Software Process Line Modeling and Evolution

Jocelyn Simmonds, Daniel Perovich, Cecilia Bastarrica and Luis Silvestre
Computer Science Department
University of Chile
{jsimmond, dperovic, cecilia, lsilvest}@dcc.uchile.cl

***Abstract*—Companies formalize their software processes as a way of organizing their development projects. These companies usually work with families of processes, since differences in project requirements and objectives means that a "one-size-fits-all" approach does not work well in practice. This family can be a collection of predefined processes, but can also be a Software Process Line (SPrL), where a general process is automatically tailored to a project's context. The latter approach generates the process that best suits the project's requirements, but requires formalization and tool support to be successful. Model-Driven Engineering provides a formal framework for defining the models and transformations required for automated process tailoring. In this approach, various types of models must be specified and evolved, limiting industrial adoption. To address this problem, we propose a megamodel for SPrL definition and evolution, that provides the required formality while hiding the complexity of the approach. A megamodel is a model in itself formed by references to models. By defining a megamodel, we can provide uniform support for process definition including variability, tailoring and evolution, which we formalize in this article. We report the application of our approach to the software development process of Mobius, a Small Software Enterprise based in Chile.**

I. Introduction

Small Software Enterprises (SSE) define their processes in order to manage their development projects in a systematic way, being able to plan, assign resources and control the progress of the project. Having a defined process allows companies to analyze projects' results in terms of quality and productivity as a basis for improvement. Moreover, if the process is rigorously defined, the company can get an ISO certification or a CMMI evaluation that may give them a commercial advantage. Formally defining the software process allows for using supporting tools for formal process analysis and management, as well as making evolution easier. Software processes can evolve in different ways such as adding or removing activities, changing a work product template, or assigning a new role to a particular task, among others.

Defining, documenting and following a software process is an expensive endeavor. However, software companies get involved in different kinds of projects: a large software development from scratch, software development with an experienced or a novice team, bug fixing, software projects using well known or highly innovative technologies. Therefore, the same process is not appropriate for addressing all kinds of projects being equally productive and effective in all cases. An obvious conclusion is that it is necessary to count on a series of processes each one for each kind of project, i.e., a process family. Evolving all these processes independently implies a big effort, as well as a risk of introducing inconsistencies among them.

If defining a software process is expensive, defining a whole process family is almost always unaffordable, especially for SSEs, which is the case for most of the software companies in Chile. A software process line (SPrL) is a software product line where the products are software processes. In this way a SPrL defines its reusable process assets and a mechanism for obtaining particular processes within the SPrL scope; this mechanism is called process tailoring. There are different approaches to process tailoring such as counting on a set of predefined processes, building a software process by configuring a series of process elements, or generating particular processes by customizing a general predefined process that includes its potential variability. All these approaches may be supported by tools if the process is formalized, but each of them has some challenges. Configuring processes is a bottom up approach and thus compatibility among process elements is not always easy to achieve. Generating customized processes requires systematic tailoring rules in order to achieve repeatability. We have found that model-driven engineering (MDE) techniques are a feasible solution for dealing with consistently generating processes by defining processes as models and implementing tailoring rules as model transformations that resolve variable process elements. In this way we are sure to obtain the most appropriate process for each project just defining the project's characteristics -its context- and executing the tailoring transformation.

An MDE-based tailoring approach obtains the optimal process but it requires defining the software process model that conforms to a particular metamodel, identifying the process potential variability, the project context model and its corresponding metamodel, as well as programming the tailoring transformation itself. All these activities are very sophisticated and require highly specialized professionals, limiting industrial adoption. Moreover, it is not enough to count on a specialized professional for defining these modeling artifacts since all them evolve: variable process elements, context attributes and potential values, or rules implemented in the tailoring transformation. We propose to deal with this complexity using a megamodel, i.e., a model whose elements represent models and therefore capturing all automatic tailoring elements in a uniform way that ensures integrity by construction. This solution also helps ensuring the consistency of the resulting elements after applying all the evolution actions we have found that exist in industrial practice: general process, context and

tailoring rules evolution.

Megamodeling is a new theoretical proposal that has not been widely applied in real world applications yet. In this article we show how a megamodel can support process modeling, tailoring and evolution by applying it to the process of Mobius, a small Chilean company that develops software and hardware integrated systems for the public transportation.

The rest of the article is structured as follows. In Sec. II we introduce Mobius and its general process model, the elements of its SPrL and discuss evolution in this context. Section III describes background concepts involved in our proposed solution such as process modeling, software process lines, decision models and megamodeling. The proposed megamodel is described in detail in Sec. IV, including how process, process variability, and project context are modeled. Section V summarizes how the megamodel is used for product derivation, using the elements modeled in the previous section, and Sec. VI describes how this megamodel can be used for evolving the SPrL. Finally, Sec. VII describes related work in this area, and Sec. VIII states conclusions and current and expected work based on our megamodel proposal.

II. Motivating Example

Mobius is a three year old software services company based in Santiago, Chile, that develops integrated software and hardware solutions for Santiago's public transportation system. Mobius has 20 employees; 8 are directly working in software maintenance and development. Employees perform more than one role in the company, according to the traditional software engineering disciplines (e.g., developer, analyst, tester, etc). Project typically take from a couple of days for incidents to three or four months for large development projects.

This company started formalizing its development process two years ago, as part of the ADAPTE project[1]. ADAPTE proposes an MDE-based approach for software process tailoring, creating a Software Process Line (SPrL) [25]. In order to create a SPrL, Mobius specified a general process model that represents its organizational development process (including variation points), a context model that can be used to characterize new projects, and a variation decision model that documents the company's tailoring know-how. This last model documents how variability is resolved based on context values. These models are used to automatically create a tailoring transformation, which outputs an adapted process that is tailored to the input project context.

Figure 1 shows the main phases of Mobius' general software development process, which is loosely based on the Rational Unified Process (RUP). It is quite detailed in its definition, with 104 tasks, 10 roles and 44 work products, grouped into 4 phases (see Figure 2). A significant amount of effort went into defining the SPrL: ten 3 to 4 hour sessions were required to define the organizational process model; and the organizational context model and variation decision model were defined over a period of five days. This setup cost has been amortized over

[1]http://www.adapte.cl

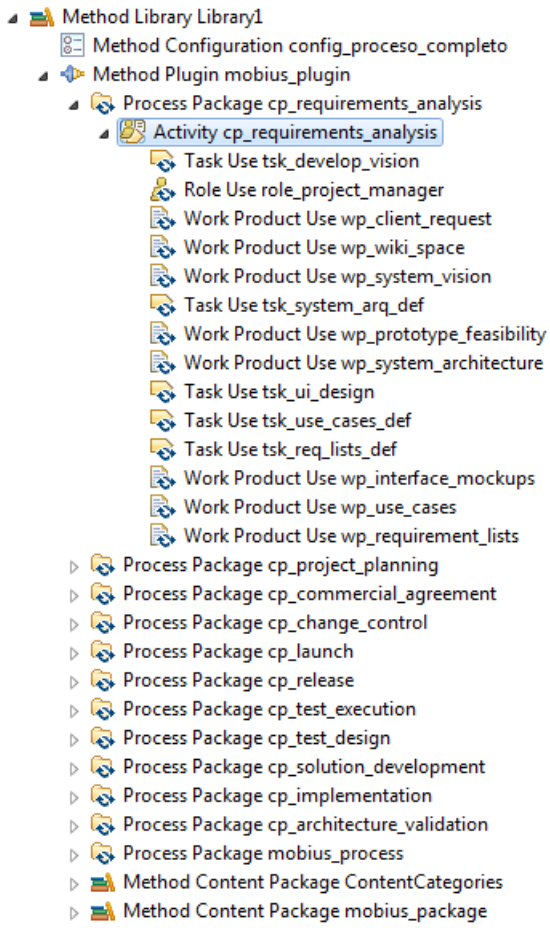

Fig. 2. Mobius' general software development process.

several projects, as it now only takes a few seconds to run our tool chain to produce an adapted software process for a new project.

Moreover, the tailored software process only includes the tasks and work products that are strictly necessary for the project, and the process configuration step required by RUP is now carried out in a systematic and replicable manner. However, as there are multiple dependencies between the different models and transformations, and managing these artifacts as part of a Software Process Improvement (SPI) initiative is non-trivial. Given the initial cost of formalizing all the models and transformations, this may hinder the industrial adoption of our approach for other companies.

What is even worse, if the SPrL definition is not adequately maintained and evolved, it will erode, making it less useful in time. This evolution task may be even harder than defining the SPrL since relationships and integrity must be correctly preserved. For example, if a new task is added to the organizational process model and it is defined as a variation point, then the process engineer must remember to update the variation decision model, specifying how this variation point is resolved during tailoring taking into account the project context po-

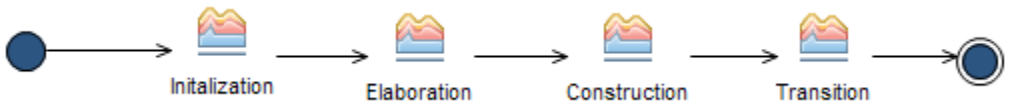

Fig. 1. Main phases of Mobius' development process.

tential values. This rule must be consistent with the existing rules, otherwise the tailoring transformation will generate incorrect adapted software processes. Removing elements from a model also requires careful change propagation. If a context attribute is removed from the organizational context model, then references to this attribute must be removed from the variation decision model, and the tailoring transformation must be regenerated. The variation decision model may be changed directly, especially when the adapted software process is not as expected. These are just some examples of possible evolution; we will discuss in the rest of this article how these and other evolution actions can be formalized using a megamodel.

III. Background

In this section, we give an overview of software process modeling, software process lines, decision models and megamodeling.

A. Software Process Modeling

A software process is a structured set of activities required to develop a software system [31], with related artifacts, human and computerized resources, organizational structures and constraints. A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. A process model provides definitions of the process to be used, instantiated, enacted, or executed. Therefore, a process model can be analyzed, validated, simulated or executed if it is appropriately defined according to these goals.

SPEM 2.0 (Software and Systems Process Engineering Metamodel) [35] is a standard proposed by the Object Management Group (OMG). It is based on the Meta Object Facility and it is the most popular language used to specify software processes [28]. In order to encourage process maintenance and reuse, SPEM 2.0 makes a difference between the definition of process building blocks and their latter use in (possibly more than one) processes.

Process elements, like tasks, roles and work products, are defined and stored in a Method Library. These definitions can later be reused in multiple process definitions. In SPEM, 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. Figure 3 shows an activity diagram representing the Architecture Validation activity, which is part of the Elaboration phase (produced using the EPF Composer tool[2]) of Mobius' process. This tool shows role and work

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

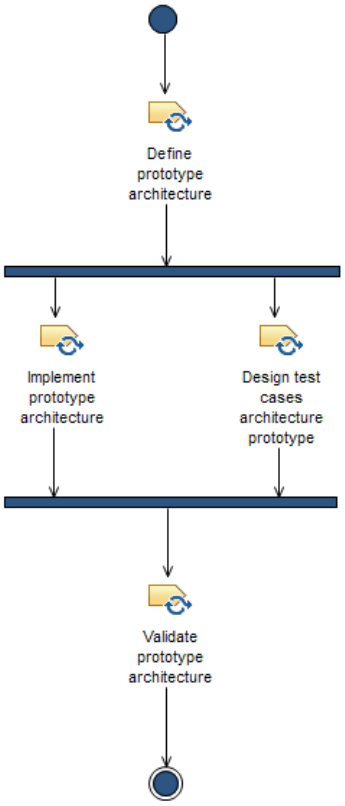

Fig. 3. Mobius' Architecture Validation activity.

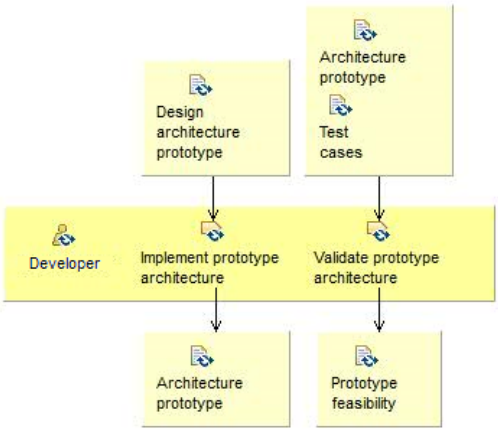

Fig. 4. Partial view of task and work products associated to the Developer as part of the Architecture Validation activity.

products separately. Figure 4 shows that the Developer role is in charge of the Implement prototype architecture task, which takes as input the Design architecture prototype work product and outputs the Architecture prototype.

Some benefits of software process modeling are [12]: ease of understanding and communication, process management support and control, provision for automated orientation and process performance, provision for automated execution support and process improvement support. A well-defined software process model is a determinant factor for achieving quality products and productive projects [22]. However, defining a software process model demands an enormous effort for making explicit common practices and defining practices that may not yet exist within the company. Standards such as the ISO/IEC15504 and maturity models such as the Capability Maturity Model Integration (CMMI) are commonly used as guidelines for defining processes.

B. Software Process Lines (SPrL)

Software Process Lines (SPrL) are software product lines (SPL) in the software development process domain [37]. Table I shows the different SPL stages [40], [16] and the corresponding SPrL phases. The process of instantiating a SPrL to a particular project context is called process tailoring [4]. As a typical SPL instantiation, software tailoring is the activity in which a software process's variation points are resolved so that it is adapted to the particular characteristics required for a project.

Software process tailoring requires knowledge about the company's process as well as possible project contexts. This activity must be carried out in a structured manner [11] in order to ensure that the resulting process meets the needs of the input project. Moreover, the resulting processes may not be consistent if tailoring is done manually– two similar projects may end up with different processes – meaning that the performance data for these projects cannot be directly compared [24].

There are several strategies for process tailoring, including template-based tailoring [10], framework-based tailoring [8], constructive tailoring [42] and automated tailoring [25]. We give an overview of automated tailoring based on Model Driven Engineering (MDE) in the rest of this subsection. In this approach, processes and possible project contexts are formalized as models, and tailoring is formalized as a model transformation, specified using the Atlas Transformation Language (ATL)[3]. The process engineer must also define the variation points in the organizational process model, as well as the relationships between context attributes and variation points.

Figure 5 shows Mobius' Requirements activity, which includes two types of variation points: role, task and work product *optionality* and *alternatives*. We have included the variation points in the activity diagrams using annotations as a visual aid for the process engineer. Each optional process element is linked to an annotation that specifies a predicate over project context attributes, the element is included in the adapted process if the predicate holds for the given input context. For example, the Develop vision task is only included if Project type is Non Corrective.

[3]http://www.eclipse.org/atl

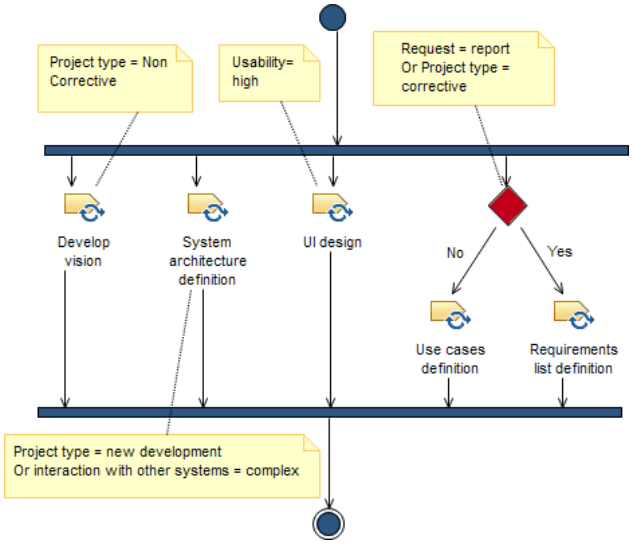

Fig. 5. Mobius' Requirements Activity.

Red decision nodes with annotations denote alternatives (regular decision nodes are blue). For example, if Request = Report or Project type = Corrective, then the requirements are defined using a list (Requirements list definition, otherwise they are defined using use case diagrams (Use cases definition). All variation points are specified using the SPEM 2.0 variability primitives: tasks, roles and work products represent optional process elements by setting to true the value of a boolean attribute *isOptional*, and alternatives are specified by linking variation points to possible realizations using the *replaces* SPEM 2.0 variability primitive.

There is an annotation for each variation point, from which we extracted the organizational context model, as a set of context attributes and their possible values. Table II lists all the context attributes that appear in Mobius' process model. All these are set-up activities executed by the process engineer. Before starting a new project, the project manager must carefully characterize the project in terms of the context attribute values. This project context model, as well as the organizational process model, are input of the tailoring transformation, generating a process tailored to the project.

C. Decision Models

A decision model is a set of rules of the form *condition* $\Rightarrow$ *conclusion*, where a *condition* is a predicate about the application domain, and *conclusion* indicates how a variation point is resolved when the *condition* is true. Each solution model of a decision model represents a product in the product line. Several notations for representing decision models have been proposed in the literature (cf. [54], [44], [13]) In this work we use add-or trees [32].

TABLE I
SOFTWARE PROCESS LINES AS PRODUCT LINES.

Domain Modeling	**Architecture Modeling**	**Process Line Implementation**	**Process Line Testing**
• Organizational Process (including variability) • Organizational Context Definition	• Organizational Process Formalization • Organizational Context Modeling	• Method Library Definition • Organizational Process in Use	• Organizational Process in Use Testing
Product Requirements	**Product Architecture Definition**	**Product Instantiation**	**Product Testing**
• Project Context Modeling	• Process Tailoring or Process Selection	• Process Configuration	• Process Configuration Testing

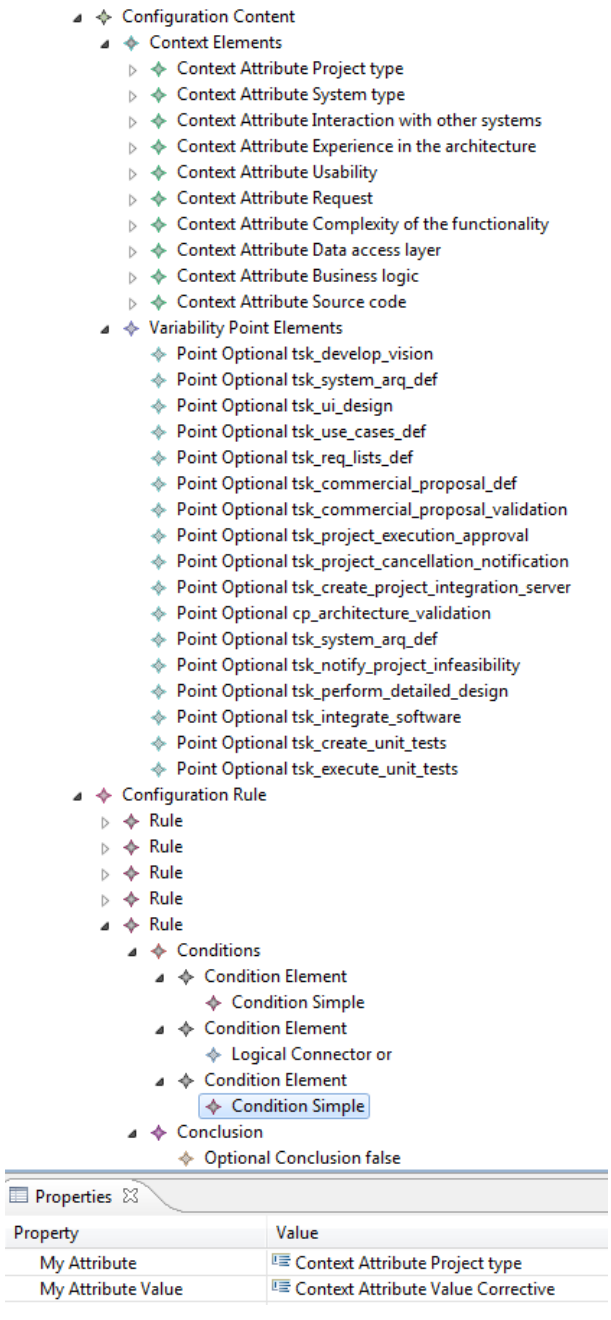

Fig. 6. Mobius' Variation Decision Model.

TABLE II
MOBIUS' ORGANIZATIONAL CONTEXT MODEL.

Attributes	**Values**
Project type	New development Corrective Non-corrective
System type	Guarantee No guarantee
Interaction with other systems	Simple Complex
Team experience	Yes No
Usability	High Low
Request	Report No report
Complexity of the functionality	High Average Low
Data access layer	Yes No
Business logic	Yes No
Source code	Yes No

Specifying the required tailoring transformation in ATL is not an easy task [25], even for an experienced process engineer. Provided that when specifying the process model including its variability, the process engineer's tailoring know-how has already been included in the organizational process model using annotations, we have formalized this information using a decision model [48]. We then use this decision model as input to a Higher-Order Transformation (HOT) [50] for automatically generating the process tailoring transformation [47]. Figure 6 shows the Variation Decision Model for Mobius, which includes 16 rules.

D. Megamodeling

Practical applications of MDE are increasingly intensive in modeling artifacts. They involve a large number of heterogeneous and interrelated models which change over time. *Megamodeling* is the model-based approach for coping with the complexity of managing and evolving such large model repositories [3]. It is centered on the notion of *megamodel* introduced in [7], which conveys the idea of modeling-in-the-large by establishing and using the global metadata and relationships on the modeling artifacts while ignoring their

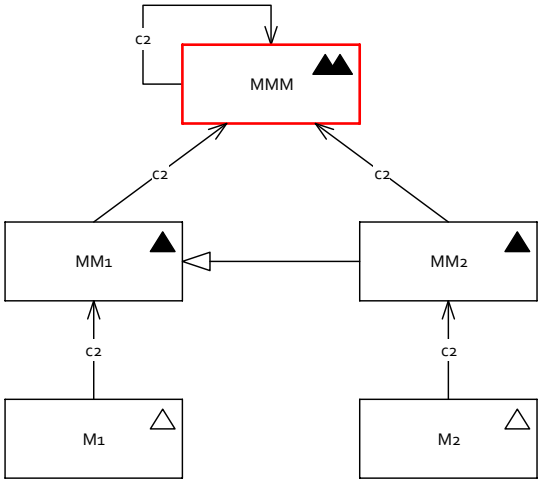

Fig. 7. Modeling Notation for Models

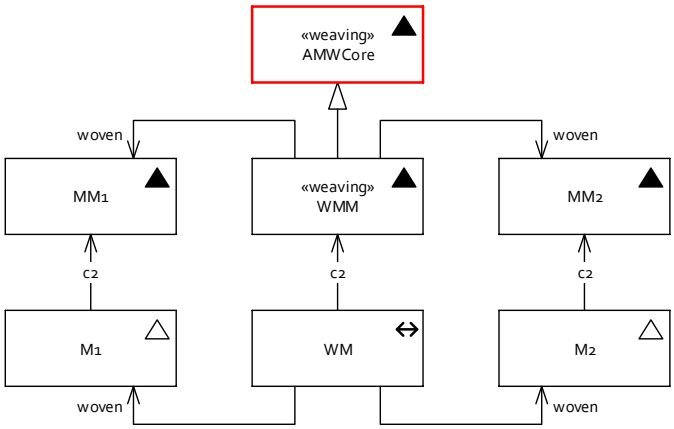

Fig. 8. Modeling Notation for Weavings

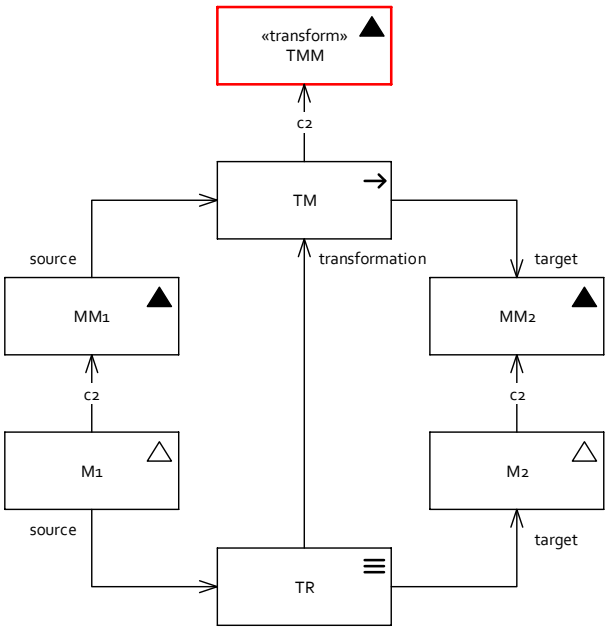

Fig. 9. Modeling Notation for Transformations

internal details. Global Model Management (GMM) [34] is a megamodeling approach that offers Eclipse-based tool support for managing modeling artifacts. It has a metamodel that characterizes the different kinds of modeling artifacts and their interrelations. The full metamodel can be found in Appendix A. In the rest of this section we give an overview of its key constructs and explain the notation used in this article.

A *terminal model* is a representation of certain aspects of a physical, abstract or hypothetical reality, called *system*. A terminal model *conforms to* (c2) a *metamodel*, i.e., it is expressed in the modeling language defined by that metamodel. In turn, a *metamodel* conforms to a self-conforming metametamodel that is available in the modeling environment. For example, Figure 7 illustrates a terminal model M1 that conforms to the metamodel MM1 that conforms to the self-conforming metametamodel MMM available in the environment. Thus, in GMM, models are organized according to the 3+1 metamodel hierarchy [6]. Besides, a metamodel can *extend* one or more metamodels to merge and refine their constructs. In the same figure, metamodel MM2 extends the base metamodel MM1, and then, the modeling constructs from MM1 and MM2 can be used to build the terminal model M2.

A *weaving model* is a special kind of terminal model that represents a semantic relationship between elements in different models. As illustrated in Figure 8, a weaving model WM has two *woven* terminal models M1 and M2. WM defines links between elements in those woven terminal models. The kinds of links and constructs available in WM are those defined by the metamodel WMM marked with the weaving stereotype. For this kind of metamodel we capture the woven metamodels to constrain the kind of terminal models that can be woven. GMM relies on the AMW toolset for weaving models, which defines a base and generic metamodel AMWCore which is extended to define concrete weaving metamodels. An *annotation model* is a special kind of weaving model in which only one model is woven. It is useful to append information to an existing model without changing its metamodel and favoring the separation of concerns.

A *transformation model* is a special kind of terminal model that implements the update or transformation of a set of source models into a set of target models. For example, as illustrated in Figure 9, a transformation model TM takes a source model that conforms to the metamodel MM1 and produces a target model that conforms to the metamodel MM2. A transformation model conforms to a metamodel which defines the model transformation language in which the transformation is implemented, exemplified as TMM annotated as transform in the same figure. A model transformation language can vary from a general-purpose imperative language such as Java to a purpose-specific language such as ATL. Currently, the toolset we developed uses both languages. A *transformation engine* executes transformation models. Each execution is captured by a *transformation record* which registers which transformation

model was executed, which specific modeling artifacts were used as sources, and which were used as targets. In Figure 9, the transformation record TR registers that the transformation model TM was executed using the terminal model M1 as source and generating the terminal model M2. The source and target models must conform to the corresponding metamodels expected by the transformation model.

Finally, a *megamodel* is a special kind of terminal model that captures the the semantic relationships among a set of modeling artifacts. In particular, Figures 7 to 9 illustrates fragments of a megamodel. Every megamodel conforms to the metamodel defined in Appendix A.

In this article, we further characterize the modeling artifacts according to their provenance or scope. For each modeling artifact we introduce, we use a colored outline in the figures to denote its scope:

- A red outline denotes that the modeling artifact is publicly available and that was developed by a research or industry community. For instance, in Figure 8 we colored AMWCore with red as this artifact is available in the Eclipse-based tool support for GMM.
- A green outline denotes that the modeling artifact was developed by our research team and that it is independent of the application scenario of our approaches and tools. We reuse these modeling artifacts at every SSE we work with.
- A blue outline denotes that the modeling artifact is specific to the organizational context of a SSE. However, these modeling artifacts are independent of any software development project that the SSE enacts.
- A black outline denotes that the modeling artifact is specific to a project of a particular SSE.

IV. SPrL Modeling

In this section we present the actual proposal of a megamodel for formalizing and evolving a SPrL. Particularly, potential strategies for modeling software processes in Sec. IV-A, process model variability in Sec. IV-B, variability modeling strategies in Sec. IV-C, and context models in Sec. IV-D. In all cases we state and justify our choice.

A. Process Modeling Strategies

We identify three strategies for modeling software processes: using an ad hoc modeling language, using a standard modeling language, and using a controlled modeling language. We have decided to follow the last one. In this section we discuss the three strategies and justify our choice.

1) Strategy M1: Use an ad hoc software process modeling language.: According to our experience with Chilean SSEs, this strategy is the most commonly applied. SSEs use an ad hoc and informal modeling language such as text documents or wikis to capture the best practices, guidance, and frequent workflow. By means of this strategy, SSEs at least count on an explicit representation of the shared knowledge on the process their teams follow, but the lack of formalism in the modeling language renders ill-formed documentation and reduces the chances of using purpose-specific tool support for maintenance, validation and evaluation of the process.

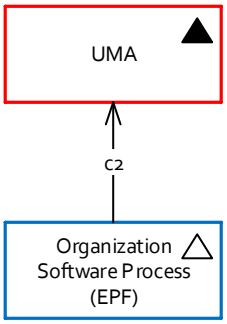

Fig. 10. Process Model conforms to the UMA metamodel

2) Strategy M2: Use a standard software process modeling language.: The advantages of using a standard process modeling language are that it is well-defined, it is possibly well-specified or documented (this is the case for SPEM 2.0 but not for UMA), and it is also possible to count with tool support.

In 2002, OMG proposed the Software Process Engineering Metamodel (SPEM) [36] specified as both, an independent metamodel based on the UML superstructure and a UML profile, both based on UML 1.4. SPEM 2.0 also includes some features of UML 2. The Unified Method Architecture (UMA) [19], a standardized method metamodel from IBM, addresses some of the weaknesses of SPEM and therefore it also influenced the SPEM 2.0 specification [35]. UMA is implemented as part of the freely available Eclipse Process Framework (EPF) [15] and the Rational Method Composer (RMC), a commercial tool from IBM. EPF and RMC provide the functionality for modeling concrete methods based on UMA. These methods can be exported in various output formats (e.g., HTML) and provide them to the development team. In this context therefore, method metamodels are a means for documentation [53]. Then, EPF Composer produces models conforming to UMA[4] and it does not support SPEM 2.0 yet. Most Chilean SSEs that have formalized their process use EPF Composer. Then, the process engineer produces (by means of the tool) a terminal model Organization Software Process conforming to the UMA metamodel. Figure 10 shows the modeling artifacts involved.

Neither UMA nor SPEM 2.0 provides direct support for the specification of the fine-grained behavior of the process. According to the SPEM 2.0 specification, it does not aim to be a generic process modeling language, nor does it even provide its own behavior modeling concepts. SPEM 2.0 rather defines the ability for implementers to choose the generic behavior modeling approach that best fits their needs. It provides specific additional structures to enhance such generic behavior models that are characteristic for describing development processes such as UML 2 activity and state machine diagrams, or alternatively BPDM/BPMN. This is also the case

[4]http://www.eclipse.org/epf/composer_architecture/

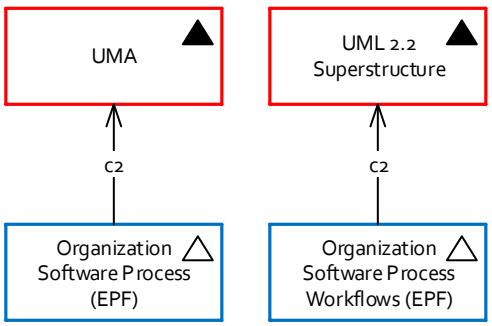

Fig. 11. Two Process Models Generated by EPF Composer

for UMA. Behavior can only be specified by means of work breakdown, and fine-grained behavior must be specified by means of a separated purpose-specific modeling language. In particular, EPF Composer uses UML activity diagrams to define workflows which are preserved in a separated model. In the current implementation of EPF Composer (version 1.5.1.6), diagrams are handled by the Eclipse UML2 plugin version 3.0.0 that, according to its documentation, is compliant with UML 2.2 Superstructure specification.

Then, EPF Composer can be used but it produces two separate models, as illustrated in Fig. 11. This is relevant only if we are interested in capturing the workflows of the software process, for instance, to achieve executability, otherwise UMA (or SPEM 2.0) is enough. However, from the perspective of tool developers, designing and building tool support that relies on a specific version of a specific modeling language has serious drawbacks. Particularly, the evolution of the modeling language has a direct impact of the tool implementation. From the perspective of tool developers, different customer organizations may rely on different modeling languages and hence, it requires a family of products covering those languages. In our experience with Chilean SSEs, while some of them use EPF Composer, others use Enterprise Architect. In these cases, UMA is the common modeling language but it poses the requirement to protect the investment on our tools from this kind of variability.

Strategy M2 is well suited for using process modeling for the sake of formal documentation, and also for scenarios where the process engineer requires to apply additional tools to manipulate the defined process (validate, evaluate, optimize, etc.). This benefit comes from the fact that the modeling language is well-defined and that tool support is available.

3) Strategy M3: Use a controlled software process modeling language.: Strategy M3 takes care of the variety of process specification languages used by practitioners. Instead of using a standard modeling language, we use a controlled modeling language, namely eSPEM. This modeling language is inspired on those available in UMA and SPEM 2.0, and its constructs are a subset of the constructs of those languages, mainly those that are relevant for our goals of tool support for the specification, enactment and improvement of SPrLs for SSEs. By having a controlled language, we control the evolution of the language and thus, the evolution of this language and the corresponding tool support can be consistently updated. The same approach was taken by the developers of EPF Composer and RMC, that instead of using the SPEM 1.1 specification, they devise and build a proprietary language.

However, in our case, our goal is not to define yet another modeling language, but rather to preserve the tool support in use by SSEs, but augmenting it with our tool set. As a consequence, we needed a mechanism for importing processes modeled in a standard language into processes modeled in our controlled language. To this end, we developed an Injector from UMA into eSPEM, and an analogous one can be implemented to import from SPEM 2.0. Figure 12 illustrates the modeling artifacts involved in Strategy M3 for the case of UMA. The current implementation of our toolset follows Strategy M3 and includes the Injector from UMA.

B. Modeling Variability in Software Processes

A single Organization Software Process model rarely satisfies the particular needs of every software development project executed in the organization. As a consequence, we also need to capture the variability in the software process and the mechanisms to resolve variable parts, in order to systematize the tailoring process. Using modeling techniques, we can fully automate this tailoring process. In this section we discuss how to model variability in order to allow such automation.

1) Units of variability: Provided that we use UMA as the modeling language for Software Processes, any kind of construct defined by UMA, any property of those constructs, and any association between them, can be considered as a unit of variability. In this case, any part of the method content and the process work breakdown can be a unit of variability. Moreover, in the case that the fine-grained behavior of the process is also captured, any of those constructs can also be used as units of variability.

In our case studies, and accordingly in the features available in our toolset, variability is localized on tasks. In the SSEs we worked with, they have defined two kinds of variations: optional tasks and alternative tasks. By declaring a task as optional the process engineer is stating that when determining the actual process for a particular project, it must be decided whether the task must be included or not. By declaring a task as having alternatives, the process engineer is stating that when determining the actual process to follow in a particular project, instead of the task, exactly one of its alternatives must be selected to be included in the actual process. These kinds of variation can be captured in terms of the constructs available in UMA. Tasks have a dual representation in UMA: the Task metaclass represents a reusable specification of a task, and the TaskDescriptor metaclass represents the concrete usage of a task in the process work breakdown. Task instances are defined in method libraries and TaskDescriptor instances are used in process work breakdown. Also, each TaskDescriptor can be linked to a specific Task in order to indicate that the task descriptor proceeds according to the specification of the linked Task. For TaskDescriptor instances with no linked

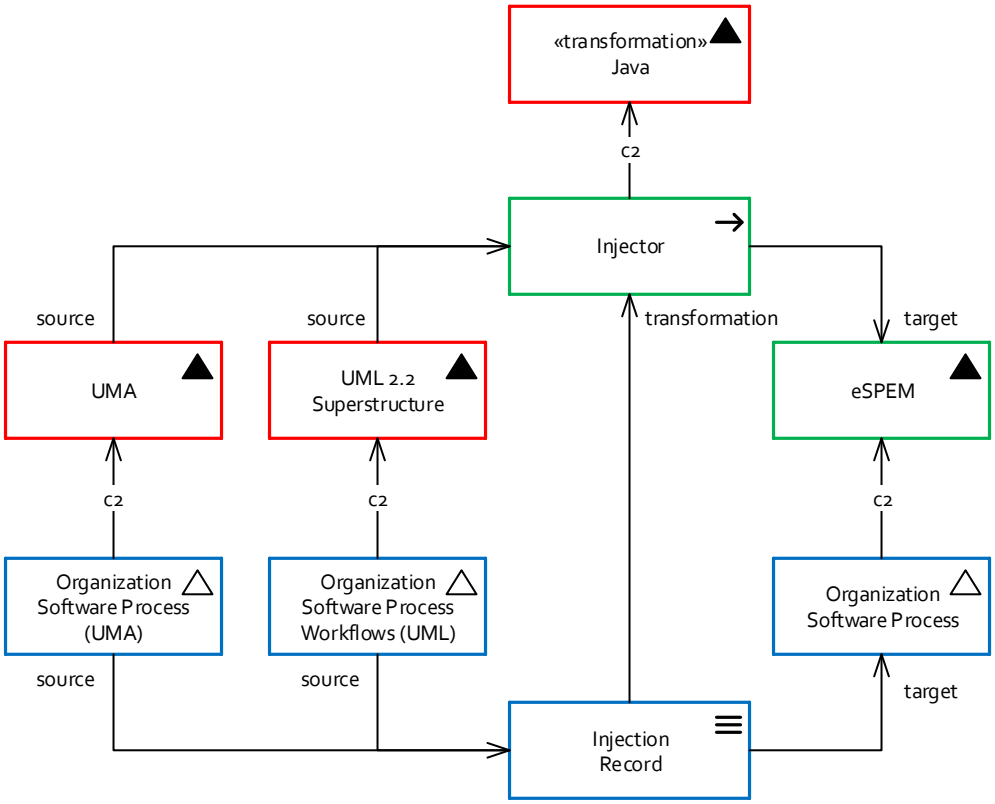

Fig. 12. Injector from a UMA Process Model into eSPEM

task, the TaskDescriptor instance itself must specify the task to perform.

We are not using the complete expressive power of UMA regarding variability. In our experience with SSEs in Chile, their processes are captured by means of optionality and alternatives, and they tend to avoid any extra complexity in their process specification. As a consequence, we just use the subset of constructs of UMA that allows us to characterize both optionality and alternatives.

Optionality can only be defined in TaskDescriptor. The BreakdownElement (super metaclass of TaskDescriptor) defines an IsOptional property to declare that the breakdown element may or may not be used when enacting the process. Optionality cannot be stated in a Task since it is a reusable declaration. Alternatives, however, cannot be stated in TaskDescriptor instances. It must be defined in the Task and it must be done by means of the variability constructs available in UMA. In particular, we use the replaces variability type to state that a task replaces another.

In Fig. 13 we use a UML 2.4 object diagram to present an organization software process in UMA. The process contains four tasks in the method library – t1, t2, t2a and t2b – and a delivery process with two sequential tasks – td1 and td2. In the method library, the process engineer declared that t2a and t2b are alternatives to t2. In the process work breakdown, the process engineer declared that td1 is optional and uses t1, and that is followed by td2 that uses t2. Then, this organization software process actually defines four potential concrete processes, depending on the selection on the variability: (I) t1, t2a, (II) t1, t2b, (III) t2a, (IV) t2b. By these means, we are using UMA constructs to specify the variable parts and we are also using UMA constructs to specify the variants: the actual TaskDescriptor element marked as optional and the alternative Task elements; in the figure, td1 in the first case, and t2a and t2b in the second case.

However, we are not using UMA constructs to establish the conditions on which the variants must be decided upon. The process marks td1 as optional, but it does not specify in which development scenario td1 must be performed. The same applies to the alternatives t2a and t2b. We capture this information in a separate model, as we explain in Sec. III-C. Besides, we automate the variation mechanism by which the concrete adapted process is produced from the organization process. We study this in Sec. V.

C. Process Variability Modeling Strategies

A condition is a rule for deciding which variant is used for a variable part. We devise three different strategies to represent process model variability conditions: using a predefined software modeling language, using a specialization of an existing language, and capturing conditions in a separate artifact.

1) Strategy V1: Use a predefined software process modeling language: This strategy uses the constructs that are already available in the process modeling language to capture conditions in order to preserve tool support.

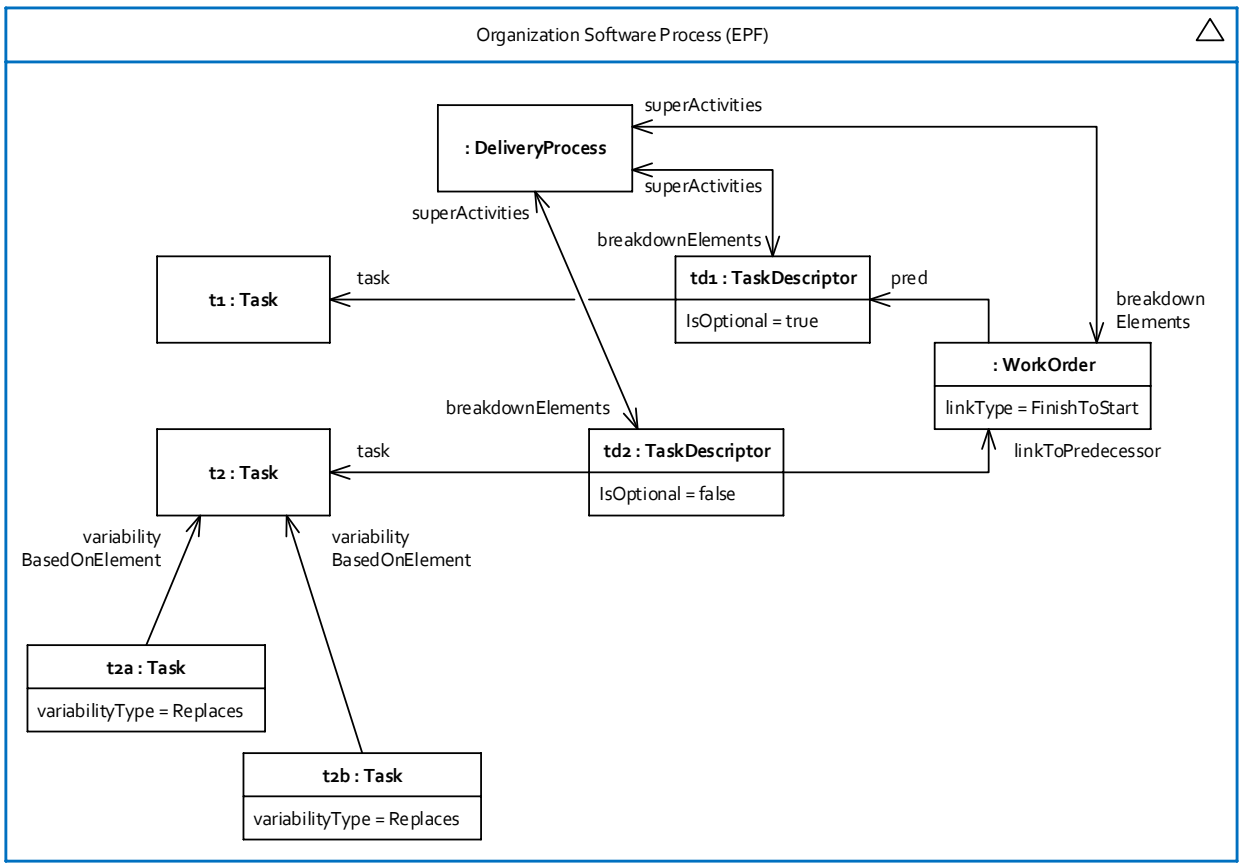

Fig. 13. Organization Software Process Specified in UMA

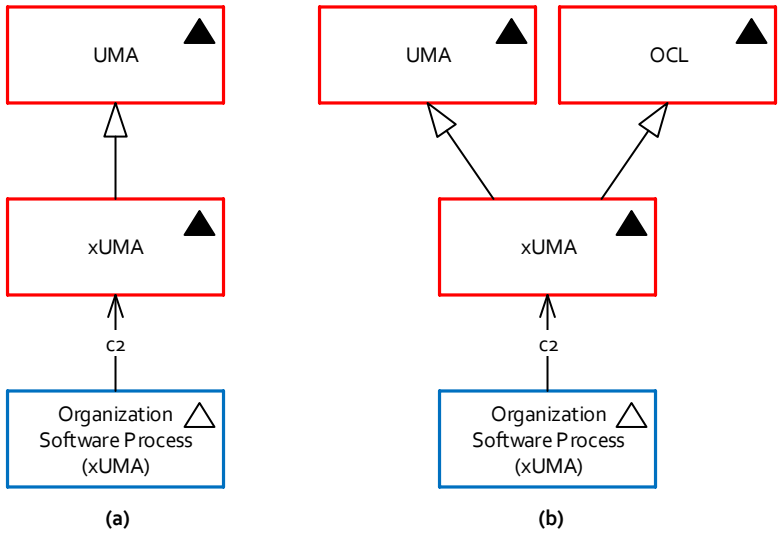

Fig. 14. UMA extensions to support variability specification

V1 has several advantages: (1) requires a single modeling language, (2) requires a single terminal model, and (3) there is tool support already developed and maintained by a large community. However, it also has several drawback: (1) there are no specific constructs for establishing variability conditions resolution, (2) there are no constructs for characterizing the development scenario, and (3) it limits the process specification to a particular modeling language.

2) Strategy V2: Use a specialization of a predefined software process modeling language: The goal of Strategy V2 is to define an extension of the process modeling language introduced by purpose-specific constructs. Thus, by this strat-

egy we define xUMA, an extended metamodel of UMA, that includes constructs for conditions, and that refines the Task and TaskDescriptor metaclasses in UMA in order to attach conditions to their instances. In order to model conditions, we can follow two strategies. We can define ad hoc constructs directly in xUMA. This strategy provides the flexibility of choosing the constructs that best fit our needs but it is also hard to define and maintain. We can use constructs that are defined in a purpose-specific language, i.e., using an expression language for models such as OCL. In this case, the ad hoc constructs in xUMA are only those needed to attach conditions to process elements. In particular, we need to define a sub-metaclass of Task and TaskDescriptor that appends a condition attribute to their respective base metaclass. The advantage of this strategy is that the language and the tool support is defined and maintained by a larger community. The main drawback is that it may be harder to learn. Figure 14 shows both strategies. In (a), we define xUMA as an extension of UMA with ad hoc constructs, and in (b) we define xUMA as an extension of both UMA and OCL.

Conditions must also refer to properties that are external to the process itself, and that characterize the scenario in which the process is enacted. So we need purpose-specific constructs to characterize the context. Again, we have two strategies: either to include those constructs directly in xUMA, or to define them in a separate metamodel and make xUMA extend it. Figure 15 illustrates the application of this latter strategy.

Then, in xUMA, the MethodLibrary metaclass must also be extended to attach a Context to it, and thus, allowing the Condition expressions to refer to the context of the project enacting the process.

The main advantage of Strategy V2 is that it provides purpose-specific constructs to capture conditions. If we use ad hoc constructs in xUMA we still have a single modeling language. However, if we make xUMA to extend from separated modeling languages (like OCL and Organization Projects Context language), although they are unified in a single metamodel (xUMA), practitioners need to learn and apply more than one language. However, this strategy requires only a single metamodel xUMA for defining Organization Software Processes. Having a single terminal model makes no separation of concerns as everything is captured together. In other words, there is no separated artifact that centralizes all conditions, but rather, conditions are tangled throughout the elements of the software process terminal model. On the other hand, V2 also has some drawbacks We lose purpose-specific tool support since EPF Composer does not use the specialized constructs for specifying conditions, so we would have to transform the UMA model into xUMA to be later manually edited by practitioners.

3) Strategy V3: Capture conditions separately in a specialized artifact: The goal of Strategy V3 is to provide separation of concerns between process modeling and the conditions on which variability is resolved, and to count on tool support. Strategy V3 consists of using separated artifacts to capture the process, the context for enacting the process, and the conditions to decide which variant must be used for each variable part. By this means, conditions (i.e., the decision rules) are captured in a centralized model. Then, a complete software process with variability is defined by means of three artifacts, as shown in Fig. 16.

While the Organization Software Process and Organization Projects Context terminal models are self-contained, the Variation Decision Rules terminal model makes references to model elements in the other two. First, each decision is attached to a variable part and a variant in the Organization Software Process. For instance, a decision states whether a specific optional task in the process should be performed or not. For example, if Task t1 is included then Task t2 is also included is a valid condition, and Task t1 is included if Project type is corrective maintenance.

The main problem with this representation is that the relationship of elements in the Variation Decision Rules terminal model with elements in the other terminal models is by name and there is a replication of model elements in the different terminal models. Instead we use a native MDE mechanism to connect the models, namely model weaving, as shown in Fig. 17.

However, this model weaving only partially solves the problem because the Variation Decision Rules terminal model still includes a representation of the elements in the other terminal models. Ideally, the Variation Decision Rules terminal model should not have another representation of the model elements in the other terminal models, but the actual links to them. Then, the Variation Decision Rules terminal model can be considered itself as a semantic relationship between elements in the software process and elements in the organization context. In other words, it can be itself a weaving model as shown in Fig. 18.

In this case we need a customized metamodel for the weaving model as the simple links provided by AMWCore[5] are not enough for expressing conditions. If a predefined expression language is used for capturing conditions, then, the weaving metamodel must also extend the metamodel of such language. Figure 19 shows the use of OCL as such language.

Strategy V3 allows for tool support for modeling software processes as it relies on UMA. However, a separated modeling tool must be used to capture the other two terminal models. To this end, we have developed a purpose-specific tool for the edition of these terminal models and then complementing EPF Composer for the complete definition of organization software processes with variability and variation resolution. The current implementation does not support OCL, it defines an ad hoc language that supports equality, conjunction and disjunction, as well as parenthesis.

4) Strategy V4: Use a controlled process modeling language: The goal of Strategy V4 is to improve Strategy V3 by making our modeling artifacts and tools independent of any specific process modeling language or version of modeling language. To this end, instead of using UMA as the modeling

[5]http://www.eclipse.org/gmt/amw

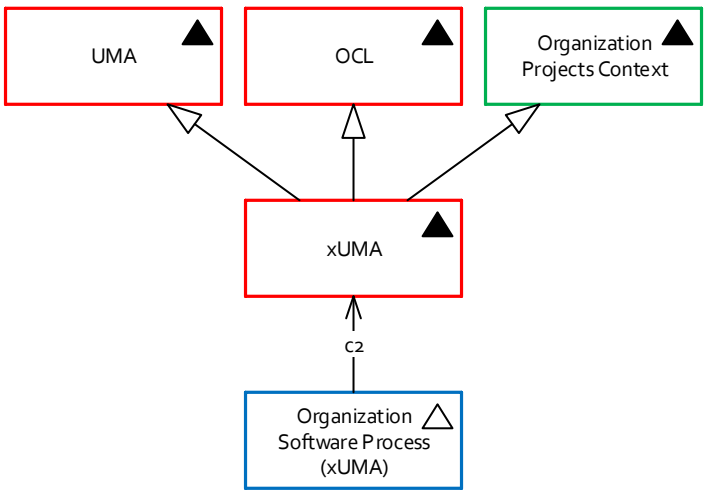

Fig. 15. UMA extensions to support variability and context specification

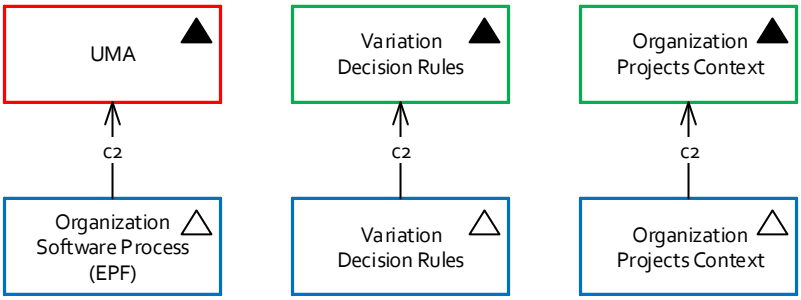

Fig. 16. Separated models for specifying process, context, and decision rules

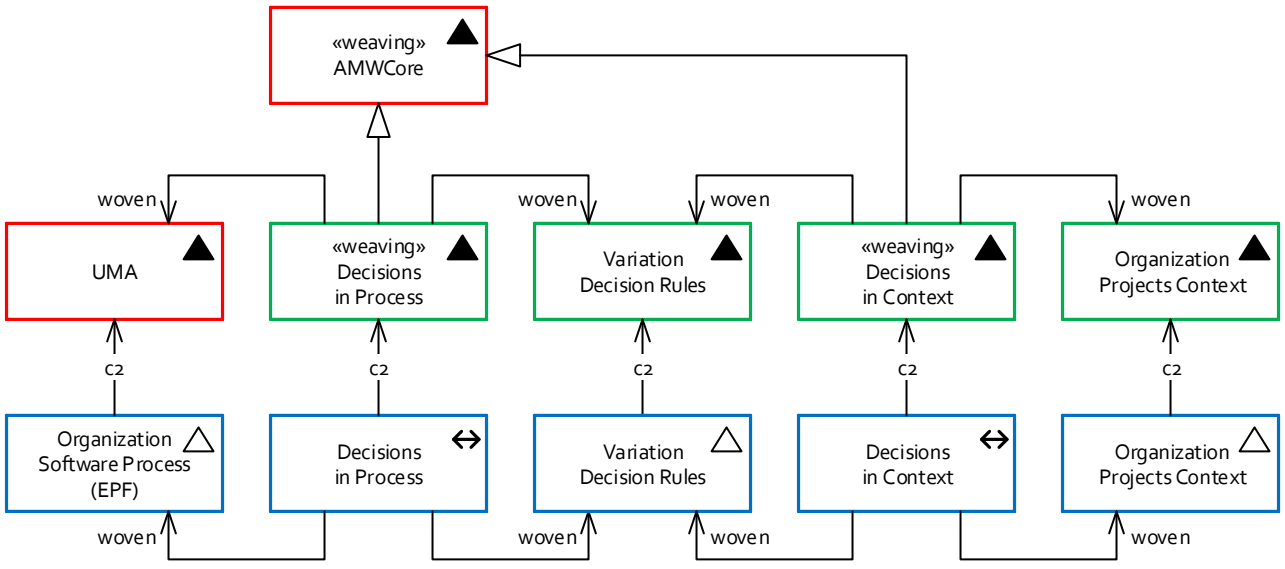

Fig. 17. Relating process, context, and decision rules with weaving models

language, this strategy relies on eSPEM that we defined in Sec. IV-A, as seen in Fig. 20.

Strategy V4 is independent on a specific version of a process modeling language. However, it requires additional tools. As we discussed in Sec. IV-A, by using a controlled process modeling language we can make our approach to work with different source modeling artifacts, such as different versions of UMA and SPEM. However, we need a custom tool for

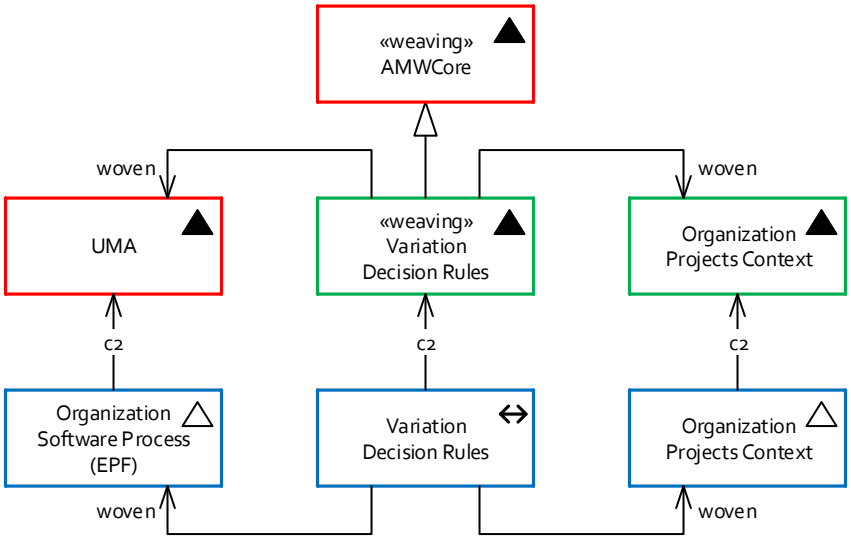

Fig. 18. Relating process, context, and decision rules with one weaving model

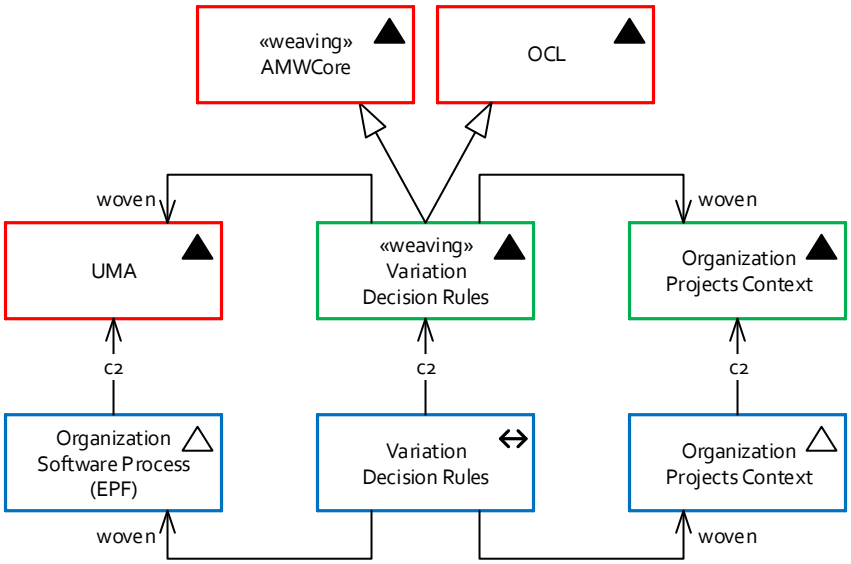

Fig. 19. Weaving model and a language for conditions (e.g., OCL)

generating process models conforming to eSPEM from those sources. The injectors discussed fulfill this purpose.

5) Resolution: The current implementation of our toolset follows a combination of Strategies V3 and V4. Following Strategy V3 we decided to use separated models to capture the process, the decision rules and the organization context. In particular, we use the first variant of this strategy in which the interrelation of the models is implicit and that does not rely on weaving models. The drawbacks of replication of model elements and the lack of validation are overcome by means of the toolset. We developed a user-friendly front-end for populating the Variation Decision Rules and Organization Projects Context terminal models. This tool is in charge of preserving the consistency of the relationships between the model elements of the terminal models. Following Strategy V4 we decided to use a controlled modeling language, particularly eSPEM. Then, the artifacts involved are illustrated in Fig. 21.

D. Context Modeling

The Organization Projects Context terminal model captures the characterization of the contexts in which the Organization Software Process can be enacted. This model provides the vocabulary for the Variation Decisions Rules to select the specific variants for particular development scenarios (contexts). Then, at the beginning of a software development project, the project manager determines the concrete characterization of the context for the specific project at hand. This concrete characterization is used to select the concrete software process that the development team has to enact. This characterization is a configuration of the Organization

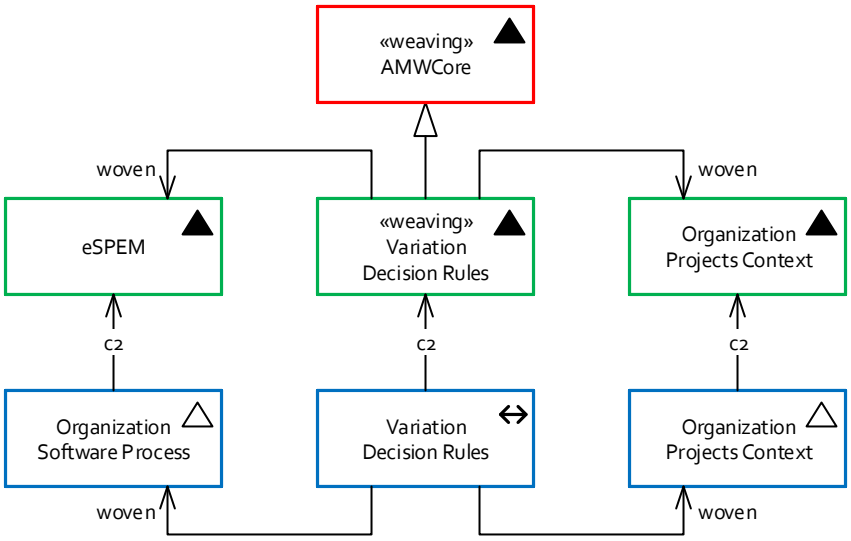

Fig. 20. Weaving model with eSPEM for process modeling

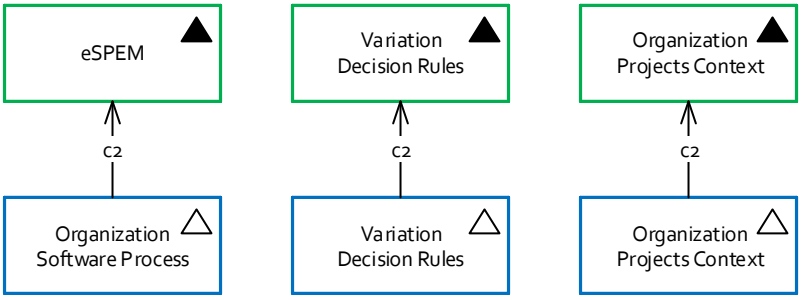

Fig. 21. Adopted Solution Structure

Projects Context in the sense that it establishes a single value for each attribute of each dimension. While the Organization Projects Context terminal model is specific to the organization, the Project Context terminal model is specific to a single project of the organization. We devise different modeling strategies to capture the concrete context for a particular project: a shared metamodel,separated metamodels, a metamodel extension, a weaving model, and an annotation model.

1) Strategy C1: Shared metamodel: Strategy C1 relies on the fact that a Project Context is a configuration of the Organization Projects Context. While the latter defines the set of possible Values for each Attribute of each Dimension, the former consists of the selection of a single Value from each set. Thus, the same modeling constructs and practically the same model elements in the Organization Projects Context also appear in a Project Context. Then, following this strategy we have a single modeling language, captured by the Organization Projects Context metamodel, to express the Organization Projects Context and every Project Context as shown in Fig. 22.

The advantage of this strategy is its simplicity, and thus it is easy to learn. However, provided that both are different

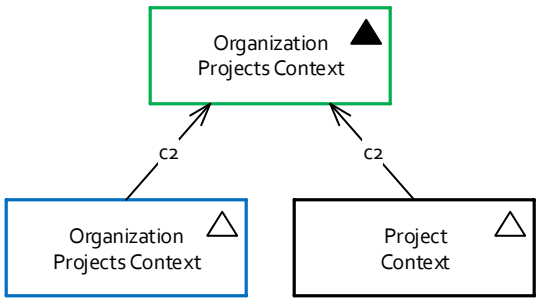

Fig. 22. Organization Projects Context and Project Context

kind of model, there are invalid configurations. For the process engineer that builds the Organization Projects Context, the metamodel must allow him to capture a set of Values for each Attribute of each Dimension, but the ability to select a single Value is not just irrelevant for the process engineer, but forbidden. Similarly, the project manager cannot select a set of Values, but only one for each Attribute. There are also implicit relationships since a Project Context is a configuration of a given Organization Projects Context such as the one-to-one relationship between Dimensions and Attributes in both

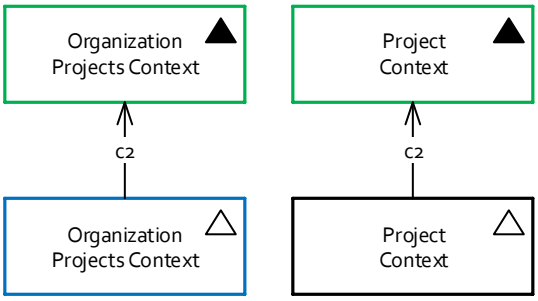

Fig. 23. Separated Organization Projects Context and Project Context

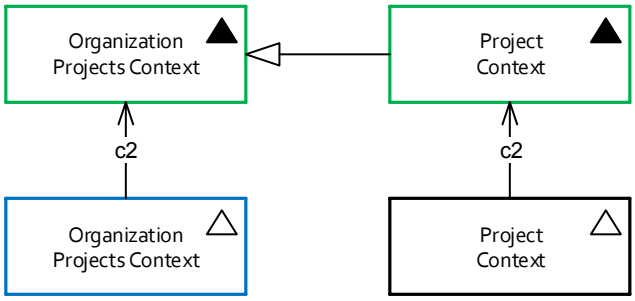

Fig. 24. Extended Separated Organization Projects Context and Project Context

models. Having only one metamodel makes this relationship implicit and the validation of well-formedness is delegated by an external tool. In addition, there is a replication of model elements in both kinds of models, requiring additional effort to build them.

2) Strategy C2: Separated metamodels: Strategy C2 consist of defining two different metamodels instead of a single shared one. One metamodel defines the modeling language to capture Organization Projects Context terminal models, and the other metamodel defines how to capture Project Context terminal models, as show in Fig. 23. The integrity is achieved at the expense of simplicity.

3) Strategy C3: Metamodel extension: Strategy C3 consist of using the metamodel extension capability for the definition of the modeling languages. We devise two possible applications of metamodel extension in this scenario. First, we can define the Project Context metamodel as an extension of the Organization Projects Context metamodel. Thus, the Project Context metamodel has all the constructs of the Organization Projects Context, and adds those specific to the selection of Values. This is shown in Fig. 24.

This solution resolves the replication of modeling constructs, but it reintroduces the problem of forbidden constructs. In particular, the Project Context metamodel provides constructs for defining sets of Values for Attributes, which should not be possible. Nevertheless, given that we now have two separate metamodels, the usage of forbidden constructs can be structurally validated by means of invariants (well-formed rules attached to a metamodel). Thus, the constructs are available but the modeling environment alerts the modeler

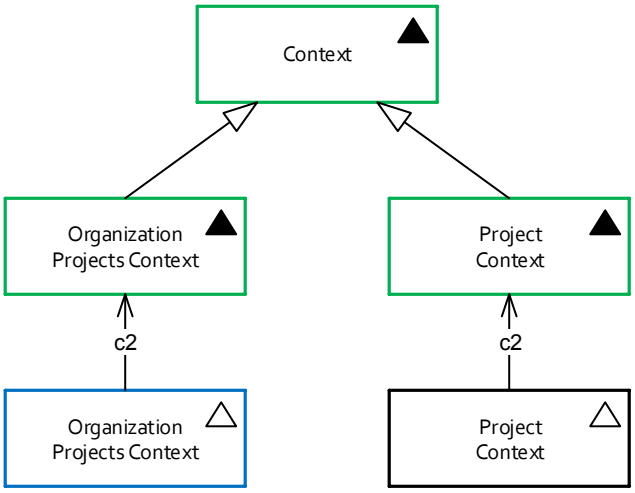

Fig. 25. Another Extended Separated Organization Projects Context and Project Context

that they must not be used. A second application of metamodel extension that does not pose this issue is by defining a base metamodel with the common set of constructs, and two extensions, one for each kind of terminal model (shown in Fig. 25). In this case, each metamodel only defines the set of constructs that is expected, and we also eliminate the replication of modeling constructs as they are defined in the base Context metamodel. However, this strategy introduces another modeling artifact to maintain.

4) Strategy C4: Weaving model: The goal of Strategy C4 is to make explicit the relationship between the model elements of the two terminal models by defining a weaving model to capture the links between the corresponding model elements. This strategy is shown in Fig. 26.

While the Organization Projects Context metamodel remains the same as in previous strategies, the Project Context metamodel must be rethought. If this metamodel provides the same set of constructs as in previous strategies (i.e., Dimensions, Attributes and selected Value), we again face the problem of replicated language constructs. However, provided that this strategy includes a weaving model, the Project Context metamodel does not need to define Dimensions and Attributes, it just need to define a Selection, which is linked to the selected Values defined in the Organization Projects Context. By this means we do not replicate the constructs. But, as a consequence, the Project Context metamodel just defines a single Selection construct for the sole purpose of having something to link with by the weaving model, and hence, the Project Context terminal models just define a single model element instance of such selection. In other words, the actual selection is in the weaving model and is captured by the defined links. Then, this strategy uses too many artifacts and some of them, particularly those concerning project context, only capture information that is available in other models.

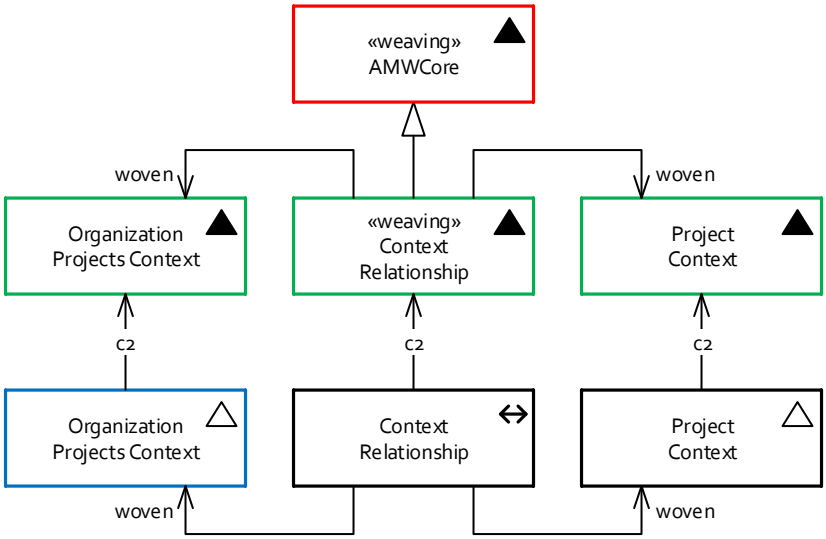

Fig. 26. Weaving Context Models

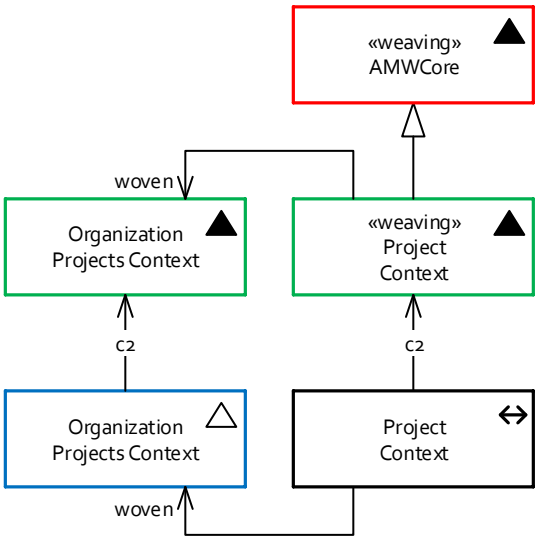

Fig. 27. Annotated Weaving Context Models

5) Strategy C5: Annotation model: The goal of Strategy C5 is to solve the problem of strategy C4 while preserving all the advantages gained by the previous strategies. Strategy C5 relies on the fact that a Project Context terminal model is a configuration of the Organization Projects Context, and as such, the former solely appends information to the latter. In terms of MDE constructs, this can be achieved by an annotation model, the particular kind of weaving model that has a single woven model. This strategy is shown in Fig. 27.

The Project Context weaving metamodel only defines the specific kind of link that represents the selection of a Value defined in the woven models. By this means, this strategy does not pose any of the problems of the previous strategies.

6) Resolution: The current implementation of our toolset follows Strategy C1 for its simplicity, and overcome its drawbacks by means of a user-friendly toolset for building the terminal models and keeping them consistent. In practice, the process engineer of a SSE uses our toolset to define the single Organization Projects Context terminal model for the organization. Later, at the beginning of each project, the project manager uses our toolset to define the specific Project Context terminal model. The toolset uses the defined Organization Projects Context terminal model to offer the project manager the available values to select, and hence, capturing a well-formed Project Context terminal model. As future work, we plan to fully embrace Strategy C5 and make our toolset rely on standard MDE constructs and tools to preserve and validate the well-formedness of these terminal models.

V. SPrL Tailoring

Given the megamodel defined in Section IV, we can now formally define SPrL process tailoring. Process tailoring is one possible application enabled by our megamodel, other uses include for example process analysis and evolution. In our approach, process tailoring is divided into two steps: 1) generation and 2) application of the tailoring transformation. Figure 28 and 29 show how these steps are formalized using our megamodel, respectively.

A. Tailoring transformation generation

Once the Organization Projects Context and Variation Decision Rules terminal models have been defined by the process engineer, we can generate the tailoring transformation. This is done using an organization-independent Higher Order Transformation (HOT) [50] that takes these two models as input and automatically generates an organization-specific process tailoring transformation. In Fig. 28, Tailor Generation and Tailor represent the HOT transformation and the tailoring transformation, respectively. As reported in [46], the HOT has been implemented in Java, and the tailoring transformation

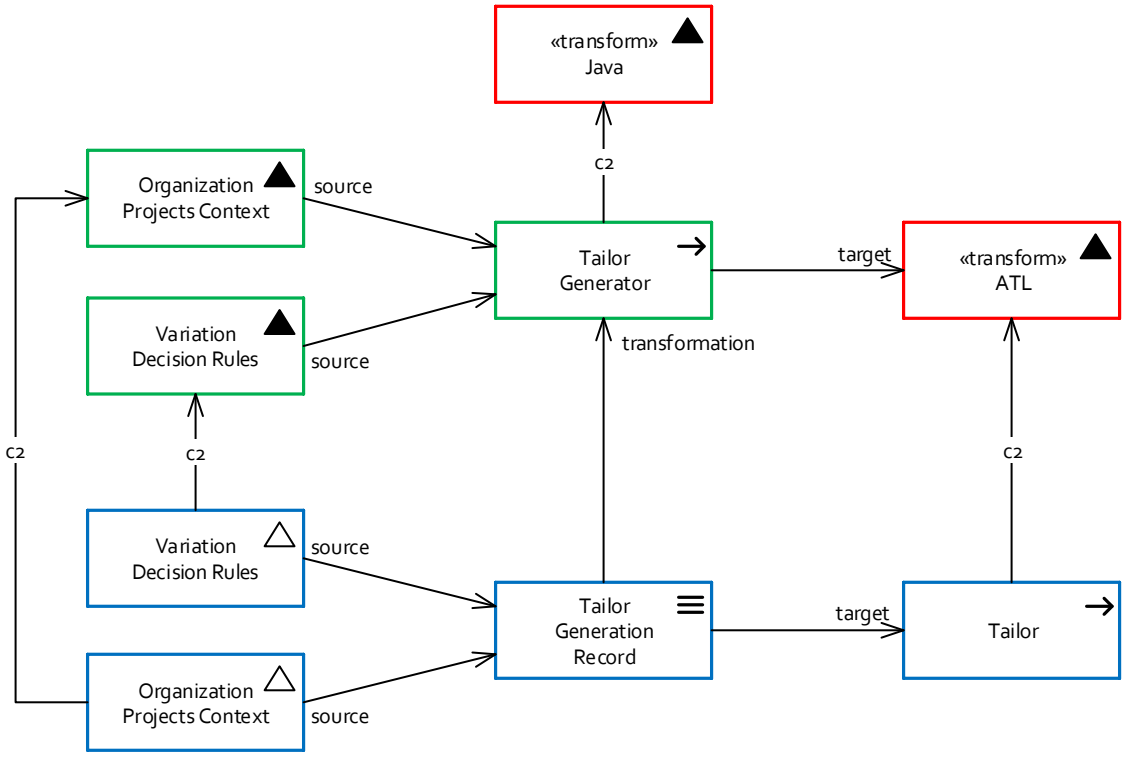

Fig. 28. Tailoring transformation generation.

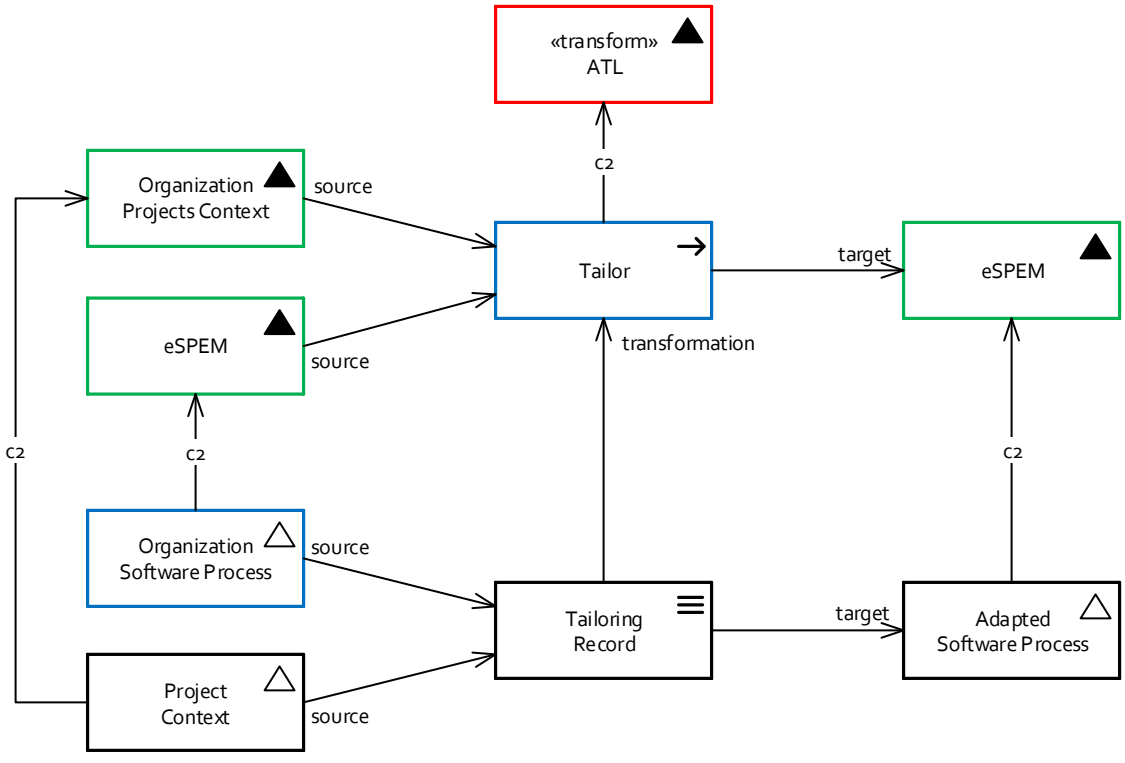

Fig. 29. Tailoring transformation application.

conforms to ATL. Figures 30 shows an excerpt of the tailoring transformation, which was generated using the models shown in Fig. 2 and 6. Moreover, a Tailor Generation Record is registered each time the HOT transformation is executed.

	Project			System				Team	
	Project type			*System type*		*Interaction with other systems*		*Experience in the architecture*	
	New development	Corrective	Non-corrective	Guarantee	No guarantee	Simple	Complex	Yes	No
Context A	x			x			x		x
Context B		x			x	x		x	
Context C			x	x		x		x	

Product												
Usability		*Request*		*Complexity of the functionality*			*Data access layer*		*Business logic*		*Source code*	
High	Low	Report	No report	High	Average	Low	Yes	No	Yes	No	Yes	No
x			x	x			x			x		x
	x	x			x			x	x			x
x			x			x		x		x	x	

Fig. 31. Three different project context configurations defined by Mobius.

```
--Variability points: Requirements
helper def:rule1():Boolean=
  if(thisModule.getValue('Project type')='Non-corrective')
    then true
    else false
  endif;

--Optional Rules: Requirements
helper def:rule2():Boolean=
  if(thisModule.getValue('Project type') = 'New development' or
      thisModule.getValue('Interaction with other systems') = 'Complex')
    then true
    else false
  endif;

helper def:rule3():Boolean=
  if(thisModule.getValue('Usability')='High')
    then true
    else false
  endif;

helper def:rule4():Boolean=
  if((thisModule.getValue('Project type') = 'Non-corrective' or
      thisModule.getValue('Project type') = 'New development' ) and
      thisModule.getValue('Request') = 'No report')
    then true
    else false
  endif;

helper def:rule5():Boolean=if(thisModule.getValue('Request') = 'Report' or
    thisModule.getValue('Project type') = 'Corrective' )
  then true
  else false
endif;
```

Fig. 30. Excerpt of the tailoring transformation generated for Mobius' SPrL.

B. Tailoring transformation application

Before starting a new project, the project manager must define the project's context values (Project Context terminal model in Fig. 29). Figure 31 shows three project contexts for Mobius in tabular form: New Development project (A), Corrective Maintenance project (B), and Non-corrective Maintenance project (C). The tailoring transformation Tailor

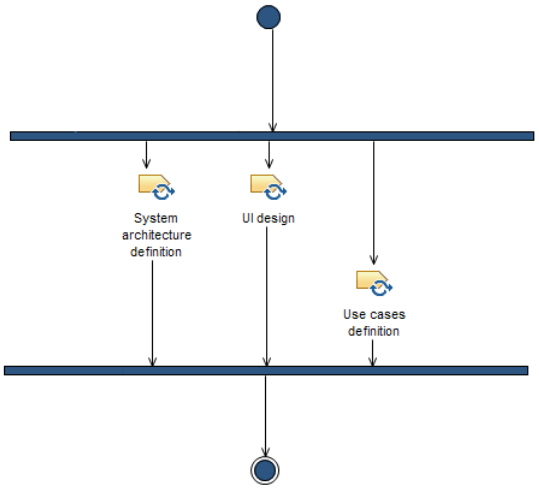

Fig. 32. Requirements activity, tailored to Context A.

generated in the previous subsection takes as input one of these contexts, as well as the Organization Software Process, generating the Adapted Software Process for that project context. For example, if the project manager chooses Context A, the process shown in Fig. 32 is generated for the new project. Choosing Context B results in the process shown in Fig. 33, which is quite different. Each application of the tailoring transformation is registered (see Tailoring Record in Fig. 29).

VI. SPrL Evolution

We now discuss how our megamodel enables process line evolution, using as examples change requests that Mobius is now planning to carry out on their SPrL.

A. Unit of Evolution

Change can occur at any level of our megamodel: both in the organization-independent and -specific models, as well as in the project-specific models. Moreover, the standards used

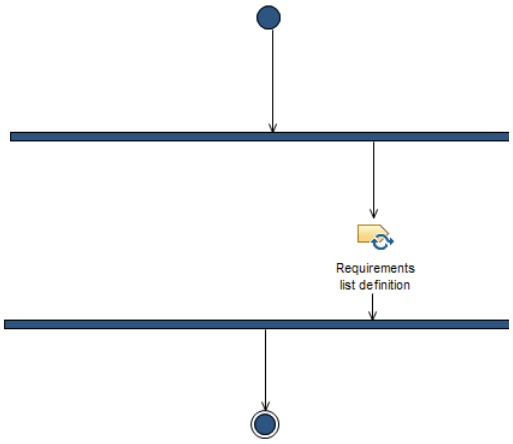

Fig. 33. Requirements activity, tailored to Context B.

by the ADAPTE project can also change (SPEM 2.0, UMA, ATL, etc.). Not being able to deal with these types of changes reduces the expected lifetime of an SPrL, and not dealing with them in a uniform and systematic manner reduces the chances of industrial adoption of our approach.

Also, we expect that some types of changes will be more frequent than others. For example, the organizational process model, the organizational context, and tailoring rules may change soon after SPrL definition, to account for differences between the modeled process and the actual process carried out by the company. These artifacts may also change as a result of an SPI process [23]. Other elements, like the HOT or the eSPEM model, are less likely to change, as they have stabilized over the course of the ADAPTE project.

Thus, according to the evolution profiles defined in [20], our SPrLs are under *continuous evolution*: evolution can occur at the domain or application level; model elements can be added, removed or modified; and changes to the relationship between context elements and variation points may lead to a reorganization of the variation decision rules, as well as changes to the transformations.

In practice, continuous evolution means an increased maintenance effort and model complexity for large systems in relation to small and simple systems [20]. Weak tool support also makes it hard to carry out preventive changes in the SPrL that would improve its evolution (and shelf-life). All this leads to Product Line Erosion [38], [26]: an increasing deviation from the requirements specification, such that key properties of the product line no longer hold.

Good tool support for evolution should enable a typical software change process, like the one defined by Rajlich [41]. Here, (semi-)automated change impact analysis and propagation are key to empowering the change process, allowing the engineer to better understand the impact of the proposed evolution of the SPrL. This is one of the main advantages of our megamodeling-based SPrL approach: as dependencies between the different models and transformations are explicit in the megamodel, our tools can offer automated change impact analysis without requiring additional effort than that already required to define the SPrL. We can also offer versioning, as the megamodel is a catalog of the artifacts in the model repository, and we keep track of all changes using transformation records. By reducing the SPrL maintenance effort, we expect to see an increase in the lifetime of the SPrLs defined by our partner companies.

B. Organizational Process Management

The definition of a software process using EPF Composer is labor-intensive, even for a small process. So taking advantage of this investment is appealing. Changes such as adding a new template for a work product, or changing the role in charge of a task should be easy to carry out, as these kinds of changes are quite frequent. Moreover, process variation points may change over time. For example, if the template assigned to the Architecture Prototype work product needs to be changed, it can be done directly in the EPF Composer and then the Injector needs to be re-run. The updated Organizational Software Process will refer to the new template for all future projects.

Having a formally defined process does not necessarily mean that it is the most efficient or appropriate process. If the process is analyzed with a tool, we can determine if there is an overloaded role that may cause inefficiencies, or if there is a role that does not interact with any other. In both cases, the EPF Composer definition of the process can be modified so that overloaded roles delegate some of their responsibilities, and that isolated roles are either assigned to tasks, or removed from the process definition.

For example, Mobius wants to update their Organizational Software Process, removing the System architecture definition task from the Requirements activity shown in Fig. 5. In order to evolve the SPrL, the process engineer must first update the Organizational Software Process using EPF Composer. This will leave the Variation Decision Rules in an inconsistent state, as it now refers to a process element that no longer exists. As our toolset checks the consistency between these models, it can notify the process engineer, who must confirm the proposed changes to the decision model. This in turn triggers the execution of the HOT, producing a new tailoring transformation (see Sec. V). Now, when the process manager picks Context A for a new project, the resulting adapted process is similar to the one shown in Fig. 32, but it does not include the System architecture definition task because it is no longer in the Organizational Software Process.

C. Context Evolution

Context values vary for each project, a new Project Context should be created. Then, the tailoring transformation can be executed in order to obtain the Adapted Software Process. This kind of change is not dramatic, but is quite frequent, so tool support is essential. For example, if the project manager picks Context A, but then changes the value of the context attribute Usability to Low, the resulting adapted process is

similar to the one shown in Fig. 32, but it does not include the UI design task.

On the other hand, the Organization Projects Context may require changes such as adding, modifying or deleting context attributes and/or their values. This affects the creation of new Project Contexts. If attributes or values are removed or modified in the Organization Projects Context, then the Variation Decision Rules must be checked for consistency. Adding, updating or removing attributes or values may require adding new rules, updating existing rules, or removing existing rules.

Either way, the Tailor Generator must be re-run if the Variation Decision Rules changes, creating a new version of the tailoring transformation. For example, if the context attribute Request is removed from the Organization Projects Context, the variation decision rule attached to the decision node in Fig. 5 must be updated to Project = corrective. Re-running the HOT will create a new version of the tailoring transformation. Now, when the project manager picks Context A, the resulting adapted process is exactly the same as the one shown in Fig. 32, since Project type = corrective is true in this context.

D. Tailoring Rule Evolution

Tailoring rules should evolve when the Adapted Software Process generated by the tailoring transformation is not as expected. If the problem is in the Organization Software Process, then the procedure described in Sec. VI-B must be followed; if it is in the Project Context, then the procedure in Sec. VI-C must be followed. However, if both models are correct, it means that the relationship between the two is what is incorrectly defined, and therefore the Variation Decision Rules must be analyzed, checking if the mapping between the context attributes and variation points is correctly specified. Once this model is updated, the HOT must be re-run to create a new tailoring transformation.

VII. Related Work

In this section, we survey current work on process variability modeling and evolution, and compare it with our approach.

A. Process Variability Modeling

SPEM 2.0 defines four primitives for specifying variability between two process elements of the same type [35]: *contributes*, *replaces*, *extends* and *extends-replaces*. It is hard to predict how variability relations interact with each other, as instances of these relations may override each other, which limits the practical use of this approach. Martínez et al. [33] propose vSPEM, a SPEM extension that allows the direct specification of process variability. In this proposal, the process engineer defines process variation points, and in a separate model documents variants and their relationships to variation points. vSPEM is a concise notation with the added advantage that it is specific to the process domain. However, existing tool support is at an academic prototype level, without graphical user interfaces, making it a poor candidate for industrial adoption.

Feature Models [27] (FM) can also be used to model process variability. Using FMs, each process element is represented as a feature. Optional variation points are modeled as optional features, and process elements that can be realized in different ways are modeled using alternatives. Requires constraints are added to ensure that existing links between roles and tasks, and tasks and work products are preserved. This approach has been explored in [25], [17], but it is limited in that variability is "lost" in the FM when working with process lines (many common elements and few variation points). Moreover, the FM and the organizational process model must co-evolve, introducing additional evolution challenges.

Business Process Models (BPM) [51] are used to model general business processes. There have been several proposals for capturing variability in BPMs. For example, Hallerbach et al. [18] introduce the notion of "options", sequences of change operations over the base BPM (*insert*, *delete* or *move* BPM fragment, and *modify* element attribute) that are applied at "adjustment" points which are explicitly identified in the base BPM. Variant BPMs are then created by executing one or more options. The drawback of this type of approach is that variability is specified operationally, so changing the order in which options are applied can lead to different variants. It also gets harder to understand how options interact as the set of options grows, or as the individual options become more complex.

Configurable Event-Driven Process Chains [43], [29] (C-EPC) is another approach for modeling business process variability. EPCs are directed graphs, nodes are events or functions, and logical connectors are used to define control-flow. C-EPC extends EPC by adding configuration nodes: functions and decision nodes are annotated with constraints to indicate whether they are mandatory or optional; these constraints are predicates over C-EPC elements. Variant C-EPCs are created by processing these annotations. Notation-wise, development processes specified using C-EPCs can be hard to understand: 1) all process elements must be modeled as function nodes, 2) context information cannot be used to resolve variability, and 3) alternatives must be modeled using decision nodes. This is especially confusing to the end-user, as some represent control-flow decisions whereas others represent variability.

The DOPLER [13] approach is the closest to ours in spirit. Assets are defined using models, and the relationships between them are specified using a decision model. This allows automatic product derivation and reconfiguration: based on the decisions made by the stakeholder, components can be automatically selected for inclusion in the configured system and components can be instantiated and composed as required. Our megamodel can be seen as a generalization of this approach, as the input metamodels can be changed as needed by the problem domain. Currently, DoplerVML is more expressive than our decision model; however, our decision model is sufficiently expressive to allow the specification of the

variation decisions found in the process domain, and we can improve the expressiveness of our approach by changing the corresponding metamodel in the megamodel.

B. Variability Evolution

Recent empirical studies [9], [55] argue that existing approaches offer weak tool support for product line evolution. Without adequate tool support for change propagation, it is hard for engineers to understand the possible impact of their changes to feature hierarchies, and as a result, evolution is primarily limited to adding features and decisions [5]. As a consequence, the models supporting SPL definition and derivation are seldom refactored, leading to an increase in model complexity over time.

Several ad hoc traceability models have been proposed as a way to control product line evolution (cf. [1], [45], [2]). Model-based approaches for modeling and evolving product lines have also been defined in the literature (cf. [49], [52], [30], [14], [21], [39]); however, to our knowledge, we are the first to generalize this approach by defining a megamodel for product lines, enabling product line evolution in a controlled and uniform manner.

VIII. Conclusions and Future Work

There are several factors affecting the adoption of software processes in industry: (a) the complexity, expressiveness and understandability of the notations and languages, (b) the cost of coping with variability and of successfully achieving variability resolution for particular development scenarios, (c) the degradation of the expensively-captured process model with respect to the actual process enacted process due to an ill-managed change and evolution, and (d) the availability and usability of tool support.

While MDE techniques and tools can be regarded as a solid foundation to address those challenges, a naïve application of MDE relying only on modeling-in-the-small constructs rapidly degenerates in a large and complex set of modeling artifacts, and versions of them, that are ill-managed and cannot evolve consistently. As a consequence, the erosion of the consistency renders those artifacts unreliably and the investment on process modeling unproductive.

On the other hand, an application of MDE relying on modeling-in-the-large constructs promotes control. As we show in this article, by applying a megamodeling approach, modeling artifacts are well-classified and cataloged and their interrelations are explicit. Thus, the impact of change can be readily identified and visualized. For instance, whenever a change is required in a particular modeling artifact, the megamodel can be navigated to analyze which related modeling artifacts should be preserved, re-examined, or re-generated.

Also, the megamodel provides the big picture of all involved artifacts and the role they play in the overall solution. As a consequence, we can improve our toolset in a focused manner to improve industrial adoption, by mitigating risks, addressing problems, or assisting in their resolution. For instance, the choice to introduce the proprietary process modeling language eSPEM was to shield our solution from changes to the process modeling approach used by SSEs. While it is not frequent for a SSE to change its process modeling language, this scenario is recurrent for our research team as we interact with multiple SSEs.

Another example is the introduction of the tailor generator. While our toolset initially relied on a custom-built tailoring transformation, this approach did not scale as we worked with more SSEs, so we abstracted the variation decision rules in a purpose-specific model instead of embedding them in the tailoring transformation. This change significantly improved the usability of our toolset.

To conclude, we have shown that megamodeling is a feasible solution for the definition and evolution of software process lines. Megamodeling allows managing the complexity of approaches that are intensive in modeling artifacts, and it is even more relevant when those modeling artifacts evolve. The advantages of using a megamodel include: 1) classification and catalogization of all the modeling artifacts in use, 2) the relationships between artifacts are made explicit and 3) megamodeling serves as a backend for centralized and integrated tool support. This allows us to (semi-)automate change propagation among the modeling artifacts. Moreover, for a solution that is intensive in weaving models, the underlying platform can assist validation and traceability of change at the model element level, not only at the modeling artifact level.

Future work. Further integration of tool support is required by our industrial partners. As EPF is a standalone product, process modeling is performed in a non-MDE environment. Provided that its underlying platform is Eclipse, we plan to integrate EPF toolset into an Eclipse Modeling edition of Eclipse, and provide a single-IDE experience to process engineers and project managers.

We also need to improve the visualization of the impact of evolution. In this article we set the foundation for the identification of the impact of change. We want to help the process engineer manage evolution of their process line by generating evolution "previews", which are automatically computed from the megamodel, the process engineer can accept these changes as-is if she/he is happy with them.

References

[1] S. Ajila and B. A. Kaba. Using traceability mechanisms to support software product line evolution. In D. Zhang, . Grgoire, and D. DeGroot, editors, *IRI*, pages 157–162. IEEE Systems, Man, and Cybernetics Society, 2004.

[2] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J.-C. Royer, A. Rummler, and A. Sousa. A model-driven traceability framework for software product lines. *Software & Systems Modeling*, 9(4):427–451, 2010.

[3] M. Barbero, F. Jouault, and J. Bézivin. Model Driven Management of Complex Systems: Implementing the Macroscope's Vision. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS'2008), 31 March - 4 April 2008, Belfast, Northern Ireland*, pages 277–286, Washington, DC, USA, 2008. IEEE Computer Society.

[4] V. R. Basili and H. D. Rombach. Tailoring the Software Process to Project Goals and Environments. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 345–357, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[5] T. Berger, D. Nair, R. Rublack, J. Atlee, K. Czarnecki, and A. Wsowski. Three cases of feature-based variability modeling in industry. In J. Dingel, W. Schulte, I. Ramos, S. Abraho, and E. Insfran, editors, *Model-Driven Engineering Languages and Systems*, volume 8767 of *Lecture Notes in Computer Science*, pages 302–319. Springer International Publishing, 2014.

[6] J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

[7] J. Bézivin, F. Jouault, and P. Valduriez. On the Need for Megamodels. In *Best Practices for Model-Driven Software Development Workshop, at the Third International Conference on Generative Programming and Component Engineerings (GPCE'2004), co-located with the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2004), 24-28 October 2004, Vancouver, Canada*, 2004.

[8] P. Borges, P. Monteiro, and R. Machado. Mapping RUP Roles to Small Software Development Teams. In S. Biffl, D. Winkler, and J. Bergsmann, editors, *Software Quality. Process Automation in Software Development*, volume 94 of *Lecture Notes in Business Information Processing*, pages 59–70. Springer Berlin Heidelberg, 2012.

[9] L. Chen and M. Babar. Variability management in software product lines: An investigation of contemporary industrial challenges. In J. Bosch and J. Lee, editors, *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, pages 166–180. Springer Berlin Heidelberg, 2010.

[10] A. Cockburn. *Crystal Clear a Human-powered Methodology for Small Teams*. Addison-Wesley Professional, first edition, 2004.

[11] K. Conboy and B. Fitzgerald. Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. *ACM Trans. Softw. Eng. Methodol.*, 20(1):2:1–2:30, July 2010.

[12] B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Commun. ACM*, 35(9):75–90, Sept. 1992.

[13] D. Dhungana, P. Grünbacher, and R. Rabiser. The DOPLER Metatool for Decision-oriented Variability Modeling: A Multiple Case Study. *Automated Software Eng.*, 18(1):77–114, Mar. 2011.

[14] D. Dhungana, T. Neumayer, P. Grunbacher, and R. Rabiser. Supporting evolution in model-based product line engineering. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 319–328, Sept 2008.

[15] Eclipse. Eclipse process framework project 1.5.1.6, 2013. http://projects.eclipse.org/projects/technology.epf.

[16] M. C. B. Felipe González, L. Silvestre, and M. Solari. Template-Based vs. Automatic Process Tailoring. In *XXXIII International Conference of the Chilean Society of Computer Science (SCCC 2014)*, November 2014.

[17] G. GröNer, M. BošKović, F. Silva Parreiras, and D. GašEvić. Modeling and validation of business process families. *Inf. Syst.*, 38(5):709–726, July 2013.

[18] A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: The provop approach. *J. Softw. Maint. Evol.*, 22(67):519–546, Oct. 2010.

[19] P. Haumer. Ibm rational method composer: Part 1: Key concepts. Technical Report http://www.ibm.com/developerworks/rational/library/dec05/haumer/index.html, IBM: The Rational Edge.

[20] W. Heider, R. Froschauer, P. Grünbacher, R. Rabiser, and D. Dhungana. Simulating evolution in model-based product line engineering. *Inf. Softw. Technol.*, 52(7):758–769, July 2010.

[21] W. Heider, R. Rabiser, D. Dhungana, and P. Grnbacher. Tracking evolution in model-based product lines, 2009.

[22] W. S. Humphrey. *Managing the Software Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[23] J. Hurtado Alegria, M. Bastarrica, and A. Bergel. Avispa: a tool for analyzing software process models. *Journal of Software: Evolution and Process*, 26(4):434–450, 2014.

[24] J. Hurtado Alegría, M. Bastarrica, S. Ochoa, and J. Simmonds. MDE software process lines in small companies. *J. of Systems and Software*, 86(5):1153–1171, 2013.

[25] J. Hurtado Alegria, M. Bastarrica, A. Quispe, and S. Ochoa. An MDE approach to software process tailoring. In *Proc. of ICSSP*, pages 43–52, 2011.

[26] John D. McGregor. The Evolution of Product Line Assets. Technical Report CMU/SEI-2003-TR-005. ESC-TR-2003-005, Carnegie Mellon Seoftware Engineering Institute, 2003.

[27] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, January 1998.

[28] M. Kuhrmann, D. M. Fernández, and R. Steenweg. Systematic software process development: Where do we stand today? In *Proceedings of the 2013 International Conference on Software and System Process*, ICSSP 2013, pages 166–170, New York, NY, USA, 2013. ACM.

[29] M. La Rosa, J. Lux, S. Seidel, M. Dumas, and A. ter Hofstede. Questionnaire-driven configuration of reference process models. In J. Krogstie, A. Opdahl, and G. Sindre, editors, *Advanced Information Systems Engineering*, volume 4495 of *Lecture Notes in Computer Science*, pages 424–438. Springer Berlin Heidelberg, 2007.

[30] J. Liu, J. Dehlinger, H. Sun, and R. Lutz. State-based modeling to support the evolution and maintenance of safety-critical software product lines. *Engineering of Computer-Based Systems, IEEE International Conference on the*, 0:596–608, 2007.

[31] J. Lonchamp. A Structured Conceptual and Terminological Framework for Software Process Engineering. In *In Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, 1993.

[32] G. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving (5th Edition)*. Pearson Addison Wesley, 2004.

[33] T. Martnez-Ruiz, F. Garca, M. Piattini, and J. Mnch. Modelling software process variability: an empirical study. *IET Software*, 5(2):172–187, 2011.

[34] ModelPlex Project. Deliverable D2.1.a: "Global Model Management Principles", March 2008. http://docatlanmod.emn.fr/AM3/Documentation/D2-1-a_Global_Model_Management_Principles_v1-1.pdf (accessed on August 2014).

[35] Object Management Group. Software Process Engineering Metamodel SPEM 2.0 OMG Specification. Technical Report ptc/07-11-01, 2008.

[36] OMG. Analysis and design task force. spem final adopted specification. Technical Report ptc/02-01-23, Object Management Group, 2002.

[37] L. Osterweil. Software Processes Are Software Too. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[38] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, Oct. 1992.

[39] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski. Model-driven support for product line evolution on feature level. *J. Syst. Softw.*, 85(10):2261–2274, Oct. 2012.

[40] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[41] V. Rajlich. *Software Engineering: The Current Practice*. CRC Press, 2011.

[42] J. Ralyté, R. Deneckère, and C. Rolland. Towards a generic model for situational method engineering. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, CAiSE'03, pages 95–110, Berlin, Heidelberg, 2003. Springer-Verlag.

[43] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, Mar. 2007.

[44] K. Schmid and I. John. A customizable approach to full lifecycle variability management. *Sci. Comput. Program.*, 53(3):259–284, Dec. 2004.

[45] L. Shen, X. Peng, and W. Zhao. A comprehensive feature-oriented traceability model for software product line development. *2014 23rd Australian Software Engineering Conference*, 0:210–219, 2009.

[46] L. Silvestre, M. C. Bastarrica, and S. F. Ochoa. Generating Transformations with Two Input Models. In *SCCC 2013*, 2013.

[47] L. Silvestre, M. C. Bastarrica, and S. F. Ochoa. A Model-based Tool for Generating Software Process Model Tailoring Transformations. In *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, Lisbon, Portugal, 7 - 9 January, 2014*, pages 533–540, 2014.

[48] Software Productivity Consortium Services Corporation. Reuse-Driven Software Processes Guidebook, Version 02.00.03. Technical Report SPC-92019-CMC, 1993.

[49] R. Sudarsan, S. Fenves, R. Sriram, and F. Wang. A product information modeling framework for product lifecycle management. *Computer-Aided Design*, 37(13):1399 – 1411, 2005.

[50] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the use of higher-order model transformations. In *Proceedings of the 5th*

European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA '09, pages 18–33, Berlin, Heidelberg, 2009. Springer-Verlag.

[51] W. M. P. Van Der Aalst, A. H. M. T. Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[52] M. Voelter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *Proceedings of the 11th International Software Product Line Conference*, SPLC '07, pages 233–242, Washington, DC, USA, 2007. IEEE Computer Society.

[53] O. Vogel, I. Arnold, A. Chughtai, and T. Kehrer. *Software Architecture - A Comprehensive Framework and Guide for Practitioners*. Springer], year = 2011, note = Publisher online: http://link.springer.com/book/10.1007%2F978-3-642-19736-9,.

[54] D. M. Weiss and C. T. R. Lai. *Software Product-line Engineering: A Family-based Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[55] B. Zhang, M. Becker, T. Patzke, K. Sierszecki, and J. E. Savolainen. Variability evolution and erosion in industrial product lines: A case study. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 168–177, New York, NY, USA, 2013. ACM.

APPENDIX

Figure 34 shows our metamodel for megamodels. This is a refined version of the GMM megamodel metamodel, with respect to three different aspects:

1) We have eliminated tool-specific elements. This allows us to raise the level of abstraction of the presentation of our approach and illustrate it in a tool-agnostic manner.
2) The Relationship metaclass hierachy has been fused into the corresponding metaclasses of the Entity metaclass hierarchy. This change was made to simplify the number of elements in the figures illustrating parts of the megamodel.
3) We have explicitly modeled Weaving Metamodel and Annotation Model as metaclasses. This allows us to give a finer characterization of frequently used modeling artifacts in our domain.

These changes were made to simplify the presentation of our megamodel; however, the original metamodel is used in our toolset.

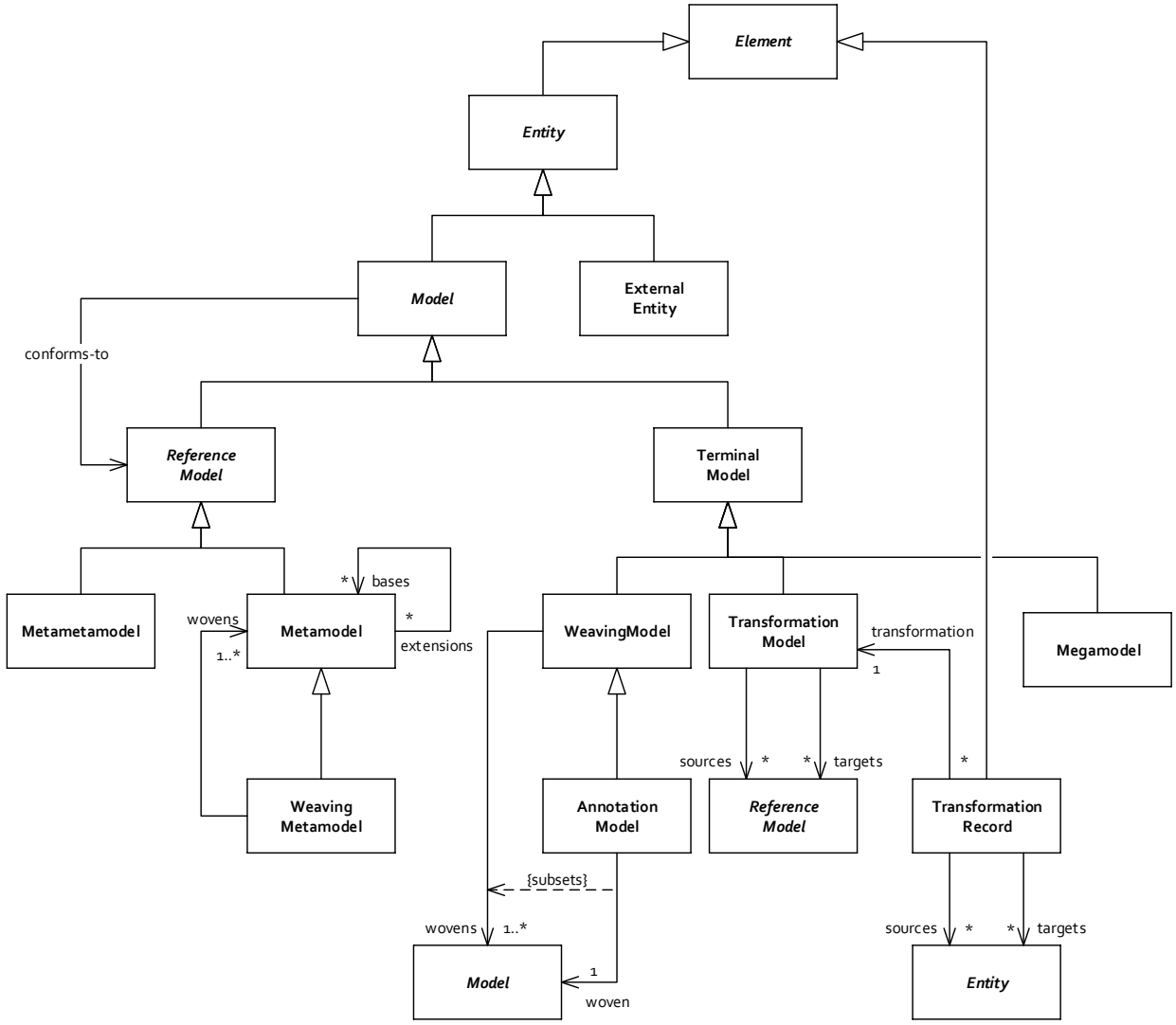

Fig. 34. Metamodel for megamodels.