in uses cases ∨ Analyst with use case skills)
- (¬Establish requirements baseline without test cases ∨ Final SRS)
- (¬Establish requirements baseline and test cases ∨ SRS and test cases)

Figure 7: Screenshot of the SPLOT tool.

7. SUMMARY AND FUTURE WORK

In this paper, we have described a methodological and technological approach for modeling software process models, along with their potential variability. In this approach, the process engineer models the general process separately from its variability, and we ensure that only reasonable variability is specified by analyzing the mapping between these two models. Process models are specified in SPEM 2.0, a standard process specification language, while process variability is specified using feature models, which are expressive enough to model the process variability that we encountered in practice. The mapping between the two models is managed using a weaving model.

Our tool chain consists of three tools: the general process model is specified using the EPFC tool, variability is expressed using the SPLOT toolkit, and finally, the weaving model is created using the Modisco/AMW tool. The resulting process model, which includes variability, can now be tailored to a specific project context (the next phase of the ADAPTE project). These tools are all open-source, and we use standard file formats to exchange data in our framework, thus meeting our first two quality requirements. However, our tool chain stills requires some duplication of work in writting the process and the feature models, which reduces its overall usability. We aim to improve this aspect in future work.

Our initial experience with an industrial case study shows that our tool chain can be used to successfully specify a software process model and its variability. The separation of concerns we achieved by modeling variability separate from the general process dramatically improved the process engineer's understanding of the process specificacion. Moreover, the warnings produced by the weaving tool were useful for diagnosing limitations of the original process model.

Future Work. We would like to streamline the creation of the variability model. The SPLOT tool is easy to use, but for large processes without much variability, it is time-consuming to make the process engineer create the feature model from scratch. Another advantage of generating this feature model is that we can ensure some of the weaving constraints by construction. We believe that we can use the results from [1] to generate an initial feature model from the general process model.

The rules for defining weaving constraints (described in Section 3.2) are actually at the metamodel level, since they talk about relationships between types of process elements. We believe that, in future, these weaving constraints can be automatically generated. In this case, the process engineer would only have to revise the warnings produced by the Modisco/AMW tool.

8. REFERENCES

[1] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On Extracting Feature Models From Product Descriptions. In *Proceedings of the Sixth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '12)*, pages 45–54, 2012.

[2] 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.

[3] 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, October 2010.

[4] Eclipse Foundation. Atlas Model Weaver website. `http://www.eclipse.org/gmt/amw`, Accessed February 2012.

[5] Eclipse Foundation. Eclipse Modeling Framework. `http://www.eclipse.org/emf`, Accessed February 2012.

[6] Eclipse Foundation. Eclipse Process Framework Project. `http://www.eclipse.org/epf/`, Accessed February 2012.

[7] Jeff Gray, Yuehua Lin, and Jing Zhang. C-SAW website. `http://www.cs.ua.edu/~gray/Research/C-SAW`, Accessed February 2012.

[8] Iris Groher and Markus Voelter. XWeave: Models and Aspects in Concert. In *Proceedings of the 10th International Workshop on Aspect-Oriented Modeling*, AOM '07, pages 35–40, New York, NY, USA, 2007. ACM.

[9] Julio Ariel Hurtado Alegria, M. Cecilia Bastarrica, Alcides Quispe, and Sergio F. Ochoa. An MDE Approach to Software Process Tailoring. In *International Conference on Software and Systems Processes, ICSSP 2011, Hawaii, USA, May 21-22, 2011. Proceedings*, pages 43–52. ACM, 2011.

[10] 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.

[11] Computer Systems Group / Generative Software Development Lab. SPLOT - Software Product Line Online Tools. `http://www.splot-research.org/`, Accessed February 2012.

[12] Generative Software Development Lab. Feature Modeling and Model Templates. `http://gsd.uwaterloo.ca/featureModelingAndModelTemplates`, Accessed February 2012.

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

[14] Tomás Martínez-Ruiz, Jürgen Münch, Félix García, and Mario Piattini. Requirements and constructors for tailoring software processes: a systematic literature review. *Software Quality Journal*, (1):229–260, March 2012.

[15] Software Systems Engineering Research Group/ University of Duisburg-Essen. VARMOD-PRIME Tool-Environment. `http://www.sse.uni-due.de/wms/de/?go=256`, Accessed February 2012.

[16] OMG. Software and Systems Process Engineering Metamodel specification (SPEM) Version 2.0. `http://www.omg.org/spec/SPEM/2.0`, Accessed February 2012.

[17] 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, Proceedings*, page 219, August 2006.

[18] pure-systems GmbH. Pure::variants Variant Management Tool website. `http://www.pure-systems.com/3.0.html`, Accessed February 2012.

[19] O. Rohlik and A. Pasetti. XFeature Modeling Tool. Automatic Control Laboratory, ETH Zürich, Accessed February 2012. `http://www.pnp-software.com/XFeature/Home.html`.

[20] 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.

[21] 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. *To appear in Software Quality Journal Special Issue on Quality Engineering for Software Product Lines*, August 2011.

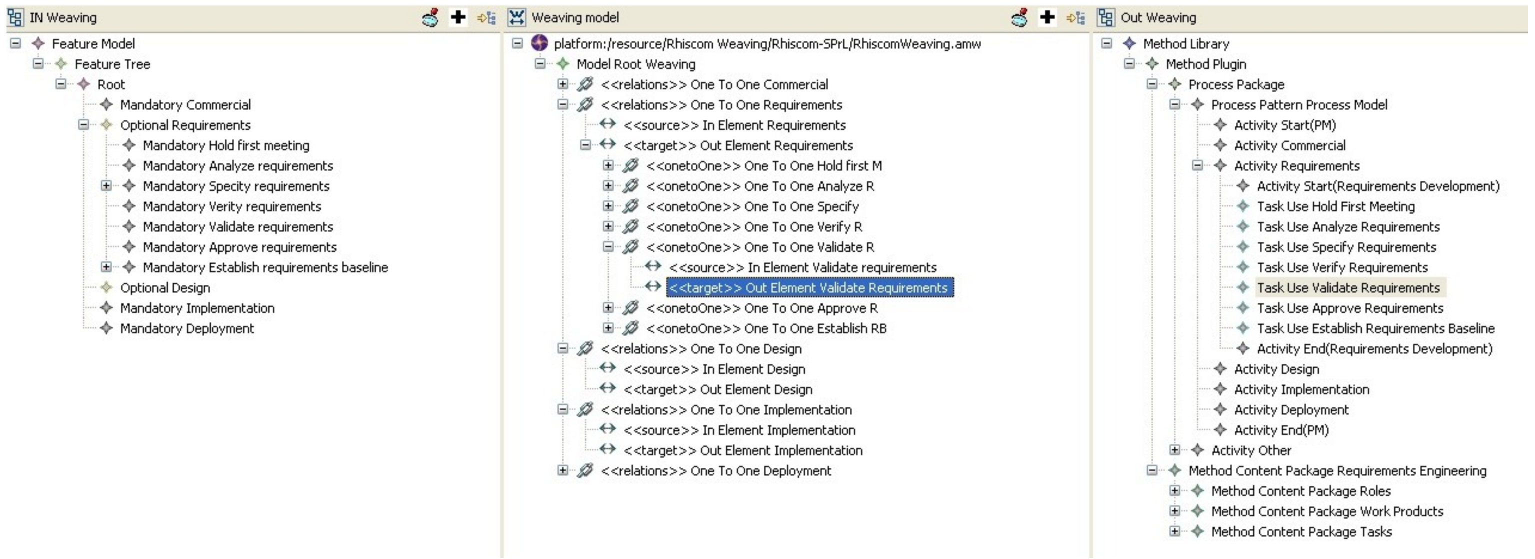
[22] Pablo Ruiz, Alcides Quispe, María Cecilia Bastarrica, and Julio Ariel Hurtado Alegría. Formalizing the Software Process in Small Companies. Technical Report TR/DCC-2012-2, Computer Science Department, Universidad de Chile, January 2012.

[23] José R. Salazar. Herramienta para el modelado y configuración de modelos de características, (in Spanish). `http://caosd.lcc.uma.es/spl/hydra/`, Accessed February 2012. Universidad de Málaga.

[24] Jocelyn Simmonds, María Cecilia Bastarrica, Luis Silvestre, and Alcides Quispe. Analyzing Methodologies and Tools for Specifying Variability in Software Processes. Technical Report TR/DCC-2011-12, Universidad de Chile, Departamento de Ciencias de la Computación, November 2011.

[25] 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.



```
IN Weaving
Feature Model
  Feature Tree
    Root
      Mandatory Commercial
      Optional Requirements
        Mandatory Hold first meeting
        Mandatory Analyze requirements
        Mandatory Specity requirements
        Mandatory Verity requirements
        Mandatory Validate requirements
        Mandatory Approve requirements
        Mandatory Establish requirements baseline
      Optional Design
      Mandatory Implementation
      Mandatory Deployment
```

```
Weaving model
platform:/resource/Rhiscom Weaving/Rhiscom-SPrL/RhiscomWeaving.amw
  Model Root Weaving
    <<relations>> One To One Commercial
    <<relations>> One To One Requirements
      <<source>> In Element Requirements
      <<target>> Out Element Requirements
        <<onetoOne>> One To One Hold first M
        <<onetoOne>> One To One Analyze R
        <<onetoOne>> One To One Specify
        <<onetoOne>> One To One Verify R
        <<onetoOne>> One To One Validate R
          <<source>> In Element Validate requirements
          <<target>> Out Element Validate Requirements
        <<onetoOne>> One To One Approve R
        <<onetoOne>> One To One Establish RB
    <<relations>> One To One Design
      <<source>> In Element Design
      <<target>> Out Element Design
    <<relations>> One To One Implementation
      <<source>> In Element Implementation
      <<target>> Out Element Implementation
    <<relations>> One To One Deployment
```

```
Out Weaving
Method Library
  Method Plugin
    Process Package
      Process Pattern Process Model
        Activity Start(PM)
        Activity Commercial
        Activity Requirements
          Activity Start(Requirements Development)
          Task Use Hold First Meeting
          Task Use Analyze Requirements
          Task Use Specify Requirements
          Task Use Verify Requirements
          Task Use Validate Requirements
          Task Use Approve Requirements
          Task Use Establish Requirements Baseline
          Activity End(Requirements Development)
        Activity Design
        Activity Implementation
        Activity Deployment
        Activity End(PM)
      Activity Other
    Method Content Package Requirements Engineering
      Method Content Package Roles
      Method Content Package Work Products
      Method Content Package Tasks
```

Figure 8: Screenshot of the AMW tool.