Generative Adaptation of Model Transformation Assets
Experiences, Lessons and Drawbacks

Fábio Paulo Basso, Raquel Mainardi Pillat, Toacy Cavalcante Oliveira
Universidade Federal do Rio de Janeiro (UFRJ) - COPPE - Rio de Janeiro - RJ, Brazil
{fabiopbasso, rmpillat, toacy}@cos.ufrj.br

Marcos Didonet Del Fabro
Universidade Federal do Paraná (UFPR), C3SL - Curitiba - PR, Brazil
marcos.ddf@inf.ufpr.br

ABSTRACT
Model transformation is a central activity in Model Driven Engineering (MDE) as it specifies how models are consumed to generate other models or code. Complex scenarios typically involve the execution of several transformations that, due to variability of solutions to develop software projects, need to be tailored to attempt different implementation technologies, libraries, patterns, etc. Recent proposals to tailor these assets suggested that current Software Product Line (SPL) techniques can be used to promote reuse of model transformation assets. However, in a recent case study, we have found lacks in techniques, including ours, to apply reuse for this domain in particular. Thus, this paper presents this case study and discusses implications and challenges in tailoring these assets with SPL techniques.

Categories and Subject Descriptors
D.3.2 [Design]: Theory; D.2.8 [Software Engineering]: Design Tools and Techniques—Computer aided software engineering (CASE), Model driven development (MDD)

Keywords
Transformation Chain; Transformation Process; Model Driven Engineering; Feature Model; Software Product Line.

1. INTRODUCTION
Model Driven Engineering (MDE) [12] focuses on designing software models used to generate code or other models. In this scenario, generation occurs by means of Model Transformations (MTs) that are executed by MDE tools such as FeatureIDE [33] and Andromeda [4]. These tools must be tailored to apply transformations for specific software implementation details. With the increasing adoption of MDE techniques [22], it is becoming common for MDE practitioners to specify model transformations for different software projects that require different framework versions, different libraries and also different implementation patterns [33]. In this context, software engineers must deal with communalities and variants of model transformation assets [36]. Assets are composed by model transformations, fragments of source-code of text-based (white box) transformations, modules of transformations used in black box programs (object files), transformation rules, MDE tool configuration files and chains of model transformations [27].

These assets can be specified with diverse languages, techniques and tools [12]. Aiming to promote the reuse of model transformation assets, some proposals focuses in how to ensure consistency in transformation compositions [38], while others focuses in high-level specification and management of transformation chain that can be further generated for specific MDE tool configurations [35]. Other proposals in [2, 3] applies reuse through generative techniques based on Software Product Line (SPL) [11]. These proposals are useful to adapt model transformations with different reuse techniques.

Our proposal is to develop large scale model transformations managed as an SPL. In order to find characteristics that compose communalities and variabilities in model transformation assets, a designer perform a domain analysis [24]. Currently a widely used notation in domain analysis is the Feature Model (FM) [16]. The Feature Model is useful to define system characteristics (functional, architectural, technological, mixed, among other ones) from a particular implementation domain and to use features relationship to configure different target implementations in MDE [32]. In this sense, our work uses FM to tailor model transformation assets with some SPL-based reuse techniques.

The application of SPL techniques in this context is not a novelty, since works [2, 3] have applied it to generate Model Transformation Chains (MTCs) and Volter et al. in [36] exemplified the use of dynamic reconfiguration of MTCs. However, literature lack in case studies on the application of such techniques for other reusable assets such as configuration files and, more important, transformation rules. In order to drive future researches, these experiences are important to understand challenges to adapt model transformation assets in real scenarios. In this direction, this paper presents a case study conducted in a real scenario and discusses positive results and drawbacks that are important to improve reuse techniques for this domain in particular.

This paper is organized as follows. Section 2 summarizes...
2. MOTIVATING SCENARIO

The development of information systems require the design and implementation of Graphic User Interface (GUI) as illustrated in Figure 1 (A). To allow the representation of details concerning information systems, e.g. actions performed by user on GUI elements (known as action semantics [25]), each GUI component can receive annotations that differentiate it from a common component such as illustrated with stereotypes and tags in Figure 1 (B). Besides provide information to generate source-code through transformations [17], annotations can be used to increment/enrich a preliminary GUI model. This is possible by transforming a GUI model to other ones (e.g. to generate a different GUI representation that allows one to generate source-code for a specific WEB user interface).

The main reason to design this type of model is to reuse these specifications in software projects. However, one must consider that implementation technologies used to produce software evolves, meaning that the same model can be transformed to different target technologies used in legacy or in current software projects. This requires one to deal with different model transformations that generates code mapped for some technologies that in legacy systems are different than in current systems. For example, in order to generate source-code taking as input the model shown in Figure 1, three transformations from model-to-text (M2T) programmed with Velocity Templates [14] could be used as shown in Figure 2. Each transformation have communalities (fragments that should be reused) and differences (fragments that should be moved for a specific transformation).

The example highlight in bold the differences between three technologies used to produce a View layer: JSP [10] is the acronym for Java Server Pages, a script language used to generate dynamic content for web pages, as shown in Figure 2 (A); JSTL and JSF have a similar goal and are exemplified in Figure 2 (B) and (C).

This type of variability is different than those related to application domains. These ones requires one to deal with variances in a single web-shop domain, such as the model shown in Figure 1, while the variants we are interested for are more like target platform features applicable for diverse application domains. In fact, implementation variabilities are cross-application domain and implies in manage communalities and differences in relation to model transformations assets, not to domain models.

Software companies whose applications are derived from a base framework, that do not mutate, should not consider the techniques presented in this paper. However, for companies that need to tailor model transformations for different software projects (legacy and modern), managing implementation variability in code generators is crucial to core business. This would allow to generate new functionalities in increments for legacy systems [6], using model transformations for old-fashioned technologies, and developing new ones with modern technologies.

With this in mind, our goal in next sections is to apply SPL techniques for model transformations assets. Moreover, one must focus in a cross-application domain variabilities, such as information systems domain, and not in application domain, such as the model illustrated in Figure 1. Therefore, we divided our study in two parts: the first one (Section 3) introduces concepts used to tailor transformation assets, while Section 4 presents the second part of the case study that focuses in application of SPL techniques to generate a MTC.

3. CASE STUDY (PART I)

This case study uses a small part of what we did along five years in industry, whose technical architecture and model transformations are composed by variant technologies in support for Java platforms [34]. The study applies some reuse techniques in order to configure model transformation engines to support transformations for different software project
3.2 Representing Variability in a TCDM

A TCDM allows to orchestrate variant and mutually exclusive Model-to-Model (M2M), Model-to-Code (M2C) or Model-to-Text (M2T) transformations [17]. For example, the transformer number 3 in sequence is abstract (see Almeida et al. in [1] for more information about this concept) and is replaced in a transformation chain by one of their children: a) JSTL Transformer is used if the feature JSTL is selected in the PDM (see \texttt{requires} stereotyped dependencies); b) JSF Transformer is used if the feature JSF feature is selected. The mutually exclusion of these transformations is ensured by the \texttt{XOR} relationship owned by the Web feature and also by the abstract transformer. This example is different in the transformer number 2 because two transformations can be used in the same sequence: a) Mobile Transformer is used if the features Mobile and Remote Layer are selected in the PDM; b) Web Transformer is used if the Web feature is selected. The inclusive \texttt{OR} relationship owned by the View Layer feature ensure that at least one of these transformations is used, but it is possible to use both.

Transformers illustrated in Figure 4 allow to generate models and code taking as input the model shown in Figure 1. They are organized chained as follows: 1. \texttt{Generate Mockup}, 2. \texttt{Mockup to Concrete UI}, and 3. \texttt{Generate Web View code}. Transformations are composed by parameters (input, output, specialization points, etc.), dependencies and pre-conditions. Parameters may have dependence links such as between the parameter named \texttt{sa:ScreenArea} to the transformer number 1 and also be classified as metamodels or data objects (default).

Moreover, MTs may have other kinds of dependencies stereotyped with \texttt{requires} or \texttt{excludes}, as shown in Figure 4 in transformers Web Transformer, JSTL Transformer and JSF Transformer. In pre-conditions, a target of a dependency is always a feature available in the PDM. A dependence relationship define when a model transformer must be used in a transformation chain sequence. In this sense, the Web Transformer will be used only if the feature Web is selected. As one may notice, our transformation chain modeled in Figure 4 does not require an \texttt{IF clause} to determine when a model transformer is used. Therefore, dependencies between features and transformers are responsible to apply this binding.

4. CASE STUDY (PART II)

This Section shows the second part of the case study towards a generation of a transformation assets using as input the domain models exemplified in Section 3. This is executed in an instantiation process [24] where the PDM features are selected and, as sequence, transformation assets are adapted.

4.1 Adapting Text-based Assets

Any textual based transformation asset can be fragmented for specific features and then be merged considering the selected features in PDM [2]. In this sense, Figure 2 highlighted, in bold, differences between three text-based model transformations used to produce the View Layer for web applications: JSP, JSTL and JSF. Each one is represented as specific transformers placed into the transformation chain named Generate Web View Code TC as shown in Figure 5 (B) and the generic algorithm (transformation rule) owned by the transformer Generate Web View code is shown in Figure 5 (A). Thus, after identify differences, it is neces-

![Figure 3: Platform Domain Model.](image-url)
A Specialization Point is used to cut text fragments into specialized units. Further, these units are combined and compose a main model transformer considering their owned pre-condition rules in relationship with features. Figure 5 exemplifies these points applied over the generic model transformation named Generate Web View code in which content is clipped into transformation units shown in middle and right-part of the Figure 5 (B). Each unit provides text fragments constrained for each web view technology modeled in the PDM.

4.2 Model Transformation Chain

Figure 6 illustrates a transformation chain used by Andromeda tool [21]. This tool is largely used in MDE projects to execute transformations, such as in MDArte framework [28] used by Brazilian government institutions. It contains a lot of configuration files that demonstrate applicability of SPL-based techniques in generating transformation assets. In this example, model transformers are programmed as XSL templates (lines 5 and 8), but they also could be expressed as ATL [23], Velocity Templates, Java binaries (compiled classes), etc.

In order to generate a MTC taking as input the TCDM, the bold content shown in Figure 6 must be changed. The highlighted content must be replaced by those sequences of model transformers shown in Figure 4. Besides, because transformation rules were split into several fragments mapped for specific technologies shown in Figure 3, they must now be merged and packaged together with the generation of a MTC. Figure 7 illustrates part of source-code, developed with Velocity templates, required to generate a concrete transformation chain used in Andromeda. The velocity requires input parameters to apply a model-to-text (M2T) transformation. As we aim to transform the TCDM into a content used by Andromeda configuration file, we must set the diagram as the velocity input parameter. Thus, the input for this transformation is an instance of the metaclass TransformationDiagram that, due to reasons to save space, is part of a metamodel not discussed in this paper.

The diagram is placed as a velocity parameter in the line 11 as a parameter named transDiagram. The algorithm shown in Figure 7 checks if the diagram contains valid sequences (line 2-3) and then iterates over each concrete model transformer specified at the diagram (line 4). The operation called in line 2, named getConcreteTransformationSequences, verifies the pre-conditions owned by each model transformer to include it or not into the returned list. This means that it is responsible to verify which model transformers match a PDM selection, to include them into an ordered list used to generate the transformation chain. The rest of lines generates the content shown in Figure 6, using following model transformer properties: the name (line 5); the output (line 8) the transformation file (line 9).

4.3 Generating Configuration Properties

In order to configure other required details by Andromeda, it is also necessary to generate file content shown in Figure 8 (A). This requires modifications into the model transformer that tailors Andromeda tool, as shown in Figure 8 B and C. In this case, one must deals with configuration properties: important details to configure Andromeda with a set of required libraries, application layers and other relevant information used to generate source-code for information system applications. In this sense, notice that the properties in lines 4 and 5, shown in Figure 8 (A): they can be mapped to the Struts feature shown in Figure 3 using UML tagged values and stereotypes. This is possible because TCDM and PDM metaclasses inherit from the UML superstructure. Therefore, one can add any extension representation into the designed elements, differently than other solutions restricted to expressing only information allowed by the MTC metamodel.

In this sense, stereotypes and tags can be used to specify properties required by some MDE tool. For example: by annotating a feature from the PDM with stereotype «Namespae», one grant that it will be used as a namespace when tailoring Andromeda with transformations; besides, this information is used to generate the bold lines shown in Figure 8 (A).

To adapt the example show in Figure 7, a modification...
in the model transformer shown in Figure 8 (B) is required: to add an input parameter (between lines 3 and 6). This parameter is a list of features coming from the PDM that are used on the white box transformer shown in Figure 8 (B) to generate those properties on Figure 8 (A). Thus, this transformation generates the namespace tags into XML content by iterating over a list of features and checking those selected and with Namespace, merging the namespace tag with main transformation chain content. Finally, the return of the model-to-text transformation may be saved in a specific file representing the Andromeda’s transformation chain.

4.4 Final Steps

In this stage it is required to tailor domain models for a target application (in our case, for Andromeda tool). Taking as input assets exemplified previously, model transformation assets are generated to attempt to a specific software project (target platform). Accordingly, Andromeda MTC is generated with the exemplified set of model transformations, using a M2T generation from TCDM to a MTC-based text specification shown in Figure 6. This way, the feature selection process (the PDM instantiation) is started and M2T transformations are executed.

The instantiation process also merges model transformer fragments exemplified in Figure 5. It generates MT assets that are complete to support executions. Then, transformation assets are packed and Andromeda tool is configured to support a tailored transformation assets.

Therefore, the result is a set of adapted transformation assets that run in a specific model transformation engine.

4.5 Lessons Learned

Our experiences reinforce that SPL-based techniques are important to reuse model transformation assets, as suggested Volter et al. in [36], Asztalos et al. in [3] and Aranega et al. in [2]. This allows to adapt transformation assets to different software project needs, considering legacy systems and modern ones. However, it is important to notice that some benefits and drawbacks relies in each SPL-based technique regarding generative or runtime (also known as dynamic SPLs). They must be carefully considered when maintaining reusable model transformation assets.

We have been using runtime and generative techniques since 2006 [6, 7] and have learned that it is preferable to use runtime compositions for assets of type MTCs and model transformations. This is because they simplify adaptation, since runtime-based executions does not require a generation of MTCs, while a generative technique generates a new unit that requires to repeat tests for each new generated MTC. Moreover, as reported in [9], transformation fragments are mostly compiled Java programs, not textual fragments. This allows to use a previously validated component, not requiring retesting as it is necessary in generative adaptations of transformation assets. However, drawbacks of runtime SPL techniques are: 1) runtime-based techniques are strictly applicable for the WCT transformation execution engine and; 2) compiled fragments can be achieved only with Java programs because other transformation languages such as ATL or QVT do not support runtime-based SPL techniques discussed in this paper.

Moreover, we have experienced that due to new requirements for specific implementation technologies, inevitable non-planned changes will be necessary in existing transformations, requiring refactoring of transformation assets (especially for model transformers). Refactoring requires a tiring procedure to test previous solutions (legacy model trans-
formers that now were modified). In this regard, in [5] we reported about an industrial case study that required modification in some existing model transformations while software development already had been started. It was necessary to change some transformations to fit to new variants of Ajax technology (rich web user interfaces). The target of techniques is a tailored model transformation, derived from a single Ajax technology (initially in Dojo Toolkit) for Dojo Toolkit or jQuery.

In this regard, in order to reduce the number of tests cases and save time, we found that tests also should be oriented to features. Thus, this would facilitate executions of test cases for both generative and runtime techniques.

4.6 Limitations

Our work did not consider a scenario where model transformations, application models and metamodels evolves together. Since fragments of model transformations must be evolved when metamodel evolves, this figures a lack in our proposal that did not consider automatic techniques for co-evolution. In order to automatically apply changes in these three transformation assets, Lopez-Herrejon et al. [27] proposed a reuse technique for co-evolution. Corrêa et al. in [15] highlighted some co-evolution tasks that can be automatically and also must be manually applied in models, metamodels and Graph-based transformations. Although they provide interesting perspectives in this direction, we find that due to a necessity to break transformations in fragments hampers the application of current co-evolution proposals.

This work is focused in applying generative-based techniques and not depicted runtime-based techniques in support for model transformation reuse. However, runtime-based compositions for PLA are discussed in details in [6, 7, 31, 36]. Another limitation in this study is that we have used generative-based techniques applied in model transformation developed only with Velocity templates. In this sense, we are exploring other languages.

4.7 Considerations

Some threats to validity constraints our findings and lessons we have learned in this case study.

Internal Validity: The first part of this study was conducted in an industrial environment by using a developer that has Senior knowledge about Java J2EE Architecture. Moreover, this professional worked in many other software companies who used different frameworks and APIs to produce information systems. In this sense, this study is valid only for information systems development with Java target platform.

External Validity: The results reported in this study can be generalized for others SPL-based techniques. Although domain models (PDM and TCDM) are designed with a particular tool named WCT, they can also be designed with other set of tool support discussed in Section 5. Moreover, SPL-based techniques can be applied in any text-based transformation asset. However, check of consistency in transformation compositions must not be considered, since the techniques presented in this paper are not directed to this goal.

Reliability: This work is trustable guarded boundary conditions that the study presented: management of model transformation assets, considering communalities and variabilities in information system development with variant Java J2EE architectures. In other words, this work is invalid in software development scenarios whose target architecture not varies.

5. RELATED WORKS

The combination of FM and MTs was suggested by Tekinerdogan et al. in [32] and inspired our work. However, so far we have not identified any other approach that uses feature model to generate transformation assets such as ours do, considering a merge of a TCDM and a PDM in the same design. In this regard, since our study is based on SPL techniques, following related works can be mentioned: Kang et al. [24], Fernandes et al. [20], Benavides et al. [11] and Volter et al. [37]. They have not tackled adaptations on model transformation assets, but provided theory to apply the case study.

In regard to variability in model transformations, as far as we know it, in [6, 7] we provided a first methodology an tool support to deal with variants in MTs. Volter et al. in [36] has a similar proposal. These works execute dynamic transformation chains ruled by a feature model. However, variants are specified directly into the MTC, a solution dependent from a model transformation execution engine. On the other hand, our solution allows specifying a model named TCDM that is used to generate MTCs used by any transformation execution engine.

Recently, Aranega et al. in [2] applied existing SPL-based techniques to generate MTCs. However they also have not considered a MTC domain model as we demonstrated in a TCDM. In other words, they do not design MTCs, but they do design a PDM. As consequence, given that features are directly mapped for a MTC, Aranega et al. proposal is limited since a PDM must be defined for each transformation execution engine. In our proposal a TCDM is used as input for model transformations that allow to generate any text-based asset (MTCs, configuration files, model transformations with merged fragments and rules, and so on), independently from a transformation execution engine that will make use of them.

Moreover, other approaches also generate MTCs, but they did not considered variants. In [32, 35, 1] it is discussed about importance of a language to connect transformations to target platforms. As a complement, Etien et al. in [19, 18] had been applied heterogenous model transformations developed with different metamodels. Although these proposals are important to generate MTCs, we considered in this work a wider set of asset types, including fragments transformation rules, configuration files and properties. These related works, being focused only in MTC generation are limited, since other assets must also be generated as exemplified in previous sections. In this sense, our proposal is to generate automatically a MTC using domain models that compose model transformation assets (not only model transformers) considering variability and communality. Moreover, differently than in [6, 7, 36] that deals with variants in runtime, our work allow generating diverse assets without requiring a specific model transformation engine.

In Almeida et al. [1] was proposed a solution to define MT compositions with consecutive model-to-model transformations. They defined transformations tiers as abstract platforms represented with extensions of MOF [25] pack-
ages using a lightweight MDA approach. Nevertheless, it is not possible to have a notion about the sequence, a detail that makes their solution difficult to design transformations sequences. Since existing SPL-based techniques consider only variabilities and sequence is important to generate valid MTCs, their proposal is complementary for our work. As a new contribution, we propose to organize transformations in a domain of model transformation assets ruled by features of a platform domain.

In Vanhooff et al. [35] target platforms are also used as a part of the MTCs. Target platform and MTs was composed using UML activity diagram, in a similar solution to our previous work [6], since our current solution is based on a DSM to draw MTCs. Some other approaches to support transformation orchestration are presented as graphical extensions from OMG’s UML [13] like Abstract Platforms [1] and MARTES UML Profile [35] and also not ensure consistency in transformation composition. In [38] proposed an approach to ensure consistency in composing MTCs. A lack in Yie’s approach is that they did not considered a composition between heterogeneous MTs, programmed with different supporting languages than ATL [23] and using different languages to specify MTCs as discussed in Asztalos et al. [3]. Similarly, this paper has the same lack, an ongoing work with some result presented in Pillat et al. [30].

6. CONCLUSIONS REMARKS

This paper presented a case study on tailoring model transformation assets. In order to use SPL-based techniques for this domain in particular, it was introduced the design of a domain model composed by a Platform Domain Model (PDM) and a Transformation Chain Domain Model (TCDM). These models allows one to specify variabilities in model transformation chains as well as in other types of assets such as transformation rules, configuration files, and so on. This study allowed to configure assets for a specific transformation engine/tool named Andromeda. Finally, experiences and shortcomings were pointed, allowing to find drawbacks useful for further reuse researches.

Considering a literature review and also current and previous experiences, we can highlight to some points that challenges the application of reuse techniques in order to adapt model transformation assets as follows.

Co-evolution: Currently co-evolution proposals tackled by Lopez-Herrejon et al. [27] and Corrêa et al. [15] did not considered a scenario where model transformations are fragmented. Thus, a specific solution for this reuse technique is necessary.

Heterogeneity of model transformation languages: Co-evolution proposals were strictly applied for graph-based transformations. However, a TCDM can chain transformations programmed with ATL or specified in a graph-based language such as QVT and also data objects (Java data types). Yie et al. [38] and Guy et al. [21] presents some contributions, but considering only homogeneous compositions. Thus, another challenge is to validate transformation compositions in a TCDM considering a heterogeneity of solutions for specific transformation engines.

Refactoring of model transformations: As mentioned in Section 4.5, refactoring of model transformations is inevitable since implementation technologies evolves. Thus, as a new contribution, we propose to organize transformations in a domain of model transformation assets ruled by features of a platform domain.

In Section 4.5, refactoring of model transformations is inevitable since implementation technologies evolves. Thus, another challenge is to ensure consistency in composing MTCs. A lack in Yie’s approach is that they did not considered a composition between heterogeneous MTs, programmed with different supporting languages than ATL and using different languages to specify MTCs as discussed in Asztalos et al. [3]. Similarly, this paper has the same lack, an ongoing work with some result presented in Pillat et al. [30].

Thus, an open question is how to apply these steps considering SPL-based techniques presented in this paper?

Agile frameworks: Finally, TDD is a method recommend in agile frameworks such as Scrum. We can point to a central difference between agile and the presented procedure: while PDM and TCDM are designed in a top-down procedure, considering existing assets (first domain models than after development of MT), agile recommends a bottom-up procedure, starting with tests. Thus, another interesting question is how to apply a bottom-up design of PDM and TCDM?

7. ACKNOWLEDGMENTS

This work was partially supported by the Brazilian agencies CAPES, CNPq and FINEP.

8. REFERENCES


[8] F. P. Basso, T. C. Oliveira, and K. Farias. Extending junit 4 with java annotations and reflection to test...


