A Model-based Tool for Generating Software Process Model Tailoring Transformations

Luis Silvestre, María Cecilia Bastarrica and Sergio F. Ochoa
Computer Science Department, Universidad de Chile, Santiago, Chile

Keywords: Software Process Tailoring, Model Transformation, Generative Approach, Model-based Tool.

Abstract: Tailoring is the mechanism of adapting a software process to the needs of a project. Model-Driven Engineering (MDE) provides a formal basis and tools infrastructure for automatic software process tailoring. However, the use of a MDE approach can become awkward for most process engineers, because it requires knowledge of MDE concepts and formalisms to implement the required models and tailoring transformations. Proposals trying to address this problem should balance the formality required by MDE and the usability needed by the users. This paper presents a model-based tool and its associated procedure that allow process engineers to automatically generate tailoring transformation rules using a graphical user-interface, obtaining the desired balance. The proposal is illustrated with a running example.

1 INTRODUCTION

Software process tailoring is the adaptation of a software process so that it is adjusted to the needs of a particular project. There are a variety of approaches to address tailoring, ranging from self-emerging processes as in XP (Beck, 1999), to template-based tailoring as in Crystal methodologies (Cockburn, 2000), and ultimately automatic software process tailoring based on MDE techniques (Bendraou et al., 2010) (De Oliveira Barros et al., 2002) (Hurtado et al., 2013).

MDE promotes building software by defining software models and successively refining them through formal model transformations (Schmidt, 2006). In this way, this approach has allowed software modeling at different abstraction levels and addressing different application domains (Kleppe et al., 2003). However, all this power requires mastering new concepts and formalisms relating model definition and writing model transformations in specific languages. These complexity issues have sometimes prevented these techniques to be applied in industrial settings.

MDE-based tailoring considers software processes as models, and process tailoring as model transformations. Model transformations are programs that generate one or more output models from one or more input models. Transformations may be written in general purpose languages such as Java or C++, but transformation-specific languages such as ATL (AtlanMod Group, 2006) or QVT (OMG, 2001) provide higher abstraction level constructs for writing transformations.

For software process tailoring, the tailoring transformation takes the organizational process model including its variability, and the project context model as input, and generates the project adapted process model. For each variable process element in the process model, there will be a rule in the transformation that determines if it is to be included or not (for optional elements), or which realization of the process element should be included (for alternative elements), according to the values of the project context model attributes.

Generating appropriate tailoring transformations requires two different kinds of knowledge. On the one hand, the process engineer, who is in charge of this activity, should know precisely how the context attribute values impact the process variation. On the other hand, she/he should be able to write the model transformation, mastering the syntax and semantics of the transformation language used to implement the tailoring rules. The company's process engineer usually has the first kind of knowledge (i.e., how to tailor the process), but she/he is almost never experienced in the use of transformation languages and MDE concepts. While it has been shown that it is technically feasible to apply MDE to tailor software process models, the complexity of this

Silvestre L., Bastarrica M. and Ochoa S..
A Model-based Tool for Generating Software Process Model Tailoring Transformations.
DOI: 10.5220/0004715805330540
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 533-540
ISBN: 978-989-758-007-9
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

solution limits its use in the software industry.

To address this challenge, we present a model-based tool to automatically generate tailoring transformation rules through a generative approach. This tool, that we have called Architect of Tailoring Rules (ATR), allows process engineers to interactively define rules using a graphical user interface, taking advantage of the formality provided by MDE but hiding its inherent complexity. Therefore, the process engineer can define transformation rules to tailor the organizational software process, only by selecting on a graphical user interface the values of project context attributes that impact variable process elements.

The rest of the paper is structured as follows. Next section presents and discusses the related work. Section 3 presents the general strategy used to tailor software processes, and shows how the proposed tool contributes to such an activity. The proposed model-based tool is described in Sect. 4, along with its main components. Finally, conclusions and future work are presented in Sect. 5.

2 RELATED WORK

There are several kinds of model transformations according to different criteria: declarative or imperative; in-place or new-target; deterministic, non-deterministic, or interactive (Czarnecki et al. 2006). However, building the appropriate model transformation requires expertise for choosing the right kind of transformation, and also for mastering the transformation language syntax and semantics. These knowledge-gap barriers are partly addressed by transformation-by-example techniques (Kappel et al. 2012). Writing model transformations is usually difficult, and the required knowledge for writing any kind of transformation is not generally available for process engineers, that are the people in charge of process design and tailoring.

MOLA (Kalnins et al., 2004) allows specifying transformation rules through visual mapping patterns. Similar to GREaT (Balasuramanian et al., 2006), MOLA specifies rules and mappings using class diagrams, but considering an environment inspired in activity diagrams. Both works define the possibility of establishing relationships between metamodel attributes and elements. A limitation of MOLA and GREaT is that they need the user to directly interact with metamodels and class diagrams, which still represents a strong restriction for process engineers in terms of usability.

Varró and Balogh, through the VIATRA framework (Varró et al., 2002), provide a text-based rule editor. Although this proposal is supported by Eclipse, it does not provide an easy-to-use environment that can be used by process engineers for defining tailoring rules.

There are also some recent proposals such as MTBE (Model Transformations By Example) (Wimmer et al., 2007) (Varró and Balogh, 2007) and MTBD (Model Transformation By Demonstration) (Sun et al., 2009) that present innovative solutions for simplifying the implementation of model transformations, by using strategies and patterns with a visual support. These strategies generate part of the code required for the model transformations, however, the process engineer still needs to understand and complete such a code. Therefore, this represents a semi-automatic process to generate model transformation rules.

Hurtado et al. (Hurtado et al., 2013) present a proposal that generates an adapted process model from a general process model, which is tailored according to a context model that specifies the characteristics of a particular project. The tailoring transformation is written in ATL (AtlanMod Group, 2006) and its rules consider information from the project context model to decide the elimination or not, or the choice of variable elements in the general software process model. This proposal demonstrates the feasibility of the MDE-based tailoring approach, but rules still need to be directly written.

These proposals highlight the need for developing new solutions for simplifying rule definition while still providing all the expressive power required for software process model tailoring. Software process tailoring is only an example of the application scenarios that are not well supported by the current way of developing model transformations. This knowledge gap has lately been addressed by new proposals such as Domain-specific transformation languages (Rumpe et al., 2011).

3 SOFTWARE PROCESS TAILORING STRATEGY

Figure 1 shows the general architecture of the MDE-based process tailoring. This approach requires two input models: an *organizational software process model* that conforms to the eSPEM (experimental SPEM) metamodel that is a subset of SPEM (Software Process Engineering Metamodel) (OMG, 2008a) and a *project context model* that is an instance of the Organizational Context Model.

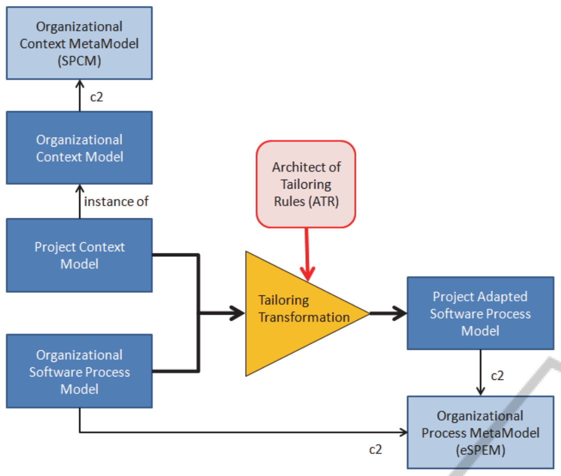

Organizational Context MetaModel (SPCM)
c2
Organizational Context Model
instance of
Project Context Model
Organizational Software Process Model
Architect of Tailoring Rules (ATR)
Tailoring Transformation
Project Adapted Software Process Model
c2
Organizational Process MetaModel (eSPEM)
c2

Figure 1: MDE-based software process tailoring.

This proposal uses a model-to-model transformation to generate a *project adapted software process model* as output. The resulting process model also conforms to eSPEM.

The *organizational software process model* has been defined using the Eclipse Process Framework Composer (EPFC), along with its variabilities (Simmonds et al., 2013). This tool has been well received by software companies' process engineers. However, the process, as specified in EPFC, conforms to the UMA (Unified Method Architecture) metamodel in its internal representation, and therefore the tool exports an xml file that cannot be directly used as input for the tailoring transformation. Therefore, an injector has been built for converting the process representation between formats obtaining an *organizational software process model* in xmi format and conforming to eSPEM as needed.

The *organizational context model* indicates the project attributes that may influence the process tailoring along with their potential values. A *project context model* is an instance of this organizational context model. The organizational context model is defined using Eclipse Modeling Framework (EMF) and conforming to the SPCM (Software Process Context Metamodel) metamodel (Hurtado et al., 2013).

The tailoring transformation in this proposal is written in ATL. For each variable element identified as part of the organizational process, there is a rule included in the transformation. For optional process elements, the rule decides, according to the values in the project context model attributes, if it should be included or not in the adapted process. For process elements defined with alternatives, the rule decides which of them will realize the process element in the adapted process. Even though this strategy seems quite clear, translating it into ATL rules is a challenging task.

Although this tailoring proposal has shown to be technically feasible in real scenarios, it clearly has important limitations when process engineers have to use it. For that reason we have developed the Architect of Tailoring Rules (ATR), a tool that allows process engineers to interactively define the process tailoring rules using a graphical user interface. The tool's output is the tailoring transformation that can be used to adapt the organizational software process (Fig. 1). The following section describes the proposed model-based ATR tool, along with is associated procedure.

4 MODEL-BASED TOOL FOR GENERATING TAILORING TRANSFORMATIONS

Figure 2 presents the architecture of ATR, the model-based tool that allows the automatic generation of the tailoring transformation.

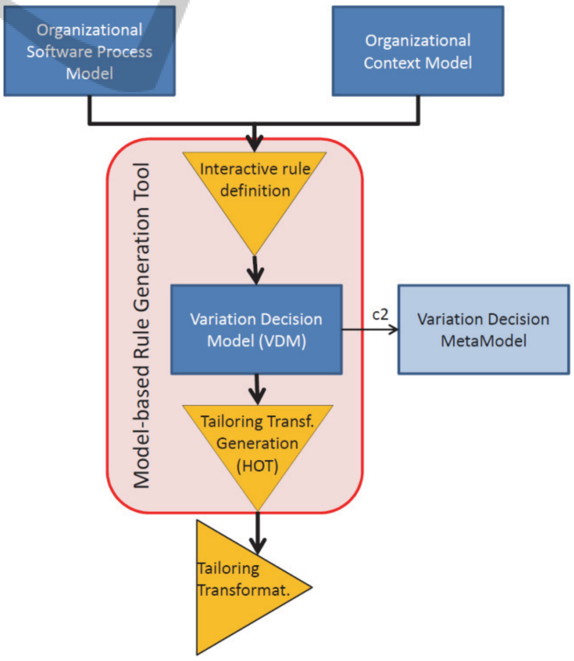

Organizational Software Process Model
Organizational Context Model
Model-based Rule Generation Tool
Interactive rule definition
Variation Decision Model (VDM)
c2
Variation Decision MetaModel
Tailoring Transf. Generation (HOT)
Tailoring Transformat.

Figure 2: Architecture of the proposed heuristic.

The ATR tool uses the *organizational software process model* as an input because such a model contains the information about the variable process elements for which the process engineer must define tailoring rules. The tool also uses the *organizational* *context model* as an input, because the conditions of the tailoring rules are defined according to the values of the attributes in this context model.

The process engineer uses a visual interface to indicate the models that will be used in the definition of tailoring rules (Fig. 3). After that, she/he can define tailoring rules for each process variation point. This activity involves two steps: the interactive definition of a decision model (using the visual user interface) and the automatic generation of the tailoring transformation, based on the previously built decision model.

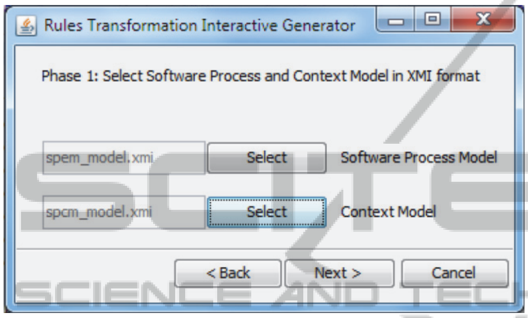

Rules Transformation Interactive Generator
Phase 1: Select Software Process and Context Model in XMI format
spem_model.xmi
Select
Software Process Model
spcm_model.xmi
Select
Context Model
< Back
Next >
Cancel

Figure 3: Interface for input models selection.

During the first step, the process engineer uses the ATR tool to interactively define the relationships between the context attribute values and process variable elements yielding a Variation Decision Model (VDM). This VDM is a high-level representation of the transformation rules. The VDM is then used as input for a Higher Order Transformation (HOT) to automatically generate the tailoring transformation that will be used to adapt the organizational software process model.

Thus, the proposed tool allows process engineers to apply MDE concepts to generate tailoring transformations, hiding the inherent complexity of such concepts. Their complexity is encapsulated mainly in the VDM and the HOT.

Next sections describe each component of the tool. In order to illustrate its capabilities, we will use, as a running example, the generation of the tailoring rules for Rhiscom's process. Rhiscom is a medium-sized software company that develops software for the retail industry. It has around 70 employees and offices in four Latin-American countries. This company has a software process formalized in SPEM 2.0.

4.1 Interactive Definition of the Variation Decision Model

Once the process engineer has specified the models that will be used as input, she/he can start with the interactive definition of the variation decision model. Figure 4 shows five variation points for Rhiscom's process: *requirements*, *environment definition*, *environment checklist*, *requirement specification* and *design*. If the user selects a variation point (e.g. requirements) and clicks on the "Create Rules" button, she/he can define the rules that will be used to tailor the organizational process in such a point, depending on the values of the context attributes of a specific project.

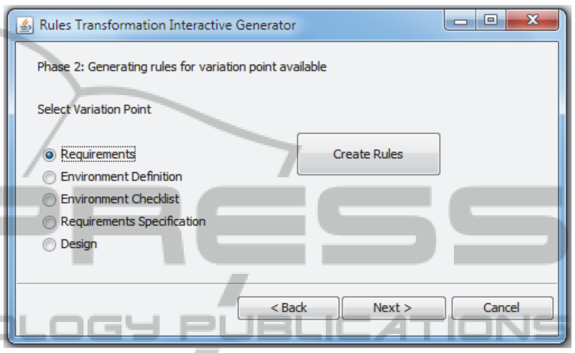
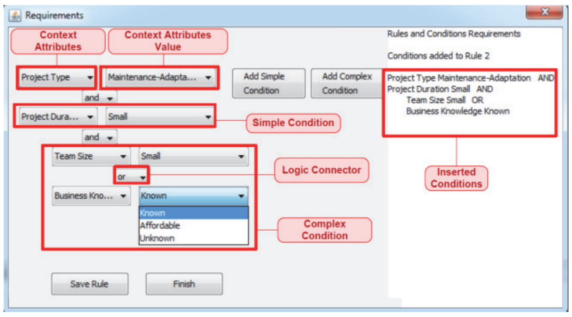

Rules Transformation Interactive Generator
Phase 2: Generating rules for variation point available
Select Variation Point
Requirements
Environment Definition
Environment Checklist
Requirements Specification
Design
Create Rules
< Back
Next >
Cancel

Figure 4: Selection of process variation points.

Figure 5 shows the interactive interface that allows the process engineer to define the decision model. Each decision has a *condition* and a *conclusion*. The condition is a predicate that could be simple or complex. Simple predicates are typically a list of context attributes linked to particular values through a logical operator. Complex conditions consider the use of predicates connected through logical connectors. In the right part of the figure we can see the conditions defined so far.



Rules and Conditions Requirements

Conditions added to Rule 2

Project Type Maintenance-Adaptation AND
Project Duration Small AND
Team Size Small OR
Business Knowledge Known

Figure 5: Interactive Interface.

In this example, the engineer defines that the Requirements activity should not be included when the *project type* is "Maintenance-Adaptation", when the *project duration* is "Small" and also when the *team size* is "Small" or *business knowledge* is "Known". These decisions are part of the adaptations defined by Rhiscom for its organizational process. Table 1 shows other adaptations also considered for such a process.

Table 1: Rules for Rhiscom's tailoring transformation.

5	Context Attribute	Context Attribute Value	Conclusion
Decision 1	Project type	Maintenance-Correction	Remove: *Environment Definition*
	Project duration	Small	
Decision 2	Project type	Maintenance-Adaptation	Remove: *Requirements*
	Project duration	Small	
	(Team size = Small) **or** (Business Knowledge = Known)		
Decision 3	Project type	Incidents	Remove: *Environment Checklist*
	Team size	Small	

In the "context attribute" column of Table 1, we can see four context attributes that influence the process tailoring. For optional process elements, the decision establishes, according to the values in the "context attribute value" column, if it should be included or not in the adapted software process model. We can also see in the "conclusion" column the process element to be removed according to each decision. We will use decision 2 as an example to show how to derive a transformation rule from it.

Next section describes the variation decision model in detail.

4.2 Variation Decision Model

The variation decision model is generated by the interactive rule definition of the ATR tool. The tool establishes relationships between context attribute-values and variable process elements. Then, ATR automatically builds a VDM, which is used as an input for the HOT that actually generates the tailoring transformation.

Decision models have been used in software product lines for establishing the conditions for configuring particular products of the line. There is no standard formalization for these models. In this work we provide a formal metamodel: the Variation Decision MetaModel (VDMM), as shown in Fig. 6.

VDMM organizes a VDM in two main parts: configuration content and configuration rule, similar to the work of Weiss on Decision Models (Weiss et al., 2008) and the work on Semantics of Business.

Vocabulary and Business Rules (SMVB), used

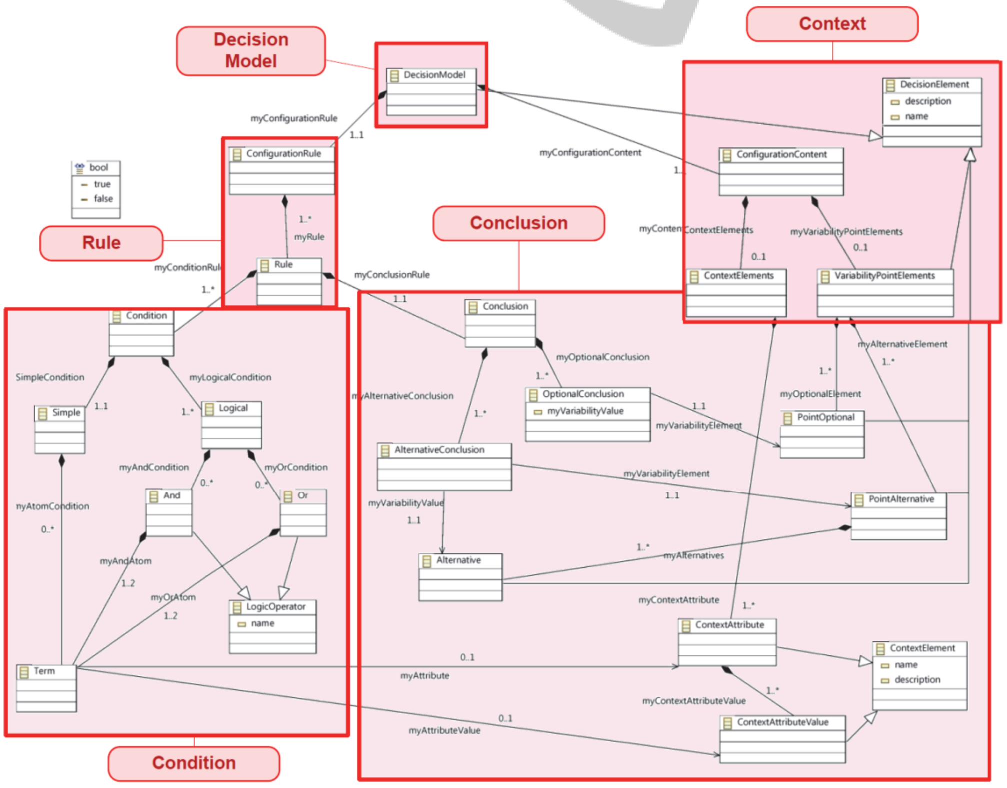
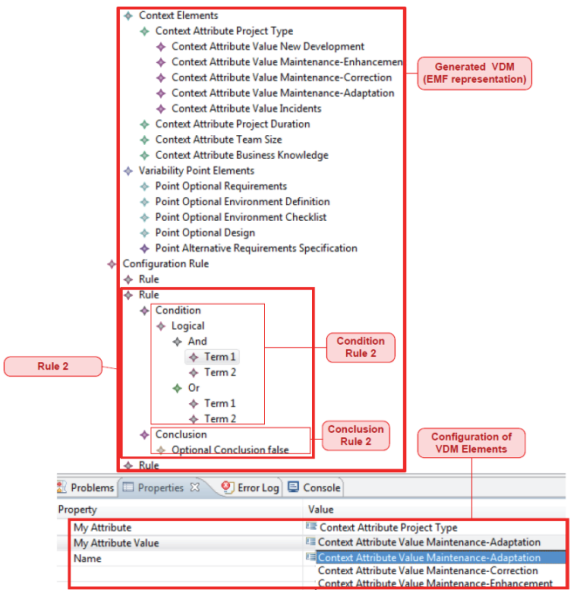


Figure 6: Variation Decision MetaModel.

for building decision rules (OMG, 2008). The configuration content's goal is to incorporate specific information that will be needed for modeling decisions including ContextElements and VariabilityPointElements. The configuration rule defines the tailoring rules using domain concepts. They have two subcomponents: Condition and Conclusion. Conditions may be simple (just one condition), or complex (several conditions with logical connectors). According to the literature on decision models, conditions should have a left and a right side that in our case are called myAttribute (left) and myAttributeValue (right). Similarly, the conclusion must also have a left and a right side; in our case they are myVariabilityElement (left) and myVariabilityValue (right).

Figure 7 shows the VDM generated through the interactive interface. We can see that the Configuration Content is formed by the Context Elements and the Process Variability Point Elements. On the other hand, in the Configuration Rule we can see that Term 1 is highlighted and in the lower part we can see that the Project Type attribute has been assigned the Maintenance-Adaptation value, as stated in Fig. 5. As part of this rule conclusion, Requirements is set to False.



```
Context Elements
    Context Attribute Project Type
        Context Attribute Value New Development
        Context Attribute Value Maintenance-Enhancemen
        Context Attribute Value Maintenance-Correction
        Context Attribute Value Maintenance-Adaptation
        Context Attribute Value Incidents
    Context Attribute Project Duration
    Context Attribute Team Size
    Context Attribute Business Knowledge
Variability Point Elements
    Point Optional Requirements
    Point Optional Environment Definition
    Point Optional Environment Checklist
    Point Optional Design
    Point Alternative Requirements Specification
Configuration Rule
Rule
Rule
    Condition
        Logical
            And
                Term 1
                Term 2
            Or
                Term 1
                Term 2
    Conclusion
        Optional Conclusion false
Rule
```

Property	Value
My Attribute	Context Attribute Project Type
My Attribute Value	Context Attribute Value Maintenance-Adaptation
Name	Context Attribute Value Maintenance-Adaptation
	Context Attribute Value Maintenance-Correction
	Context Attribute Value Maintenance-Enhancement

Figure 7: Variation Decision Model Instance.

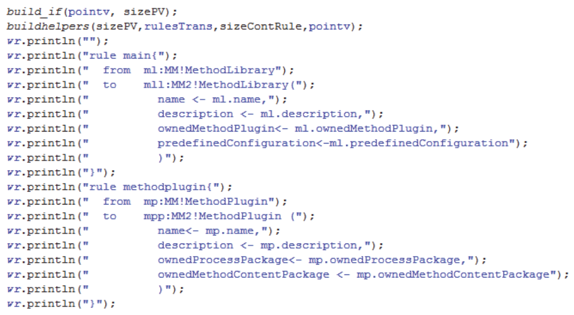
4.3 HOT for Generating Tailoring Transformations

Provided that model transformations can also be considered as models conforming to their language metamodel (Bézivin et al., 2006), a Higher-Order Transformation (HOT) is a transformation in itself, but it either takes a transformation model as input or generates a transformation model as output (Tisi et al., 2010). We use a HOT to generate the tailoring transformation, thus avoiding writing it directly. Our HOT takes the VDM previously built as input, and its output is the desired process tailoring transformation.

There are two approaches for building HOTs: model-to-model (M2M) and model-to-text (M2T) transformations. We choose M2T and therefore the output is the ATL source code of the tailoring transformation.

To build a M2T transformation it is possible to use transformation-specific or general-purpose languages. Transformation-specific languages for this task like ATL, Acceleo and MOFScript provide transformation-specific abstractions. However, we have decided to use a general purpose language such as Java to build the HOT, at least for the first version, because it is a mature language that is easily mastered by developers (Silvestre et al., 2013). A final version of the HOT will be probably implemented in a transformation-specific language.

The M2T transformation built in Java has two aspects: its fixed and its variable parts. The fixed parts are instructions that do not change and that will be present in all tailoring transformations created, e.g., the head of the input metamodels and code to build the output transformation model. The variable parts are statements that make use of libraries to read the information and statements that recursively create the tailoring transformation according to the VDM.



```
build_if(pointv, sizePV);
buildhelpers(sizePV,rulesTrans,sizeContRule,pointv);
wr.println("");
wr.println("rule main{");
wr.println("  from  ml:MM!MethodLibrary");
wr.println("  to    ml1:MM2!MethodLibrary(");
wr.println("          name <- ml.name,");
wr.println("          description <- ml.description,");
wr.println("          ownedMethodPlugin<- ml.ownedMethodPlugin,");
wr.println("          predefinedConfiguration<-ml.predefinedConfiguration");
wr.println("          )");
wr.println("}");
wr.println("rule methodplugin{");
wr.println("  from  mp:MM!MethodPlugin");
wr.println("  to    mpp:MM2!MethodPlugin (");
wr.println("          name<- mp.name,");
wr.println("          description <- mp.description,");
wr.println("          ownedProcessPackage<- mp.ownedProcessPackage,");
wr.println("          ownedMethodContentPackage <- mp.ownedMethodContentPackage");
wr.println("          )");
wr.println("}");
```

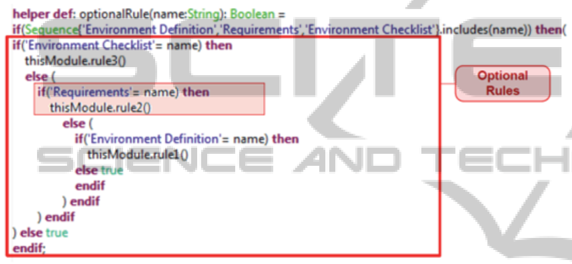
Figure 8: HOT implemented in Java.

Figure 8 shows an excerpt of the Java code that builds the body of the tailoring transformation and recursively generates the transformation rules considering the information in the VDM. After executing the HOT, the process tailoring transformation is obtained in text format (i.e., ATL source code).

4.4 Resulting Tailoring Transformation

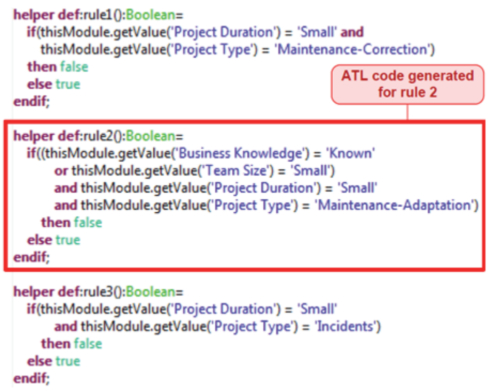
Once we have the process and the context models, as well as the generated tailoring transformation, it is possible to encapsulate them into an interactive tool: ATR. This tool includes building the VDM and the HOT that takes this model as input and automatically generates the tailoring transformation.

Figure 9 shows the transformation automatically generated for tailoring Rhiscom's process. As stated in Figure 5, Requirements is optional, and it has an associated rule in the tailoring transformation. Figure 10 highlights the helper called by rule 2 for deciding about the deletion of the Requirements activity.



```
helper def: optionalRule(name:String): Boolean =
if(Sequence{'Environment Definition','Requirements','Environment Checklist'}.includes(name)) then(
if('Environment Checklist'= name) then
   thisModule.rule3()
   else (
      if('Requirements'= name) then
         thisModule.rule2()
         else (
            if('Environment Definition'= name) then
               thisModule.rule1()
            else true
            endif
         ) endif
      ) endif
) else true
endif;
```

Figure 9: Tailoring transformation. "Requirements" is optional so a decision is made with respect to its inclusion.



```
helper def:rule1():Boolean=
   if(thisModule.getValue('Project Duration') = 'Small' and
      thisModule.getValue('Project Type') = 'Maintenance-Correction')
   then false
   else true
endif;

helper def:rule2():Boolean=
   if((thisModule.getValue('Business Knowledge') = 'Known'
         or thisModule.getValue('Team Size') = 'Small')
         and thisModule.getValue('Project Duration') = 'Small'
         and thisModule.getValue('Project Type') = 'Maintenance-Adaptation')
      then false
   else true
endif;

helper def:rule3():Boolean=
   if(thisModule.getValue('Project Duration') = 'Small'
         and thisModule.getValue('Project Type') = 'Incidents')
      then false
   else true
endif;
```

Figure 10: Tailoring transformation excerpt. Decision about deleting the "Requirements" activity is highlighted.

5 CONCLUSIONS

We have presented ATR, a model-based tool for interactively defining and automatically generating process tailoring transformations. ATR combines MDE and generative programming aspects. We have also described the implemented user interface, the underlying VDM model and the involved HOT. The resulting tool is powerful enough to generate the tailoring transformation for a real world company's process.

The main purpose of building an interactive tool was aiding the process engineer tailoring her/his process. We provided a running example that shows how to apply MDE concepts without interacting with the code or knowledge about transformation languages. Transformations in general could be quite complex. However, we have shown that building process tailoring transformations requires only a few types of rules that may be automatically generated from a VDM. Although we have been able to generate transformations automatically, this kind of tool is only applicable for software process domain, but this experience can be the starting point to be extended to other domains.

ATR generates complex rules using simple conditions, logical operators and complex conditions (with logical operators). In this sense, if there are rules with different conclusions on the same variability point, ATR still does not solve it; this can be addressed adding constraint definitions. Future work is necessary to extend the VDM to support constraints definition between software process elements and complex rules.

Finally, we need empirical evidence that help us validate the tool usability for real world process engineers.

ACKNOWLEDGEMENTS

This work is partly funded by Project Fondef D09I1171, Conicyt, Chile. The work of Luis Silvestre was supported by PhD Scholarship Program of Conicyt-AGCI (Chile), "Doctorado Nacional para Estudiantes sin Permanencia Definitiva- Convocatoria 2013".

REFERENCES

AtlanMod Group (2006). *Atlas Transformation Language*. ATL Eclipse Project. Online http://www.eclipse.org/atl/.

Balasubramanian, D., Narayanan, A., vanBuskirk, C., and Karsai, G. (2006). The graph rewriting and transformation language: GReAT. In *Proceedings of the Third International Workshop on Graph Based Tools*, pp. 1–8.

Beck, K. (1999). Embracing Change with Extreme Programming. *IEEE Computer* 32(10): 70-77.

Bendraou, R., Jezequel, J., Gervais, M.P., and Blanc, X. (2010). A Comparison of Six UML-Based Languages for Software Process Modeling. *Software Engineering, IEEE Transactions on*, 36(5):662–675.

Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., and Lindow, A. (2006). Model transformations? transformation models! In *MoDELS'06*, LNCS 4199, pp. 440–453. Springer.

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

Czarnecki, K., Helsen, S. (2006). Feature-based survey of model transformation approaches, *IBM Systems Journal* 45(3): 621-645.

De Oliveira Barros, M., Werner, C. M. L., and Travassos, G. H. (2002). A system dynamics metamodel for software process modeling. *Software Process: Improvement and Practice*, 7(3-4):161–172.

Hurtado, J.A., Bastarrica, M.C., Quispe, A., Ochoa, S.F. (2013). MDE-Based Process Tailoring Strategy. *Journal of Software: Evolution and Process*, in press. DOI: 10.1002/smr.1576.

Kalnins, A., Barzdins, J., and Celms, D. (2004). Model Transformation Language MOLA. *In Aßmann, U., Aksit, M., and Rensink, A., (Eds), Model-Driven Architecture: European MDA Workshops: Foundations and Applications*, *LNCS* 3599, pp. 62–76.

Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M. (2012). Model Transformation By-Example: A Survey of the First Wave. In the *Conceptual Modeling and Its Theoretical Foundations*, LNCS 7260, pp. 197-215.

Kleppe, A. G., Warmer, J., and Bast, W. (2003). MDA Explained: The Model Driven Architecture: Practice and Promise. *Addison-Wesley Longman Publishing* Co., Inc., Boston, MA, USA.

OMG (2001). *Meta Object Facility (MOF) 2.0 Query/View/Transformation V1.1*. Object Management Group. OMG doc. formal/2011-01-01.

OMG (2008). *Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.0*. Object Management Group. OMG dtc/07-09-04.

OMG (2008a). *Software Process Engineering Metamodel SPEM 2.0 OMG Specification.* Object Management Group. OMG Technical Report ptc/07-11-01.

Rumpe, B., and Weisemöller, I. (2011). A Domain Specific Transformation Language. In *Proceedings of the ME'11 - Models and Evolution*, Wellington, New Zealand, Oct. 2011.

Schmidt, D.C. (2006). Guest Editor's Introduction: Model Driven Engineering. *IEEE Computer*, 39(2):25-31.

Silvestre, L., Bastarrica, M.C., and Ochoa, S.F. (2013): Implementing HOT's that Generate Transformations with Two Input Models. *Accepted in XXXII International Conference of the Chilean Computer Science Society (SCCC'13)*, Temuco, Chile.

Simmonds, J., Bastarrica, M.C., Silvestre, L., and Quispe, A. (2013). Variability in Software Process Models: Requirements for Adoption in Industrial Settings. In *4th International Workshop on Product line Approaches in Software Engineering (PLEASE'13)*, San Francisco, California, USA.

Sun, Y., White, J., and Gray, J. (2009). Model Transformation by Demonstration. In *MoDELS'09*, pp. 712–726.

Tisi, M., Cabot, J., and Jouault, F. (2010). Improving Higher-Order Transformations Support in ATL. In *Tratt, L. and Gogolla, M. (Eds), ICMT*, *LNCS* vol. 6142, pp. 215–229. Springer.

Varró, D. and Balogh, Z. (2007). Automating model transformation by example using inductive logic programming. In *Proc. of the 2007 ACM Symposium on Applied Computing (SAC'07)*, Seoul, Korea.

Varró, D., Varró, G., and Pataricza, A. (2002). Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227.

Weiss, D. M., Li, J., Slye, H., Dinh-Trong, T., and Sun, H. (2008). Decision-Model-Based Code Generation for SPLE. In *12th International Software Product Line Conference* (*SPLC'08)*, pp. 129–138.

Wimmer, M., Strommer, M., Kargl, H., and Kramler, G. (2007). Towards Model Transformation Generation By-Example. In *HICSS'07*, IEEE Computer Society: 285-294.