Benefits of reverse engineering technologies in software development makerspace

M.H. Aabidi[1,*], *Bouziane El Mahi*[1], *Chafik Baidada*[1], *Abdeslam Jakimi*[1], *and Hany Ammar*[2]

[1]Software Engineering & Information Systems Engineering Team, Computer Sciences Department, Moulay Ismaïl University, FST Errachidia, Morocco
[2]West Virginia University, Lane Department of Computer Science and Electrical Engineering, Morgantown, USA

Abstract. In the recent decades, the amount of data produced by scientific, engineering, and life science applications has increased with several orders of magnitude. In parallel with this development, the applications themselves have become increasingly complex in terms of functionality, structure, and behavior. In the same time, development and production cycles of such applications exhibit a tendency of becoming increasingly shorter, due to factors such as market pressure and rapid evolution of supporting and enabling technologies. As a consequence, an increasing fraction of the cost of creating new applications and manufacturing processes shifts from the creation of new artifacts to the adaption of existing ones. A key component of this activity is the understanding of the design, operation, and behavior of existing manufactured artifacts, such as software code bases, hardware systems, and mechanical assemblies. For instance, in the software industry, it is estimated that maintenance costs exceed 80% of the total costs of a software product's lifecycle, and software understanding accounts for as much as half of these maintenance costs. To facilitate the software development process, it would be ideal to have tools that automatically generate or help to generate UML (Unified Modeling Language) models from source code. Reverse engineering the software architecture from source code provides a valuable service to software practitioners. Case tools implementing MDA and reverse-engineering constitute an important opportunity of software development engineers. So MDA and reverse engineering is an important key witch make makerspace more productive and more efficient.

1. Introduction

The architecture of a system is a specification of software components, interrelationships, and rules for component interactions and evolution over time. In 2001 OMG adopted an architecture standard, the Model Driven Architecture (MDA). With the emergence of internet applications, the interoperability problem moved from the integration of platforms and programming languages on a company intranet to the integration of different middleware on the Internet. In this situation, the middleware is part of the problem itself [1]. The original inspiration around the definition of MDA had to do with this internet middleware integration problem. Apart from interoperability reasons, there are other good benefits to use MDA such as to improve the productivity, code and processes quality and, software maintenance costs. MDA is an architectural framework for improving portability, interoperability and reusability through separation of concerns. It uses models to direct the complete lifecycle of a system; all artifacts such as requirement specifications, architecture descriptions, design descriptions and code, are regarded as models. MDA provides an approach for specifying a system independently of the platforms that it supports, specifying platforms, selecting a particular platform for the system, and transforming the system specification into one implementation for the selected particular platform. It distinguishes Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) and Implementation Specific Model (ISM). The Unified Modeling Language (UML) [2,3] combined with the Object Constraint Language (OCL) [4] is the most widely used way for writing either PIMs or PSMs. Model Driven Development (MDD) refers to a range of development approaches that are based on the use of software models as first class entities. MDA is the specific realization of MDD proposed by OMG. It is carried out as a sequence of model transformations: the process of converting one model into another one of the same system preserving some kind of equivalence relation between them. The idea behind MDA is to manage the evolution from CIMs to PIMs and PSMs that can be used to generated executable components and applications. The high-level models that are developed independently of a particular platform are gradually transformed into models and code for specific platforms. The concept of metamodel, an abstract language for describing different types of models and data, has contributed significantly to some of the core principles of the emerging MDA. The Meta Object Facility (MOF), an adopted OMG standard, (latest revision MOF 2.0) provides a metadata management framework, and a set of metadata services to enable the development and interoperability of model and metadata driven systems [5]. MDA reverse engineering can be used to recover architectural models of legacy systems that will be later used in forward engineering processes to produce new versions of the systems. OMG is involved in a series of standards to successfully modernize existing information systems. Modernization supports, but are not limited to, source to source conversion, platform migration, service oriented architecture migration and model driven architecture migration. Architecture Driven Modernization (ADM) is an OMG initiative related to extending the modeling approach to the existing software systems and to the concept of reverse engineering [6]. One of ADM standards is Knowledge Discovery Metamodel (KDM) to facilitate the exchange of existing systems meta-data for various modernization tools [7]. The following section presents the concepts of model, metamodel, transformation, MDA and reverse engineering in more details and finally presents our study and how to share it in makerspaces.

2. MDA AND REVERSE ENGINEERING

To understand automated reverse engineering, we must first understand model driven development/architecture [8] [9] and the transformation framework. Model driven development and code generation from models is called in literature forward engineering. In a model driven development approach, given two meta-models, i.e., a source meta-model and a target meta-model and the transformation rules that can transform the source meta-model into the target meta-model, any given platform independent model that adheres to the source meta-model can be translated into a platform specific model (PSM) that adheres to the target meta-model. The resulting PSM can then be translated into various implementation artifacts on the target platform. This is called forward engineering. By reversing this approach, platform independent models can be extracted from platform specific models and implementation artifacts. Extraction of models from existing artifacts of a business application is termed reverse engineering.

Figure 1 shows forward engineering transformation approach and reverse engineering transformation approach. The figure represent software transformations that automatically translate artifacts on the left to the artifacts on the right of the arrows they reside. Model transformation is the process of converting one model into another model of the same system preserving some kind of equivalence relation between both of these models.

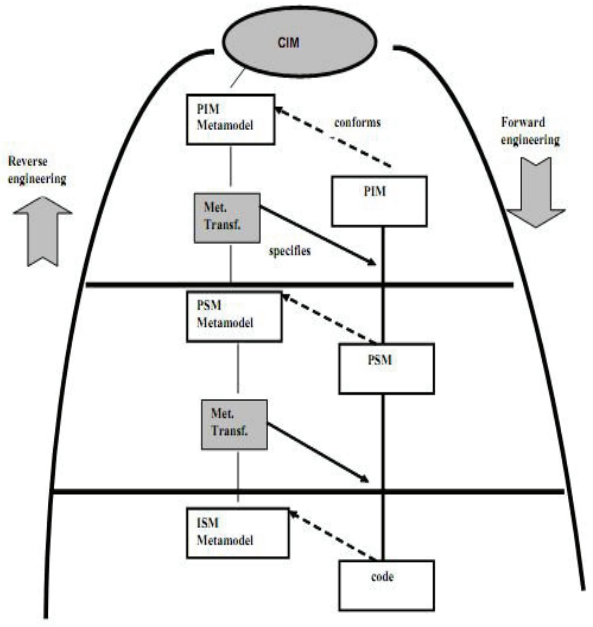

Fig. 1. Model, metamodels and transformations

The idea behind MDA is to manage the evolution from CIMs to PIMs and PSMs that can be used to generate executable components and applications. The high-level models that are developed independently of a particular platform are gradually transformed into models and code for specific platforms. The transformation for one PIM to several PSMs is at the core of MDA. A model-driven forward engineering process is carried out as a sequence of model transformations that includes, at least, the following steps: construct a CIM; transform the CIM into a PIM that provides a computing architecture independent of specific platforms; transform the PIM into one or more PSMs, and derive code directly from the PSMs.

The success of MDA depends on the existence of CASE tools that make a significant impact on software processes such as forward engineering and reverse engineering processes, however all of the MDA tools are partially compliant to MDA features. The major developments taking place in the framework of the Eclipse project [10]. For instance, the Eclipse Modeling Framework (EMF) was created for facilitating system modeling and the automatic generation of Java code. The major developments taking place in the framework of the Eclipse project [10]. For instance, the Eclipse Modeling Framework (EMF) was created for facilitating system modeling and the automatic generation of Java code. EMF started as an implementation of MOF resulting Ecore, the EMF metamodel comparable to EMOF. EMF has evolved starting from the experience of the Eclipse community to implement a variety of tools and to date is highly related to Model Driven Engineering (MDE). For instance, ATL (Atlas Transformation Language) is a model transformation language in the field of MDE that is developed on top of the Eclipse platform [11]. Commercial tools such as IBM Rational Software Architect, Spark System Enterprise Architect or Together are integrated with Eclipse-EMF [12] (figure 2).

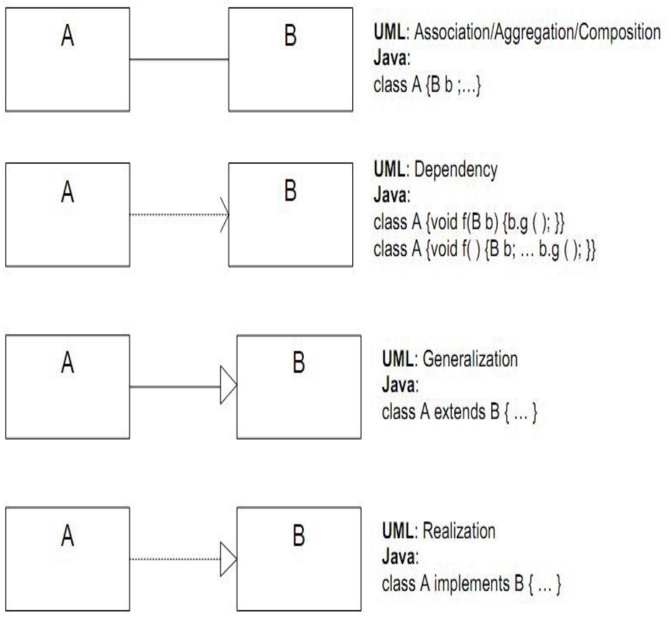
A
B
UML: Association/Aggregation/Composition
Java:
class A {B b ;...}
A
B
UML: Dependency
Java:
class A {void f(B b) {b.g ( ); }}
class A {void f( ) {B b; ... b.g ( ); }}
A
B
UML: Generalization
Java:
class A extends B { ... }
A
B
UML: Realization
Java:
class A implements B { ... }

Fig. 2. ISM Java constructs versus PSM Java constructs

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at higher levels of abstraction [13]. Reverse engineering is needed when the process to understand a software system would take a long time due to incorrect, out of date documentation, complexity of the system and the insufficient knowledge of the maintainer of the system. Reverse Engineering is used to recover lost information, to improve or provide documentation, to detect side effects, to reuse the components and to reduce the maintenance effort.

This technique is widely used in several disciplines including new technologies such as mechanics, electronics, etc. In computer science, Reverse engineering dynamic models is to convert the code to models such as UML sequence diagram, state diagram, etc, the goal is to increase the level of abstraction to better understand the behavior of software.

Reverse engineering dynamic UML models is a very efficient way to understand complex software whose source code is absent or documentation is outdated. It is also used in software engineering activities such as testing and validation.

In this paper, we consider UML diagrams as target design models for reverse-engineering. This means that we focus only on RE of UML diagrams (i.e. the process that analyzes the execution of the system and creates UML diagrams that depicting its structure and its behavior). UML is an object-oriented language for software system modeling [2,3]. It is composed of a set of diagrams which allow specifying the several aspects of a system. The two main aspects are: static (structural diagrams) and behavior (dynamic) aspects. The static aspect of system can be specified using class or component diagrams, while the behavior aspect can be specified using sequence diagrams, state machines and activity diagrams...etc.

In the earlier approaches, reverse engineering of object oriented code resulted in generation of class diagrams, interaction diagrams and use case model in general. However they represent only the static nature of the object oriented systems.

3. Related Works

Several studies have been performed on the reverse engineering of UML diagrams [14,15,16,17,18,19,20,21]. We distinguish two categories in existing approaches: static and dynamic. Static analysis is to use the code structure to generate the sequence diagram. One of the main works based on static analysis is that Rountev et al. [14]. They proposed an approach for the extraction of UML sequence diagrams from code through building the control flow graphs. The dynamic analysis is to analyze the performance of the application. Several studies try to generate the sequence diagram by analyzing the execution traces. In [15] is proposed an approach to build a high-level sequence diagram incrementally from basic diagram using the operators introduced by UML 2.0. In [16] they try to build a high-level sequence diagram from combined fragment using the state vector describing the system. In [17], it is proposed an approach completely dynamic based on the LTS (labeled transition system) for merger traces collected and generate a high-level sequence diagram.

These approaches have succeeded in generating representative UML behavior. But they recognize some limitations. These limitations include the information filtering problem. Thus, the resulting sequence diagram contains a lot of useless information that does not help to understand the software.

Figure 3 shows an MDA-based Reverse Engineering Framework for reverse engineering that distinguishes three different abstraction levels linked to models, metamodels and formal specifications.

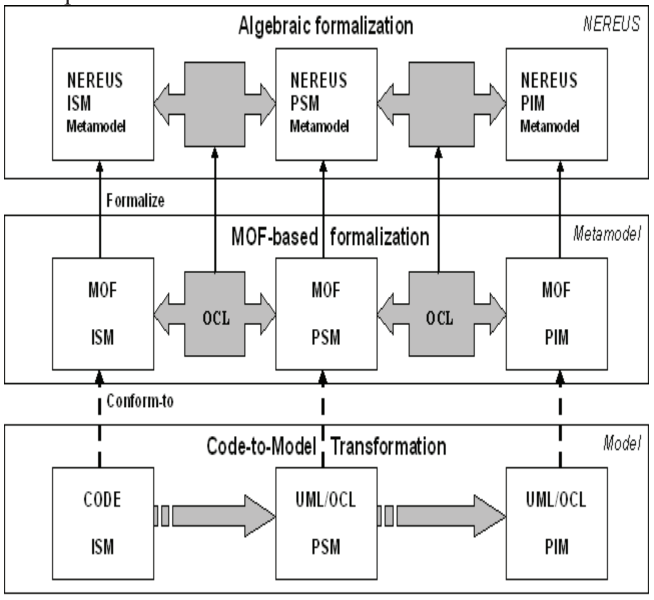

Fig. 3. MDA-based Reverse Engineering Framework

The model level includes code, PIMs and PSMs. A PIM is a model with a high-level of abstraction that is independent of an implementation technology. A PSM is a tailored model to specify a system in terms of specific platform such J2EE or .NET. PIMs and PSMs are expressed in UML and OCL [2,3,4]. The subset of UML diagrams that are useful for PSMs includes class diagram, object diagram, state diagram, interaction diagram and package diagram. On the other hand, a PIM can be expressed by means of use case diagrams, activity diagrams, interactions diagrams to model system processes and state diagrams to model lifecycle of the system entities. An ISM is a specification of the system in source code.

At model level, transformations are based on static and dynamic analysis. The metamodel level includes MOF metamodels that describe the transformations at model level. Metamodel transformations are specified as OCL contracts or QVT transformations [30] between a source metamodel and a target metamodel. MOF metamodels "control" the consistency of these transformations. The level of formal specification includes specifications of MOF metamodels and metamodel transformations in the metamodeling language NEREUS that can be used to connect them with different formal and programming languages. This framework could be considered as an MDA-based formalization of the process described by [22]. In this chapter we exemplify the bases of our approach with Class Diagram reverse engineering. However, our results include algorithms for extracting different UML diagrams such as interaction diagram, state diagram, use case diagram and activity diagram [23,24,25,26].

4. Methodology

Our approach consists on converting platform specific artifacts into a platform independent model (Figure 4). Platform specific code, artifacts, UI elements and schema are processed in a Model Generator Module to generate a platform specific model. The platform specific code, artifacts, UI elements and schema could be present in many forms and formats including code written in programming languages such as Java, or C, or C++ and schema and other artifacts represented as xml files or other files. A Model Generator Module processes the platform specific artifacts in their various formats and extracts a platform specific model from them. In order to do this, it has to know the metamodel of the underlying platform. If one exists, then the implementation artifacts can be mapped to such a platform specific model. But in cases where one does not exist, we use a semi-automated approach to derive metamodels from specific platforms. In general, extracting the meta-models for non-standards based and proprietary platforms is an engineering challenge. Depending on the platform, varying amounts of manual effort may be required to extract the meta-modal of the platform. If the meta-models are not published or not accessible, then one may have to resort to manual observation of exemplars to derive the meta-model from the exemplar. This means an exemplar with all possible types of elements needs to be constructed. An exemplar contains the implementation artifacts which include code, schemas, xml files etc. The meta-model extraction may be automated using exemplar analysis tools available in vendor tools such as IBM's Rational Software Architect (RSA).

However, an exemplar must be created first to conduct the exemplar analysis. In our work, for the two vendor platforms chosen, we were able to obtain the metamodels for one of the vendor platforms while we had to manually create the other using exemplar creation and exemplar analysis.

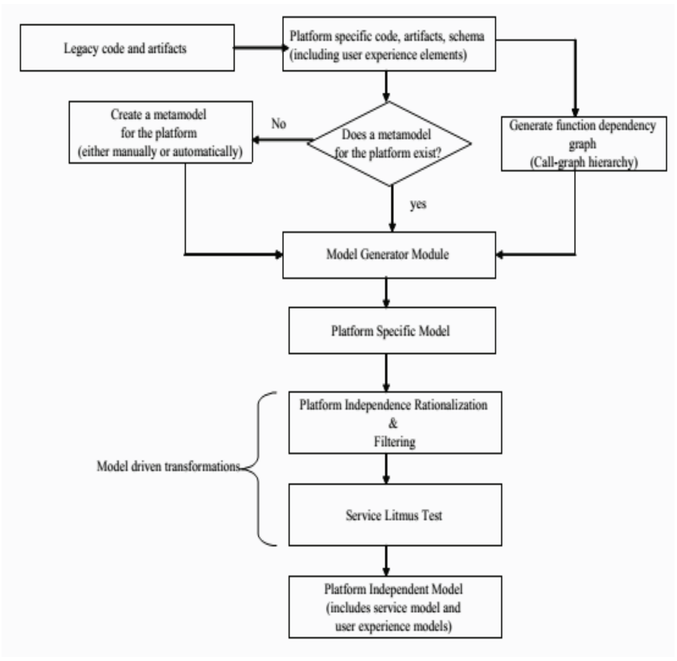
Legacy code and artifacts
Platform specific code, artifacts, schema (including user experience elements)
Create a metamodel for the platform (either manually or automatically)
No
Does a metamodel for the platform exist?
Generate function dependency graph (Call-graph hierarchy)
yes
Model Generator Module
Platform Specific Model
Platform Independence Rationalization & Filtering
Model driven transformations
Service Litmus Test
Platform Independent Model (includes service model and user experience models)

Fig. 4. Our approach to deriving platform independent models from implementation artifacts

This metamodel is then used by the Model Generator Module to generate a platform specific model for specific model instances. Then, filtering is performed to extract only those elements that would be of 'value' at platform independent level in an SOA environment. The rationalization and filtering mechanism can employ predefined rules to perform this. For example, models of artifacts such as factory classes for business objects, and auxiliary data structures and code that setup environment variables and connectivity with legacy systems etc need not be translated onto platform independent models. These types of business objects, data structures, application services, their operations are cleansed and filtered at this stage. Then from the platform specific model, we extract service models and apply a service litmus test as given in IBM's SOMA method [27] to categorize services as process services, information services, security services, infrastructure services etc. SOMA method defines these categories of services. Each service along with its ecosystems of services can be examined in detail to derive this information either automatically or manually. Once done, additional tagging is done on services to note which ones are exposed externally and which ones are internal implementations. The litmus test can be administered manually or can be automated if there is enough semantic information about the code/artifacts to know about the behavior and characteristics.

In our work, we used a user- directed mechanism for doing this filtering. A tool has been developed to enable a developer to conduct the filtering. This along with the user experience elements and models are all extracted into a platform independent model via model-driven transformations. In addition one can use code analysis tools to understand the call-graph hierarchy to retrieve an ecosystem of mutually dependent services. Several vendor tools are available for doing this for various programming languages. We use IBM's Rational Software Architect (RSA) [28] tool to do code analysis [29].

This information is captured and reflected in a platform specific model which then gets translated into a platform independent model via model driven transformations. This helps generate a service dependency model at the platform independent model. The service model and the service dependency information together provide static and the dynamic models at the platform independent level.

5. Reverse Engineering in makerspace

This study can be shared through experimental workshops working on concrete case studies using one of the MDA software, these workshops should simply be equipped with a few computers or a single computer with a video projector can do the trick. The sharing can also be done by videoconference or by recorded videos illustrating the use of these tools by demonstrations or by other means allowing sharing like a local network.

6. Case Study

Transformations create elements in a target model (domain) based on elements from a source model' [6]. A model driven transformation is a set of mapping rules that define how elements in a given source model map to their corresponding elements in a target domain model. These rules are specified between the source and target platform metamodels. Depending on what need to be generated there could be multiple levels of transformations such as model-to-model, model-to-text, model-to-code and code-to-model. Also, depending on the domain and the desired target platform multiple levels of transformations might be required to transform a PIM into implementation artifacts on a target platform in the case of forward engineering and vice versa for reverse engineering. For example, transformations may be required across models of the same type such as a transformation from one PSM to another PSM to add additional levels of refinement or across different levels of abstraction such as from PIM to PSM or from one type of model to another such as from PSM to code or even PIM to code. In our case (Figure 5), we use the traditional PIM-to-PSM and PSM-to-code transformations for forward transformations and code-to-PSM and PSM-to-PIM transformations for model extraction or reverse engineering. Operationally, multiple levels of transformations can be chained so that the intermediate results are invisible to the consumer of the transformations. The proposed case tool to prove our approach is Rational Software Architect or Eclipse-EMF (the Eclipse Modeling Framework).

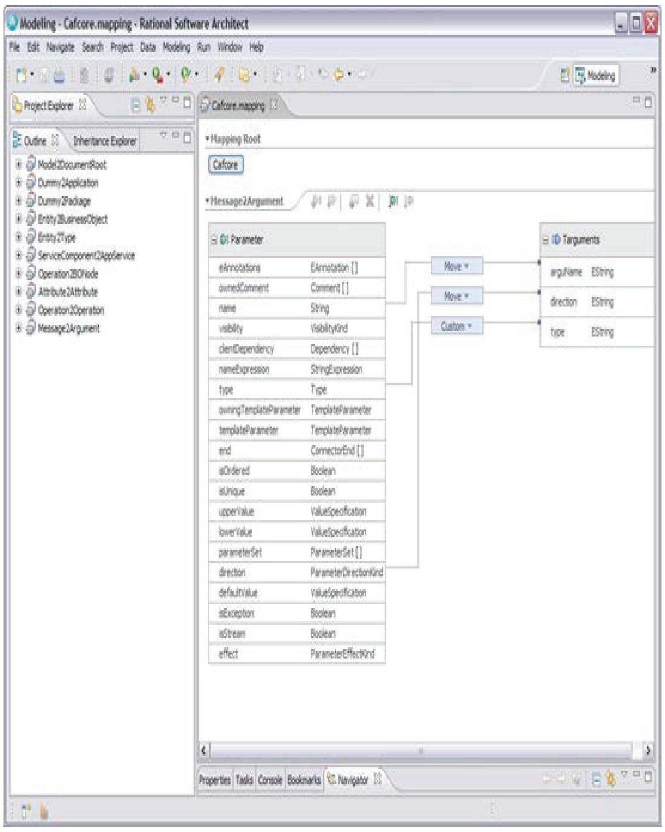
Modeling - Cafcore.mapping - Rational Software Architect
File Edit Navigate Search Project Data Modeling Run Window Help
Project Explorer
Outline
Inheritance Explorer
Model2DocumentRoot
Dummy2Application
Dummy2Package
Entity2BusinessObject
Entity2Type
ServiceComponent2AppService
Operation2BOMode
Attribute2Attribute
Operation2Operation
Message2Argument
Cafcore.mapping
Mapping Root
Cafcore
Message2Argument
Parameter
eAnnotations EAnnotation [ ]
ownedComment Comment [ ]
name String
visibility VisibilityKind
clientDependency Dependency [ ]
nameExpression StringExpression
type Type
owningTemplateParameter TemplateParameter
templateParameter TemplateParameter
end ConnectorEnd [ ]
isOrdered Boolean
isUnique Boolean
upperValue ValueSpecification
lowerValue ValueSpecification
parameterSet ParameterSet [ ]
direction ParameterDirectionKind
defaultValue ValueSpecification
isException Boolean
isStream Boolean
effect ParameterEffectKind
Move
Move
Custom
Targuments
arguName EString
direction EString
type EString
Properties Tasks Console Bookmarks Navigator

Fig. 5. Case tool for deriving platform independent models

7. Results/Discussion

Reverse engineering is one the essential parts of software maintenance. It involves high risk especially when the maintenance and development teams are different. Extracting the static model from the source code may not be motivating factor for software maintenance. Many reverse engineering tools and approaches are proposed in literature. However each represents only a subset of operational requirements. The process of reverse engineering is resulting in models consisting of only the static parts of design. Though the communication among objects is transformed, but reverse engineering of the internal state of the object is not presented. Static models are limited in their usefulness. It is important to realize that quality attributes such as performance and reliability can be predicted from the dynamic behavioral models of the system.

In this paper we proposed a new approach to extract UML dynamic behavior diagrams from the Java source code using both the static and dynamic analysis.

There are two main categories of existing UML behavior reverse engineering approaches: the first category refers to approaches which are based on static analysis while the second concerns dynamic analysis based approaches. Static analysis is done on static information which describes the structure of the software as it is written in the source code. However, dynamic analysis is based on the system runtime behavior information which can be captured by separated tools as in [17], by instrumentation techniques as in [18], or by debugging techniques. There approaches [31,32, 33] which combine both static and dynamic approaches for more efficiency.

We show in this paper the importance of the reverse engineered static and dynamic views of the system architecture in order to predict the system quality attributes and analyze possible plans of the system evolution.

ACKNOWLEDGMENTS

We would like to thank the FST Errachidia for providing us with assist us in the development part of our research. My sincere thanks also goes to Moulay Ismail University for its financial support of the project "Algorithmic and software support for transformational approaches to UML".

8. CONCLUSION

Reverse-engineering aims to provide design models from existing software. Hence, this can facilitate program comprehension and by consequence maintenance and evolution of systems.

In this paper, we have presented an overview on the reverse engineering of static and behavioral diagrams. Our approach helps software development engineers to reverse engineer object-oriented software. This paper gives also manners of how to share and benefit of the reverse engineering technologies in software development Makerspace.

A Makerspace is a shared community space where people can come together and collaborate while sharing tools, resources and knowledge. Makerspaces can include a range of equipment and instructions from industrial arts, electronic & robotic technologies and 3D prototyping to software development and digital arts. A Makerspace represent an opportunity to share our software development approach. So, in this paper we presented our approach to porting software solutions on multiple software middleware platforms. We propose the use of model-driven transformations to achieve cross-platform portability. We propose approaches for two scenarios. First, in cases where no software solution exists on any of the desired target middleware platforms, we advocate developing a platform independent model of the software solution in a formal modeling language such as UML and then applying model-driven transformations to generate implementation artifacts such as code and schemas from the models on the desired target platforms. Second, if a software solution already exists on one specific middleware platform, we propose applying reverse transformations to derive a platform independent model from the implementation artifacts and then applying forward transformations on the derived model to port that software solution on to a different target platform. We advance the traditional model-driven technique by presenting a service-oriented approach to deriving platform independent models from platform specific implementations. Reverse-engineering aims to provide design models from existing software. Hence, this can facilitate program comprehension and by consequence maintenance and evolution of systems. Our approach helps software development engineers to reverse engineer object-oriented software.

Our future work is to evaluate our approach on more complex system such as embedded systems

REFERENCES

1. The Model Driven Architecture. www.omg.org/mda. MDA (2005).
2. Unified Modeling Language: Infrastructure. Version 2.3. OMG Specification formal/ 2010-05-03. www.omg.org. UML (2010a).
3. UML: Unified Modeling Language: Superstructure. Version 2.3. OMG Specification: formal/2010-05-05. www.omg.org. UML (2010b).
4. OCL: Object Constraint Language. Version 2.2. OMG: formal/2010-02-01. www.omg.org. OCL (2010).
5. MOF: Meta Object Facility (MOF ™) 2.0. OMG Specification formal/2006-01-01. www.omg.org/mof. MOF (2006).
6. Standards Roadmap. ADM Task Force. adm.omg.org. ADM (2010).
7. Knowledge Discovery Meta-Model, Version 1.3-beta 2, March 2011. OMG specification formal 2010-12-12. http://www.omg.org/spec/kdm/1.3/beta2/pdf. KDM (2011).
8. Mellor, S. J., Clark, A. N., and Futagami, T. Model-driven development. IEEE Software, Vol-**20**, Issue-5, pp. 14–18, (2003).
9. Frankel, David S.: Model Driven Architecture: Applying MDA to Enterprise Computing. OMG Press: 2003. OMG Unified Modeling Language Specification, Object Management Group, (2003).
10. The eclipse modeling framework. http://www.eclipse.org/emf/. Eclipse (2011).
11. ATL Documentation. www.eclipse.org/m2m/atl/documentation. ATL (2011).
12. CASE MDA. www.case-tools.org. (2011).
13. Elliot j. Chikofsky and James H. Cross II. Reverse engineering and design recovery : a taxonomy. IEEE Software, (1990).
14. A. Rountev, O. Volgin, and Miriam Red-doch. Control flow analysis for reverse engineer-ing of sequence diagrams. Technical Report OSU-CISRC-3/04-TR12, Ohio State University, (March 2004).
15. M. K. Sarkar , T. Chaterjee ; Reverse Engineering : An Analysis of Dynamic Behavior of Object Oriented Programs by Extracting UML Interaction Diagram, International Journal of Computer Technology & Applications, Vol **4** (3),378-383, (2013).
16. R. Delamare, B. Baudry, Y.L/ Traon "Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces" Workshop on Object-Oriented Reengineering at ECOOP 06, Nantes, France, (July 2006).
17. T. Ziadi, M. Aurélio A. da Silva, L. Hillah, M. Ziane, "A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams", 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011, pp. 107-116, (IEEE Computer Society). http://www.ptidej.net/material/inanutshell. (August 2009).
18. L. C. Briand, Y. Labiche, J. Leduc, "Towards the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software", IEEE Transactions on Software Engineering, vol. **32** (9), pp. 642-663, (2006).
19. R. Zhao, and L. Lin, "An UML Statechart Diagram- Based MM-Path Generation Approach for Object-Oriented Integration Testing",Enformatika, Vol. **16**, p259, (2006).
20. D.H.A. van Zeeland, "Reverse-engineering state machine diagrams from legacy C-code", proceedings of 12th Conf. on Entity-Relationship Approach - Arlington-Dallas, (Dec. 1993).
21. S. Christian, "Extracting n-ary relationships through database reverse engineering", B. Thalheim (Ed.), Proc. 15th Internat. Conf. on Conceptual Modeling, Cottbus, Germany, pp. 392–405, (1996).
22. Tonella, P., & Potrich, A. Reverse Engineering of Object-oriented Code. Monographs in Computer Science. Heidelberg: Springer-Verlag, (2005).
23. Favre, L. (2010). Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution. Engineering Science Reference, IGI Global, USA.
24. Favre, L., Martinez, L., & Pereira, C. MDA-based Reverse Engineering of Object- oriented Code. Lecture Notes in Business Information Processing (Vol **29**, pp. 251-263). Heidelberg: Springer-Verlag, (2009).
25. Pereira, C., Martinez, L., & Favre, L. Recovering Use Case Diagrams from Object- Oriented Code: an MDA-based Approach. In Proceedings ITNG 2011, 8th. International Conference on Information Technology: New Generations (pp. 737-742), Los Alamitos: IEEE Computer Press, (2011).
26. Martinez, L., Pereira, C., & Favre, L. Recovering Activity Diagrams from Object- Oriented Code: an MDA-based Approach. In Proceedings

2011 International Conference on Software Engineering Research and Practice (SERP 2011)(Vol. **1**, pp. 58- 64), CSREA Press, (2011).

27. Arsanjani A.: Service Oriented Modeling and Architecture (SOMA). http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/
28. IBM Rational Software Architect http://www.ibm.com/developerworks/rational/products/rsa/
29. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html Java Emitter Templates (JET).
30. QVT: MOF 2.0 Query, View, Transformation. Formal/2008-04-03. www.omg.org. QVT (2008).
31. A. Jakimi, L. Elbermi and M. El Koutbi, “Software Development for UML Scenarios: Design, fusion and code generation”. International Review on Computers and Software, Vol. **6**. n. 5, pp. 683-687, (2011).
32. M. H. Abidi, A. Jakimi, E. H. El Kinani. A New Approach the Reverse Engineering UML State Machine from Java Code. International Conference on Intelligent Systems and Computer Vision (ISCV’2015), Fes, Morocco, (March 25-26 2015).
33. Chafik Baidada, El Mahi Bouziane and Abdeslam Jakimi, An Analysis and New Methodology for Reverse Engineering of UML Behavioral, International Journal of Advanced Engineering, Management and Science (IJAEMS), Vol-**2**, Issue-4, (April- 2016).