World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:2, No:6, 2008

A Middleware Transparent Framework for Applying MDA to SOA

Ali Taee Zade, Siamak Rasulzadeh, and Reza Torkashvan

***Abstract*—**Although Model Driven Architecture has taken successful steps toward model-based software development, this approach still faces complex situations and ambiguous questions while applying to real world software systems. One of these questions - which has taken the most interest and focus - is how model transforms between different abstraction levels, MDA proposes. In this paper, we propose an approach based on Story Driven Modeling and Aspect Oriented Programming to ease these transformations. Service Oriented Architecture is taken as the target model to test the proposed mechanism in a functional system.

Service Oriented Architecture and Model Driven Architecture [1] are both considered as the frontiers of their own domain in the software world. Following components - which was the greatest step after object oriented - SOA is introduced, focusing on more integrated and automated software solutions. On the other hand - and from the designers' point of view - MDA is just initiating another evolution. MDA is considered as the next big step after UML in designing domain.

***Keywords*—**SOA, MDA, SDM, Model Transformation, Middleware Transparency, Aspects and Jini.

I. Introduction

THE goal of this research is to provide a successful and usable conjunction between these two technologies. We have tried to provide a simple yet effective process which can be viewed as a framework. In the vision inspired by this framework, SOA is the product and MDA makes its production line. During this process, input model is provided via XMI [2] standard and with a high level of abstraction. Proposed framework analyses the elements and their relations within the given model and tries to recognize the SOA components.

In two phases (Fig. 1), the input model is first transformed into a SOA profile based model and then into a middleware independent code. Middleware transparency is achieved via the concept of Aspect. The final phase of framework is to transform middleware transparent code into an executable code based on one of known middlewares for SOA. Jini middleware and pre-process weaving is used in the last phase.
Rest of the paper is organized as follows: section II introduces proposed SOA profile. Sectιn III, IV and V relatively focus on the 1st, 2nd and 3rd phases of the framework. Section VI contains some implementation details and finally, section VII will conclude the paper.

Authors are with Islamic Azad University, Naragh Branch and Malayer Branch, Iran (e-mail: alitaee@gmail.com).

II. Profile for Service-Oriented Architectures

As shown in Fig. 1, generating a profile for service-oriented architecture is the first step to produce such a framework. This profile enables the designer to describe the platform specific model based on SOA. Profiles are standard techniques for extending UML. By using profiles for precise modeling, we ensure that the designed model can be used in different views of MDA with the same concepts, as we are following the MDA for defining standard models.

The elements of SOA profile is selected based on knowledge of the main elements of service-oriented architectures (Fig. 2).

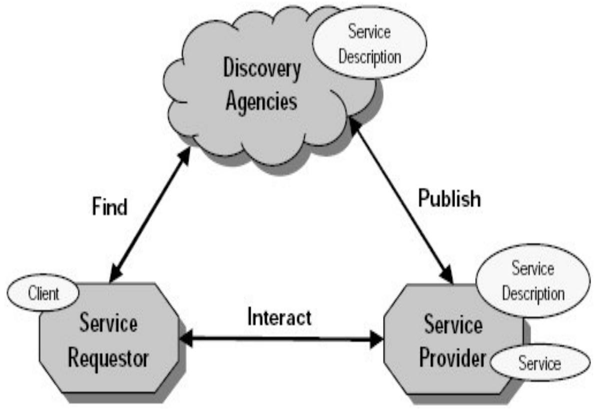

Fig. 2 Components of SOA [3]

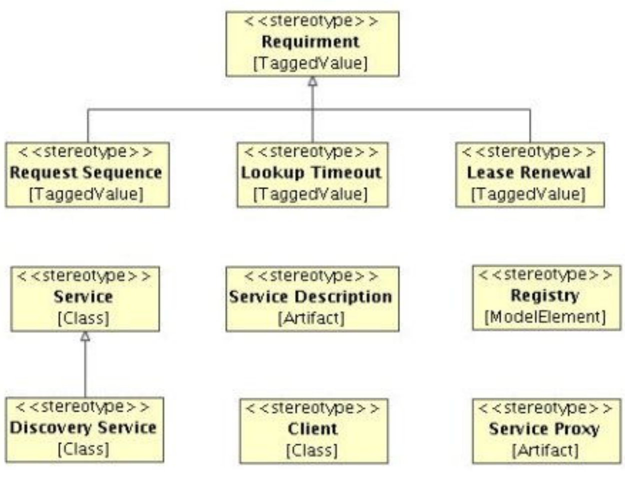

Fig. 3 Proposed SOA Profile

Although there are different approaches for implementing of service-oriented systems [4], these elements are used in all of them and it shows a correct implementation of such architecture. Based on this idea, besides studying different service-oriented systems, and identifying their core components, the profile shown in Fig. 3 is proposed.

III. From PIM to PSM

This phase can be considered as the most important and complex part of the framework. In this step, the platform independent model - based on UML standard profile - is transformed to the platform specific model - based on proposed SOA profile. Although we have tried to apply MDA to SOA for simpler model [5], the approach taken here has more capabilities and can handle relatively more complex cases.

In this approach, the input model (PIM) has no direct information about SOA. Obviously using such an abstract input - based on standard UML - needs a more autonomous model transformer. By autonomous we mean a model transformer which tried to depend on the specification of model rather than human guidelines. Such a model transformation is beyond what we expect from an MDA based model transformer and also beyond most of the current frameworks.

Second notable point about our framework is its declarative approach in PIM to PSM phase. Declarative transformation is considered as a huge advantage, since designer is not directly involved in the logic of model transformation. What is considered the logic of transformation is generated automatically, based on a declaration of how and what should be done. Using declarative approach, we need a formal way to express transformation for which a unique and correct code is generated. We have used story driven modeling profile [6] for this formal definition of transformation [7, 8, 9].

A. Input Model

Input model is based on standard UML. Designer has designed this model having SOA in mind but has not placed any SOA specific details in it. Transformer uses graph specifications of the input model (such as relations etc) to determine SOA components. We categorize these graph cifications into two main groups:

- Conditions over vertices: which shows what tagged values a vertex, can have.

- Conditions over edges: This shows the type, and specification of edges connecting vertices.

Considering these two categories and the general SOA model (Fig. 2) we conclude the following conditions:

- A vertex of sterotype *Interface*
- A vertex of type *Class*, implementing above interface.
- A vertex *using* above class.
- An edge connecting above class of type *use*.
- A vertex presenting *registry* service.

B. Formal Definition of Transformation

We have used SDM to present a formal definition of model transformation. A Detailed discussion on SDM is beyond the scope of this paper.

To have a general perspective of it, SDM uses a combination of activity and collaboration diagrams to define a story based presentation of the model. Fig. 4 shows a view of our main activity diagram. Swim lanes divide the sequences into two parts, i.e. human interactive and machine interacted. Steps of this diagram fall into two groups:

- Conditional steps which test occurrence of a specific case in the model
- Functional steps which perform a change in the model

Details of each step are as follows:

1. Start is the very first node of the diagram. A state without any input and only one exit.
2. Print defined with <<*code*>> stereotype and prints out an informative message.
3. Initial check of the input model, where we check whether input model contains at least one UML package and four classes (Fig. 5).

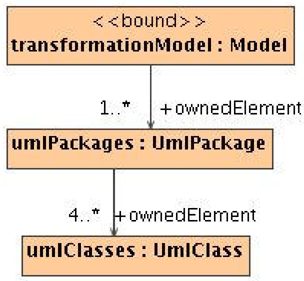

Fig. 5 Initial check of the input model

4. Selecting input model components and iterate over them. (Fig. 6)
5. Copying the selected model into the target model.
6. Initial check of the selected element which checks whether this element has at least 3 connecting edges.
7. Applying SOA profile to the selected element, which has passed the initial checking. This step contains a number of UML sequence and collaboration diagrams which is shown in Fig. 7.

As it can be seen, this diagram checks the selected elements precisely and applies SOA profile to them by the weaves of some consecutive conditional expression.

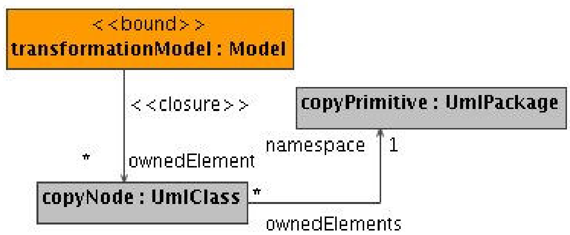

Fig. 6 Selecting input model components

Here we present conditions and function of detecting a service interface and applying it (Fig. 8).

IV. Generating Aspect-Oriented Codes from PSM

The second stage is, transforming PSM model to the middleware independent codes. As we have labeled the elements of model in the first stage, the process of generating code is a simple one-to-one mapping and we have used a template based approach for this mapping.

Another point considered during code generation, is the middleware that PSM model specifies it and in this case it is a middleware for service-oriented architecture. But one of the goals of the framework is middleware independency. For this reason and considering the fact that we can see the most resulting changes of using middleware as Aspects, it is possible to generate a middleware independent code with required Aspects for service-oriented architecture.

In fact, this code has the ability to implement the service-oriented architecture. But only when Aspects are used in a particular middleware environment (this process is illustrated in final stage).

A. Required Aspects for Service-Oriented Architecture

It is possible to consider aspects as profiles in code level. This property makes it easy to select Aspects which are related to the profile. Therefore, we will introduce Aspects that are selected based on service-oriented architecture profile. These Aspects are presented in code level (Java) based on JSR 175 standard [11] (annotations).

- *Interface* that is recognized with *ServiceDescription*
- *Service* that is recognized with Service.
- *Client* with the Aspect named Client and
- *Registry* which is recognized with Registry.

These Aspects are appeared at the beginning of definition of classes and variables.
For example:

```
@Client
public class SamleClient {
...
}
```

B. Templates for Code Generation

As said before, code generation based on templates is one of the ideal methods for easily code generation. Easy, because the most complex part of transformation is done in first stage and in second stage, there is a one-to-one transformation.

We have used templates based on Velocity template engine [12] for code generation. This template engine is used with AndroMDA [13], for making the code generation easier.

A simple instance of such templates is shown bellow:

```
package $service.packageName;
@Service
public class ${service.name}Service \
implements
${service.name}ServiceDescription{
#foreach($operation in
${service.operations})
$operation.visibilty
$operation.returnType
$operation.signature;
#end
}
```

V. From Aspect to Executable Code

The last phase of framework, transforms SOA enabled code to a full executable code based on an SOA middleware. Two main questions of this phase are 1) weaving technique and 2) target middleware

A. Pre-Process Weaving

There are various techniques for weaving aspects and converting them to executable code, like compile time, and deploy time to name a few. To select a weaving technique, one must consider various factors such as coordination of weaving technique and problem, ease of use, tools etc. Considering these factors, we have decided to use pre-process weaving in this framework. This type of weaving techniques are used in cases that code changes inspired by aspect are few considered to other changes such as configuration stuff.

Pre-process weaving is in fact a special kind of AOP, known as Attribute Oriented Programming. XDoclet [14] is a famous attribute oriented programming tool. With standardizing annotation in Java 5 most of the time attributes are defined using annotation and we have used the same approach.

B. Choosing Middleware

Nowadays there are various middleware which support SOA development and bring facilities to ease this architecture. Among them are Java EE, Microsoft .NET, and Web Services. But we have used less know middleware Jini [15] as our SOA enabled middleware. Although Jini is less know, but this middleware has build in and complete features for SOA development among them: platform independent, PnP, and interface base design.

VI. Implementation

According to the previous sections, this framework has been formed from different multiple parts and each part has its own complexities and requirements. The main part of this framework is description of model transformation based on story driven approach and its implementation. And these activities are done, adherence to the FOTS team from Antwerpern University. The required transformation in this stage, are implemented according to the extension of MoTMoT [16]. In second stage, we have used from AndroMDA [13]. The structure and testing of templates is done based on Java language and APT software.

VII. Conclusion

Although MDA can be considered a successful movement in model based software development, but this approach still has ambiguous questions to face especially when applied to complex real world systems. One of these questions - which has taken the most attempts in this way - is model transformed across different abstraction layers, MDA propose. What this paper presented, was how to use story driven modeling and aspect oriented programming to ease model transformation from PIM to PSM and from PSM to code. We have also tried to use SOA as our target model and test the proposed method in a functional environment.

REFERENCES

[1] Object Management Group. Model-Driven Architecture. www.omg.org/mda/.
[2] Object Management Group, XML Metadata Interchange Specification, Version 2.0, OMG Document: formal/03-05-02, May 2003.
[3] L. Baresi, R. Heckel, S. Thöne and D. Varró. Modeling and Analysis of Architecture Styles Based on Graph Transformation - A Case Study on Service-Oriented Architecture, European Research Training Network (SegraVis), 2003.
[4] L. Baresi, R. Heckel, S. Thöne and D. Varró. Modelling and Validation of Service-Oriented Architectures: Application Vs. Style. In Proc. ESEC/FS 03 European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, Pages 68-77. ACM Press, 2003.
[5] Adel Torkama Rahmani, Vahid Rafe, Saeed Sedighian, Amin Abbaspour. An MDA-Based Modeling and Design of Service Oriented Architecture. International Conference on Computational Science (3) , Volume 3993 of LNCS, Pages 578-585, Springer Verlag, 2006.
[6] Thorsten Fischer, Jorg Niere, Lars Torunski, and Albert Zundorf. StoryDiagrams: A New Graph Rewrite Language Based on UnifiedModeling Language, and Java. In Proceedings of the 6th International Workshop on Theory and Applications of Graph Transformation (TAGT) , volume 1764 of LNCS , pages 296-309. Springer Verlag, November 1998.
[7] Hans Schippers, Pieter Van Gorp. Standardizing SDM for Model Transformation. Formal Techniques in Software Engineering, Universiteit Antwerpern, Belgium. Fujaba Days Programme , 2004.
[8] Hans Schippers, Pieter Van Gorp, and Dirk Janssens. Leveraging UML Profiles to Generate Plugins from Visual Model Transformation. Accepted at Software Evolution through Transformation (SETra). Satellite of the 2nd Intl. Conference of Graph Transformation. October 2004.
[9] L.Baresi and R. Heckel. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In Proceedings of the First International Conference on Graph Transformation (ICGT 2002), volume 2505 of Lecture Notes in Computer Science, Pages 402-429. Springer-Verlag, 2002.
[10] Albert Zundorf. Rigorous Object Oriented Software Development, Habiliation hesis, 2001.
[11] Java Community Process. JSR 175 : A Metadata Facility for the Java Programming Language. http://jcp.org/en/jsr/detail?id=175 , Sep 2004.
[12] Velocity 1.3.1, The Apache Jakarta Project, http://jakarta.apache.org/velocity/, March 2003.
[13] M. Bohlen, AndroMDA - From UML to Deployable Components, version 3.1, http://www.andromda.org/, 2002-2005.
[14] XDoclet, Attribute Oriented Programming, http://xdoclet.sf.net/, Sep 2003.
[15] Jim Waldo, Alive and Well: Jini Technology Today. IEEE Computer, 33(6), pages 107-109, June 2000.
[16] Formal Techniques in Software Engineering. Model driven, Template based, Model Transformer (MoTMoT). http://sf.net/projects/motmot/ , 2004.

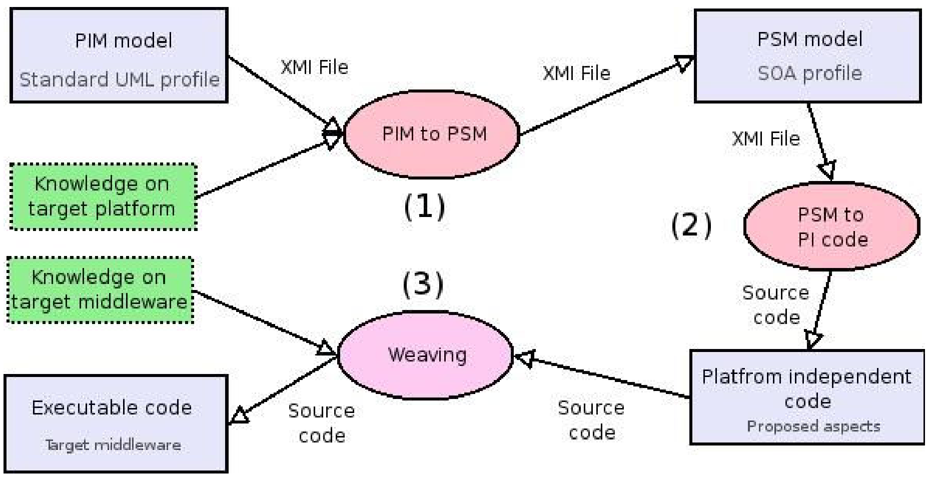

Fig. 1 Framework Components

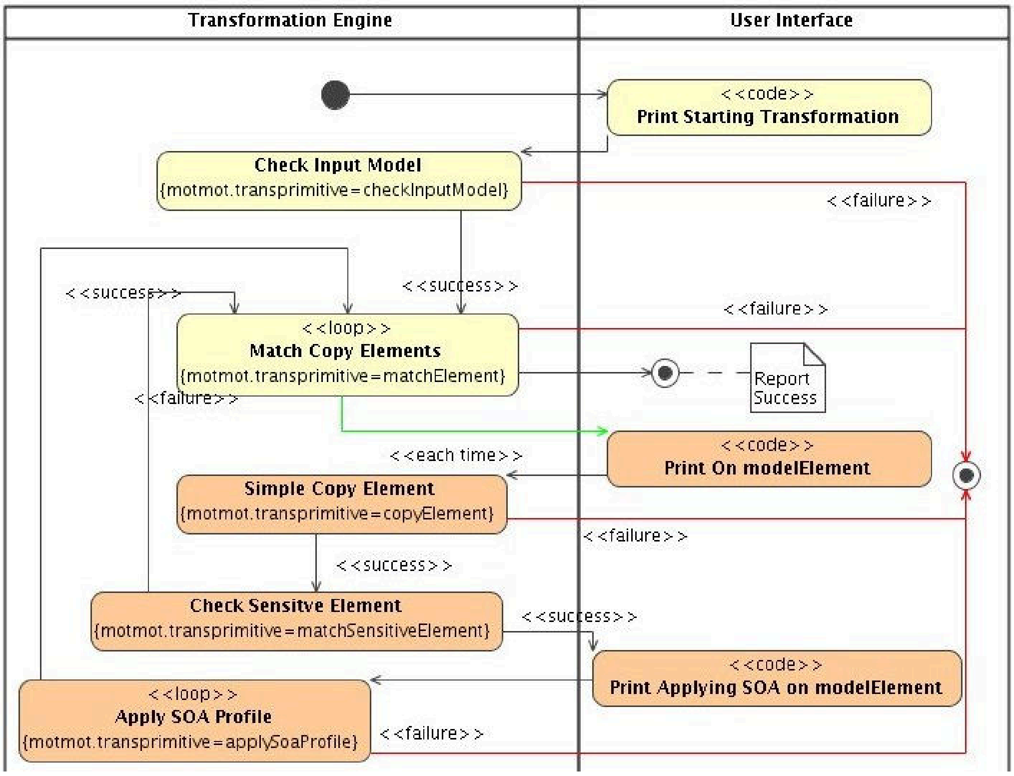

Fig. 4 Flow Diagram of Transformation

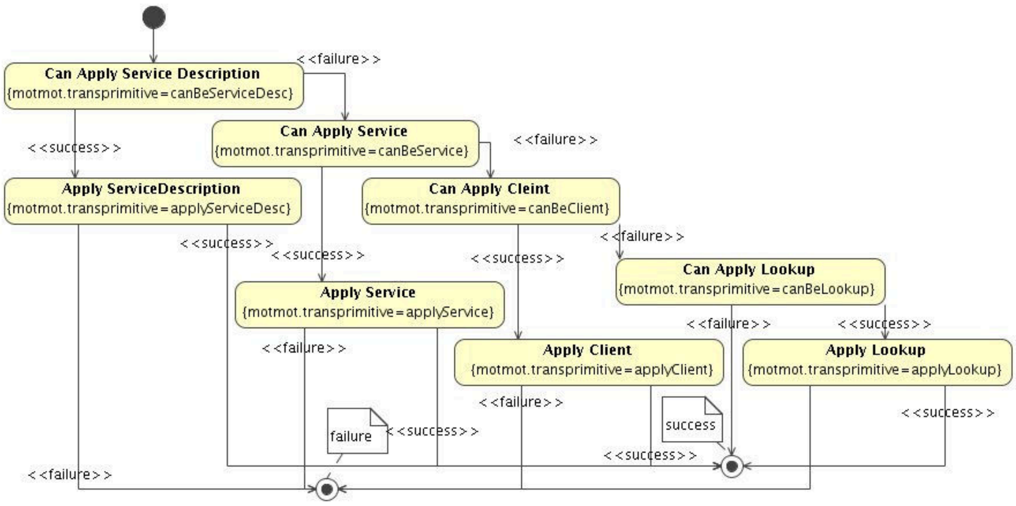

Fig. 7 Appling SOA profile

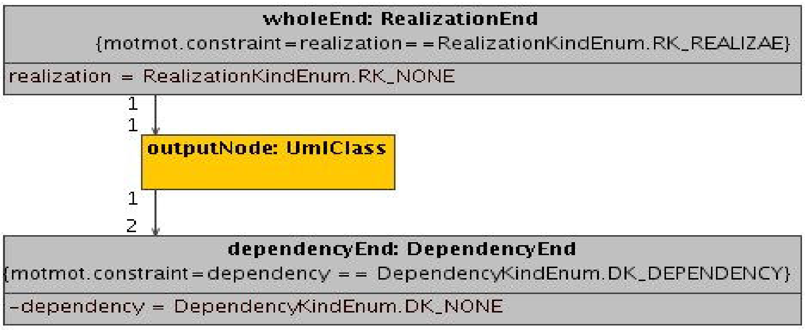

Fig. 8 Detecting a service interface