Odyssey-MEC: Model Evolution Control in the Context of Model-Driven Architecture

Chessman Corrêa Leonardo Murta Cláudia Werner

Federal University of Rio de Janeiro
COPPE - System Eng. and Computer Science
{chessman, murta, werner}@cos.ufrj.br

ABSTRACT

Model-Driven Development aims to use models as first class artifacts in software development. Therefore, the need to control model evolution in this context became as important as to control the evolution of source-code. In Model-Driven Architecture, a target model is generated from a source model through a transformation process. Consequently, there is a relationship among them. However, these models may evolve independently due to modifications, making them inconsistent with each other. In this scenario, traditional versioning is fundamental, but it is not sufficient to control the evolution of different interconnected models that represent the same software. In this paper, we propose a server side transformation, synchronization and versioning approach to control the evolution of models.

Keywords

Version Control, Model Versioning, Model-Driven Development, Model-Driven Architecture, Model Evolution.

1. INTRODUCTION

Model-Driven Architecture (MDA) is the Object Management Group (OMG) framework for **Model-Driven Development** (MDD) [18]. One characteristic of this approach is the generation of a target model from a source model using a transformation engine. It means that the software is represented by different models, most of the time in different abstraction levels.

In large software projects, multiple people assuming specific roles and located at different places may modify related models independently. For example, a PSM (Platform Specific Model) generated from a PIM (Platform Independent Model) may need to be modified because it does not have all the necessary details to derive the source-code. In other words, models may need to be updated in order to be used to generate other models or source code.

Since these models are related with each other, modifications applied to a model may create inconsistencies between them. However, as these models represent the same software, inconsistencies cannot be allowed. For example, inconsistencies between PIM and PSM introduce some difficulties to generate PSMs tailored to other platforms. This is especially true if PIM level changes are made in PSM instead. In this case, the generation of PSM to a new platform would not have PIM details that exist in the other platform. It means that if a MDD project aims to create software for different platforms, PSMs of each platform have to be consistent with the corresponding PIM and with PSMs of other platforms. It is also true for models in the same abstraction level.

Model versioning is essential to control model evolution. However, if source and target models are versioned independently, there will be no guarantee that they are consistent with each other. Since these models have to evolve together, versioning is not enough to control their evolution in MDA context. Therefore, these models have to be synchronized before versioned.

Models synchronization is achieved from round-trip engineering through bidirectional transformations that preserve previous versions of existing models. However, if a synchronization tool is not automatically executed, software engineers may forget to use them, leading to inconsistent models.

Based on these facts, this paper proposes a server side model transformation, synchronization and versioning approach to control model evolution in MDA.

The rest of this paper is organized as follows. Section 2 briefly describes the Model-Driven Architecture. Section 3 discusses the key aspects of our approach. Section 4 presents some related works. Finally, the conclusion and future work are presented in Section 5.

2. MODEL DRIVEN ARCHITECTURE

OMG was inspired by constantly shifting infrastructures, requirements changing, and new emerging technologies to create the **Model-Driven Architecture** (**MDA**) [18]. This approach considers models as first-class development artifacts and uses them not only for understanding and communication, but also for design, construction, deployment, operation, maintenance, and modification of a system.

2.1 MDA Models

MDA specifies four kinds of models: **Computation Independent Model** (CIM), **Platform Independent Model** (PIM), **Platform Model** (PM) and **Platform Specific Model** (PSM) [18]. **CIM** represents the system requirements. It takes into consideration domain concerns, such as the vocabulary used by the domain practitioners. CIM represents a view of the system without computational details. **PIM** is a view of the system considering computational solutions that aim to be generic to any platform. Thus, it represents a system that can be tailored to multiple platforms, assuming that these platforms are compatible to the architectural styles adopted in the corresponding PIM. **PM** provides the technical concepts, requirements, and services of a specific platform. **PSM** is a view of a system considering the platform details. It can be seen as a merge of PIM and PM, augmented by some changes specific to the target platform.

2.2 Model Transformation

Model transformation is the process of creating a target model from a source model of the same system. Although it could be made manually, the MDA approach aims at automating this operation. This is a key factor to the increasing MDA adoption over traditional software development.

In **forward engineering**, a model-to-model transformation uses the CIM and other information to generate a PIM. Subsequently, another model-to-model transformation combines PIM and PM to create the corresponding PSM. Finally, PSM is used by a model-to-text transformation to generate the source-code to the specific software platform.

A model transformation uses **mappings** to create target model elements from source model elements. Mappings provide specification of how one or more target elements are derived from source elements. It also may have mapping rules based on specific **marks**, like stereotypes and tagged values. For example, a PIM class with the stereotype <<*entity*>> may generate an EJB (Enterprise JavaBean) class for the JEE[1] platform.

During model generation, the model transformation should also generate the **record of transformation**. It includes the **traceability links** between source and target model elements and informs which parts of the mapping were used during the generation. It is an important resource to support synchronization.

It is important to notice that CIM, PIM and PSM are in different abstraction levels. This mean that CIM-PIM and PIM-PSM transformations are vertical transformations among different abstraction levels. However, it is also possible to generate models in the same abstraction level through horizontal transformations, such as PIM-PIM and PSM-PSM.

2.3 MDA Application

The application of MDA is relatively simple. It can be divided into two main phases: infrastructure setup and transformation. The infrastructure setup starts with the creation of mappings based on a platform or a set of platforms. These mappings will be used by transformations. In addition, the marks to be applied on a PIM may also be defined, usually through a Profile [19].

After this initial setup, the software engineer uses a modeling tool to create a model (e.g., a PIM). Afterwards, that model may be marked according to the available Profiles. Finally, the transformation is executed, using the mappings and the marked model to generate the corresponding model (e.g., a PSM) and the record of transformation.

This scenario focuses on forward engineering. However, it is also possible to occur reverse engineering transformations, generating a PIM from a PSM.

3. ODYSSEY-MEC

In this section we introduce Odyssey-MEC (Odyssey for Model Evolution Control), a server side transformation, synchronization and versioning approach to control MDA models evolution.

In the following, we detail our approach presenting its architecture, model infrastructure, model repository, model versioning, model transformation, record of transformation, element search, and model synchronization.

3.1 Architecture

The architecture of the approach is shown in Figure 1. It has four types of repositories: Transformation Mappings, PIM, PSM, and Record of Transformation. The **Transformation Mappings Repository** (TMR) stores the transformation mappings to be used by the transformation engine. These transformation mappings are created by a transformation engineer, as specified by Bacelo et al [1]. **PIM and PSM Repositories** store PIMs and PSMs, respectively. It is worth to notice that these repositories persist versioned models. Moreover, each platform has its own PSM repository. **Record of Transformation Repository** (RTR) stores the Record of Transformations (RT).

Our approach comprises three main components to control model evolution: Odyssey-VCS [15, 17], Odyssey-MDA [1], and a synchronization engine (SE). It also uses a Transaction Manager (TM) component to control the synchronization and versioning process in a transaction context. The

[1] http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

Odyssey-VCS component is used for model versioning and the Odyssey-MDA component for model transformation.

Odyssey-VCS has hooks that execute the TM and the SE when a model is checked in. SE uses Odyssey-MDA to generate target models, and Odyssey-VCS to access the models to be synchronized and their versioning data. It also uses RT as an auxiliary resource to synchronize the models. Finally, an Odyssey-VCS client is used to communicate with Odyssey-VCS server (it can be any CASE tool that exports models through XMI 2.1 format). Odyssey-VCS client communicates with Odyssey-VCS server through Web Services [4].

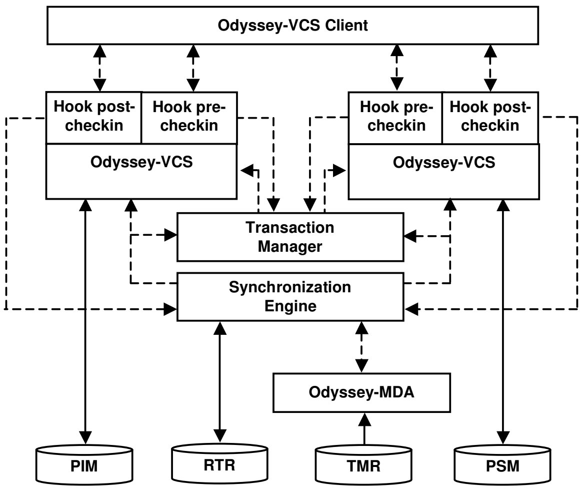

Figure 1. Odyssey-MEC Architecture

3.2 Model Infrastructure

OMG chose UML (Unified Modeling Language) as the standard modeling language for MDA. Therefore, our approach controls the evolution of UML models.

Although OMG uses MOF (Meta Object Facility) as UML meta-model, Odyssey-MEC uses the Eclipse Ecore meta-model [5]. The use of Ecore instead of MOF is not an obstacle to control the evolution of UML models because EMF uses XMI (XML Metadata Interchange) [20] for externalizing UML models. Client tools just have to use the same XMI version used by EMF (version 2.1).

3.3 Model Repository

Model repositories are used to store models. In our case, it is necessary to store all versions of a model to control the model evolution.

Due to the lack of versioning repositories for EMF, we adopted Odyssey-VCS as our versioning component, as detailed in Section 3.4.

3.4 Version Control

Version control is a key resource to control the evolution of models during development and maintenance. It is used to generate a history of model versions and maintain information like when, why, and who has made modifications. This history of model versions and modification information are stored in a repository. The basic functionalities of version control systems are: check-in (save a model into the versioned repository), check-out (get a model from the versioned repository), merge (join two models) and detect conflicts (identify concurrent modifications that cannot be resolved)[2].

Our model-based version control component is Odyssey-VCS. This component has a client/server architecture and offers all the requirements discussed above. It uses the EMF reflective API to support the versioning of any UML model element[2]. It can also execute external code trough hook implementation.

Odyssey-VCS works at fine-grained model versioning. This means that it is capable of identifying a new version of any UML model element. When a model element is composed from other model elements, if one of these elements is changed, the composing model element also receives a new version number. This is propagated recursively up to the outer model element, frequently a model package.

3.5 Model Transformation

Model transformation depends on a set of mappings and rules to create elements in a model from elements of another model. There are different ways to generate a new model using transformations [18]. Some existing approaches to model-to-model transformations are: ATL (ATLAS Transformation Language) [11], Triple [3], OptimalJ [7]. UMT [16], UMLX [8], and Odyssey-MDA[1]. A further discussion about transformation can be found in [21].

One of the requirements for controlling model evolution is the support for bidirectional transformations. This means that transformations should be able to generate PSM elements from PIM elements, and PIM elements from PSM elements. This feature is needed because different people may be working over different models, and new elements inserted in a PSM may have to be represented in its corresponding PIM. From the approaches presented above, ATL [11] and Odyssey-MDA [1] allow transformations in both directions. However, ATL requires the writing of a particular transformation mapping for each direction. On the other hand, Odyssey-MDA allows the specification of bidirectional transformations in the same mapping. In addition, it is also shipped with a tool for model marking, named ModelMarker. Due to that, we adopted Odyssey-MDA as our transformation engine component.

[2] A model element is any UML element defined in its metamodel, for example, a class, attribute, operation, component, association, etc.

Odyssey-MDA is capable to execute vertical and horizontal transformations. Therefore, although this paper is focused in vertical transformations, our approach can also be applied to control the evolution of models at the same abstraction level.

3.6 Record of Transformation

A record of transformation (RT) [18] is used to identify source models from target models and vice-versa. This is a very important resource to synchronize models, as it helps to identify existing model elements that have to be updated instead of being overwritten. Traceability links are particularly important for the synchronization activity when some relevant information is lost during the transformation [22].

In Odyssey-MEC approach, RTs are represented as a Traceability Links (TL). This is an Ecore model element that we created to reference the source and target model elements and the mapping that was used to generate the target element. Traceability links are generated by our Odyssey-MDA component during model transformations. Since more than one source or target model may be involved in transformation, it is possible that more than one traceability link references the same source element or target element.

3.7 Transaction Control

The synchronization and versioning of source and target models should be performed in a transaction context. In other words, if one of these steps fails, the whole process has to be canceled to avoid model inconsistencies.

To solve this problem, we adopted a Transaction Manager (TM) component that implement a two-phased transaction commit. When a model is checked in, the Odyssey-VCS pre-checkin hook uses TM to verify if there is any existing transaction in progress. If not, it asks for a new transaction and informs Odyssey-VCS that the model can be versioned. Odyssey-VCS starts its own transaction to create the model version. After versioning the model, Odyssey-VCS executes the post-checkin hook. This hook initiates the transformation and synchronization process. During this activity, other Odyssey-VCS instances may start their own transactions, as well as RTR. If all Odyssey-VCS instances finish their transactions successfully, TM navigates trough all Odyssey-VCS instances asking them to confirm their transactions. This also happens with RTR. Finally, the global transaction is confirmed.

3.8 Element Search

The versioning process depends on finding prior element versions. The synchronization process depends on finding PIM and PSM elements that have a trace relationship. Due to that, our element search occurs in two dimensions: time (different versions) and space (different models).

UML model elements are identified in XMI files by unique identifiers. Unfortunately, most tools do not preserve the value of these ids when models are exported. Therefore, this identifier cannot be used do identify model elements. To solve this problem, Odyssey-VCS uses a unique identifier as a tagged value.

The Odyssey-VCS meta-model has an element called Version. This element represents a version of a UML model element, and stores some versioning data, such as the element version number. It also has references to the UML model element it represents and references to the prior and next versions. Therefore, there is a list of versions for each element, which constitutes the element version history. This version history is useful to find prior and next versions of an element. However, due to the use of separate repositories for PIM and PSM, elements in different models have their own version history.

The combination of version history list and traceability links can be seen in Figure 2. Together, these two references make it possible to freely navigate from one version to another and from an element of a model (e.g., PIM) to another element of another model (e.g., PSM). This capability supports the versioning and synchronization processes discussed in Section 3.9.

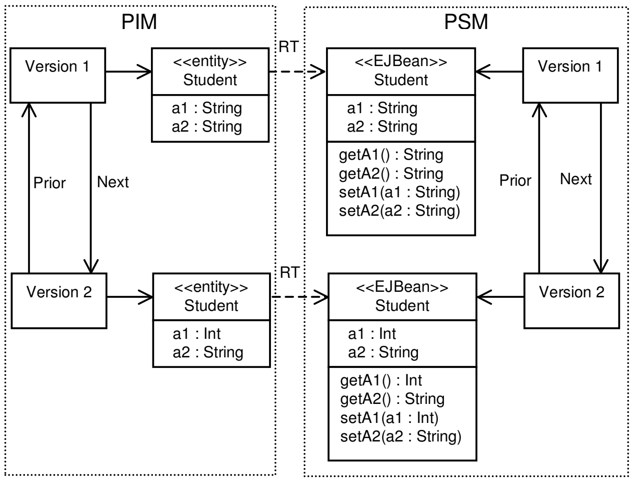

Figure 2. Version, PIM and PSM references

3.9 Model Synchronization

Interrelated models have to be consistent with each other. Therefore, it is necessary to synchronize them during development and maintenance, but preserving prior modifications. The ability to automatically synchronize models without information loss is called **roundtrip engineering** [22], and the lack of this ability usually leads to legacy systems [13].

Odyssey-VCS is designed to control the evolution of independent models. It means that this component alone is not capable of controlling the evolution of models that have traceability links among them. Therefore, it is necessary to adopt a synchronization engine together with Odyssey-VCS. This synchronization engine is triggered by Odyssey-VCS hooks.

The synchronization engine, which is a component of Odyssey-MEC, depends not just on PIM and PSM version control information, but also on existing record of transformations of prior PIM and PSM versions. It also depends on Odyssey-MDA to generate models.

When a model is checked in, Odyssey-VCS tries to create a new version of the model. The Synchronization Engine (SE) is executed only if there is no version conflict during the versioning process. This avoids synchronization effort in cases of conflicts. If there is no conflict, SE selects the transformation mapping to be used and sends it to Odyssey-MDA, together with the new model version that was checked-in. Odyssey-MDA generates the target model (TM) and the RT of each target element, and returns them to SE. SE uses Odyssey-VCS of the target element to verify if there is any existing version available. If no previous version is found, SE considers the target model as a new model. In this case, SE checks in the target model using the Odyssey-VCS repository designated to it.

If there is an existing version of the generated target model, SE has to pre-process it in order to allow Odyssey-VCS to match the model with its prior version. This pre-processing starts with the recovery of versioning information. After that, the versioning information is interwoven into the generated target model.

This pre-processing process is composed of the following steps: (1) SE navigates trough all elements of the generated target model; (2) Using the traceability link, SE finds the related source model; (3) SE searches for the most recent version that has a traceability link dependency to an element of the target model; (4) When this element is found, SE retrieves its version information and puts into the respective generated target model element.

After the process is finished, SE checks in the model. It is worth to notice that, at this moment, the generated target model has all the necessary versioning information to allow Odyssey-VCS to interpret it as a new version of a model under version control. The generation of a new version of the model element means that the differences between the existing version and the checked-in version were merged. In other words, the synchronization was performed. If some conflicts occur during this merge process, all the operations are canceled.

When a source element has traceability links to more than one target element, the part of the transformation used to generate the target element is used to identify the correct element. This information is specified together with the traceability link that exists between the source and target models.

4. RELATED WORK

Gîrba et al. [10] proposes Hismo as a meta-model based solution to control model versions. However, this approach does not take into consideration synchronization and does not support UML models.

Matheson et al. [14] proposed an architecture for capturing models evolution in MDD. They suggest the use of a repository centric solution that is independent from client tools and stores model versions and their relationships in fine granularity. XMI is proposed as the data exchange mechanism for UML artifacts, and it uses XML and XML Schema to specify the transformation specifications. Besides the similarities with our approach, nothing was mentioned about the execution of model synchronization and model versioning.

There are some other researches [6, 9, 12] that take model evolution into consideration in some different ways, but do not consider versioning and model synchronization, as we do in our work.

5. CONCLUSIONS

This paper presented an approach to control the evolution of MDA models considering the model synchronization and versioning in a client/server architecture. Therefore, any CASE tool that can export models using XMI format is a potential client to Odyssey-MEC.

The way that PIM and PSM are versioned in Odyssey-MEC eliminates the need of any special mechanism to synchronize them. This synchronization is made when Odyssey-VCS merges the model that is being checked in with its last available version.

The client/server architecture of Odyssey-MEC makes it possible to implement distributed MDD using the MDA approach. The automatic model synchronization avoids the errors that can be introduced during manual synchronization. It also guarantees that models will always be consistent.

Although this paper focused on PIM and PSM, models in the same abstraction level may also be generated, synchronized and versioned. This can be done via horizontal transformation definitions during the MDD project creation. Moreover, we were mostly focused in this paper on PIM and PSM synchronization and versioning. Therefore, only two abstraction levels where considered. However, the approach works with unlimited abstraction levels. In this case, a PSM can be considered a PIM for the next abstraction level. It is also possible to support PSM for multiple platforms.

Currently, Odyssey-MDA works just with static models (i.e., class and component models). It means that Odyssey-

MEC cannot synchronize dynamic models, such as sequence model. Nevertheless, Odyssey-VCS can still be used to version control these models, but without synchronization among them. Moreover, the current version of Odyssey-MDA is able to deal with just one model as input and generates another model as output. Therefore, Odyssey-MEC supports model evolution in a one-to-one basis.

Our next step is to evaluate the proposed approach by applying some selected cases that will take into consideration conflict resolutions, forward and reverse transformations, transformation mapping change, etc. The results will be evaluated through precision and recall analysis [23], comparing them to the expected values.

As future work, we intend to: (1) expand our support to CIM and source-code; (2) develop an additional tool to help de visualization of MDA models evolution during the project execution and system maintenance; (3) control the evolution of transformation mappings and register in the RT the version of the transformation mapping used during the transformation; (4) expand our support to other UML models, such as the behavioral models; (5) modify Odyssey-MDA to receive and generate more than one model; and (6) use rules do control modifications that can be applied on interrelated models.

6. ACKNOWLEDGMENTS

Our thanks to the members of the Software Reuse Group at COPPE/UFRJ, especially Hamilton Oliveira, Cristine Dantas, Luiz Gustavo Lopes, João Gustavo Prudêncio, and Natanael Maia, who contributed to Odyssey-VCS and Odyssey-MDA. We also want to thank CNPq for the financial support.

7. REFERENCES

1. Bacelo, A., Maia, N. and Werner, C.M.L., Odyssey-MDA: A Transformational Approach to Component Models. in *Proceedings of Conference on Software Engineering and Knowledge Engineering*, (Boston, USA, 2007), 9-14.
2. Berczuk, S. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration* Addison-Wesley, Boston, MA, USA, 2002.
3. Billig, A., Busse, S., Leicher, A. and Süb, J.G., Platform Independent Model Transformation Based on Triple. in *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, (Toronto, Canada, 2004), 493-511.
4. Booth, D., Hass, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchand, D. Web Services Architecture - W3C Working Group Note, World Wide Web Consortium (W3C), 2005.
5. Budinsky, F., Steiberg, D., Merks, E., Ellersick, R. and Grose, T.J. *Eclipse Modeling Framework: A Developer's Guide*. Addison Wesley, 2003.
6. Chen, F., Yang, H., Qiao, B. and Chu, W.C.-C., A Formal Model Driven Approach to Dependable Software Evolution. in *Proceedings of 30th Annual International Computer Software and Applications Conference - Cover*, (Chicago, Illinois, USA, 2006), 205 - 214.
7. Compuware. OptimalJ - Model-driven Java Development Tool, 2007.
8. Eclipse. UMLX A Graphical Transformation Language for MDA, 2007.
9. Engels, G., Küster, J.M., Heckel, R. and Groenewegen, L., Towards Consistency-Preserving Model Evolution in *Proceedings ICSE Workshop on Model Evolution*, (Florida, USA, 2002), 129-132.
10. Girba, T., Favre, J.-M. and Ducasse, S.e. Using Meta-Model Transformation to Model Software Evolution. *Electronic Notes in Theoretical Computer Science*, *137*. 57-64.
11. Jouault, F. and Kurtev, I., Transforming Models with ATL. in *Proceedings of the Model Transformation in Practice Workshop at MoDELS*, (Montego Bay, Jamaica, 2005), 128-138.
12. Lin, Y. and Gray, J., A Model Transformation Approach to Automatic Model Construction and Evolution. in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, (Long Beach, CA, USA, 2005), ACM, 448-451.
13. Maciaszek, L.A., Roundtrip Architectural Modeling. in *Proceedings of the 2nd Asia-Pacifc Conference on Conceptual Modeling*, (Newscastle, Australia, 2005), Australian Computer Society, Inc. , 17-23.
14. Matheson, D., France, R., Bieman, J., Alexander, R., DeWitt, J. and McEachen, N., Managed Evolution of a Model Driven Development Approach to Software-based Solutions. in *Workshop on Best Practices for Model Driven Development*, (Vancouver, Canada, 2004).
15. Murta, L.G.P., Dantas, H.L.R., Lopes, L.G.B. and Werner, C.M.L. Odyssey-SCM: An Integrated Software Configuration Management Infrastructure for UML Models. *Science of Computer Programming*, *65* (3). 249-274.
16. Oldevik, J. UML Model Transformation Tool - Overview and User Guide Documentation, 2004.
17. Oliveira, H., Murta, L. and Werner, C.M.L., Odyssey-VCS: a Flexible Version Control System for UML Model Elements. in *International Workshop on Software Configuration Management (SCM-12)* (Lisbon, Portugal, 2005), 1-16.
18. OMG. MDA Guide Version 1.0.1, Object Management Group, 2003.
19. OMG. Unified Modeling Language (UML) Infrastructure Specification. Version 2.0, Object Management Group, 2006.
20. OMG. XML Metadata Interchange (XMI) Specification. Version 2.0, Object Management Group, 2005.
21. Sendall, S. and Kozaczynski, W. Model Transformation - the Heart and Soul of Model-Driven Development. *IEEE Software*, *20* (5). 42-45.
22. Sendall, S. and Küster, J., Taming Model Round-Trip Engineering. in *Workshop on Best Practices for Model-Driven Software Development*, (Vancouver, Canada, 2004).
23. Yates, R.B. and Neto, B.R. *Modern Information Retrieval*. ACM press, 1999.