A Megamodel for Process Tailoring and Evolution

María Cecilia Bastarrica
Comp. Science Department
Universidad de Chile
Chile
cecilia@dcc.uchile.cl

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

Luis Silvestre
Comp. Science Department
Universidad de Chile
Chile
lsilvest@dcc.uchile.cl

ABSTRACT

Companies formalize their software processes as a way of organizing their development projects. In practice, a family of processes is required, in order to ensure that each project is handled appropriately. This family may be a collection of predefined processes, but can also be generated by tailoring a general process to a project's context. Automated process tailoring yields the most appropriate process for a project, but requires formalization and tool support to be successful – the general process, its potential variability and tailoring transformations must be formalized up-front, and the context characterizing the project must be specified. MDE provides a formal framework for defining the models and transformations required for automated process tailoring, but as various types of models must be specified, evolution of process families thus defined is hard to manage in practice, limiting the industrial adoption of this approach. To address this problem, in this paper we propose a Megamodel for automated process tailoring. Megamodeling provides an integrating framework for modeling in the large, and as such, enables a controlled evolution of a process family. We report the application of our approach to the software development process of a small Chilean company, showing how different types of evolution are handled in an integrated manner.

1. INTRODUCTION

Software development companies face different kinds of projects, each one with its own characteristics (its "context"). Defining a new process for each project is economically unfeasible, an alternative is to rely on a series of predefined processes [4]. However, this solution is not only suboptimal for almost all projects (since one must manually decide which process best "fits" the project at hand), but it is also an expensive solution, as these processes must be evolved consistently [5]. Defining a general process that is tailored for each project seems to be a good trade-off between these two approaches: here, a general process, including its potential variation points, is defined, and it is adjusted to each new project's context. In this way, the set-up costs are limited, and the adjustment costs are distributed among projects [11].

Manually tailoring a general process has a series of drawbacks: it requires a skilled process engineer, who must have experience in process tailoring, and project managers must be able to consistently characterize new projects [22]. In other words, it is a labor-intensive task. Moreover, the results are not necessarily consistent, and two similar projects may end up with different processes, and thus their performance cannot be objectively compared [7]. Automated process tailoring is not only faster, but also it yields consistent results; however, it requires formalizing all the elements involved in the tailoring process.

We use Model-driven engineering (MDE) to formalize the tailoring process. MDE is a software development approach that focuses on formally defining models that may be automatically transformed into lower-level models and eventually into source code [23]. Its goal is to raise the abstraction level when dealing with domain-specific concepts. The most relevant elements in MDE are models and transformations. Models are formally defined using metamodels. A metamodel is a special type of model, one that specifies the concepts that may be present in models conforming to that metamodel, as well as their allowed relationships. Transformations can be thought of as programs that take models as input and generate other models as output; admissible input and output models are specified by the metamodels the input and out models conform to.

Applying an MDE approach to process tailoring, software processes and project contexts are formalized as models, and process tailoring is a model-to-model (M2M) transformation [7]. Here, software process models are specified in SPEM 2.0, using the EPF Composer[1] tool. This tool allows the graphical visualization of processes, which helps improve process understandability. Since tailoring is a M2M transformation, but the EPF Composer uses XML files as input/output, we also developed an injector and extractor, a text-to-model (T2M) and model-to-text (M2T) transformation [2], respectively. In practice, we have encountered two types of process variation points: 1) optionality, i.e., process elements that must either included in or excluded from the adapted process, depending on the project's context, and 2) alternatives, i.e., process elements that can be implemented in different ways, and one of these alternatives must replace the variation point in the adapted process (also depending on the project's context). We have developed our own metamodel for modeling project contexts, and we have developed tailoring transformations using ATL [10].

We acknowledge that writing and maintaining tailoring transformations is a complex task, so we have developed a Higher Order Transformation (HOT) [27] that interactively generates the tailoring transformation [24], allowing practitioners to deal only with familiar concepts. Using an interactive interface, the process engineer creates a Decision Model [28], which is then processed by the HOT, generating

[1]EPF Composer is part of the Eclipse Process Framework Project (http://www.eclipse.org/epf/).

the tailoring transformation. Therefore, several models and transformations must be defined and consistently evolved in order for our MDE approach to process tailoring to be successful.

One way of managing these models and transformations is through a megamodel. A megamodel is a model in itself formed by references to models – including transformation models – and metamodels [3]. A megamodel does not actually contain the modeling artifacts, it is a model that defines the metadata of existing modeling artifacts that reside in a model repository. In this work, we adopt Global Model Management (GMM) [16], the megamodeling approach proposed by the AtlanMod team, which has developed the AtlanMod MegaModel Management (AM3) tool, an Eclipse plugin that implements the GMM approach to megamodeling [17]. This tool has a well-established community of developers, and is fully open-source.

In this paper we describe our approach for automated process model tailoring using a megamodeling as the underlying formalism. We describe each metamodel, model, and transformation involved. We provide an industrial case study for illustrating the megamodel definition and we describe how this definition eases the coordinated evolution of the whole approach.

The rest of the paper is structured as follows. Section 2 describes related work. Then, Sect. 3 presents software process tailoring using a megamodeling approach. An industrial case study is presented in Sect. 4 including the definition of the megamodel and its use in process evolution. Finally, we present some conclusions and further work in Sect. 5.

2. RELATED WORK

A software development process can vary in a number of ways. For example, process elements like tasks, work products and roles may be defined as optional (*opt*), where which elements should be included must be decided before starting each project. More complex types of variabilities can also be required, such as having alternative (*alt*) work products or ways of performing a task.

The specification of these kinds of variabilities is not straightforward, as their specification also depends on how the development process has been formalized. As a result, there are different proposals on how to manage variability. For example, SPEM 2.0 defines four primitives for specifying variability between two process elements of the same type [19]: *contributes*, *replaces*, *extends* and *extends-replaces*. Instances of these relations may override each other, and variability relations must be resolved in the presented order. This complexity is why the SPEM 2.0 variability mechanisms are rarely used in practice [14]. However, if we restrict ourselves to *opt* and *alt* variabilities, both can be directly expressed in SPEM 2.0 – *opt* variability can be specified by setting the *optional* attribute of the corresponding process element, whereas *alt* variability can be modeled using the *replaces* primitive (defining a replacement option for each alternative).

Other process modeling formalisms like VModell-XT [9] are limited in the type of variability they allow. VModell-XT provides a generic process, and before starting a new project, the project manager must decide which modules will be included in the project's development process, based on what work products must be produced. This is akin to modeling process elements as optional. This solution is limited since VModell-XT only provides a generic process, and it does not provide a way of formally specifying alternatives. Moreover, most of the documentation for this approach is in German.

Taking into account the limitations of the SPEM 2.0 variability mechanisms, mainly from the understandability point of view, Martínez et al. [15] proposed vSPEM, a SPEM 2.0 extension based on OVM [21], that allows the direct specification of process variability. In this proposal, the process engineer defines process variation points, as well as variants that can fill the variation points. The relationship between a variation point and its variants is a SPEM 2.0 *replaces* relation, and during process instantiation, each variation point is replaced by exactly one variant. Lack of standardization hinders the adoption of this approach and existing tool support is at an academic prototype level [13], without graphical user interfaces.

To improve the industrial adoption of our work, it is important to use standards, which should be supported by robust tools [26]. Since SPEM 2.0 supports the types of process variability that we have encountered in practice, and existing tool support for it is quite robust, we have decided to model processes along with their variabilities using SPEM 2.0 and EPF Composer following current trends [12].

3. OVERVIEW OF OUR APPROACH

The two main stakeholders in software process tailoring are: 1) the process engineer, who is in charge of defining, evaluating and evolving the process, and 2) the project manager, who follows the tailored process during a project. We provide two user interfaces, one for each stakeholder, that enable their activities and that hide the complexities of the underlying megamodel. Figure 1 shows the architecture of our megamodel-based solution for software process model definition, tailoring and evolution. The megamodel includes models, their corresponding metamodels, and M2M, M2T and T2M transformations, as well as higher order transformations. We now describe the individual components in more detail.

3.1 User Interfaces

Process Engineer's Interface. The company's process engineer is in charge of the Organizational Process Definition. We use the EPF Composer tool for this task, which allows the specification of process models in SPEM 2.0 [19], the OMG standard for organizational process modeling. SPEM 2.0 is based on MOF and is the most popular approach for specifying software processes [12]. The process engineer must also indicate the variation points of the process model.

Project characteristics determine which is the most appropriate process for a project, these characteristics can be modeled as a project "context". We have built a web-based tool for the Organizational Context Model Definition, the process engineer uses this tool for defining the context attributes that affect the variable elements of a process, as well as their potential values [20], generating the Organizational Context XML as output.

The most challenging task the process engineer faces is the Context Affected Elements Definition, i.e., specifying the relationships between context attributes and variable process elements. These relationships define how variability is resolved during the tailoring process. This is done through an interactive tool [25], generating the Context Affected Process

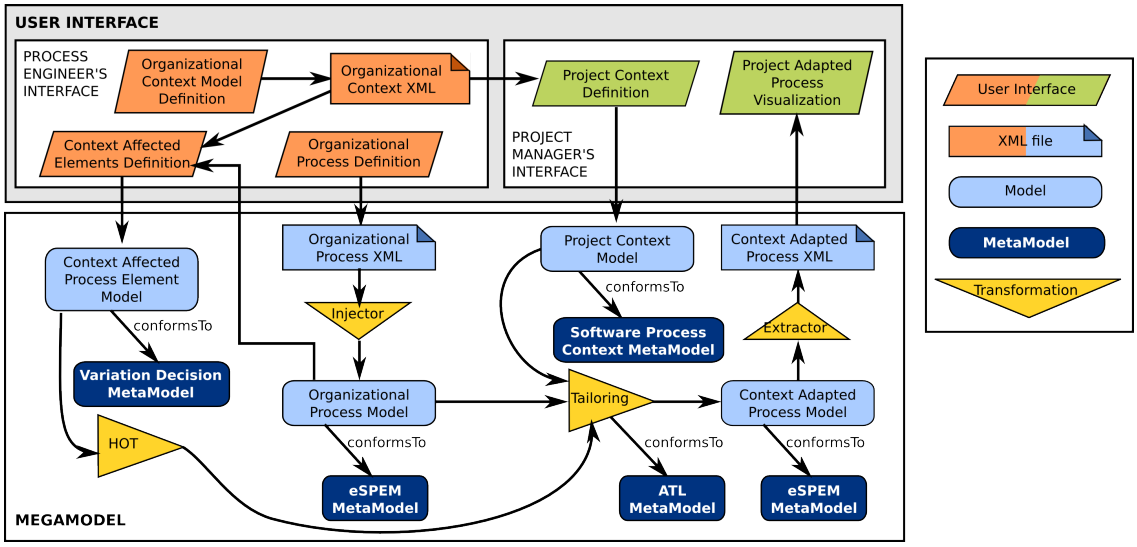

Figure 1: Solution architecture: the megamodel along with the user interfaces

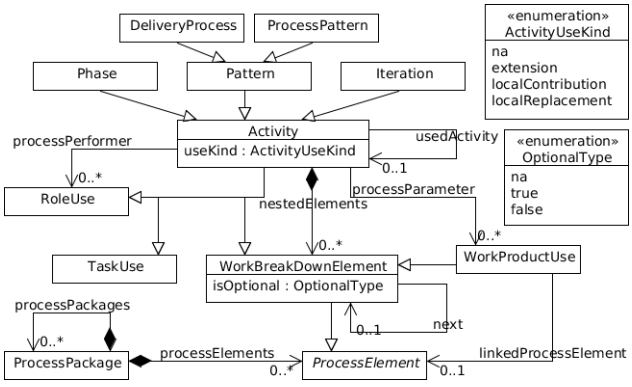

Figure 2: Fragment of the eSPEM MetaModel

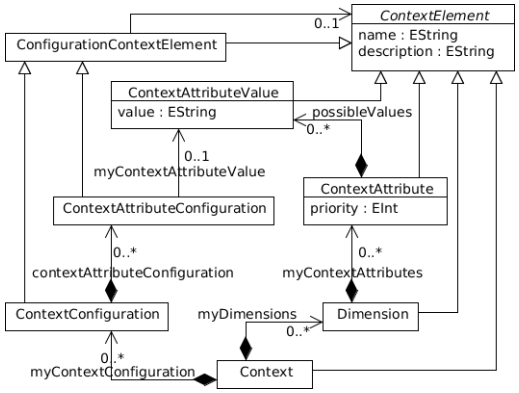

Figure 3: Software Process Context MetaModel (SPCMM)

Element Model.

Project Manager's Interface. The project manager is in charge of the Project Context Definition; this is done using a tool that takes the Organizational Context XML as input and allows her/him to choose the particular values for the attributes of the project at hand, generating the Project Context Model as output. The EPF Composer tool is used for Project Adapted Process Visualization, allowing the project manager to examine the tailored process that will be followed during the project.

3.2 Megamodel

Organizational Process Model. EPF Composer can export the process model as an XML file, but this text file is not a model, and cannot be used as an input for a transformation. To remedy this, we developed an injector [2], a T2M transformation that generates the Organizational Process Model from the XML created by the EPF Composer. This model conforms to eSPEM (see Fig. 2), a subset of SPEM 2.0 that includes the concepts and relations necessary for process tailoring. *TaskUse*, *RoleUse*, and *WorkProductUse* represent potentially variable process elements, they all inherit the *isOptional* attribute from *WorkBreakDownElement*, which is used to specify optionality. Alternatives are modeled through the *usedActivity* association between *Activity* and itself; the *useKind* attribute defines the type of SPEM variability primitive (thus far, we only use the value *replaces*).

Project Context Model. We have defined Software Process Context Metamodel (SPCM) for specifying the context model for each project (see Fig. 3). SPCM has three core concepts: *ContextAttribute*, *Dimension* and *ContextAttributeConfiguration*. Every element in SPCM is a subclass of the *ContextElement* class, which defines *name* and *description* attributes. *ContextAttribute*s represent relevant project context characteristics, and each one is associated to zero or more possible *ContextAttributeValue*s. The Project Context Model is a *ContextConfiguration*, which is a set of *ContextAttributeConfiguration*s, where every *ContextAttributeValue* corresponds to a valid *ContextAttribute*. Therefore, each *ContextAttributeConfiguration* associates a *ContextAttribute* to one possible *ContextAttributeValue*.

Context Affected Process Element Model. The process engineer uses the Context Affected Elements Definition tool

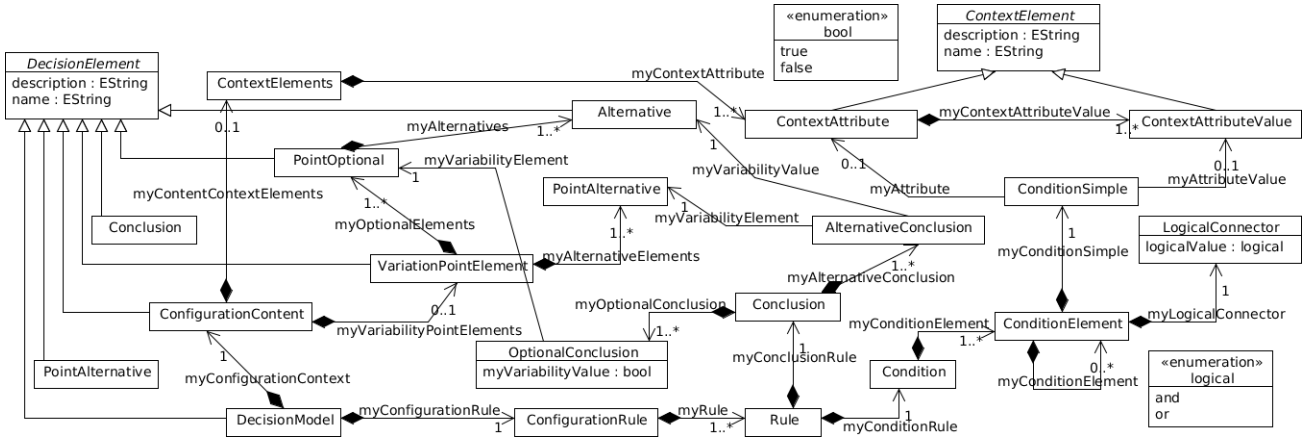

Figure 4: VDMM -Variation Decision MetaModel

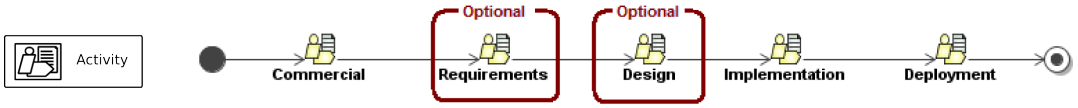

Figure 5: Rhiscom's general software development process.

to establish the relationship between context attribute values and variable process elements as simple and complex rules. These rules are formalized using a domain-specific Decision Model, which conforms to the Variation Decision MetaModel (VDMM) (shown in Fig. 4, and defined in [25]). The VDMM organizes Variability Decision Models (VDM) into three parts: 1) context elements that affect process variability (*ContextElement*), 2) process variation points (*VariabilityPointElement*), and 3) configuration rules that specify how context elements affect process variability (*ConfigurationRule*). Configuration rules have two subcomponents: *Condition* and *Conclusion*. Conditions may be simple (only one predicate), or complex (several predicates joined by logical connectors). This is similar to the work by Weiss on Decision Models [28].

Higher Order Transformation (HOT). Writing tailoring transformations in ATL is not an easy task, even for a process engineer [8], so we built a Higher-Order Transformation (HOT) that automatically generates the process tailoring transformation. This HOT is written in Java [24], and it takes the Context Affected Process Element Model as input and generates the Tailoring transformation as output, hiding the complexity of the task from the process engineer.

Tailoring. The Tailoring transformation takes the Organizational Process Model and a Project Context Model as inputs, and produces a Context Adapted Process Model as output, which also conforms to eSPEM.

Context Adapted Process Model. The tailoring process generates the Context Adapted Process Model, but this model cannot be directly manipulated by the project manager because it is not in a format supported by the tools that she/he uses. To remedy this we built an extractor, a M2T transformation [2] that transforms the Context Adapted Process Model from its XMI format back into its original XML format, so that it can be imported by the EPF Composer for visualization (see Project Adapted Process Visualization).

4. INDUSTRIAL CASE STUDY

We have defined the complete megamodel for Rhiscom, a small Chilean software development company. Rhiscom has around 70 employees and develops software for retail, mainly point-of-sale applications. They started defining their development process about five years ago, and in the past two years they have moved towards process formalization and automated tailoring. Figure 5 shows the general software development process followed by Rhiscom, and Fig. 6 shows the detail of the *Requirements* activity. Variation points in these figures are highlighted as a visual aid for the reader, this is not the standard graphical notation in EPF Composer.

4.1 Megamodel Definition

Organizational Process Model. Figure 7 shows Rhiscom's *Requirements* activity modeled in EPF Composer. In Fig. 6, the *Specify Requirements* task was marked as having alternatives, these alternatives can be seen in Fig. 7: *Specify Requirements in Use Cases* and *Specify Requirements in plain text*. Internally, these alternatives are linked to the task they can *replace*.

Project Context Model. Figure 8 shows two project contexts: (a) "Maintenance-Adaptation" and (b) "New Development". The *ContextConfiguration* for each context is highlighted in Fig. 8, both "Properties" tabs show the attribute value corresponding to the "Project Type" attribute for both *ContextConfiguration*s. Note that although the Organizational Context Model also has the "Team Size" attribute, it was not considered in either configuration.

Context Affected Process Element Model. Figure 9 shows the interface used by the process engineer to define the Context Affected Process Element Model, the model generated for tailoring the Rhiscom process is shown in Fig. 10. The *ConfigurationContent* includes the *ContextElement*s defined as part of the Organizational Context Model, as well as the vari-

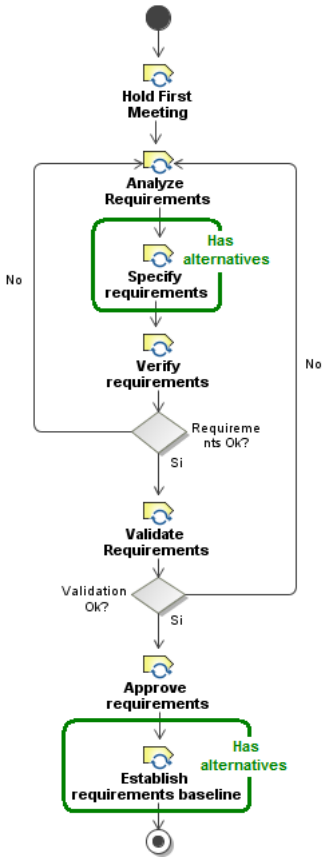

Figure 6: Rhiscom's requirements activity.

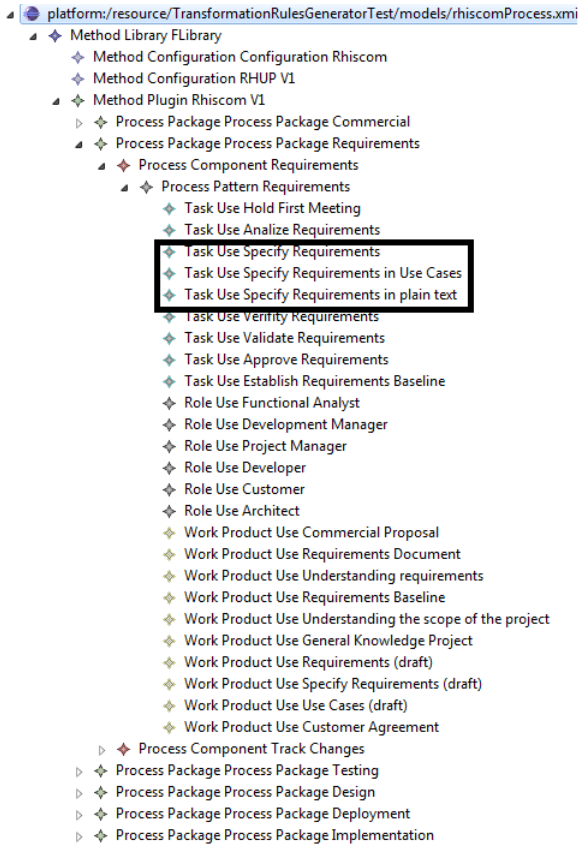

Figure 7: Rhiscom's requirements process model.

able process elements of the Organizational Process Model. Among these latter ones we can see the *Requirements* activity, that we have identified as optional. As part of the *Configuration Rule* there is a simple condition stating that the *Requirements* activity must be omitted when the value of "Project Type" is "Maintenance-Adaptation".

Higher Order Transformation. The code fragment shown in Fig. 11 processes the input from Fig. 9, generating helper rules for *opt* and *alt* variation points, like the ones seen Fig. 12. These rules, along with some bootstrapping rules, make up the tailoring transformation.

Tailoring. The tailoring transformation defines which optional process elements should be included in the adapted process, as well as which alternatives should be implemented (in the case of alternative variation points). Figure 12 shows two helper rules that are part of Rhiscom's tailoring transformation: `ruleOpt1`, which implements one of the rules involving the *opt*ional *Requirements* activity, and `ruleAlt2`, which implements one of the rules involving the *Establish Requirements alt*ernative variation point. For example, `ruleOpt1` states that if "Project Type" is "Maintenance-Adaptation", "Project Duration" is "Medium" and "Business Knowledge" is "Known", then the corresponding process element (in this case, the *Requirements* activity) will be omitted (*false*) from the adapted process, otherwise it will be included (*true*). On the other hand, `ruleAlt2` states that if "Project Type" is "New-Development", "Project Duration" is "Medium", and "Business Knowledge" is "Unknown", then the *Establish Requirements Baseline and Test Cases* task should replace the *Establish Requirements* task in the development process.

Context Adapted Process Model. After applying the tailoring transformation to the *Maintenance-Adaptation* context model, we obtain the Context Adapted Process Model shown in Fig. 13. Here we see that the *Requirements* activity has been omitted, as defined by the Tailoring transformation. If the *New-Development* context had been used as input instead, this activity would not have been removed.

4.2 Megamodel Evolution

4.2.1 *Software Process Management*

Software process definition using EPF Composer is labor-intensive, even for a small process. So taking advantage of this investment is appealing. Changes such a 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. Managing the software process as a model supported by a a megamodel enables these changes. For example, if the template assigned to the *Requirements Document* work product needs to be changed, it can be done directly in the EPF Composer and then the injector needs to be rerun. The updated Organizational Process Model 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 such as AVISPA [6], we

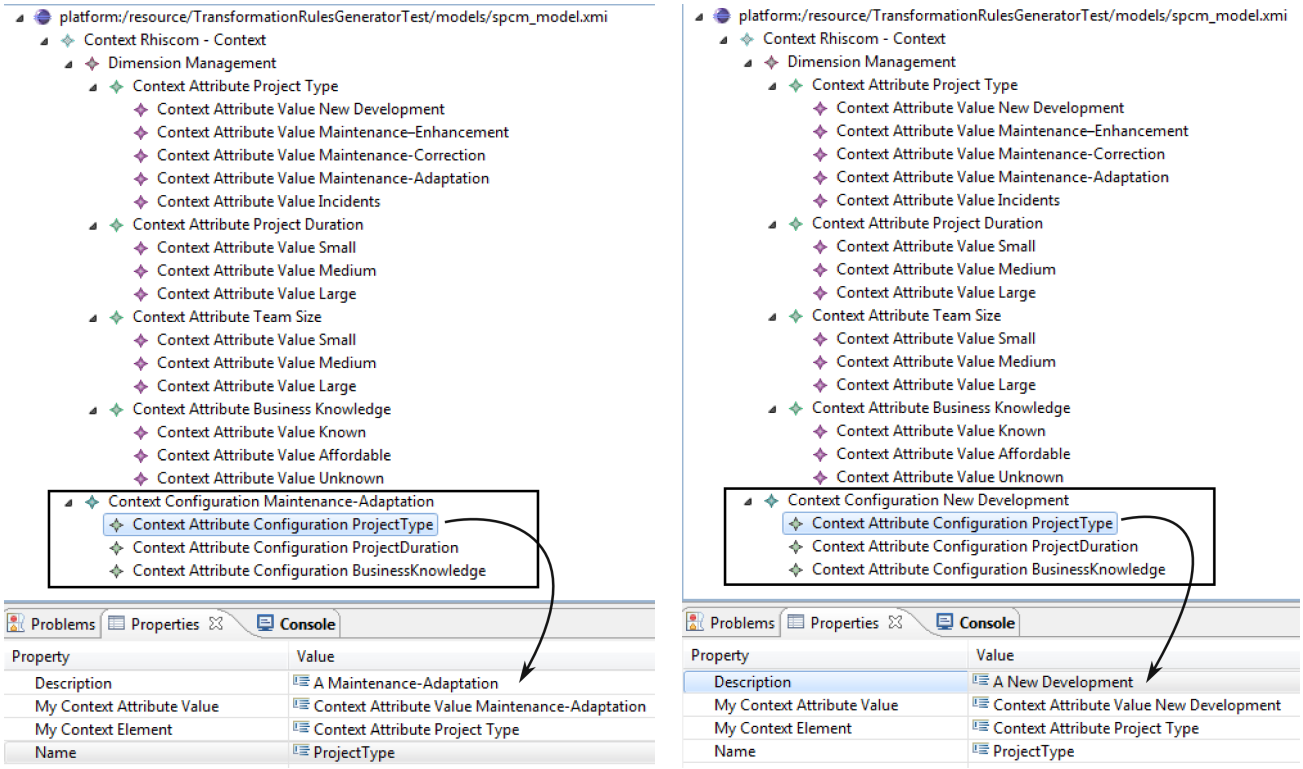

Figure 8: Context models for (a) "Maintenance-Adaptation" and (b) "New Development".

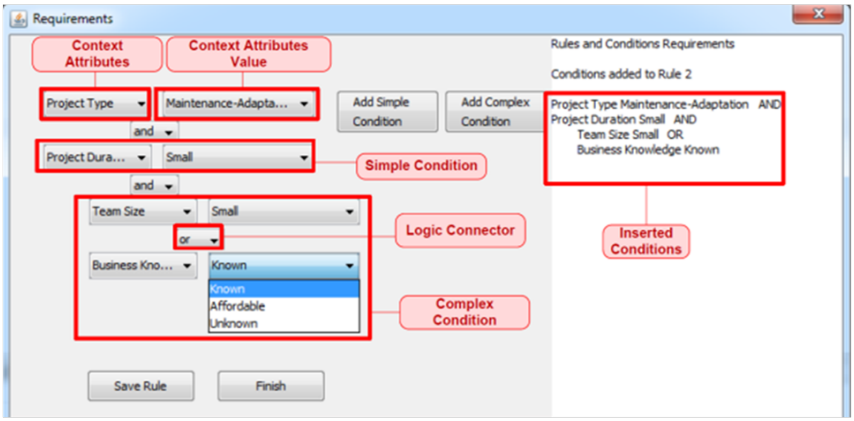

Figure 9: Context Affected Element Definition

can determine whether there is an overloaded role that may cause inefficiencies, or whether there is a role that does not interact with other roles. 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.

4.2.2 *Context Attribute Evolution*

Context values vary for each project. In each case the Project Context Definition should be executed in order to regenerate the Project Context Model. Then, the Tailoring transformation can be executed in order to obtain the Context Adapted Process Model. This kind of change is not dramatic, but is quite frequent, so tool support is essential.

On the other hand, the Organizational Context XML may require drastic changes such as adding, modifying or deleting context attributes and/or their values. In these cases, the Organizational Context Model Definition needs to be rerun so that a new Organizational Context XML is generated; this model will be used for assigning context values for all future projects. Moreover, as the Organizational Context XML is input to the Context Affected Elements Definition, it may be necessary to regenerate the Context Affected Process Element Model. If this model changes, the HOT must also be rerun, so that a new version of the Tailoring transformation

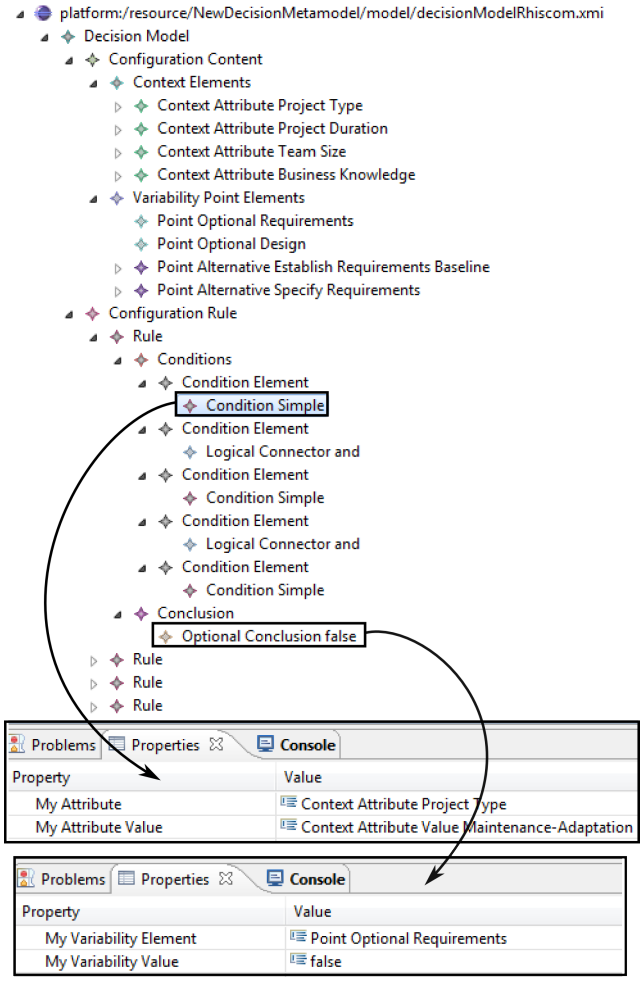

Figure 10: Fragment of the Context Affected Process Element Model.

is generated.

4.2.3 *Tailoring Rule Evolution*

Tailoring rules should evolve when the Context Adapted Process Model generated by the tailoring process is not as expected. If the problem is in the Organizational Process Model, then the procedure described in Sect. 4.2.1 must be followed; if it is in the Organizational Context XML, then the procedure in Sect. 4.2.2 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 Context Affected Elements Definition should be rerun and a new Context Affected Process Element Model must be generated. Finally, the HOT must be rerun generating the new Tailoring transformation.

```
public void saveConditionsOfRule(DecisionModel decisionModel,
      RulesTransformation ruleTransformation[], String titleGui,
      int ruleCounter){

   // add a new rule to the decision model
   decisionModel.addNewRule();
   decisionModel.addRuleToDecisionModel();
   ...

   // update the rule conditions for existing rules
   for(int i=1;i<addArrayConditionCounter-1;i++){
      // update the corresponding predicate
      decisionModel.addSimpleConditionToRule(
         attributeComboBox[i]....);
      // .. and its logical connector
      decisionModel.addLogicalConnectorToRule(
         logicConnector[i]....);

      // update the rule's condition
      ruleTransformation[ruleCounter].setRule(i,
         attributeComboBox[i].., attributeValueComboBox[i]..);
   }

   // add the new condition to the decision model
   decisionModel.addSimpleConditionToRule(
      attributeComboBox[addArrayConditionCounter-1]...,
      attributeValueComboBox[addArrayConditionCounter-1]...);

   // set the new rule's condition attribute
   ruleTransformation[ruleCounter].setRule(
      addArrayConditionCounter-1,
      attributeComboBox[addArrayConditionCounter-1]...,
      attributeValueComboBox[addArrayConditionCounter-1]....);
}

public void saveConclusionOfRule(
      RulesTransformation ruleTransformation[], int ruleCounter,
      String conclusionValueExt){

   // set the new rule's conclusion attribute
   ruleTransformation[ruleCounter].setConclusionValue(
      conclusionValueExt);
}
```

Figure 11: Fragment of the HOT.

```
helper def:ruleOpt1():Boolean =
   if (thisModule.getValue('Project Type')
         = 'Maintenance-Adaptation'
      and thisModule.getValue('Project Duration') = 'Small'
      and thisModule.getValue('Business Knowledge') = 'Known')
   then false
   else true
endif;

helper def:ruleAlt2(tu:MM!TaskUse): MM!TaskDefinition =
   if (thisModule.getValue('Project Type') = 'New-Development'
         and thisModule.getValue('Project Duration') = 'Medium'
         and thisModule.getValue('Business Knowledge')
            = 'Unknown')
   then thisModule.getTaskDefinition(
            'Establish Requirements Baseline and Test Cases')
   else thisModule.getTaskDefinition(tu.name)
endif;
```

Figure 12: Fragment of the Tailoring Transformation.

5. CONCLUSIONS

In this paper, we show that megamodeling is a feasible solution for the definition and evolution of the model-based elements involved in automated software process tailoring. In this approach, the process engineer defines a general software process, as well as variation points (optionality and alternatives), an organizational project context model, and a decision model that establishes the relationship between the two, whereas the project manager must only define the context of the project she/he is working on. A series of M2M, T2M and M2T transformations automate the tailoring process, hiding the megamodel from our users.

We illustrated the usefulness of our approach by defining the tailoring megamodel for a small Chilean company,

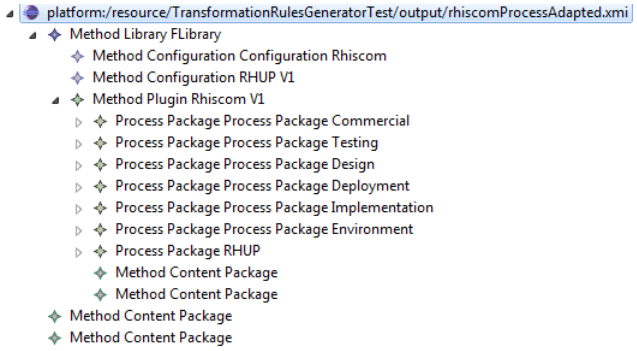

Figure 13: Adapted process models for "Maintenance-Adaptation"

which can now be used to generate new adapted processes. We also described how typical changes to the general software process, the organizational context model and the tailoring rules are now simple to carry out, since these changes prompt user-interactions or trigger transformations in order to maintain a consistent megamodel.

Megamodeling is an appealing approach for structuring large MDE-based applications, and our initial results are promising. However, tools for supporting megamodeling are not as stable and supported as we would like them to be, e.g., AM3 is only available through svn and it only runs with an older version of Eclipse. We expect this to be an accidental drawback of the approach.

6. REFERENCES

[1] R. Alarcón and P. Barceló, editors. *Proc. of SCCC '12*. IEEE Computer Society, 2012.

[2] A. Bertero, L. Silvestre, and M. C. Bastarrica. Text-to-Model and Model-to-Text Transformations between Software Processes and Software Process Models. In Alarcón and Barceló [1].

[3] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. Modeling in the Large and Modeling in the Small. In *ECMFA '03*, pages 33–46, 2003.

[4] A. Cockburn. Selecting a Project's Methodology. *IEEE Software*, 17(4):64–71, July 2000.

[5] D. Firesmith. Creating a Project-Specific Requirements Engineering Process. *J. of Object Technology*, 3(5):31–44, 2004.

[6] J. A. Hurtado Alegría, M. C. Bastarrica, and A. Bergel. AVISPA: A Tool for Analyzing Software Process Models. *J. of Softw. Evolution and Process*, 2013. doi: 10.1002/smr.1578. To appear.

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

[8] J. A. Hurtado Alegria, M. C. Bastarrica, A. Quispe, and S. F. Ochoa. An MDE approach to software process tailoring. In *ICSSP '11*, pages 43–52, 2011.

[9] IABG. Das VModell-XT, 2004. `http://v-modell.iabg.de/index.php?option=com_frontpage&Itemid=1`. Accessed Nov. 2013.

[10] F. Jouault and I. Kurtev. Transforming Models with ATL. In J.-M. Bruel, editor, *Satellite Events at the MoDELS '05*, volume 3844 of *LNCS*, pages 128–138, 2005.

[11] P. Kruchten. *The Rational Unified Process: An Introduction*. Object Technology Series. Addison-Wesley Professional, third edition, 2003.

[12] M. Kuhrmann, D. M. Fernández, and R. Steenweg. Systematic software process development: where do we stand today? In Münch et al. [18], pages 166–170.

[13] T. Martínez-Ruiz, F. García, M. Piattini, and F. D. Lucas-Consuegra. Process variability management in global software development: a case study. In Münch et al. [18], pages 46–55.

[14] T. Martínez-Ruiz, F. García, M. Piattini, and J. Münch. Modelling software process variability: an empirical study. *Software, IET*, 5(2):172–187, 2011.

[15] T. Martínez-Ruiz, J. Münch, F. García, and M. Piattini. Requirements and constructors for tailoring software processes: a systematic literature review. *Software Quality Journal*, 20(1):229–260, 2012.

[16] ModelPlex Project. Deliverable D2.1.a: Global Model Management Principles. `http://docatlanmod.emn.fr/AM3/Documentation/D2-1-a\_Global\_Model\_Management\_Principles\_v1-1.pdf`, March 2008. Accessed Nov. 2013.

[17] ModelPlex Project. Deliverable D2.1.b: Global Model Management Supporting Tool. `http://docatlanmod.emn.fr/AM3/Documentation/D2-11-b\_Global\_Model\_Management\_Supporting\_Tool\_v3-0.pdf`, October 2008. Accessed Nov. 2013.

[18] J. Münch, J. A. Lan, and H. Zhang, editors. *Proc. of ICSSP '13*. ACM, 2013.

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

[20] D. Ortega, L. Silvestre, M. C. Bastarrica, and S. Ochoa. A Tool for Modeling Software Development Contexts. In Alarcón and Barceló [1].

[21] K. Pohl, G. Böckle, and F. J. Linden. *Software Product Line Engineering*. Springer, 2011.

[22] C. Rolland. Method Engineering: State-of-the-Art Survey and Research Proposal. In *SoMeT'09*, pages 3–21. IOS Press, 2009.

[23] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

[24] L. Silvestre, M. C. Bastarrica, and S. F. Ochoa. HOTs for Generating Transformations with Two Input Models. In *SCCC '13*, 2013.

[25] L. Silvestre, M. C. Bastarrica, and S. F. Ochoa. A Model-Based Tool for Generating Software Process Model Tailoring Transformations. In *To appear in MODELSWARD '14*, January 2014.

[26] J. Simmonds, M. C. Bastarrica, A. Quispe, and L. Silvestre. Variability in Software Process Models: Requirements for Adoption in Industrial Settings. In *PLEASE '13*, pages 33–36, May 2013.

[27] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the Use of Higher-Order Model Transformations. In R. F. Paige, A. Hartman, and A. Rensink, editors, *ECMDA-FA*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2009.

[28] D. M. Weiss, J. J. Li, J. H. Slye, T. T. Dinh-Trong, and H. Sun. Decision-Model-Based Code Generation for SPLE. In *SPLC*, pages 129–138. IEEE Computer Society, 2008.