Supporting Large Scale Model Transformation Reuse

Fábio P. Basso  Raquel M. Pillat  Toacy C. Oliveira
Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ - Brazil +55 21 2562-8672
{fabio.basso,rm.pillat,toacy}@cos.ufrj.br

Leandro B. Becker
Federal University of Santa Catarina (UFSC)
Postal 476 - 88.040-900 - Florianópolis - SC - Brazil
lbecker@das.ufsc.br

Abstract
The growth of applications developed with the support of model transformations makes reuse a required practice, specially when applied to transformation assets (e.g., transformation chains, algorithms, and configuration files). In order to promote reuse one must consider the different implementations, communalities, and variants among these assets. In this domain, a couple techniques have been used as solutions to adapt reusable assets for specific needs. However, so far, no work has discussed their combined use in real software projects. In this paper, we present a new tool named WCT, which can be used to adapt transformation assets. Moreover, through lessons learned in industry, we address some reuse techniques devoted to adapt these assets.

Categories and Subject Descriptors D.3.2 [Software Engineering]: Design Tools and Techniques - Computer-aided software engineering (CASE); Object-oriented design methods, Model driven development (MDD).

General Terms Design, Theory.

Keywords Model transformation chain, feature model, transformation reuse, product line technique, MDE.

1. Introduction
Model Driven Engineering (MDE) [19] promotes the software development through the execution of a sequence of Model Transformations (MTs). In real world scenarios, where models can evolve attempting to new technologies, the transformation assets (e.g., MT algorithms, transformation tool configuration files, MT test cases, etc.) must be properly adapted [25]. A web survey conducted in 2011 by Hutchinson et al. [29] included 250 MDE practitioner’s responses, where a huge number (67.8%) considered that source-code generation is very important. Paradoxically, in a recent study published in 2013, Petre [37] interviewed 50 industry practitioners who used the Unified Modeling Language (UML) [14] at least once in software projects. The results show that only few practitioners (6%) used it for source-code generation purposes. Hutchinson et al. [29] addressed the difficulty related with the development of MTs as a limiting factor to disseminate the MDE, since practitioners consider that it is difficult to develop, test, and maintain transformations. Moreover, each transformation must be tailored for a particular software project, demanding expertise, time, and money [21]. Therefore, it is necessary to find solutions to easy the development of new transformation assets as well as to reuse existing ones.

Some solutions for transformation reuse divide MTs into several modules, which may be combined in configuration files known as Model Transformation Chain (TC) [49]. Some related works design transformation chains with UML [46] [20], to then generate specific MDE tool configurations. Although these solutions are helpful to manage transformation assets - independently from the adopted MDE tool - they require the design of many transformation chains in order to support alternative transformation sequences. Thus, this solution requires too much effort to design many possible transformation chains [48].

In order to reduce this effort, Aranega et al. [2] claim that TCs should support variability in transformations. Variability management is a requirement to support reuse even in different contexts [24, 39]. Accordingly, other approaches support transformation asset variability in two different scopes: a) Runtime variant points are checked at runtime to execute variant transformation sequences [5, 48]; b) Generative approach for transformation chains that are adapted to execute certain sequences [2, 46]. Our experiences show that both scopes are important and complementary.

Our main contribution in this paper is reporting a multi-year industrial effort in adapting large scale model transformation assets through the combined use of these techniques. This requires the design, composition, execution, and adaptation of transformation assets. Therefore, this paper reports experiences on using a new tool named WCT that integrates these techniques in a single toolkit. WCT has been used to adapt model transformation assets, whose technologies used to implement information systems varies [4, 6, 7].

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 introduces reuse techniques through exemplifications of variant model transformation assets designed with WCT. Results and experiences are discussed in Section 4 and Section 5 presents the conclusions. Finally, future works are pointed in Section 6.
2. Related Work

WCT is a toolkit used to adapt transformation assets both at runtime and generative approach. Due to the need of applying Product Line Architecture (PLA) reuse techniques [12], it allows associating a Feature Model (FM) [32] instance with MTs in a domain model. This domain model is then adapted to a specific MT chain or is used to dynamically re-configure the chain at runtime. In this regard, it is important to mention that the combination of FM and transformations was firstly suggested by Tekinerdogan et al. [42], which inspired our first work in [5]. Since then, we have been applying these concepts to reuse model transformation assets through a set of reuse tools embedded into WCT.

Some reuse approaches support transformation orchestration through graphical extensions from OMG’s UML [14], such as Abstract Platforms [1], MARTES UML Profile [47] and FOMDA [5]. They can all be used to generate specific transformation chains. Almeida et al. [1] proposed a solution to define MT compositions with consecutive model-to-model transformations. They defined transformation layers as abstract platforms represented with extensions of MOF packages [36]. Nevertheless, it is not possible to have notions about the transformation sequence, which makes their solution inappropriate to design TCs. In another work, van Boas [45] proposed the design of a TC using workflows to model transformation processes with UML activity diagram, but lacked more formalism used to generate real TCs as Almeida suggested. The Almeida’s, Tekinerdogan’s and Boas’s proposals are complementary for TCs. However, none of them supports rules to assist the generation of TCs as we have done in this paper. Our contribution is complementary to such works, proposing to organize transformations in a domain ruled by feature’s relationships from a platform domain model.

Aiming to generate TCs used by MDE tools, some proposals allow designing transformation compositions in high-level of abstraction to then generate specific TCs. Accordingly, Vanhooff et al. [47] proposed a TC modeling language using target platforms as part of TC designs and in van Hooff et al. [46] a framework to execute heterogeneous transformations. Target platform and transformations are composed using UML activity diagrams, in a similar solution as we did in Bass et al. [5], using a UML Profile. These proposals were complemented by Wagelaar [49] that proposed a composition of black-box model transformations and by Etien et al. [20, 21] that suggested the use of heterogenous model transformations developed with different metamodels. Although an interesting work, Etien’s, Vanhooff’s and Wagelaar’s did not support transformation chain executions considering target platform variants, meaning that chains must be manually changed when a different set of transformations is necessary.

Regarding the specification of target platform variability, there exist other tools such as SPLOT [35], FeatureIDE [44], Odyssey [22] and pure::variants [13]. These and other tools listed in Benavides et al. [12] and Berger et al. [13] facilitate the management of commonalities and variabilities in PLA-based applications. Therefore, PLA-based reuse techniques presented in this paper are also supported by other tools and proposals. However, similarities remains here, since these PLA-based proposals surveyed in [12, 13] cannot be compared with the current proposal. This is because our core assets are very different, what implies in the use of different techniques. In true, they are not even related works, as discussed and justified in Section 3.8. Thus, to tailor model transformation assets one must consider more specific solutions as follows.

In [5, 6] we presented the first methodology (Features-Oriented Model-Driven Architecture) and a tool support to add variants in model transformation processes/chains, allowing adaptations in runtime and generative approach. Völter and Groher [48] proposed a strictly runtime-based approach, where variants are programmed inside transformation chain files (e.g., an XML file that chain transformations for a specific MDE tool). In other words, a TC is not generated through a model specification, but it is manually changed in code to support variants. Accordingly, authors pointed to some important shortcomings that must be considered to apply the suggested runtime-based technique: 1) it requires a specific MDE execution engine; 2) the tracing of variant points may be difficult, since one must search for programmed instructions inside the chain’s code when the feature of a FM changes. However, we consider the existence of two types of runtime-based dynamic transformation chains: one is by interpreting a model specification of a transformation chain, as we exemplified in [6], and the other one is by programming variants directly into TC code, as we exemplified in [5]. The first one not present the traceability problem, but still requires a specific MDE execution tool.

Some recent proposals have applied similar reuse techniques through generative approaches. Aranega et al. [2] uses FM to fragment some transformation assets (e.g: white-box model transformations, fragmented algorithms or “transformation rules”, diverse configuration files, and so on) using aforementioned PLA tools. However, despite techniques for PLA are applicable in diverse asset types, adaptations of model transformation assets must also consider valid compositions among transformations. While in [2, 20, 21] authors seems to move in this direction, so far, only we considered the use of FM and transformation chains that allow to validate the bindings with transformation’s input and output (IO) in design time. Thus, in our proposal these compositions are checked during the design of a transformation chain model, while in [2] they are checked only after the generation of a concrete model transformation chain. In comparison to the work from Aranega et al. [2], our WCT tool saves time in detecting problems that can occur while we are adapting these assets.

Considering consistency in transformation composition, Guy et al. [27] and Yie et al. [52] suggest that valid compositions must be ensured with parameter types. For example, IO parameters are checked during the design of a model transformation chain, validating rules such as metamodel data types (e.g., EMF-based metamodels compatibility), transformation languages (e.g., ATL and QVT transformations), etc. Moreover, it is important to consider that model transformation assets can evolve. In this regard, Lopez-Herrejon et al. [34] presented a proposal to automatic control the co-evolution of models, metamodels, and transformations. Although being important techniques related to model transformation assets tailoring, nor heterogeneity of model transformation composition nor co-evolution is used in this paper, figuring as a shortcoming discussed in Section 4.3.

The tailoring of transformation assets also requires a strong procedure to ensure that what was adapted leads to a working MDE-based tool. This way, it is necessary to ensure that a selected FM is consistent. This is usually made through some techniques that can be based on constraint
3. Reuse Techniques Assisted by WCT

In this section we discuss about reuse techniques used in WCT to support tailoring of model transformation assets. It is important to mention that any technique presented in Section 3.1 to 3.6 can be started in arbitrary order considering the availability of their inputs. Accordingly, we are assuming that all the resources used by model transformation tools are model transformation assets such as: complete algorithms, fragmented algorithms, black-box Java programs [19], QVT graph-based transformations [30], white-box templates programmed with Velocity [16] or ATL [31]), model transformation chains [47], MDE tool configuration files, and so on. Finally, it is important to notice that these techniques only applies in cases where a big set of assets must be reused.

3.1 Platform Domain Analysis

Along some experiences in developing and adapting MTs to specific target architectures, we have found that variability is important to adapt transformation assets for specific needs [5]. Variability allows one to take essential decisions, e.g.: if characteristics like persistency and file are required in a software project, then one must execute a model transformer to generate file structure, otherwise executes the MT to generate database structure. Moreover, identifying a family of software project architectures is important to adapt these assets [47]. This is detailed by techniques used to analyze technical requirements as follows.

Input: New requirements (e.g., implementation technologies, patterns “design, architectural and codification”, etc.) and existing model transformation assets.

Output: Platform Domain Model (PDM) and a data dictionary describing features.
The platform domain model shown in Figure 1 allows one to configure some features that can be used in software projects to specify the persistence layer. The relationship between the features defines that a persistency layer must contain at least a "Database" feature. In software projects that we have produced, the Database Management System (DBMS) “PostgreSQL” was used in some occasions and “Oracle” in others. However, given that more than one database can be used in the same application, we designed such relationship as an inclusive OR. Thus, both databases can be used in the same software project.

The feature “DB script” is optional, meaning that in some projects we generated the database structure with SQL scripts and in other ones we let Hibernate (the Java framework used to manage the persistency layer) to generate the structure. The ORM feature is optional to develop the persistency layer since in legacy software projects no framework for persistance layer were used. However, when object-relational mapping is present in a software project it certainly was implemented with JPA or XDoclet.

### 3.2 Transformation Chain Design

Model transformation chains are supported by tools such as openArchitectureware, Epsilon and AM3 (AtlanMod Megamodel Management). Transformation chain modeling is a common reuse technique where specification becomes independent from the execution tools. In this sense, the second technique is intended to design model transformation chains.

**Input:** Model transformation assets.

**Output:** Transformation Chain Domain Model (TCDM) with low-level information about model transformations associated with pre-condition rules.

**Main References:** Almeida et al. [1], Basso et al. [5], van Boas [45], Etien et al. [20] and Vanhooff et al. [47].

After specifying the PDM, the next step is to organize model transformations inside a Transformation Chain Domain Model (TCDM). As the name suggest, this model must specify not a single sequence of transformations, but a family of transformations that can compose model transformation chains used by more than one software project. Almeida et al. [1] and Vanhooff et al. [47] have a similar proposal to design model transformation chains, but without the link for features available in the PDM. Thus, it is necessary to constrain each transformation to a set of features, as illustrated in Figure 3, using dependency relationships stereotyped with «requires», or stereotyped with «excludes» that demands the deselection of the linked feature.

The above constraints exemplify pre-condition rules owned by MTs, in which dependencies always target a feature designed in PDM. This relationship determines (in runtime and generative approach) when a MT must be used in a transformation chain derived from the TCDM. In this sense, the transformation number 2 is used only if the “DB Script” feature is selected (note, in Figure 1, that it has been specified as an optional feature). This means that if “DB Script” is not selected in PDM, then the transformation chain does not include transformation 2.

Each MT assumes at least a name, a return type (default output parameter) and a default input parameter. Besides, transformations are sequenced in TCDM through numbered back circles. Thus, Figure 3 illustrates the following sequence:

1. **(T1 - “Apply ORM”).** It is a model-to-model (M2M) transformation programmed in Java; it is classified as a Platform Independent Model (PIM) view and provides a dialog to apply ORM annotations into Class elements, generating the model shown in Figure 2 (A).

2. **(T2 - “Generate DB script”).** It is an abstraction: a dotted model transformation that allows generating scripts to create database structures for a specific DBMS, discussed in Section 3.3.

3. **(T3 - “ORM to PSM”).** It is also an abstraction whose transformations generate source-code embedded in Java classes known as entities. Therefore, entities own those ORM annotations exemplified in Figure 2 (B) and (C).

The second and third MTs are abstract specifications (scattered elements), which the concept was first introduced by Almeida et al. [1]. This means that they propagate a transformation for their concrete children (not dotted ones). Thus, according to the example shown in Figure 3, if “JPA” is selected then “Generate JPA” is used in the third order of execution, otherwise, if “XDoclet” is selected, then “Generate XDoclet” is used in the position number 3. The same rule is valid for “Generate DB script”. However, the decision between “Oracle Script” or “PostgreSQL” is taken at runtime and not in generative approach (note the stereotype «Runtime»). Therefore, in the case of runtime decisions, the solution is strictly applied when using WCT based transformation chains and must be avoided if the intent is to tailor the TCDM for other TC tool formats.

Finally, mutual exclusion between transformations must be handled to ensure that only transformations that attempt to the selected features in a PDM are used in a resultant model transformation chain. This way, the relation between the features and transformations in a TCDM ensures the
use of only desired execution sequences when a transformation chain is generated in Section 3.6. This is represented in a TCDM by annotating abstractions with «XOR», as illustrated in Figure 4 (D). Figure 4 exemplify two chains (A) and (B) that are included into a high-level chain (C). The first chain owns a transformation for reverse engineering: from source-code to a UML class element. Note to the “source:File” parameter: it contains an arc, representing an external parameter for this chain. This means that it is visible when the “Rev Eng plus ORM TC” is included into a high-level chain, shown in Figure 4 (C). The same is valid for the output parameter “entityOutput:Class” in Figure 4 (A) that is related with first transformer in Figure 4 (C). Also note in this chain that “DB script TC” owns a single parameter: this is because the transformation chain, shown in Figure 4 (B), externalizes only “entity” as an input (in highlight).

Moreover, Figure 4 (A) illustrates a transformation process while Figure 4 (B) illustrates a regular transformation chain. Notice that “Reverse from code” is executed automatically «AutomaticTask», while “Apply ORM” is a guided procedure «GuidedTask». This one illustrates a situation where the software engineer will specify ORM annotations shown in Figure 2 with the help of a wizard, taking the time he/she need.

In this case, in [4] we reported experiences in using wizards and in [9] we exemplified a similar case where reverse engineering tasks are performed to allow the execution of assisted ones. In [6] we exemplify the use of a transformation tasks to execute model-to-model and model-to-code transformations to develop a web information system. In [8] we discuss procedures to transform annotated Graphic User Interface (GUI) for a UML-based model according to Model-View-Controller [41] application layers, that are refined and then used to generate a web information system source-code. Recently, in [9], we issued semantic rules specified with UML tagged values over Input and Output (IO) transformation parameters, that are useful to constraint a composition among different model transformation engines. Moreover, we also presented a metamodel and shortcomings to chain heterogeneous reuse tools.

3.4 Transformation Asset Fragmentation

While previous technique focused in compose model transformations, this technique allow split model transformation assets in small parts, also referred as factorization [34]. These techniques are based on product line concepts, also referred in literature as Software Product Lines (SPL) [12]. In this sense, important requirements to split transformation assets are discussed along this section.

Input: Model transformation assets, PDM, TCDM, adaptation type. Transformation split must consider the adaptation type: 1) in Runtime [48] depends from an execution engine or; 2) Generative approach [2] that is independent from an execution engine.

Output: Model transformation assets are split in smaller units.

Main References: Different perspectives regarding model transformation assets factorization are found in Aranega et al. [2], Basso et al. [5] and Völter and Groher [48].

The TCDM can also be used in the context of PLA proposals, splitting and merging parts of white box transformers (e.g., ATL transformation rules) [49]. The main references listed in this section present alternatives for tools to apply the same concepts. Considering a WCT-based exemplification, the transformer T1 = “Generate DB script” needs to generate SQL scripts to create a database structure in some DBMS (Oracle and Postgres), where examples are shown in Figure 5 (A).

The source-code shown in Figure 5 (B) illustrates the transformation content, written using the Velocity template, associated to T1. In line 7, the algorithm put character (50) as data type for a column named with the return of a Velocity attribute in line 5. This code fragment is specific to generate a script file used to create table structures for Postgres DBMS.

Figure 6 shows a new TCDM element that is used to split and merge transformation fragments: a fragmentation, also called as variants [22], variant points [12] or hotspots [35], is represented by the element stereotyped with «SpecializationPoint». This element is used to cut a white box text fragment and to handle these parts in specialized units. These units implement a specialization point such as “Oracle Script”, which merges the value “varchar2” with the content of “Generate DB script”. As exemplified before, each MT has...
Figure 5. White-box transformer named “Generate DB Script” to generate a database schema.

Figure 6. Fragmentation of transformations. The line 7 is replaced by a tag that is further replaced by the transformation fragment owned by “Postgre Script” OR “Oracle Script”.

its own pre-condition rules in TCDM and in PDM, ensuring that the value in line 7 is: character (50) or varchar2 (50).

3.5 Development and Test Support
The development of MT is well discussed in literature [31, 51]. However, due to the necessity in combining the use of PLA and TC, a problem arises: tests must assert that adapted assets return the correct results. This way, this section explains how to test MTs and transformation chains.

Input: Requirements for transformations and, if available, the PDM and TCDM.

Output: Model transformation rules and Test Cases.

Main References: Kuster et al. [33] provide good practices for test-driven development of MT, while Cuadrado et al. [31], Jouault et al. [31] and Willink [51] provide information about MT development. Fleurey et al. [23], Hervieu et al. [28] and Baudry et al. [11] provide information about test cases applied in MTCs.

For instance, we use an approach similar to Kuster et al. in [33], where MTs are developed using unit and integration tests. Accordingly, an adapted TC can be tested with the program shown in Figure 7. It is a JUnit test case derived to support the test of MT assets. Line 3 shows a Java annotation (component of WCT API) used to handle common transformation tasks, such as to open a TC, apply M2M and M2C transformation, open and save a UML model and so on. The task in Line 3 informs the test case to open a file “config/transf_chain.fomda”, the TCDM, before executing the operation in line 11. This file has exemplified model transformations, such as “Generate JPA”. Line 4 exemplifies an annotation used to constrain the use of this test case for a specific selection of features in PDM. This means that in a test suite composed by many test cases, this test case is executed only if the “ORM” feature is selected in Figure 1. These details are not supported by the default JUnit framework version 4. Thus, they are important to reduce time in adapting test suites to execute tests for every time an adaptation of MT assets is necessary.

3.6 Adapting the Assets
Other reuse techniques are used in last stages of adaptations. Following we discuss about these techniques.

Input: Model transformation assets, data dictionary, PDM and TCDM.

Output: A configured MDE-based transformation tool.

Main References: Kang et al. [32] provide information about application engineering reuse technique, while in Basso et al. [5] we discuss the application of this technique in tailoring transformation processes. Related works (Aranega et al. [2] and Lopez-Herrejon et al. [34]).

After defining a PDM and a TCDM, one can configure the target platform (also referred in the literature as concrete platform) and tailor transformation assets to the configuration necessary for a given software project. The target platform definition is known as FM selection, or as instantiation process, term coming from the FORM approach [32]. The instantiation is a procedure guided by the WCT, in which it will select automatically some features considering their relationships, such as dependencies. A wizard automates this process by offering facilities to select features, considering FM’s relationships, as discussed in [38].

In this moment, it is necessary to ensure that only valid selections from a PDM are acquired Thum et al. [43]. In this regard, due the number of a features and relations, scalability issues may be a problem. White et al. [50] proposed FAMA, a framework composed by constraint solvers to ensure the consistency in features selection and does not faces scalability problems for a model composed by up to 5000 features [50]. Alternatively, as reported in [38], we obtained good results in using a Business Rule Management System.
Accordingly, a PDM designed to specify variabilities in model transformation assets cannot contain features that are particular for web shop or for e-learning domains. The features designed in PDM must be general for both domains, because the adaptation procedure will generate configured transformation assets that can be used further to generate source code for both applications. Therefore, this is the reason why PDM contains features related to implementation details.

For more discussions regarding these differences, see Tekinerdgojan et al. [42], Basso et al. [5], Völter and Groher [48], Aranega et al. [2] and García-Alonso et al. [25].

4. Experience Report
The WCT toolkit has been improved since 2006 at Adapit, a small Brazilian company, to create MDE applications with the Java programming language. Along this time, three complete industry projects can be highlighted, where “SP” is a software project for i in 1,2,3: (1) SP1 - It is a web/desktop application for online auction with support to enterprise resource planning functionalities (system with 656 classes and 190 dynamic web pages), developed by the company’s team; (2) SP2 - It is a web and desktop application used internally to manage trainee (system with 789 classes and 149 dynamic web pages); (3) SP3 - It is a web application to manage financial support for innovation projects (system with 139 java classes and 56 dynamic web pages). The latter application has been developed by a partner company with the supervision of the Adapit’s team. Some measures of such application regarding software productivity are reported in [6]. It is important to notice that the developed applications have no similarities in their functionalities. Therefore, similarities and differences designed in a PDM refers to target implementation technologies, as discussed in Section 3.8.

In all these projects, transformations have been successfully used in order to generate models and code to produce multi-layered web information systems that varies in implementation technologies. Examples of application layers are: graphic user interface layer (desktop and web), data access layer, entity layer, integration or remote layer, xml configuration files, text files, Java classes, database scripts, models, etc. García-Alonso et al. [25] have designed a PDM composed by a similar set of layers. Their PDM is very similar to the primary versions of our model that we have presented in [6, 7]. Besides, information about Java technologies that we have been using to produce software applications is available in [40]. These papers can be accessed to understand technical details about transformation asset variants used to generate source code, as detailed in the next section.

4.1 Changes in Target Platforms
Our PDM defines that each software project contains the following layers: Model Layer (a conceptual model also known as entity classes); Data Access Layer (DAO classes where business logic accesses databases to retrieve and store objects); a Database Layer, a View Layer to display User Interfaces (UIs); a Controller Layer to handle user actions and transactions; a Remoting Layer, needed when the software project requires integrated applications (e.g.: UIs displayed for Desktop, Mobile and Web platforms that use business logic that run in a web server).

It is important to mention that each developed project uses a different target platform. First and second applications have had the following configuration of technologies: Hibernate, JPA (XDoclet for the first one), Dojo toolkit (the second one was migrated to jQuery), among many others.
technologies in support for each application layer. Dojo toolkit and jQuery are technologies to deal with Ajax actions, they are mutually exclusive web frameworks used to develop web user interfaces for the View application layer.

In this sense, the following technologies are a part of what have been changed across projects and required modification in TCDM: a) In our first experiences, XDoclet comments that apply Hibernate mappings (ORM) were used in the model layer while in subsequent projects it was used JPA; b) The first two projects included Dojo toolkit API as the web technology used to write rich user interfaces while the last project included jQuery.

4.2 Lessons Learned

We learned that a UML Profile based solution is not de quickest alternative to design TCs, since one must specify a lot of details that are difficult when using tags and stereotypes. We have experienced UML Profiles on our first works, in the development of SP1 as reported in [6], and have learned that it is difficult to apply bindings between model transformations and selection rules that will adapt them into a generated transformation chain. Since 2008, we have opted for an approach based on a domain specific modeling tool, presented in this paper. WCT provides facilities to design TCs and to validate bindings between transformation assets (e.g. FM. MTs, cutting rules, and other artifacts during design phase) in comparison to our previous tool prototype in [5].

However, a considerable time has been used to develop the specific diagrams (PDM and TCDM) and also to fix bugs. Therefore, a DSM is interesting to speed-up the PDM and TCDM modeling, but it is more expensive to produce than a UML Profile.

The integrated set of reuse techniques in a single toolkit allowed us to perform adaptations in an efficient way. This is supported by comparing our first experiences started in [6] with more recent ones [4, 7, 8]. In first experiences we generated transformation chains, modeled with FOMDA UML Profile [5] and mapped for FOMDA toolkit (the first prototype version [5]). This is a tiring and error-prone procedure, because tags and stereotypes can be wrongly specified in a TCDM, requiring to change it to then repeat the generation. In recent experiences, models of tags and stereotypes are replaced by specific metamodel elements. Also, dynamic runtime re-configuration of model transformation chains, exemplified in Figure 7 in line 8 and Figure 3, simplified the adaptation process. Moreover, the integrated set of tools and reuse techniques allowed to apply some validations (transformation parameter matching, unit test cases and integration test cases) more quickly. Therefore, these experiences showed us that reusable model transformation assets can be quickly constructed.

Moreover, we have experienced that changes in existing transformations must be carefully conducted. Accordingly, changes require a strong set of tests to ensure that transformations still work. This is the reason why we improved our first methodology on the FOMDA approach [5], since tests play an important role in adapting transformation assets. The use of test cases integrated in the same toolkit also helped to quickly ensure the consistency in runtime execution. Thus, through a customized JUnit engine for tests, it is ensured that transformations assets work.

Finally, we found that runtime-based approach is important, because it is quickest to apply than a generation time is. However, in order to avoid the tracing problem between features and transformation chain variants, in [6] we learned that the best option is to interpret a transformation chain domain model stead of program variants in TC files as exemplified in [5]. Accordingly, most of our transformation compositions are at runtime because this simplifies the process to deliver working transformation assets, since runtime adaptations are quicker than to generate a model transformation chain. Thus, we prefer the use of runtime compositions.

4.3 Limitations and Open Questions

Section 3.4 discussed about important generation-time reuse techniques used adapt transformation assets, allowing to fragment model transformations through PLA concepts. Although this approach is interesting, it complicates a little bit the adaptation process, since it requires more tests to ensure that chains and transformations fragments were correctly generated. Current proposals for model transformation asset tailoring, including this one, considered model transformations that run in a single MDE tool. In this sense, an open question is how to ensure that the generated product (model transformations and chains) are valid across different model transformation engines? This question is relevant because if the intent is to tailor model transformation assets to execute in a single execution engine, it is preferable the runtime-based approach because it simplifies the adaptation process.

Moreover, in our experiences we have not considered a solution for co-evolution of model transformations, application models and metamodels. In this sense, conform the metamodel discussed in [8] evolved, transformations and application models were manually changed. This required a maintenance procedure that, in part, was facilitated by the Java compiler, because most model transformations were programmed in Java. However, since fragments of model transformations must be evolved when metamodel evolves, this figures a lack in our proposal that do not considered automatic techniques for co-evolution. In order to automatically apply changes in these three transformation asset sets, Lopez-Herrejon et al. [34] proposed a reuse technique for co-evolution. Although it is an interesting proposal, we find that the necessity to break transformations in fragments hampers the application of current co-evolution proposals.

Thus, an open question is how to apply co-evolution reuse techniques for fragmented units of model transformations?

5. Conclusion Remarks

Model transformations are becoming the core assets of companies that produce software based on MDE techniques. Due to the growing demand of new transformations that cover new requirements, transformation assets must be changed. In this sense, this work presented the WCT toolkit as a solution to manage transformation variants in a scenario where the implementation technology used to produce software evolves together with the transformation assets. By using a set of tools available in the WCT, users are allowed to specify model transformers, transformation variants and, finally, execute model transformations and test cases that may ensure that everything is working when delivering a configured MDE tool. Thus, we presented a set of tools to deal with large scale transformation reuse.

The support for large scale transformation requires the adaptation of model transformation assets. In our work, adaptation must consider strictly variabilities of implementation solutions, but not variabilities in application domains. In other words, the set of techniques presented in this paper are not applicable to manage communalities and vari-
abilities in a single software application domain, but are interesting to adapt model transformation assets that actually are used in more than one domain. Here relies the main differences between our proposal and the existing software product line proposals: it is general for any software application domain. Therefore, this work presented experiences in adapting model transformation assets considering the same set of implementation variants used in three software projects, each one from a different application domain.

Moreover, since a single toolkit is used to perform diverse reuse techniques, we presented WCT as an integrated solution that efficiently applies adaptation procedures. Therefore, by bringing a wider set of functionalities to allow one to customize transformation assets, we present a singular tool in comparison to the related ones.

6. Future Work

This work did not detail some important aspects to tailor model transformation assets, as follows:

A process to adapt and deliver operational MDE tools. This paper discussed about reuse techniques developed with the WCT toolkit. However, more than using an adequate tool, we consider that it is necessary to follow a production planning that assist designers of reusable MTs.

Transformation compositions with heterogeneous transformation languages. Since most transformations used in software projects reported in Section 4 are of type black-box Java programs, this work is limited for a few compositions techniques. In this sense, a more general transformation composition must be considered, with heterogeneous set of model transformation languages.

Different types of test cases. In order to facilitate the test of adapted transformation assets, our techniques discussed in Section 3.5 must be compared with the state of the practice regarding test techniques for derived PLA products.

Acknowledgments

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

References


