Implementing HOTs that Generate Transformations with Two Input Models

Luis Silvestre, María Cecilia Bastarrica and Sergio F. Ochoa
Computer Science Department, University of Chile
{lsilvest,cecilia,sochoa}@dcc.uchile.cl
Beauchef 851, Santiago, Chile

***Abstract*—Model-driven Engineering (MDE) is a paradigm that promotes the use of models and automatic model transformations to handle complex software developments. Model transformations promise to reduce the effort for manipulating models. However, building transformations themselves is not easy. Higher-order Transformations (HOTs) are a means for automatically building model transformations. Building HOTs is in itself a complex task mainly because there are no standard languages for implementing them, and there are not many HOTs available in the literature to learn from. This situation is even worse when more sophisticated HOTs are required with two input models. We consider a real application to generate transformations for tailoring software process, because the generated transformation needs to have two input models: the organizational process and the project context model. In this paper, we show three different techniques for implementing this HOT and discuss their benefits and limitations.**

I. Introduction

MDE is a trend in software engineering where concepts are represented as models that can be successively transformed into lower level models, and eventually into source code [16]. Model transformations are a special kind of programs that take one or more models as input, and produce one or more models as output. There are mainly two types of transformations: model-to-model (M2M) and model-to-text (M2T). Even though these transformations may be written in any programming language, there are transformation languages that provide high abstraction level primitives for implementing model transformation. Query/View/Transformation (QVT) [15] and Atlas Transformation Language (ATL) [11] are some of the most popular languages for developing M2M transformations, while TCS [10], MOFScript [12] and ACCELEO [5] are utilized to implement M2T and T2M transformations. Writing transformations, regardless of the language, requires more than just mastering the implementation language. This makes writing good model transformations a challenging task.

Provided that in MDE everything is treated as a model, a transformation may also be considered as a model. Therefore, a transformation model can also be the output (or the input) of a model transformation. A transformation that either takes a transformation as input or produces a transformation as output is called a Higher-order Transformation (HOT) [17]. There is still no standard language for implementing HOTs, and there are few successful real world implementation experiences reported in the literature either. Therefore, it is not clear

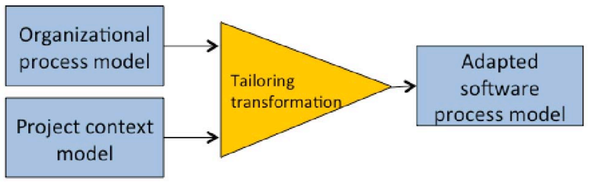


Fig. 1. Software process tailoring strategy

```
-- @path MM=/SPEM/spem.ecore
-- @path MM2=/SPEM/spem.ecore
-- @path MM1=/SPEM/spcm.ecore

 module KI_SPrLTailoring;
 create OUT : MM2 from IN : MM, IN1 : MM1;
 -- To obtain a Attribute Configuration Value
 helper def: getContextAttributeConfiguration(nameAttribute: String):
             MM1!ContextAttributeConfiguration =
             MM1!ContextAttributeConfiguration.allInstances()
            ->asSequence()
            ->select(a | a.myContextElement.name =nameAttribute )
            ->first();
 -- To obtain the next WorkBreakDownElement
 helper def:nextElement(a:MM!WorkBreakDownElement):
    MM!WorkBreakDownElement =
    MM!WorkBreakDownElement.allInstances()-
    >select(t|t=a.next)->first();
 -- Start point to optional choices
 helper def: optionalRule(name:String): Boolean =
          if(Sequence{'Problem and Project Scope Definition',
        'Early Change Management',
        'Environment Specification'}.includes(name)) then
         (thisModule.taskRule1(name) and
       thisModule.activityRule2(name)and
       thisModule.activityRule3(name) and
       thisModule.taskRule4(name))
```

Fig. 2. Excerpt of the complexity of writing model transformations

for software engineers how to proceed when they have to implement HOTs.

In several scenarios, for instance when we want to automatically adapt a software process to a particular project context, the HOT must generate a *tailoring transformation* with two input models, and produce an output model. According to the proposal presented in [8] (see Fig. 1), a *tailoring transformation* can use the organizational process model and the project context model, in order to generate the software process particularly adapted to a specific project.

Given the complexity and the skills required to perform this tailoring activity, using a MDE approach to automate it seems to be an appealing idea. Although automatic process tailoring has the potential to be highly useful for software companies, the complexity of writing model transformations prevents process engineers from taking advantage of it.

Figure 2 shows part of the tailoring transformation that was used in MDE strategy for tailoring software process [8]. Although this tailoring proposal has shown to be technically feasible in real scenarios, it clearly has important limitations when the users (i.e., the process engineers) have to use it in practice. Therefore, using a HOT for automatically generating the *tailoring transformation* is also appealing. However, as mentioned before, it is not clear how to implement HOTs that use two input models.

In this paper we show three different techniques for implementing HOTs with to input models: MOFScript [12], Atlas Model Weaver [4] and a Java program. We analyze the feasibility of implementing HOTs with each alternative, and we discuss the benefits and limitations of each one.

Next, Section II reports the main related work. Section III describes the implementation of the HOT for the application example. Section IV presents the conclusions and further work.

II. Related Work

According to Tisi et al. [17], *"A higher-order transformation is a model transformation such that its input and/or output models are themselves transformation models"*. They identify four classes of HOTs: transformation analysis, synthesis, modification and composition [18]. Provided that we use HOTs for generating a model transformation, we are using a synthesis HOT. They acknowledge that this class of HOTs tend to be too verbose, i.e., it is required a long and complex code for generating just a small output transformation. This is mainly the problem we address in this paper: technological approaches for building HOTs provided that they are always complex.

A promising approach for developing HOTs is the Atlas Model Weaver (AMW) [4]: it defines a mapping between the source and the target model, and this mapping model is automatically translated into the model transformation between the source and the target models. However, the transformations that can be generated are quite simple, and basically taking only one input model and producing one output model. The AMW generates ATL [11] transformations using matching and allowing traceability.

Didonet del Fabro et al. [7] report the use of ATL to create a transformation that uses a weaving model to transform a bug tracking model in Mantis into an equivalent model in Bugzilla. The authors also provide other examples using AMW and ATL to generate a model transformation that translates a Kernel Metametamodel (KM3) into the Structured Query Language (SQL) data definition language, and vice versa [6]. In both cases, the input model for the HOT is a weaving model.

Tisi et al. [18] use ATL to define four patterns for building HOTs, and establish transformations aimed at improving their use. The literature also reports that there are other languages that allow implementing these HOTs, such as GReAT [1], MOFScript [12] and XSLT [3]. However, there is no evidence indicating that transformations implemented using these languages can take two input models to generate an output model.

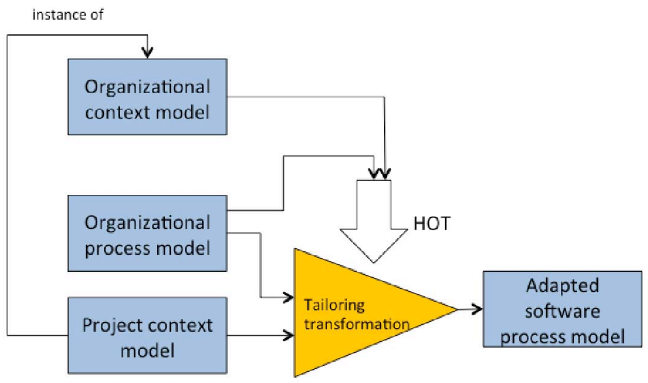

instance of
Organizational context model
Organizational process model
HOT
Project context model
Tailoring transformation
Adapted software process model

Fig. 3. HOT with two input models to generate the tailoring transformation

III. Generating Transformations with Two Input Models

In the MDE approach for tailoring software process, the tailoring transformation needs to be written in a transformation language formally defined, such as ATL. The tailoring transformation includes the constructors and rules for building the adapted software process model. The transformation constructors allow the transformation to generate the output model, and the rules allow the transformation to tailor the software process model, based on the characteristics of the project context to be addressed.

We use a HOT to generate the *tailoring transformation*, avoiding thus writing it directly. Our HOT takes two input models: *organizational process model* that conforms to the eSPEM (experimental SPEM) Metamodel and *organizational context model* that conforms to the SPCM (Software Process Context Metamodel) [9]. The eSPEM Metamodel is a subset of SPEM (Software Process Engineering Metamodel) [14].

Figure 3 shows the extended solution defined to adapt the organizational process model, based on the characteristics of the software project to be addressed. The *organizational process model* has all the activities, tasks, roles and work products for guiding the software development and includes its potential variability. The *organizational context model* indicates the project attributes that may influence the process tailoring along with their potential values. The goal is to determine the technical feasibility to generate the *tailoring transformation* using this HOT. Finally, the tailoring transformation takes the organizational process model and project context model, and generates the adapted software process model.

Four different techniques were considered for implementing this HOT. First, we used MOFScript for implementing the HOT as a M2T transformation. Second, we used the AMW tool for defining a weaving model with two input models that would allow us to automatically generate an ATL tailoring transformation. Finally, we utilized a Java program with the same purpose.

```
texttransformation ExampleTransformation
(in spemtl:"http://spem/1.0",
in ct:"http://contextmetamodel/1.0")
{
  var fileList : List;
  var auxstring: String;
  var number : Integer = 0;
  spemtl.ProcessPattern:: main () {
    file ("tailoringTranformation.atl");
    self.nestedElements->forEach(br:spemtl.Activity) {
        auxstring= br.name;
        br.nestedElements->forEach(bs:spemtl.Activity){
        auxstring= bs.name;
        }
  }
   //Print ATL code
 println("-- @path MM=/SPEM/spem.ecore");
 println("-- @path MM2=/SPEM/spem.ecore");
 println("-- @path MM1=/SPEM/spcm.ecore");

 println("module Tailoring;");
 println("create OUT : MM2 from IN : MM, IN1 : MM1;");
```

Fig. 4. Excerpt of the HOT implemented in MOFScript

A. Implementing the HOT using MOFScript

MOFScript is a M2T transformation imperative language. This language implements explicit rule calls and a flat structure that manages only a few constructors; these characteristics make it simple and usable. A MOFScript transformation consists of a set of rules/methods that allow evaluating input model elements and producing text, either in the standard output or into a file. Generating a tailoring transformation written source code can also be considered as a model [2], and thus a MOFScript M2T transformation plays the role of a HOT.

Figure 4 shows part of the HOT implemented in MOFScript, which takes two input models: the organizational software process model (spemtl) and the organizational context model (ct). The MOFScript implementation builds the tailoring transformation in two steps: (1) it establishes the relationships between organizational context model and variation elements of the organizational process model, in order to select the process elements to be included in the output, and (2) writes the tailoring transformation using functions and procedures for building the tailoring transformation.

Although the generated tailoring transformation is able to transform an organizational process model into a context adapted model as expected, this transformation is quite complex. In order to manage project context information parametrically, the use of a recursive solution is required, and this is complex to implement it using imperative rules. Moreover, building generic rules requires sophisticated data structures that MOFScript is neither able to define nor manage.

B. Implementing the HOT using AMW

AMW is a tool for establishing mappings between models, and storing them in a model that is called a weaving model [4]. This weaving model conforms to a weaving metamodel and it is defined as a mapping between elements in metamodels.

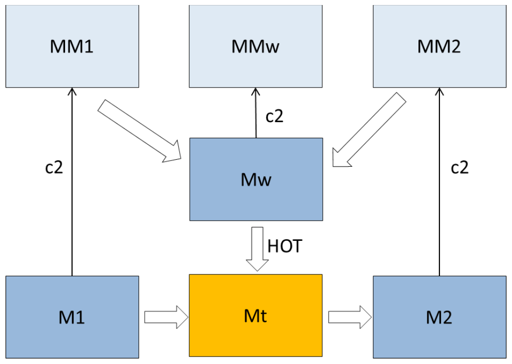


Fig. 5. HOT using the AMW tool

As shown in Fig 5, the weaving model is used as input of a HOT for generating a transformation model that conforms to the ATL metamodel. This transformation takes Model 1 conforming to the first metamodel (MM1) and produces as output a Model 2 conforming to the second metamodel (MM2).

This seems to be a clear and high level declarative solution. However, for our case we need to generate a model transformation that takes not one but two input models: the organizational process model and the organizational context model. This solution is not technically feasible, because although AMW has no theoretical limitations in the number of input metamodels for defining the mapping, it is not expressive enough for specifying conditions, such as including or not a process element in the output process model, depending on the value of a project context attribute. Moreover, the AMW tool does not allow us defining two input metamodels for the weaving model.

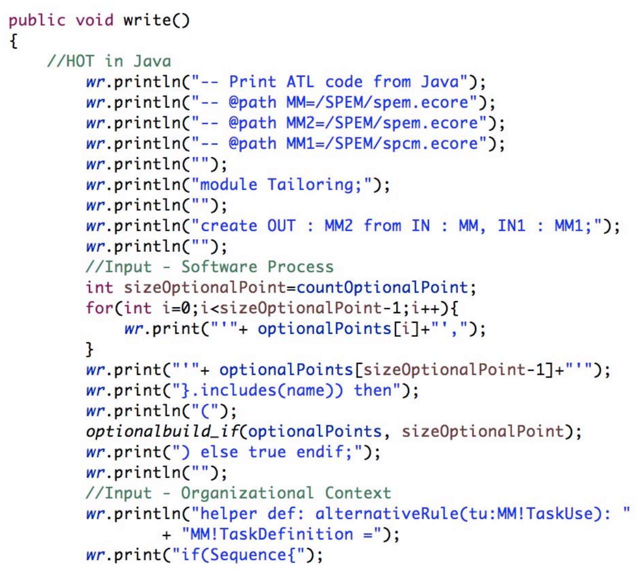
C. Implementing the HOT using Java

We used Java for implementing the HOT trying to avoid the complexity posed by ATL and MOFScript, and also gaining flexibility. Even though Java is a general purpose language, it counts on the JDOM library for dealing with XML files. We have also built another library (HashMaps) to convert XML files into XML Metadata Interchange (XMI) models and vice versa.

Figure 6 shows part of the HOT implemented in Java using procedures and functions. In this case, the HOT also takes two input models and generates the tailoring transformation in a text format (i.e., it performs a M2T transformation), following the same structure depicted in Figure 4. Given the flexibility provided by Java, we were successful in building this HOT. However, we are aware that having a domain-specific language makes building model transformations more straightforward and prevents several potential syntax errors that may be introduced in Java.

IV. Conclusions and Future Work

This article shows three techniques to determine the technical feasibility to implement HOTs that use two input models.



```
public void write()
{
    //HOT in Java
        wr.println("-- Print ATL code from Java");
        wr.println("-- @path MM=/SPEM/spem.ecore");
        wr.println("-- @path MM2=/SPEM/spem.ecore");
        wr.println("-- @path MM1=/SPEM/spcm.ecore");
        wr.println("");
        wr.println("module Tailoring;");
        wr.println("");
        wr.println("create OUT : MM2 from IN : MM, IN1 : MM1;");
        wr.println("");
        //Input - Software Process
        int sizeOptionalPoint=countOptionalPoint;
        for(int i=0;i<sizeOptionalPoint-1;i++){
            wr.print("'"+ optionalPoints[i]+"',");
        }
        wr.print("'"+ optionalPoints[sizeOptionalPoint-1]+"'");
        wr.print("}.includes(name)) then");
        wr.println("(");
        optionalbuild_if(optionalPoints, sizeOptionalPoint);
        wr.print(") else true endif;");
        wr.println("");
        //Input - Organizational Context
        wr.println("helper def: alternativeRule(tu:MM!TaskUse): "
                + "MM!TaskDefinition =");
        wr.print("if(Sequence{");
```

Fig. 6. Generating Transformations with Java

The implemented HOTs must perform the tailoring of a software process, and the alternatives used for the implementation were MOFScript, AMW and an application in Java. The AMW solution is the most clear and elegant, but it does not support two input models. Therefore, this is not an option to address the challenge.

Although MOFScript is a domain-specific language, implementing HOTs that use two input models seems to be immature, because this language introduces a series of constraints -no recursion, limited data structures, only a few iterators- that limit its applicability; or at least, it resulted too limited for our application. This language is not supported any more. MOFM2T [13] is a new language that Eclipse is considering as MOFScript successor, and some of its drawbacks are being addressed in the new proposal.

The Java solution reached the goal at the expense of clarity. This solution has the advantage that can be tuned according to the user needs, and improve the usability when required. For instance, the GUI implemented as part of the Java application eases the usability of the solution when it is used by process engineers. This helps transfer these solutions to the software industry.

Given the flexibility provided by a general programming language as Java, we were successful in building the HOT. However, we are aware that having a domain-specific language, such as the AMW tool or MOFScript, makes building model transformations more straightforward and prevents several potential syntax errors that may be introduced in Java.

We are now testing the implementation of HOTs using a transformation language, such as ATL. We expect that in the near future one of the former solutions will substitute this one, and thus taking advantage of solutions based on standard specifications.

Acknowledgment

This work has been partly funded by Project Fondef GEMS IT13I20010 and Luis Silvestre was also supported by PhD Scholarship Program of Conicyt, Chile (CONICYT-PCHA/2013-63130130).

References

[1] D. Balasubramanian, A. Narayanan, C. vanBuskirk, and G. Karsai. The graph rewriting and transformation language: GReAT. *Electronic Communications of the EASST*, 1, 2007.

[2] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow. Model transformations? transformation models! In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 440–453. Springer, 2006.

[3] J. Clark. XSL Transformations (XSLT). Technical Report Version 1.0, W3C Recommendation, 1999.

[4] M. Didonet del Fabro and P. Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and System Modeling*, 8(3):305–324, 2009.

[5] Eclipse. Acceleo, 2014. http://www.eclipse.org/acceleo/.

[6] Eclipse. Translating KM3 into SQL using AMW and ATL, 2014. http://www.eclipse.org/gmt/amw/examples/#AMW_KM32SQL.

[7] M. D. D. Fabro, J. Bézivin, and P. Valduriez. Model-driven tool interoperability: An application in bug tracking. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 863–881. Springer, 2006.

[8] J. A. Hurtado Alegria, 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.

[9] J. A. Hurtado-Alegria, M. C. Bastarrica, A. Quispe, and S. F. Ochoa. An MDE approach to software process tailoring. In D. Raffo, D. Pfahl, and L. Zhang, editors, *ICSSP*, pages 43–52, Honolulu, HI, USA, 2011. ACM.

[10] F. Jouault. TCS: Textual Concrete Syntax, 2014. https://raweb.inria.fr/rapportsactivite/RA2010/atlanmod/uid20.html.

[11] F. Jouault and I. Kurtev. Transforming models with ATL. In H. Berlin, editor, *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138, Montego Bay, Jamaica, 2006. Springer-Verlag.

[12] J. Oldevik, T. Neple, R. Grønmo, J. Ø. Aagedal, and A.-J. Berre. Toward standardised model to text transformations. In A. Hartman and D. Kreische, editors, *ECMDA-FA*, volume 3748 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2005.

[13] OMG. MOF Model To Text Transformation Language (MOFM2T), 1.0. Technical Report 2008-16-01, Object Management Group, 2008. http://www.omg.org/spec/MOFM2T/1.0/.

[14] OMG. Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Technical Report 2008-04-01, Object Management Group, 2008. http://www.omg.org/spec/SPEM/2.0/PDF/.

[15] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Technical report, Object Management Group, January 2011. http://www.omg.org/spec/QVT/1.1/PDF/.

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

[17] M. Tisi, J. Cabot, and F. Jouault. Improving Higher-Order Transformations Support in ATL. In L. Tratt and M. Gogolla, editors, *ICMT*, volume 6142 of *Lecture Notes in Computer Science*, pages 215–229, Malaga, Spain, 2010. Springer.

[18] 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, Enschede, The Netherlands, 2009. Springer.