Model-View-Controller Architecture Specific Model Transformation

Hiroshi Kazato
Tokyo Institute of Technology /
NTT DATA CORPORATION
Tokyo 152–8552, Japan
kazato@se.cs.titech.ac.jp

Rafael Weiß
Tokyo Institute of Technology
Tokyo 152–8552, Japan
rweiss@se.cs.titech.ac.jp

Shinpei Hayashi
Tokyo Institute of Technology
Tokyo 152–8552, Japan
hayashi@se.cs.titech.ac.jp

Takashi Kobayashi
Nagoya University
Nagoya 464–8601, Japan
tkobaya@is.nagoya-u.ac.jp

Motoshi Saeki
Tokyo Institute of Technology
Tokyo 152–8552, Japan
saeki@se.cs.titech.ac.jp

ABSTRACT

In this paper, we propose a model-driven development technique specific to the *Model-View-Controller* architecture domain. Even though a lot of application frameworks and source code generators are available for implementing this architecture, they do depend on implementation specific concepts, which take much effort to learn and use them. To address this issue, we define a UML profile to capture architectural concepts directly in a model and provide a bunch of transformation mappings for each supported platform, in order to bridge between architectural and implementation concepts. By applying these model transformations together with source code generators, our MVC-based model can be mapped to various kind of platforms. Since we restrict a domain into MVC architecture only, automating model transformation to source code is possible. We have prototyped a supporting tool and evaluated feasibility of our approach through a case study. It demonstrates model transformations specific to MVC architecture can produce source code for two different platforms.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Object-oriented design methods*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures, Patterns*; D.4.7 [**Operating Systems**]: Organization and Design—*Interactive systems*

General Terms

Design, Languages

Keywords

Model-Driven Development, Model Transformation, Model-View-Controller Architecture, UML Profile

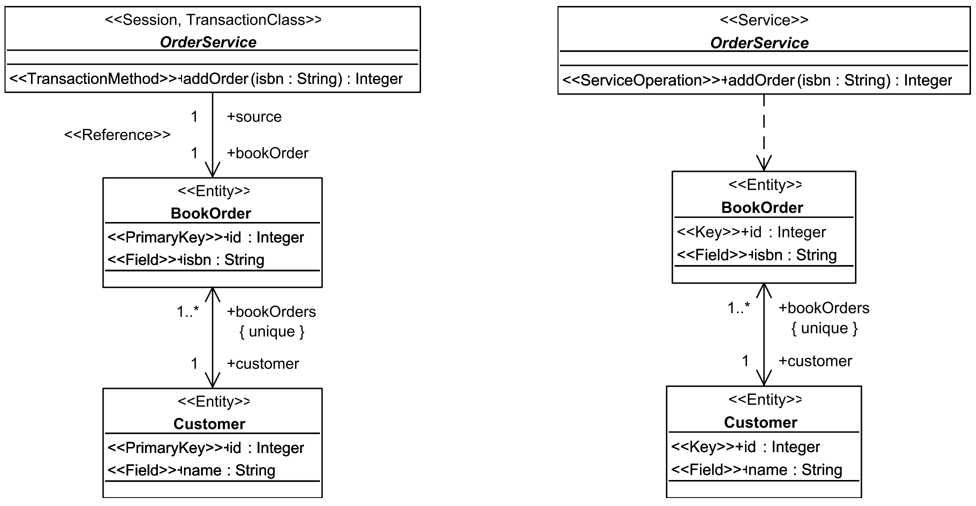
1. INTRODUCTION

Model-driven development (MDD) [16] is a development paradigm, which places models as primary artifacts and derives executable software by means of model transformation. It aims to increase productivity, maintainability and reusability of models by raising the level of abstraction above general-purpose programming and modeling languages. Some MDD tools, such as openArchitectureWare (oAW) [18] and AndroMDA [1], use their own UML profiles to include their necessary information into UML models. Since this kind of tools transform profiled UML models into source code, hereafter we refer to them as (model-driven) code generators. Along with the recent evolution in model transformation techniques, they have shown the possibility and effectiveness of MDD in practice to some extent.

However, because of the diversity of implementation platforms and code generators, there are a lot of UML profiles corresponding to various implementation concepts, and thus it is a labor-intensive and error-prone task to build, maintain and reuse these models. To cope with these problems, we should think of another level of abstraction by identifying similarities of various kind of implementation platforms and using code generators as building blocks.

In this paper, we propose a model-driven approach called ACCURATE, in which the *Model-View-Controller* architecture style is used to capture design concepts in a user-interactive application as well as to classify implementation platforms such as application frameworks and libraries. More specifically, we define a UML profile to describe architectural concepts directly in a model. Using this profile as a pivot [3], a bunch of transformation mappings is provided for each supported platform, in order to bridge between architectural and implementation concepts. By applying these mappings and code generators in sequence, our MVC-based model can be transformed into implementation models and source code for various platforms. Automating these transformations is feasible because we only cover a restricted architecture domain. The main contribution of this paper is to propose a model-driven approach specific to the *Model-View-Controller* architecture.

The rest of this paper is organized as follows. In the next section we explain our motivation by a brief example. Section 3 presents the ACCURATE approach. Section 4 briefly introduces the prototype implementation of our toolkit and following Sect. 5 evaluates the approach through a case study of an address book application. In Sect. 6, we survey some related works and close with conclusion and future work in Sect. 7.



(a) EJB Profile (b) Spring and Hibernate Profiles

Figure 1: PSM Examples Using Profiles for the Fornax-Platform [10]

2. MOTIVATING EXAMPLE

Class diagrams shown in Fig. 1 are examples taken from two different platform specific models (PSMs), one uses a profile for EJB and the other does a combination of profiles for the Spring Framework and Hibernate. Both of them specify almost same functionality, i.e. object/relational mapping between Java and relational databases, but they are different from a technical view because of their platform-specific profiles.

A problem occurs, e.g. if one would migrate a PSM based on EJB to Spring and Hibernate. In this case, the EJB profile first has to be unapplied by removing stereotypes and tagged values, then the model has to be modified structurally to conform the constraint forced by Spring and Hibernate profiles, and finally, stereotypes and tagged values defined by the target profiles have to be attached to the model. We identify that problems of profiles are closely coupled with code generators because of the following reasons:

- Each profile defines many stereotypes and tagged values whose names and possible values are closely related to the platform terminology. For example, ≪`Entity`≫ stereotypes denote EJB entity beans in Fig. 1(a), while they are plain-old Java objects (POJO's) managed by Hibernate in Fig. 1(b). Thus, developers are obliged to learn the platform first, rather than the profile itself.

- Concepts and terms introduced by profiles are technical and separated from the requirements. For example, ≪`ServiceOperation`≫ stereotype in Fig. 1(b) means that the *addOrder* operation runs an application logic within a transaction because the result should be transactional. Thus, one can hardly tell, which profile need to be applied and how to elaborate the requirements to models.

- Since profiles often put their own constraint over the UML metamodel, it is not easy to migrate a PSM from one profile to another, even if both of them offer similar functionalities and thus are alternatives for the application. For example, the relationship between *OrderService* class to *BookOrder* class needs to be an association with ≪`Reference`≫ stereotype in Fig. 1(a) and a dependency in Fig. 1(b).

For these reasons, PSMs are unsuitable to build, maintain and reuse for the further software evolution. We find it difficult to deal with a PSM once an application is built, especially when it has to be migrated to another platform. To address these issues, we propose an approach which enables developers to avoid operating with PSMs and code generators directly.

3. ACCURATE APPROACH

In this section we present the ACCURATE approach. The name ACCURATE comes from an acronym for 'A Configurable Code generator Unified with Requirements Analysis TEchniques'. As it implies, requirements play an important role in both PIM modeling and platform decision. The key idea is to capture functional and non-functional requirements into separate artifacts, a PIM and a platform configuration respectively, and join them at the downstream of the development.

Figure 2 illustrates the workflow of the approach. It defines four activities, PIM modeling, platform decision, PIM-to-PSM transformation and code generation. They are carried out by two kind of actors, application designer and requirements engineer, who are responsible for functional and non-functional aspects of the system respectively. During the proposed workflow, models have to run through different stages (e.g. a PIM is transformed into a PSM).

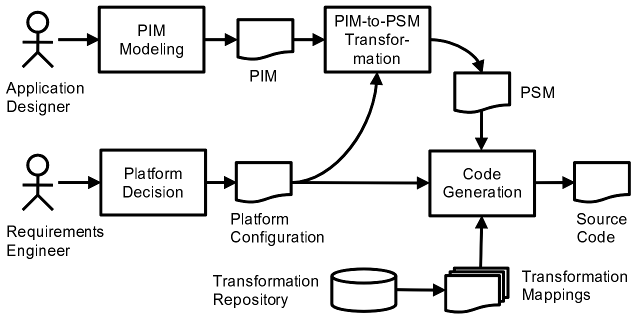


Figure 2: ACCURATE Workflow

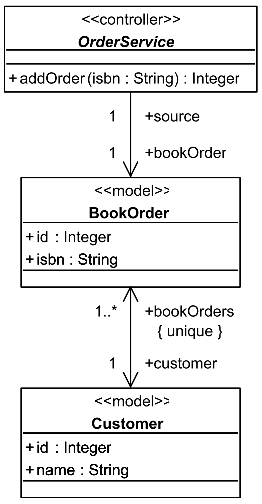


Figure 3: An Example Usage of the ACCURATE Profile

Following subsections explain these activities in terms of their inputs and outputs.

3.1 PIM Modeling

The workflow begins with definition of structure and functionality of the system as a PIM. We defined a platform-independent UML profile, called the ACCURATE profile, to describe PIMs. This profile adopts established concepts defined in the architecture styles as names and semantics of stereotypes, since architecture styles can be considered essentially immutable and independent of any platforms.

Application designers describe PIMs using UML modeling tools (such as MagicDraw [15]), which have support for defining and applying profiles to a UML model. The ACCURATE profile has fewer stereotypes and tagged values so that designers are easily able to learn and use, keeping still expressive enough to specify an application independently of any platform specific details.

Figure 3 illustrates a possible usage of the ACCURATE profile for the well-known *Model-View-Controller* architecture style [6]. According to this style, ≪`model`≫ classes provide core functionalities of an application domain and propagate changes to ≪`controller`≫ and ≪`view`≫ classes, which are responsible for inputs and outputs respectively.

3.2 Platform Decision

In parallel with the PIM modeling activity, requirements engineers communicate with stakeholders around the system to assess quality attributes expected for the system. The output from this activity is a combination of platforms, which usually tends to depend on experience and knowledge of requirements engineers since estimating quality of the system before implementing it is essentially a hard problem.

We assume our approach could be combined with certain requirement analysis and quality estimation techniques, but this topic is out of the scope of this paper due to the limitation of pages.

3.3 PIM Transformation

Once platforms are determined for a system, a PIM can be transformed to a PSM automatically by a model transformation. The output from this activity is a PSM that conforms to the UML profiles for the designated platforms. It can be used directly as an input

```
/* map a PIM class to a PSM identically */
mapping Class::toPSMClass() : Class {
  name := self.name.firstToUpper();
  isAbstract := self.isAbstract;
  visibility := self.visibility;
  ...
  ownedAttribute := self.ownedAttribute->map
    toProperty()->asOrderedSet();
  ownedOperation := self.ownedOperation->map
    toOperation()->asOrderedSet();
}

/* map a PIM operation to a PSM identically */
mapping Operation::toPSMOperation() : Operation {
  name := self.name;
  type := self.type;
  ...
  ownedParameter := self.ownedParameter->map
    toPSMParameter()->asOrderedSet();
}

/* map a Controller class to a Service class */
mapping Class::toService():Class
inherits Class::toPSMClass
when{
  self.isStereotypeApplied(ACCURATE::controller)
}{
  end {
    result.applyStereotype(Spring2::Service);
  }
}

/* map an operation on a controller class to
   a ServiceOperation */
mapping Operation::toServiceOperation():Operation
inherits Operation::toPSMOperation
when {
  self.class.isStereotypeApplied(ACCURATE::controller)
}{
  end {
    result.applyStereotype(Spring2::ServiceOperation);
  }
}
```

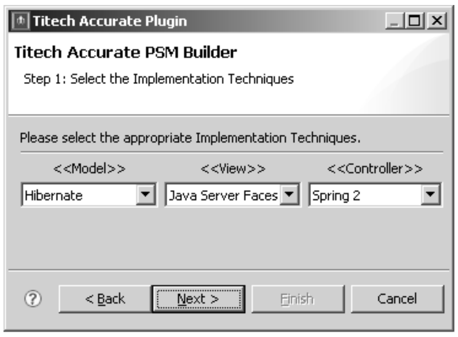
Figure 4: Mappings between the ACCURATE and the Spring2 Profiles

for the following code generation activity.

To implement this transformation, we defined mappings between elements of a PIM and a PSM for each supported platform. A transformation can be achieved by a stepwise conversion of all contained elements of the PIM due to the mappings to PSM elements. To define these mappings, we categorized existing stereotypes and tagged values of the profiles for PSMs according to the established concepts used in the architecture styles. Although architecture styles defines typical structure and behavior of the elements, they usually need to be modified due to additional constraints enforced by target platforms.

For example, let's consider a mapping from a PIM element *OrderService* with the stereotype ≪`controller`≫ (see Fig. 3) to the PSM element *OrderService* with ≪`Service`≫ and ≪`ServiceOperation`≫ stereotypes for the Spring Framework (see Fig. 1). Figure 4 shows a part of the mappings specified in the MOF QVT operational language [17]. These mappings create PSM elements from input PIM elements and map ACCURATE stereotypes to Spring ones.

As one might notice, not only the stereotypes and tagged values need to be changed, but it is also required to remove unnecessary elements (such as ≪`external`≫ stereotyped elements) or modify the structure (e.g. change associations to dependencies) in this transformation.


Titech Accurate Plugin

Titech Accurate PSM Builder

Step 1: Select the Implementation Techniques

Please select the appropriate Implementation Techniques.

<<Model>>	<<View>>	<<Controller>>
Hibernate	Java Server Faces	Spring 2

Figure 5: Platform Selection Dialog for the MVC Architecture Style

3.4 Code Generation

Source code for the application based on the platform configuration is generated at the end of the workflow. Here we make use of existing code generator frameworks (such as oAW, AndroMDA) that support various platforms by separating definitions of transformation mappings from their execution engines. Such transformation mappings are often called *cartridges* due to their replaceable character and stored in the transformation repository for reuse (as shown in Fig. 2).

According to the platform configuration determined by the platform decision activity, transformation mappings are chosen from the repository to configure a code generator specific to that platforms. Using a valid PSM from the PIM-to-PSM transformation activity, source code generation can be less error-prone.

4. SUPPORTING TOOLS

We have prototyped a PIM-to-PSM transformation tool as a plug-in for the Eclipse platform. This tool implements transformation mappings using the QVT operational language implemented by the Eclipse M2M [8] project. It offers a user interface to specify a PIM and platform decisions for the system with a wizard-style dialog shown in Fig. 5. Users just have to select appropriate platforms for the ≪`model`≫, ≪`view`≫ and ≪`controller`≫ parts of the target system from drop-down menus.

After the wizard dialog is finished, a PSM and a platform configuration file are generated. This file is used in the following oAW code generator to distinguish, which transformation mapping have to be executed from the transformation repository to generate source code conforming to the designated platforms. As for the PSM-to-PSI transformation, we make use of the oAW code generator framework. One common transformation repository for oAW is the Fornax-Platform, which offers a variety of cartridges to generate application code based on profiled UML models and thus is a possible candidate for the code generator in our tool chain.

5. CASE STUDY

In order to evaluate our approach, we have carried out a case study derived from a possible real-world scenario in which a system is using a specific platform technique. Due to changing requirements of the project, the platform decision was out-dated. As a result, the PSM and PSI have to be regenerated to adopt the new platform decision. The aim of the case study is to show that a platform, developed using the ACCURATE approach, can handle such a situation properly. Furthermore, we are going to argue on the feasibility and benefits of our approach in Sect. 5.2.

5.1 Address Book Example

Consider a company that is implementing an application for managing their customer's addresses using the ACCURATE approach.

At the beginning of the scenario, designers described a PIM and requirements. The ACCURATE profile is applied to the PIM (as shown in Fig. 6). Around the same time, requirements engineers assessed quality attributes of the system and determined to adopt Hibernate for the ≪`model`≫, POJO's for the ≪`controller`≫ and a Swing GUI for the ≪`view`≫. Using the PIM and the platform decision, the ACCURATE plug-in generated an accordant PSM (see Fig. 7(a)). After transforming the PIM into a PSM, the application consists of 28 generated Java classes (six for Hibernate and 22 POJO's). From these 22 POJO's only seven classes have to be implemented manually since the remaining 15 classes are automatically generated interfaces, abstract or implementation classes that don't need to be modified. Besides this, three Hibernate mapping files and a Hibernate property file are generated that also not have to be modified. The PIM-to-PSM transformation took about one minute and the PSM-to-PSI transformation around ten seconds with an average laptop PC (with a Pentium M processor at 1.60 GHz and 1.5 GB of memory) in this scenario.

At some point of the project, the project manager decided to adopt the Spring Framework as a ≪`controller`≫ technology. Since the PIM doesn't not hold any platform specific information by definition, no changes to the PIM have to be made. Using the ACCURATE plug-in again, another PSM conforming to the new platform is generated in about one minute (as shown in Fig. 7(b)). One might notice that Swing is still used as the ≪`view`≫ technology but the PSM elements are mapped to ≪`SpringBean`≫ instead of ≪`JavaObject`≫ this time. Afterwards, the PSM-to-PSI transformation is triggered to regenerate the source code, which took around ten seconds. At this point manual implementation of the missing parts has already been finished. Since oAW doesn't overwrite manual implementation classes during the PSM-to-PSI transformation, the number of newly generated artifact in this second scenario is lower than before. As a result, one interface, abstract and implementation class for each ≪`controller`≫ component was generated. These classes are stored at a different location due to the platform specification. Furthermore, two helper classes for enhanced access to Spring beans and three configuration files are automatically generated. As the final task, the developer has to move the manually implemented code fragments from the outdated ≪`controller`≫ classes to the newly generated ones.

5.2 Discussion

One of the main benefits shown in this case study is that the platform of the application can be switched within a small time period and without modifying the PIM at all. Since the ACCURATE profile is based on architecture styles, which have an essentially platform independent and immutable nature, PIMs using such a profile show improved maintainability and reusability. Thus, they could live on until some functional requirement changes or platform evolutions occur in the future.

Furthermore, in case that new platforms emerge they need to be adopted to our approach, e.g. another implementation technique for ≪`view`≫ classes. In such a case, we only have to define a transformation from our PIM to the PSM of the new platform, as long as this platform conforms to some architecture styles adopted in the ACCURATE approach. Compared to arbitrary PSM-to-PSM transformations like the example shown in Sect. 2, it is rather straightforward to refine PIM concepts to those of PSM and thus most of the transformation can be automated. It has to be mentioned, that our approach expect a code generator together with a PSM defi-

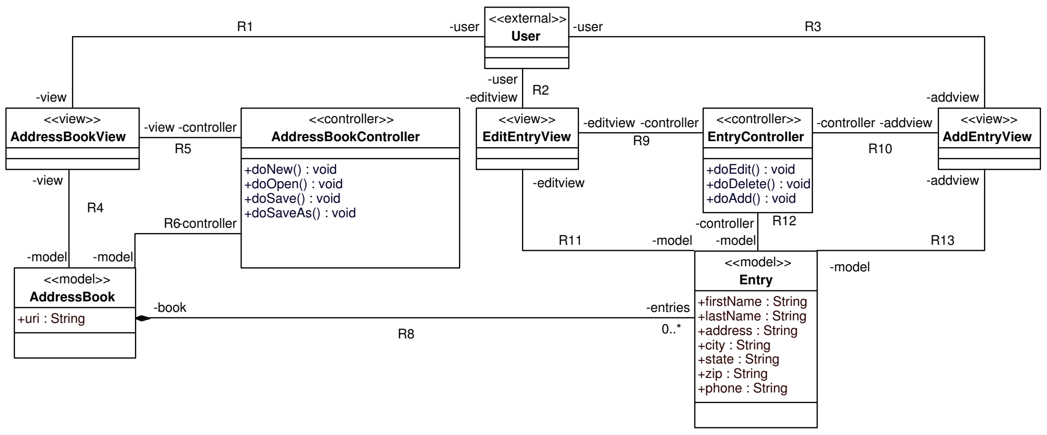
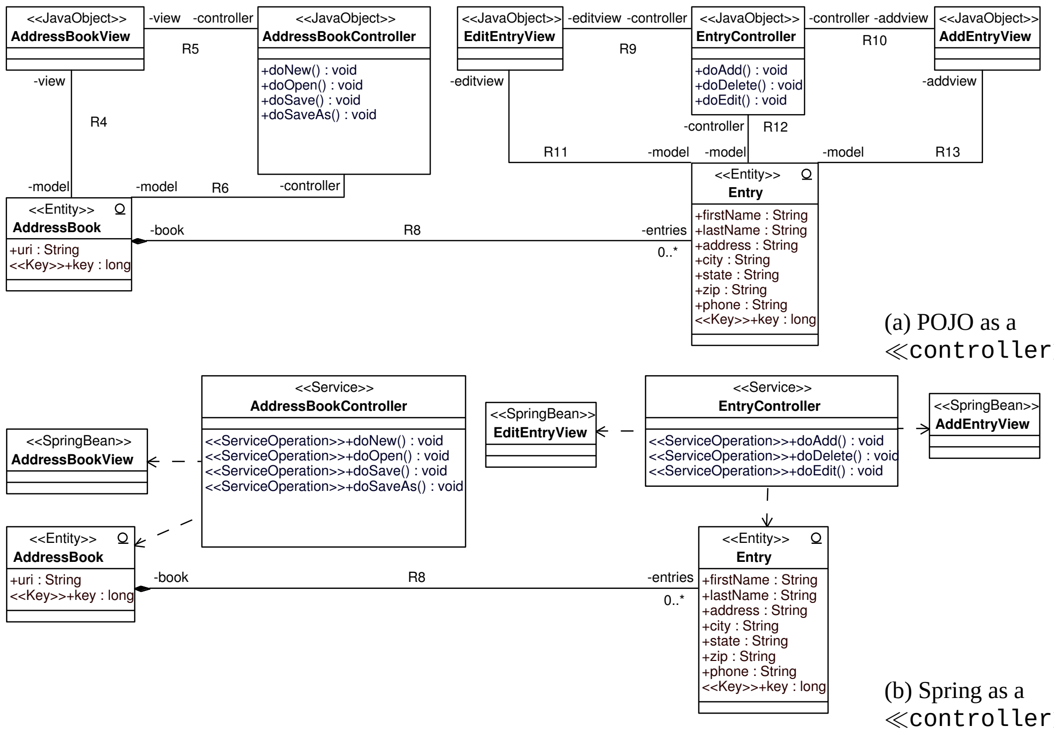


Figure 6: A PIM for the Address Book Example



(a) POJO as a ≪`controller`≫

(b) Spring as a ≪`controller`≫

Figure 7: Two PSMs using Hibernate as a ≪`model`≫ and Swing GUI as a ≪`view`≫

nition. Otherwise, we would have to implement the PSM-to-PSI transformation by ourselves.

Another advantage worth mentioning is that handwritten parts of source code are preserved during code regeneration. In the case study, Swing GUI and Hibernate classes are reused, except that their instantiation code (i.e. constructor calls) is replaced by a XML configuration file for Spring. On the other hand, the generated part for ≪`controller`≫ classes are regenerated based on the Spring service components, while handwritten part of the POJO's are left untouched. This means that there are some remaining parts, which have to be migrated manually, even though task can be achieved in a reasonable time due to the size of the handwritten code. We suppose, that generating complete source code from PIM or providing help and guidance for each possible migration are two possible solutions to address these problems.

6. RELATED WORK

There is already some existing work focusing on platform independent modeling and model transformation in a different problem domain. Bezivin et al. [4] propose to use ATL transformation [7] to transform PIMs defined by Enterprise Distributed Object Computing into PSMs for different web service platforms. Billig et al. [5] define PIM-to-PSM transformations in the context of EJB by using QVT [17]. Besides this, some related work define PIMs via UML profiles. Link et al. propose to use GUIProfile to model PIMs and transform them into PSMs [13]. Richly et al. focus on a UML profile to define PIMs for databases [11]. He et al. use template role models together with PIM profiles for templates to design PIMs, which are specific for web applications [12] . Ayed et al. propose a UML profile for modeling platform independent context-aware applications [2]. Lopez-Sanz et al. define a UML profile for service-oriented architectures [14]. Finally Fink et al. combine UML and MOF profiles for access control specifications [9]. As one can notice, there are a lot of approaches, which describe a PIM on a more abstract level than a PSM. Even so, these approaches are still tailored to a specific technology or architecture and thus need some detailed knowledge of the concrete problem domain. Furthermore, the adoption to a different problem domain or architecture such as MVC is hindered due to the specific notations of these PIMs.

7. CONCLUSION AND FUTURE WORK

In this paper, we have stated out some clear problems of MDD approach when it has to change a platform to another. To address this problem, we have introduced an approach called ACCURATE, which consists of a profile for describing PIMs and transformation mappings to bridge a PIM to PSMs of existing code generators. Our approach shows how to specify systems easily without any PSM modeling skills. The approach offers much automation of the development process and thus reducing costs under the pressure of a shorter time-to-market.

Furthermore, a prototype tool is provided, which both assures the integrity during model transformation and offers guidance through the software development process to the user. The current implementation of the tool provides a workable and extendable solution to address the stated problems. However, there are still some enhancements that we would like to adopt to our approach in the near future. These possible extensions can be summarized as follows:

1. **Further evaluation:** This paper focused on applying the ACCURATE approach to the MVC architecture style. Since this is just one possible example for an architecture, we would like to evaluate our approach to a different architecture style (e.g. Pipes and Filters or Blackboard) and on a larger scale to prove the applicability more sustained.

2. **Platform decision models:** As mentioned before, the platform decision can be supported by assessing quality attributes expected for the system. We are going to introduce platform decision models more precisely. The first model we are now focusing on is based on Bayesian networks, which allows to infer platform decisions based on predefined probability distribution metrics.

3. **Interaction with coding:** In theory, complete source code could be generated from a model. But due to unfamiliarity of graphical PIM modeling and immaturity of tool support for MDD at this moment, developers prefer to finish up implementation by complementing or adjusting generated source code in their common programming languages like Java. We would like to adopt oAW recipes to help the developer track the missing parts of the implementation, and hopefully propagate changes in source code (e.g. adding a method) to its originated PIM.

8. REFERENCES

[1] AndroMDA.org. AndroMDA.org - Home. `http://www.andromda.org/`.

[2] D. Ayed and Y. Berbers. UML Profile for the Design of a Platform-Independent Context-Aware Applications. In *MODDM'06: Proceedings of the 1st Workshop on Model Driven Development for Middleware*, pages 1–5, 2006.

[3] J. Bezivin and S. Gerard. A Preliminary Identification of MDA Components. In *GTCMDA'02: Proceedings of the OOPSLA 2002 Workshop in Generative Techniques in the Context of Model Driven Architecture*, 2002.

[4] J. Bezivin, S. Hammoudi, D. Lopes, and F. Jouault. Applying MDA approach for Web service platform. In *EDOC'04: Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference*, pages 58–70, 2004.

[5] A. Billig, S. Busse, A. Leicher, and J. G. Süss. Platform Independent Model Transformation Based on TRIPLE. In *Middleware'04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 493–511, 2004.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc., 1996.

[7] eclipse.org. ATLAS Transformation Language (ATL). `http://www.eclipse.org/m2m/atl/`.

[8] eclipse.org. Model to Model (M2M) Project. `http://www.eclipse.org/m2m/`.

[9] T. Fink, M. Koch, and K. Pauls. An MDA approach to Access Control Specifications Using MOF and UML Profiles. *Electronic Notes in Theoretical Computer Science*, 142:161–179, 2006.

[10] fornax-platform.org. The Fornax-Platform. `http://www.fornax-platform.org/`.

[11] D. Habich, S. Richly, and W. Lehner. GignoMDA: Exploiting Cross-layer Optimization for Complex Database Applications. In *VLDB'06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 1251–1254, 2006.

[12] C. He, F. He, K. He, and W. Tu. Constructing Platform Independent Models of Web Application. In *SOSE'05: Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering*, pages 85–92, 2005.

[13] S. Link, T. Schuster, P. Hoyer, and S. Abeck. Focusing Graphical User Interfaces in Model-Driven Software Development. In *ACHI'08: Proceedings of the 1st International Conference on Advances in Computer-Human Interaction*, pages 3–8, 2008.

[14] M. López-Sanz, C. Acuña, C. Cuesta, and E. Marcos. UML Profile for the Platform Independent Modelling of Service-Oriented Architectures. *Software Architecture*, pages 304–307, 2007.

[15] No Magic. MagicDraw UML. `http://www.magicdraw.com/`.

[16] OMG. MDA Guide Version 1.0.1. `http://www.omg.org/docs/omg/03-06-01.pdf`, 2003.

[17] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.0. `http://www.omg.org/docs/formal/08-04-03.pdf`, 2008.

[18] openArchitectureWare.org. Official openArchitectureWare Homepage. `http://www.openarchitectureware.org/`.