Modden: An Integrated Approach for Model Driven Development and Software Product Line Processes

Ana Patrícia Magalhães
Federal University of Bahia
Salvador, Brazil
anapfmm@dcc.ufba.br

José Maria N. David
Federal University of Juiz de Fora
Juiz de Fora, Brazil
jose.david@ufjf.edu.br

Rita Suzana P. Maciel,
Bruno Carreiro Silva
Federal University of Bahia
Salvador, Brazil
{ritasuzana,
brunocs}@dcc.ufba.br

Filipe Araujo da Silva
Faculdade Ruy Barbosa
Salvador, Brazil
araujo@filip@gmail.com

Abstract—In a Software Product Line reuse should be considered from the artifacts conception stage and, when implemented, these artifacts can be part of a library to be further reused by every domain application. In a model-driven context an application development can be seen as a set of transformations that lead to a final system. However, during the transformation process reuse techniques are not usually considered. Modden is a SPL approach that uses model-driven techniques. Through the integration of these two approaches we aim to enhance model-driven techniques by the reuse of models, code, transformation rules and so on. As a result, Modden aims to leverage the reuse of these artifacts in different software development process phases. It comprises two processes, one to develop core assets and another to develop specific assets for the product line, a SPL UML profile and a model driven development supporting tool.

Keywords—Model Driven Engineering; Product Line; Model Driven Architecture

I. INTRODUCTION

Building software with high quality and productivity as well as low cost has been a key challenge for many companies. In order to achieve this, reuse techniques have increasingly been used in scenarios that require such quality attributes. In this context, several approaches have been proposed over the years, such as: object-oriented paradigm, component-based development, service-oriented architecture, and others. Software Product Line (SPL) also aims to support the design and implementation of component reuse infrastructure, their execution and the communication between these components [11][12].

Systematic reuse should not be limited to source code artifacts. It is observed that in SPL reuse techniques are considered from the artifact conception stage and, when implemented, they can become part of a library which can be reused by every domain application. If these artifacts are obsolete, systematic reuse might be hindered.

A reusable asset is potentially made up of many lifecycle products including requirements and architecture definition, analysis model, design models and code, test programs, test scenarios and reports. According to [23], software assets encapsulate knowledge and are valuable for organizations. They are composed of a collection of related software work products that may be reused from one application to another. Additionally, to achieve systematic reuse it is necessary that this practice be inserted into the software development process context [20].

Several methods have been designed to systematize a development process based on SPL concepts [2], [3], [5]. However, as in traditional processes, models which are generated in the initial phases (i.e. domain analysis) can become obsolete as the applications evolve. As a result, these models are mostly used as additional documentation to support application construction. Some current proposals support artifact traceability and signal the models that should be changed, but they do not guarantee this update. Moreover, they do not consider an integrated SPL approach from a process enactment to the code generation.

The Model-Driven Development (MDD) [9] approach aims to establish a set of models to support software construction so that they are the main artifacts of this process. Model Driven Architecture (MDA) [7] is a conceptual framework that carries out MDD. Through the integration of SPL and MDA it is possible to reuse this transformation chain in SPL, which includes models, code, transformation rules and so on. This could leverage the reuse of these artifacts from different software development process phases. The goal of this work is to propose a SPL approach, named Modden, for software development which can be supported by MDA concepts. This approach comprises: (1) two processes, one to develop the reusable core assets, named MDA-DE and the other to develop specific products in the product line, named MDA-APP. The second process is beyond the scope of this paper, and will not be discussed here; (2) a SPL UML profile, used by the processes to support specific modeling; (3) a supporting tool for process editing and enactment to achieve a better process understanding, design, training simulation and optimization, support and enactment proposed by the use of Process Modeling Languages (PMLs). The processes are specified using SPEM [8].

Modden offers an integrated approach to enhance reuse in a process-centered perspective. This approach is specified through a standard notation that should guide software engineers in their software development tasks. Additionally, reuse is considered from the early software development
stages and process automation is supported by MDA techniques and tools.

We consider that the integration of both the MDA and SPL approaches together can enhance the software development process. Furthermore, we define transformation rules for each process stage. Using MDA in a SPL process we expect not only to enhance software artifacts reuse, but also software process artifacts (ex. transformation rules, UML profiles, specification). Thus software process is being reused.

This paper focuses on the Domain Engineering [6] process (MDA-DE) and the technological support used to enable it. Due to lack of space, the MDA-APP process will not be detailed in this paper. It is organized as follows: Section II presents some research into the support software development processes based on the MDA approach. Section III presents an overview of Modden approach and section IV details the MDA-DE process specification and automation. Section V shows an evaluation of the approach in an example of use; and finally Section 6 concludes this work and presents the next steps to be taken in this research.

II. RELATED WORK

In recent years a number of research initiatives related to software reuse have been proposed aimed at improving software development. Some of them integrate SPL and MDA approaches. For example, in [16] and [17] the authors proposed the use of feature-based model templates in model-driven development and software product line engineering contexts. The key-point in their work is to use a generative approach to support feature models, annotated models (as templates) and derived product models. They also developed supporting tools for the Eclipse environment and Rational Software Modeler. In summary, once a given feature model is created, it is used to generate a UML profile with stereotypes and constraints corresponding to the features. This profile can be applied in (structural and behavioral) UML models for annotating the model elements corresponding to each feature. This can be used as templates in order to generate SPL assets. Hence, depending on each product configuration, the UML models (as templates) will be transformed into specific UML models of a given product. However, this approach does not focus on a process definition. That is, there is no way to know and control how the SPL engineering is going on from a process-centered perspective, involving phases, tasks, roles and so on.

In [18], the authors presented an MDA framework for model-driven product derivation in the context of product line engineering. Bragança and Machado [1] propose automatic model transformations between use case models and feature diagrams for SPLs. However, [18] and [1] are other examples of tool-centered approaches integrating MDD and SPL which lack a process focus.

There have been some important contributions focusing on a process definition. However, some of these do not tackle every SPL engineering phase, or they just fail by lacking a standard language or notation in their definitions. For example, [19] presented the TENTE approach, which is a feature-oriented model-driven process to support SPL architectural design and implementation. Although they propose a process which covers domain and application engineering levels, it only encompasses architectural design and implementation stages. Despite being key-stages, other important SPL phases should be addressed. Moreover, the process was specified without a standard language or notation, lacking definitions regarding process roles and associated tasks, iterations and associated workproducts.

The DREAM approach [3] is another important contribution which details phases, process instructions and artifacts with corresponding representations in MDA, but it still lacks a standard notation for process representation.

This approach specifies what should be done in each phase. It does not indicate metamodels/profiles for each phase. It only recommends that the developer should use stereotypes and labeled values. Furthermore, it does not specify the rules, in other words, the automation needs to be built by the user according to the stereotypes that he uses. In a general way, it considers the process and artifacts that would be necessary to automate the SPL.

The Modden approach encompasses a complete process specification aimed at considering not only every SPL phase but also integrating MDA concepts in order to generate artifacts for specific domains. These artifacts will be available for reuse as well as for the construction of specific systems. This approach defines a process which is supported by the Modden Tool, based on [10] and, at the same time, it allows process instantiation to fulfill domain specific requirements. For example, it is possible to modify the process editing/creating new tasks or inserting new diagrams to support specific domains. Furthermore, the Modden process is specified by the use of well-known underlying standards such as SPEM and UML. Consequently, it not only allows better understanding and communication about the process but it also facilitates process enactment [25]. However Modden does not entirely support traceability between artifacts.

III. MODDEN APPROACH

This section presents the Modden approach, focusing on integration of software product line development and MDA. It aims to address reuse at different abstraction levels and the support of a software development process.

A. Modden Overview

Figure 1 gives an overview of the Modden approach. As mentioned before, it comprises two processes, named MDA-DE and MDA-APP. These two processes were specified using SPEM 2 [8] concepts and terminologies. The MDA-DE is a Domain Engineering process designed to develop reusable core assets. It has four phases, following the MDA structure, named CIM-DE, PIM-DE, PSM-DE and Code. Each one of these phases generates artifacts that are stored in the Asset Repository to be reused. The MDA-APP process is an Application Engineering process that develops a specific asset for the product line. Similarly, MDA-APP has four phases – CIM-APP, PIM-APP, PSM-APP and Code –, that use the artifacts stored in the Asset Repository.
The processes are supported by profiles and transformation rules. Profiles are necessary to enable specific knowledge modeling, for example variability and commonalities in feature diagrams. The transformation rules are executed by the transformation engines. We specified a profile that encapsulates feature concepts (to enable SPL modeling) and a rule that creates a profile according to the feature model. These profiles are responsible for preserving traceability between models and can be used not only in a new modeling task, but also in the maintenance activity.

These processes are supported by (i) the Modden tool which enables process editing and enactment; (ii) UML editors; and (iii) tools to execute the transformation rules.

B. MDA-DE Process

Our approach is based on SPEM/MDA metamodel [4] that specializes a SPEM metamodel for MDA context. Using this metamodel the specification could be enriched considering that we have used the stereotypes which were created from MDA concepts. The need for a language or notation especially for process modeling is not new. Process Modeling Languages (PML) have been proposed as a particular language to model and describe software processes. In this context, PMLs can be used for different purposes [26]: process understanding, design, training and education, simulation and optimization, support and enactment. Figure 2 shows a SPEM/MDA metamodel instance, that partially describes MDA-DE process.

In this context therefore, the method content is a library that describes reusable definitions such as Disciplines, Tasks, WorkProducts (specialized in SPEM/MDA metamodel as UML model, profile, transformation rule and extraModel) and Roles that will be used during the process life cycle. A Role identifies skills and responsibilities for an individual or a group. Each individual performs Tasks, a piece of work, during the process execution.

These Tasks are grouped into major areas, named disciplines, which may consume/produce WorkProducts. WorkProducts are related to every domain engineering artifact and are stored in the Asset Repository. In this approach, the WorkProduct is specialized into four kinds of artefacts: the UMLModel, produced by a process role or automatically generated by a transformation during the process execution; the TransformationRule contains the rules for model transformation and code generation during the process execution; the ExtraModel is used only for documentation and is based on text or supplementary notations; and Profile to represent an UML profile to base the modeling on each phase. Transformation rules are used in the MDA process to automatically transform UML models. Each transformation rule should refer to at least one source model and generate one or more target models.

A Process has a life cycle composed of a set of sequential Phases performed in Iterations. In terms of MDA, these phases represent the modeling of CIM (Computational Independent Model), PIM (Platform Independent Model), PSM (Platform Specific Model) and Codification. Thus, in the SPEM/MDA metamodel it is possible to stereotype these phases as <CIM>, <PIM>, <PSM> and <Codification>. Each Modeling Phase can be associated to UML profiles defined to address the specific characteristics of a particular domain or platform.

Following these definitions, the MDA-DE process structure is divided into Static and Dynamic dimensions, according to SPEM terminology the method content and process respectively.

For the MDA-DE method content ten disciplines were defined: (1) domain scope (2) asset search (3) core asset specification (4) domain analysis and design (5) domain commonalities and variability specification (6) variability management (7) asset implementation (8) asset cataloging, (9) V&V- validation and verification and (10) automation.

Roles are associated to individuals when accomplishing activities. In our MDA-DE process six roles are defined: (i) domain specialist, individuals who are familiar with the
domain concepts (ii) users, who know the domain through the use of existing systems (iii) system analyst, who is responsible for technical definitions during domain analyses (iv) architect, who is in charge of the domain design (v) product developer, who is responsible for the codification (vi) tester, who is in charge of assets testing.

For each task of each discipline we indicate the input and output workproducts. We also specify the steps that should be performed to complete the task, guiding the user during the process execution. For example, in the task Commonalities and variabilities identification, the input workproduct is the Use Case model and we have specified five steps: (1) identify related use cases (2) create a feature on the feature model to group these use cases (3) identify the variabilities associated to this new feature (4) classify features according to the profile (i.e. variable, external) and (5) identify the relationship between the features (i.e. mutex, require).

The Dynamic dimension represents the process life cycle phases and their related iterations. The MDA-DE process comprises four phases: (i) CIM-DE that models domain requirements (ii) PIM-DE which is related to the domain design artifacts (iii) PSM-DE which is related to the domain design in a specific platform and (iv) Coding, where the asset implementation is fulfilled and kept for further reuse. Each phase may have one or more iterations and comprises the discipline execution. Each discipline can be performed in one or more phases according to the tasks. Each discipline which was specified in the method content can appear in different process phases.

After each iteration every workproduct produced by the MDA-DE process is stored in the Asset Repository and remains available to be reused throughout the development stages of any other domain application.

Figure 3 summarizes the CIM phase in terms of iterations, tasks and workproducts produced by the task (in and out workproducts).

As can be seen, the first iteration (it_comprehension) aims to understand the product line needs and comprises two tasks: scoping definition and viability study.

The scoping definition task starts with the identification of the products that will make up the product line followed by the feature elicitation. As a result, the domain specialist produces the SPL objective declaration, a glossary and a product map that associates each product to its features.

The viability study task starts with the cost definitions in terms of hardware, software and human resources. It is important to analyze the product lines viability reports that already exist in the company repository.

Time viability and norm evaluations may influence SPL success and should be analyzed. As a result, the domain specialist constructs the report that will indicate the SPL viability.

The second iteration (It-analyzes) aims to detail the SPL needs at a high abstraction level (computationally independent) and comprises five tasks: domain requirements definition, CIMtoCIM refinement, conceptual modeling, commonalities and variabilities identification, and CIM to PIM transformation.

The domain requirements definition task starts mapping the requirements in terms of functional and nonfunctional requirements. Similar requirements are filtered (to be sure that they have something different), grouped and prioritized by the domain specialist. They are represented on a use case model stereotyped by the SPL profile which is defined in section IV A). This stereotype will be used by the refine CIMtoCIM task in order to generate a preliminary feature model.

In the conceptual modeling task the domain specialist analyzes the use case description and creates the conceptual class model. Besides this, in the commonalities and variabilities identification task the domain specialist completes the generated feature model.

In the CIMtoPIM transformation the following main transformations are performed: the feature model is mapped...
to a profile that will be used in the next phases to stereotype the workproducts in order to trace the models that contains each feature; and the conceptual class model is mapped to a 3 tier architecture class and component models as the input of the PIM phase.

IV. PROCESS AUTOMATION

The proposed approach was automated using Modden Tool that is based on [10]. This is a tool which was built on top of the Eclipse environment. In addition, it was based on SPEM concepts that enable general MDA process editing and enactment. Modden Tool comprises two main modules: (i) the process editor, in order to support MDA processes specification; and (ii) the process executor, which automates the process enactment. The adoption of this tool enables Modden approach editing, the automation of transformation rules and overall environment customization.

The first process step is the MDA-DE and MDA-APP specifications. In order to support SPL concepts, for example the feature model, [4] was extended in Modden Tool to support the use of a SPL profile (detailed in the following section). Furthermore, a plug-in was implemented in order to support diagram modeling according to this profile using GMF editor. Special transformation rules in ATL [24] were also implemented to fulfill the SPL requirements. Although QTV is the OMG standard for model to model transformation [8], its engines do not have the same stability, availability and maturity as ATL has. So, for this moment ATL was chosen and used. The following sections present the SPL profile, the transformation rules and the MDA tools which were adapted to support Modden.

A. SPL Profile

To support the process which was specified in section III, a product line profile was defined based on [15] providing the primary elements of Modden approach. In the next versions we are planning to evaluate others proposals in order to extend the specified profile. The profile comprises a metamodel, as illustrated in Figure 4, and a stereotype table (Table II). [13] and [14] define some concepts that were used to design the metamodel. Similarly, [15] specifies some stereotypes that were used to compose the stereotype table associating them to the diagrams which were used in the
proposed processes. Our profile differs from [21] with regard to the feature specializations (external, similar, alternate), as well as the specification of the relationships between features such as mutex and require. By adding extra information we can better generate new low level diagrams through the transformation rules.

As can be seen in Figure 4, the main concept of the metamodel is a feature which can be specialized as similar, variable or external. A feature stereotyped as Similar can appear in all the products in the same way. A variable feature is specialized as variationPoint and variant. They represent the variation point for each product in a SPL and its variation alternatives respectively. Furthermore, a variationPoint may comprise several variants. A variant can be optional, because it can be used or otherwise in a specific product; mandatory, when it is always used in the products; or alternative, indicating related features that can be selected or not in a product. These features can be exclusive, alternative_xor, or inclusive, alternative_or. Moreover, it is possible to represent some relationships such as require, to express that the related variant is required; and mutex, to express that a specialized feature can appear in all the products in the same way. A variable feature is specialized as variationPoint and variant. They represent the variation point for each product in a SPL and its variation alternatives respectively. Furthermore, a variationPoint may comprise several variants. A variant can be optional, because it can be used or otherwise in a specific product; mandatory, when it is always used in the products; or alternative, indicating related features that can be selected or not in a product. These features can be exclusive, alternative_xor, or inclusive, alternative_or. Moreover, it is possible to represent some relationships such as require, to express that the related variant is required; and mutex, to express that the related variants cannot be used together.

The SPL stereotype table definition shows the stereotype, the associated use case and class diagrams as well as a short description (partially described in Table I). For example the stereotype <<variationPoint>> in the class diagram and in the use case diagram can be used to represent an aggregation, a composition or an association. Similarly, the <<variant>> stereotype is used in the class and use case diagram as an association and represents a system variation.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Association</th>
<th>Association</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;feature&gt;&gt;</td>
<td>Inheritance</td>
<td>Association, Spec./General. &lt;&lt;inclusion&gt;&gt;</td>
<td>Feature represents an abstract class to the defined features</td>
</tr>
<tr>
<td>&lt;&lt;similar&gt;&gt;</td>
<td>Inheritance</td>
<td>Association, Spec./General. &lt;&lt;inclusion&gt;&gt;</td>
<td>Similar represents a specialized feature</td>
</tr>
<tr>
<td>&lt;&lt;variable&gt;&gt;</td>
<td>Inheritance</td>
<td>Association, Spec./General. &lt;&lt;inclusion&gt;&gt;</td>
<td>Variable represents a specialized feature</td>
</tr>
<tr>
<td>&lt;&lt;Variation Point&gt;&gt;</td>
<td>Aggregation, composition, association</td>
<td>Association, Spec./General. &lt;&lt;includes&gt;&gt;</td>
<td>The &lt;&lt;variationPoint &gt;&gt; specialization represents a class that comprises a Variant collection 1..n &lt;&lt;variant&gt;&gt;</td>
</tr>
<tr>
<td>&lt;&lt;Variant&gt;&gt;</td>
<td>Association</td>
<td>Association, Spec./General. &lt;&lt;inclusion&gt;&gt;</td>
<td>Represents a System Variant</td>
</tr>
</tbody>
</table>

TABLE I. THE SPL STEREOTYPE TABLE DEFINITION

Figure 5 shows a code fragment of an ATL transformation module that transforms the feature model to the profile. In this fragment the profile is created with the name specified in the feature model and each feature is mapped to a stereotype in the UML notation.

B. Rule Definition

Based on the profile shown in the previous section, some specific rules were implemented in order to automate the MDA transformation chain. We implemented rules to refine the CIM model (i.e. transforming a use case model in a feature model) and rules to transform CIM to PIM models.

Figure 5, for example, shows a rule fragment that transforms a feature model to a profile. The feature model is an extended UML class model with the stereotypes defined in the SPL profile. One of the most important things in the Modden process is to trace the feature in the produced models through the transformation chain. Our proposal is to automatically generate a profile from the feature model and use this profile to stereotype the model elements. As a result when a rule is executed the target models will preserve the feature stereotype from the source models.

C. Implementation Issues

As mentioned earlier, Modden tool enables general MDA process editing and enactment. This environment is divided into two main modules.

The first module provides design and customization of MDA processes offering a graphic editor to model the process structure and behavior. The set of graphic editors allows engineers to model their processes according to the proposed approach. It is also possible to specify a process using a breakdown structure and automatically generate diagrams to represent it graphically. Once Modden process is already specified, it is possible to customize it in order to fulfill specific companies requirements.

The other module is for the MDA process enactment. At this stage our environment enables the registration of professionals of a software team and the roles assigned to each, the possibility to view all the process definitions (phases, iterations, artifacts etc.) including the tasks and their status, an integrated environment for UML modeling.
and model transformation execution, and a management view of process tasks and artifacts during the MDA software project execution. At the end of the process enactment Modden tool should provide all the models created by the project team, generated code and models, and the history of tasks (manually or automatically executed).

The MDA-DE Process edition begins with static definitions. Figure 6 (B) shows the MDA-DE static definition in terms of a class diagram with the discipline, their associated tasks and the input / output work products. For example, the Scope Definition discipline comprises two tasks: (i) viability study, which generates the viability report as output and (ii) scope definition, which also generates the Requirements and other artifacts as output. Figure 6(B) presents the process edition modeled in a class diagram and duplicated in a work breakdown structure as in Figure 6(A).

Figure 6(C) shows the associated tool pallet. Similarly, the dynamic Modden definition of the Domain Engineering process is represented in a class diagram which comprises the phases, iterations and the associated taskUse.

SPL and UML profile were used in the process editing associating it to the CIM and PIM phases. Furthermore, the rules specified in section 4 (subsection B) were implemented.

Based on the process definition specified in the Modden Tool Editor, the Executor module is ready for enactment. In order to adhere to the Model Approach this module was modified. The interface enables not only diagram customization using the SPL stereotypes but also the feature diagram modeling through the class diagram.

Figure 7 shows the process executor with the Modden approach previews. They are ready to be used in any SPL project. To start a new project the first activity is to assign the process roles to the specific user, as can be seen in Figure 7(B).

The whole process specified by the Modden Editor is shown in the left window. When taskUse is selected an action is executed. For example, when Commonalities and Variability identification taskUse is selected at the Analyse iteration in the CIM-DE Modeling Phase, a feature model editor is opened with the associated SPL profile definitions in the tool pallet. As a result, the model can be constructed. When the CIM to PIM transformation taskUse is selected the respective transformation rule is executed, and so on. In section V these examples are complemented with a case study description.

V. EXAMPLE OF MODDEN APPROACH

The Modden approach and tool have been used before in several experiments [22], but never in a SPL context. Evaluating a SPL approach, such as Modden, is not an easy task due to the complexity of identifying an adequate domain, considering time and variability restrictions.

In this scenario we decided to divide the evaluation into phases to better identify and deal with problems as they emerge.

In this section we present an example of Modden use that aimed to evaluate the applicability of the SPL UML profile in the diagram modeling and transformations. To do this, we executed the CIM phase of the MDA-DE process to produce assets related to the academic applications domain.
This evaluation does not aim to validate the effectiveness of the proposed process.

Figure 8 shows the domain requirements definition in a stereotypical use case model. For example it is optional for a student to do pre-registration, although it is necessary to register. These stereotypes are necessary to enable the transformation of this diagram into the feature model.

The use case diagram shown in Figure 8 was automatically transformed into the feature model presented in Figure 8 by the selection of the transformation taskUse in the process.

Figure 9 shows the academic domain feature model produced by the transformation execution. Classes in gray were automatically generated by the defined rule. Other classes were added by the user in order to complement the model.

After this the tasks which were specified on the MDA-DE process were followed until the end of the PIM phase where the component diagram was generated. The PSM and Coding phases of Modden approach are still in progress.

After this example we can affirm that the defined SPL profile has supported the diagrams modeling and rule specifications. Nevertheless, the example results have pointed out us some process issues to be adjusted and, at the same time helped us to validate the technologies used in this project.
For example the feature model notation for the SPL, the product mapping definition in scoping task and a profile definition using the specified feature model to trace the artifacts associated to a feature. In addition, it is essential that the overall process evaluation including the PSM and coding phases be carried out.

Through the complete Modden implementation, we hope to evaluate our hypothesis that the integration of both MDA and SPL approaches together could leverage software development processes.

VI. CONCLUSIONS AND FURTHER WORK

This work presented the Modden approach. This is based on SPL activities in order to support software development through Model Driven Architecture transformations. For such, two software processes were defined: MDA-DE to develop the reusable core assets and MDA-APP to develop specific products in the product line. This text details the MDA-DE process that constitutes our first step to evaluate the hypothesis that the integration of MDA and SPL approaches can enhance reuse during MDA processes transformations.

In order to achieve this, we specified: (i) phases, tasks, roles and workproducts using the SPEM standard (ii) a UML profile for SPL, providing a lightweight extension of the UML language allowing the modeling of variabilities and commonalities in UML diagrams and (iii) ATL rules to support process automation. Moreover, the approach was put into practice using the Modden tool to perform an initial evaluation. This automation was crucial to improve process usage “with” reuse as well as “to” reuse.

MDA-DE is designed to initiate artifact reuse in the CIM phase towards code. With Modden approach it is possible to use and automated the existing SPL practices in an integrated MDA process supported by an environment that provides an interface to model the diagrams and execute the rules to generate new artifacts. With the use of SPL profile it is possible to use UML class diagram to represent the feature model. As a result the feature model can be constructed in any compatible tool to be used in the transformation rules.

Modden enables not only reuse of code but also reuse of models once they are used as documentations as well as input artifacts that can be modified and processed by transformations to generate new products.

The transformation chain is already defined to enable this automation without needing the user to understand transformation languages. The process can be reused in several domains and even in the same domain at different moments, producing complementary artifacts that express additional features. Thus it is possible to reuse the process and not just the artifacts produced by this process.

The Modden approach and Tool is based on several open standards (UML, SPEM, MDA) and open source projects such as ATL Language. The use of graphics standards can enhance the interoperability of models proposed by MDA in a more extensive manner. Methodologies and tools that assist software development will have the same conceptual and notational framework. This aspect facilitates the understanding of models both by development teams and by process automation tools.

During the example presented we realized that some artifacts that were manually constructed by the user could also be generated by the proposed approach. For example, a preliminary release of the sequence model can be generated.
from the activity and class model. The construction of new transformation rules is still in progress.

After the project specification and automation, the next steps are related to process validation and platform testing. The platform has already been used in an MDA context, but additional tests are needed to validate the integration with SPL and the new process definitions. The artifact repository is also in construction due to the need to store, search and recover assets.

ACKNOWLEDGMENT

The Modden project has been partially supported by FAPESB (Fundação de Amparo à Pesquisa do Estado da Bahia) contract number 006/2008 from edit number 001/2008.

REFERENCES


