Association for Information Systems
AIS Electronic Library (AISeL)

ECIS 2007 Proceedings

European Conference on Information Systems (ECIS)

2007

An Extended MDA Method for User Interface Modeling and Transformation

Jen-Her Wu
National Sun Yat-Sen University, jhwu@mis.nsysu.edu.tw

S Shin
x3216@mail.meiho.edu.tw

J Chien
frank.chien@msa.hinet.net

W Chao
chao@mail.nsysu.edu.tw

M Hsieh
hmz@nttu.edu.tw

AN EXTENDED MDA METHOD FOR USER INTERFACE MODELING AND TRANSFORMATION

Wu, Jen-Her, Department of Information Management, National Sun Yat-sen University, 70 Lien-Hai Road, Kaohsiung, 804, Taiwan, jhwu@mis.nsysu.edu.tw

Shin, Shin-Shing, Department of Information Management, Mei-Ho Institute of Technology, Pingtung, 912, Taiwan, x3216@mail.meiho.edu.tw; Department of Information Management, National Sun Yat-sen University, 70 Lien-Hai Road, Kaohsiung, 804, Taiwan

Chien, Juei-Lung, Department of Information Management, National Sun Yat-sen University, 70 Lien-Hai Road, Kaohsiung, 804, Taiwan, frank.chien@msa.hinet.net

Chao, William S., Department of Information Management, National Sun Yat-sen University, 70 Lien-Hai Road, Kaohsiung, 804, Taiwan, chao@mail.nsysu.edu.tw

Hsieh, Ming-Che, Department of Information Science and Management Systems, National Taitung University, 684 Sec.1, Chung-Hua Road, Taitung, 950, Taiwan, hmz@nttu.edu.tw

Abstract

This study presents a systematic methodology for MDA transformation that includes creating a platform independent model (PIM), transforming PIM into platform specific model (PSM), and transforming PSM into a Code model. The PIM is created first based on the use case, activity diagram and robustness diagram. The PIM is then transformed into a PSM based on three target platforms: Web-based user interface, Java, and relational database. The PSM is then transformed into a Code model based on three types of code: JavaServer Pages (JSP) 2.0, J2SE 1.5.0 and Oracle Database. With this methodology, systems can more easily and systematically be analyzed, designed, and generated and, thereby, increase system development productivity.

Keywords: Model-Driven Architecture, Unified Modeling Language, Object-Oriented Technique, Robustness Analysis, User Interface.

1 INTRODUCTION

Software development is a labor intensive and costly job. The United States alone devotes at least $250 billion each year to application development in approximately 175,000 projects involving several million people (Mellor et al. 2004; Mahmood et al. 2005). Software development poses numerous problems that must be solved. For instance, with a new technology, the legacy system needs to be upgraded to align with a new platform and many of them cannot afford to lag behind. This forces existing systems to operate with new systems, causing a portability problem. Systems are never built using only one technology and thus always need to communicate with other systems. The model driven architecture (MDA) approach has been adopted by the software industry to ameliorate the above problems. This approach has been considered a useful solution for improving the efficiency of software development (Uhl, 2003).

The MDA considers everything as a model or model element. Each model or model element is unit independent of the other models and elements. Model elements can be transformed into applications for other models and even be integrated together through communication bridges (Koehler et al. 2005). With such idea, software development becomes a model construction and transformation procedure (Caplat and Sourrouille, 2005). Independence and transformation provide the possibility for ameliorating the foregoing problems. However, MDA is just a principle and there is no single universal MDA framework that fits all software architectures and applications. Therefore, to make MDA model construction and transformation practical, transformation rules must be developed between models for each specific platform (Duddy et al. 2003). This is a challenging task.

Over the past few years a considerable effort has been made in the MDA transformation area (Agrawal et al. 2003; Czarnecki and Helsen, 2003; Wu et al. 2005; Kleppe et al, 2003). However, little attention has been paid to the issue of user interface transformation, especially for the web-based 3-tier systems. This study, therefore, presents a systematic methodology for MDA transformation which integrates the robustness analysis, activity diagram and state diagram to extend the MDA approach for PIM modeling and user interface transformation from PIM to PSM and to Code model for the web-based 3-tier systems.

2 LITERATURE REVIEW

2.1 Model driven architecture and transformation approach

Model Driven Architecture is a new approach for software development defined by the Object Management Group. MDA regards system development as model building and transformation. These concepts separate the application modeling away from the implementation details, and allow developers to build a model without knowledge of the other models in the system. The existing models can be combined in the final step to create a whole system. This prevents design decisions from becoming intertwined with the application. It also renders the application independent of its implementation; allowing it to be recombined with other technologies at some later time (Mellor et al. 2004; Bézivin et al. 2004).

MDA has been considered to be an enabling software development tool for increasing software development productivity and reducing the software development cost and time to market (Uhl, 2003). MDA concepts are closer to the problem domain at hand than those offered by programming languages. However, the above advantages are based on the assumption of a transformation mechanism application among models. Many approaches have been proposed to address this issue. Sendall and Kozaczynski (2003) classified these approaches into three categories: (1) Direct model manipulation, (2) Intermediate representation, and (3) Transformation language support.

In *direct model manipulation*, transformation logics are contained that can access source models and transform those into codes. The logics can be written as a series of APIs to handle model elements and properties such as get, create and update. For instance, the logics could simply be Java codes that access APIs provided by Java Metadata Interface (JMI). The other example in this category is Jamda (Czarnecki and Helsen, 2003), which provides a set of classes representing UML models and several APIs for manipulating those models. However Jamda's work does not support the MOF standard which dominates this domain. The advantage is that developers can use their preferred language to write the transformation logics such as Java. This could reduce the difficulty and complexity of constructing transformation rules. On the other hand, APIs lack high-level abstractions for specifying transformations (Liu et al. 2005) and may limit the kinds of transformations that can be performed (Mens and Van Gorp, 2005).

In *intermediate representation*, model transformations are performed through a standard intermediate form. Take the transformation from PIM to PSM as an example, a PIM model could be an input model and be transformed into a XMI (XML Meta data Interchange) document. The PSM model could then be generated according to the XMI document transformed from the PIM. Usually, the transformation is achieved using stand alone tools loosely coupled with modeling tools such as XSLT, which allows the manipulation of XML files to accomplish the transformation from the source to the target model. However, manual implementation of model transformation in XSLT quickly leads to a non-maintainable implementation because of the verbosity and poor readability of XMI and XSLT. The current XMI standard does not contain diagramming information. After a model is exported into a XMI document, all diagram information is lost (Weis et al. 2003).

The *transformation language support* uses a specific language to specify the model transformation. The most famous one in this category is the graph-transformation based approach, whose transformation rules consist of a left-hand side (LHS) and right-hand side (RHS) graph pattern. Once the LHS pattern of a transformation rule is matched in a source model, a rule is fired to replace that one using the RHS pattern in its place. Because current OMG graphical language such as MOF and UML provide a well-established foundation for defining PIM and PSM, the graph-transformation based approach seems to be a natural solution for specifying the transformation. It has even been considered the approach with the most potential (Sendall and Kozaczynski, 2003). This approach complies with the principle of divide-and-conquer. It decomposes complex transformation into several smaller parts that can be conquered individually and then reintegrated into a whole system. This approach increases the readability, modularity and maintainability of a transformation mechanism. The other advantage of this approach is the visualization using graphic transformation. On the visual notation basis, the PIM and PSM can be expressed in a visual way that could make developers more comfortable in using this approach. Several methodologies have been developed to provide this feature. For example, Agrawal et al (2003) developed a UML-based approach for specifying model transformations using a class diagram to represent the input and output transformation graph grammars. The transformation language is called Graph Rewriting and Transformation language (GreAT). Even though a few methodologies have been developed, a transformation method for user interfaces from user requirements to Code is still lacking. This motivates us to develop a transformation mechanism based on the concept of graph-transformation based approaches.

2.2 Robustness analysis

Robustness analysis introduced by Ivar Jacobson acts as a mediator to bridge the gap between modeling use case diagram and sequence diagram (Zhou and Stålhane, 2004). It provides an approach for classifying objects and their courses of interaction from the use case to build the robustness diagram which shows the objects that participate in the scenario and how those objects interact with each other as shown in Figure 1. Robustness analysis classifies objects in a system into three stereotypes: boundary object, entity object and control object. Entity objects represent data stored in a database. Boundary objects are deemed user interfaces and triggered by users to communicate with

control objects, which capture application logics and act as bridges between boundary objects and entity objects. The interaction rules among these objects can be summarized as follows (Rosenberg and Scott 2001):

- Actors can only communicate with boundary objects.
- Boundary objects can only communicate with control objects and actors.
- Entity objects can only communicate with control objects.
- Control objects can communicate with boundary objects, entity objects and the other control objects, but not with actors.

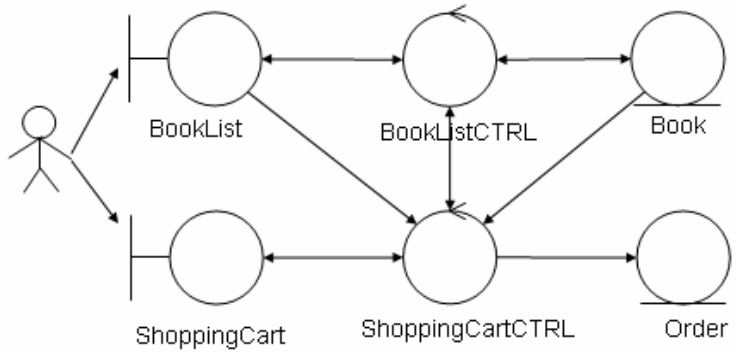


Figure 1. An example of robustness diagram

3 THE FRAMEWORK OF USER INTERFACE TRANSFORMATION

With respect to 3-tier software architecture, Kleppe et al (2003) developed an MDA framework including three models in a hierarchical order: Platform Independent Model (PIM), Platform Specific Model (PSM) and Code model. The model at the first level is PIM which provides a formal function and structure system specification without technical details. The model at the second level is PSM which is derived from PIM to target a specific technology. PSM is tailored to specify a system in terms of the implementation constructs that are available in one specific implementation technology. The model at the third level is the Code model, which is the executable program transformed from the PSM model. In addition, OMG defined the computation independent model (CIM) prior to the stage of PIM to model user requirements (Miller and Mukerji, 2003). CIM doesn't consider how a system is implemented and focuses on what the system is expected to do. CIM should be traceable to PIM and vice versa.

Based on Kleppe et al's framework, we draw an MDA architecture for 3-tier JSP applications, as shown in Figure 2. This architecture encompasses four levels: CIM, PIM, PSM and Code model. At the first level, CIM, three tools are used to analyze user requirements and produce preliminary objects including boundary, control and entity objects and communication bridges. At the second level, PIM, these objects can then be refined or enhanced into the object in the sequence diagram. Besides the boundary objects, control objects and entity objects in a robustness diagram correspond to boundary classes, object classes and entity classes in a sequence diagram in PIM, respectively. At the Third level, PSM, the MVC (model-view-controller) model is used as the fundamental system architecture. It separates the PSM into three distinct tiers: model, view and controller, so that modifications to one tier can be made with minimal impact on the others. The model dubbed relation schema is the domain-specific representation of the information on which the application operates. It contains a data model and domain logics which add meaning to the raw data. The view is termed widget class diagram which refers to how to render the model into user interfaces for interaction. The controller referred to as Java class diagram responds to the model and the view and invokes changes on those. Especially, these three tiers can be transformed from PIM plus JSP, database and Java technology.

Further, the PSM can be transformed into real executable program namely Code model. In this study, the widget class diagram can be transformed into JSP code. The Java class diagram can be transformed into Servlet. The relation schema can be transformed into JavaBean and database schema. Database schema is used to generate a database for JavaBean Code. JavaBean acts as an interface to access and store data in the database for Servlet.

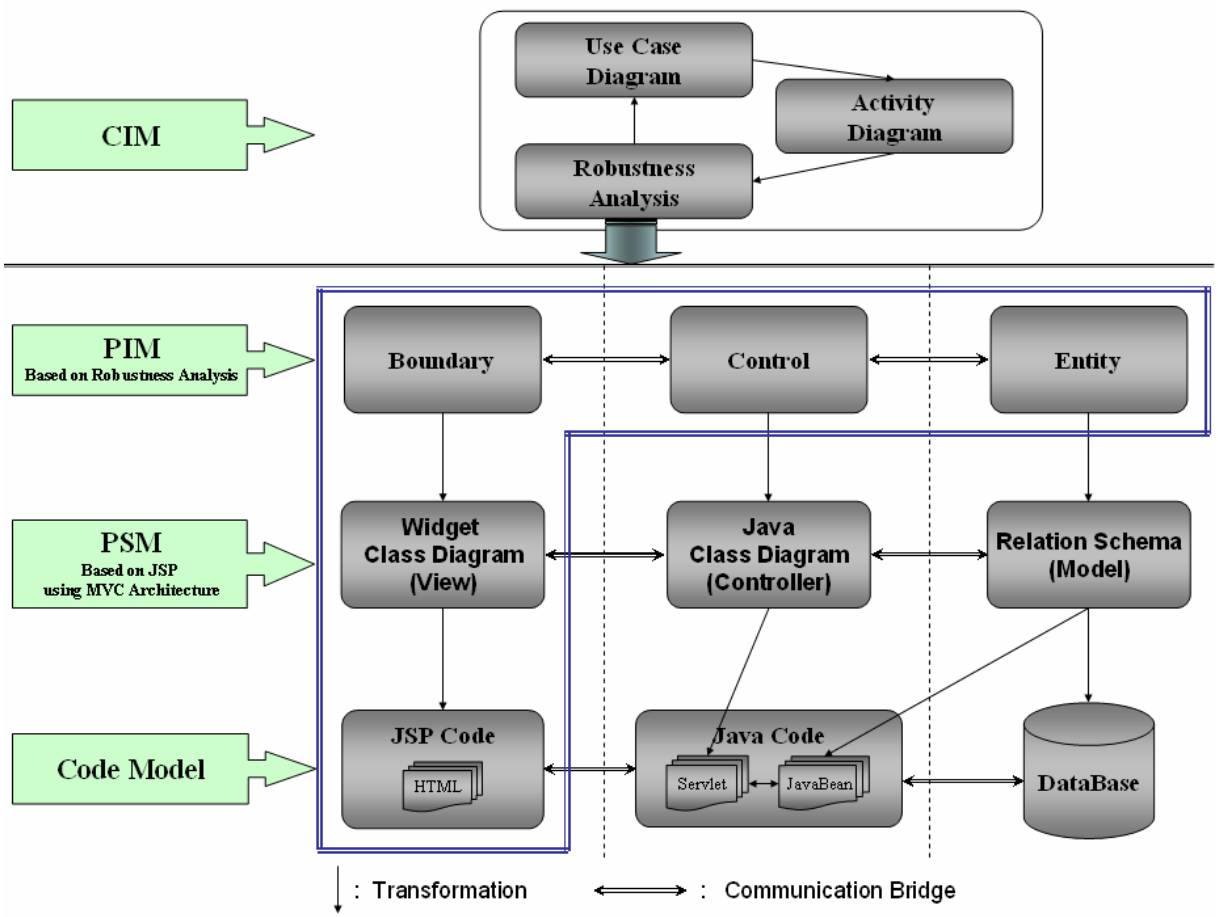


Figure 2. The MDA architecture for 3-tier JSP software.

In this study, we focus on the UI and propose a framework for MDA transformation from boundary to JSP code through widget class diagram as marked by the solid line in Figure 2. This framework consists of four levels; CIM, PIM, PSM, and JSP code. In CIM level, use case diagram and activity diagram are used to model user requirements; the robustness analysis is used to enhance the completeness and correctness of user requirements. The outcome can then be further enhanced in two facets in the PIM level: behaviour and structure. The behaviour part is presented using the sequence diagram, while the structure part is depicted using the class diagram. Once the PIM model is constructed, it then can be used to produce widget class diagram based on MVC architecture in the PSM level. JSP code is finally generated from the PSM.

3.1 Computation Independent Model

In the study, use case diagram, activity diagram and robustness analysis are used to represent the CIM. Use case diagram has been a well-known tool to model user requirements. Once a use case diagram is constructed, the outcome can then be used to construct a sequence diagram in the PIM. To do so, the robustness analysis has been used to help the identification of boundary, control, and entity objects

and enhance the completeness of the user requirement (Rosenberg and Scott, 2001). However, it's still not a straightforward task to identify objects, operations and their relationships from the use case diagram. The activity diagram can be used to alleviate the problems since it is used to describe the activity flow and the associated input/output of each use case. The activity diagram provides a visual manner to represent the actors and their related activities, needed information, flow controls and transitions. This is helpful for discovering objects and process flows and furnishes a basis for performing robustness analysis. For instance, a note of an activity could be considered as a user interface namely a boundary object. Data in the note can be regarded as attributes of the boundary object. An activity could be deemed as a control object and so forth. The transformation rules from activity diagram to robustness diagram are summarized as in Table 1. This construction procedure is not only providing an aid in establishing a robustness diagram, but also useful in clarifying user requirements.

Rule	Activity Diagram	Mapping	Robustness Diagram
1.	Activity	→	1. Activities could be transformed into a control object. 2. User behavior in an activity can be implemented by widgets. These widgets could be transformed into attributes and operations of boundary objects.
2.	Note	→	1. A note could be transformed into a boundary object. 2. Data in a note could be transformed into attributes of the boundary object. 3. While data in a note is stored in a database, entity objects with attributes could be generated from the data. Also, operations for retrieving and storing the data could be identified and delivered to the entity objects. 4. If data in a note is retrieved from a database, an operation creating the note in a control object would be generated.
3.	Transition	→	1. If transition invokes a control object to fire its action, the control object should contain an operation responding the invocation. 2. If transition send data to an activity or receives data from an activity, the data would be transformed into attributes of a control object.

Table 1. Mapping rules from an activity diagram to a robustness diagram.

3.2 PIM

The objective of PIM is to provide the formal function and structure specifications for systems without technical details. To achieve this end, two tasks must be performed: behaviour and structure modeling. The sequence diagram is the major modeling tool for behaviour modeling. However, for the user interface, the sequence diagram falls short in modeling the information about which UI widget triggers which operation, while this information is essential for programming. Therefore, we propose that each UI sequence diagram should be fraught with a UI state diagram to clearly specify the behaviour of UIs and widgets. The state diagram considers each UI as a state and widget as a sub-state, and reveals their state transitions including event, guard and action information. This concept brings the sequence diagram and the state diagram together, underpinning the PIM class diagram construction. It provides the designer with detailed specifications for deciding which UI widget could serve in a specific platform.

3.2.1 Behaviour modeling

The sequence diagram is a good tool for modeling system behaviour because it brings together objects, operations and sequences simultaneously. A sequence diagram can be built by transforming information from robustness and activity diagrams. The transformation rules were established as shown in Table 2.

Rule	Robustness/Activity Diagram	Mapping	PIM Sequence Diagram
1.	A boundary object in a robustness diagram.	→	It could be a boundary object in a sequence diagram.
2.	A control object in a robustness diagram.	→	It could be a control object in a sequence diagram.
3.	An entity object in a robustness diagram.	→	It could be an entity object in a sequence diagram.
4.	An association between two objects.	→	It could be a message between these two objects' focus of control.
5.	An operation of an object.	→	It could be an operation on a message.
6.	An attribute of an object.	→	It could be a parameter of operations or a return value of an object.
7.	An event which could be an activity or a transition in an activity diagram.	→	An event can be transformed into a chain of messages and objects involved to achieve the event' goal.
8.	A loop in an activity diagram	→	It could be a loop frame.
9.	A branch transition in an activity diagram	→	It could be an alternative or option frame.
10.	A fork transition in an activity diagram	→	It could be a parallel frame.

Table 2. The transformation rules from a robustness/activity diagram to a sequence diagram.

3.2.2 Structure modeling

Behaviour modeling concerns objects and their interactions at the instance level, while the structure modeling highlights classes and their static relationships between those. To establish a PIM class diagram, it consists of two steps: (1) generalize classes, (2) identify relationships. In the first step, objects with the same attributes and operations can be abstracted to form a class. Obviously, each object in a PIM sequence diagram can be transformed into a class in a PIM class diagram. Most of the relationships among classes can be identified based on the message passing in the sequence diagram. The rules are shown in Table 3. Table 3 indicates that rules 4-8 act as an aid in identifying dependency relationships. Usage is one kind of dependency predefined by OMG, which indicates a situation in which one class requires the presence of another class for its correct implementation or functioning. A usage represents calling an operation from another class or instantiating an object of another class, etc. It even can be stereotyped further to indicate the exact nature of a dependency. Therefore, a calling message in a PIM sequence diagram can be notated as a usage with *call*, and a return message can be notated as a usage with *send* in a PIM class diagram. If the instance of a class is created by another class, it can be notated as a usage with *instantiate*. The association relationships with multiplicity between entity objects can be identified based on the information passing among the objects. If there is many-to-many multiplicity in an association relationship, an association class needs to be created to record the association information.

Rule	PIM Sequence Diagram	Mapping	PIM Class Diagram
1.	A boundary object	→	A boundary class
2.	A control object	→	A control class
3.	An entity object	→	An entity class
4.	A message with operation from a boundary object to a control object	→	A usage with call from the boundary class to the control class
5.	A message with operation from a control object to an entity object	→	A usage with call from the control class to the entity class
6.	A message with return values from an entity object to a control object	→	A usage with send from the entity class to the control class
7.	A message with return values from a control object to a boundary object	→	A usage with instantiate from the control class to the boundary class
8.	A message with operation between two control objects	→	A usage with call between the two control classes

Table 3. The transformation rules from a PIM sequence diagram to a PIM class diagram.

3.3 PSM

At the PSM phase, a PIM is tailored to a specific platform. In this study, the technologies employed in the PSM for boundary, control and entity classes are JSP, Java and Oracle DB, respectively. This study focuses on the UI transformations from PIM through PSM to Code. The application and database transformation are beyond the scope of this study.

3.3.1 PSM class diagram

Many widgets have been developed to facilitate user interface design. Most designers don not develop UI components in house and use widgets available on the market. As noted in the previous section, a user interface is defined by a drawing, a boundary class and a UI state diagram. This is obviously insufficient for establishing a PSM class diagram including widget information. The PIM focuses on the user's initial requirements without technical information, while the PSM is forced not to ignore low level details for specific technologies. A UI function may be implemented using different widgets with the same function. It is by no means clear how to model user interfaces using UML. There is a lack of uniformity and standardisation in widget selection and integration (da Silva and Paton, 2003). It seems that there are no theoretically or even pragmatically well-developed guidelines to help software engineers tackle this problem successfully.

We believe an interactive mechanism is useful to mitigate these problems. This mechanism should provide a UI presentation structure and allow users/designers to choose adequate widgets for achieving a function. To do so, we adopted the presentation model introduced by da Silva and Paton (2003). It contains a top-level container, Form, which is composed of other Containers that consist of UI components. A UI Component is specialised into three categories: StaticDisplay, ActionInvoker and InteractionControl. The StaticDisplay category is relevant at those UI components providing visual information, such as labels. The ActionInvoker category is relevant at those UI components receiving user triggers, such as a button. The InteractionControl category is relevant at those UI components receiving user options concerning navigation through the UI, such as a menu. With this premise, we construct representation models based on JSP for each boundary class and then integrate them into a PSM class diagram. Using the BookList (boundary class) and its drawing as an example (Figure 3), the representation model is constructed using four HTML components. For instance, the BookList (drawing) is transformed into the BookList (Form), which is aggregated with a CheckOut button (ActionInvoke) and a Table (StaticDisplay). The CheckOut button comes from the command button *check out*. The grid on the drawing generates the Table (StaticDisplay), which is aggregated with several Table Records (StaticDisplay). Hence, the BookList (boundary class) is modelled as shown in Appendix F. Similarly, the other boundary class, ShoppingCart, can also be transformed.

3.4 Code

The Code phase aims at transforming the UIs and widgets constructed in the PSM phase into code. This is straightforward because every element has a specific corresponding code on JSP platform. We enumerate three basic transformation patterns including form, button and table as shown in Table 4.

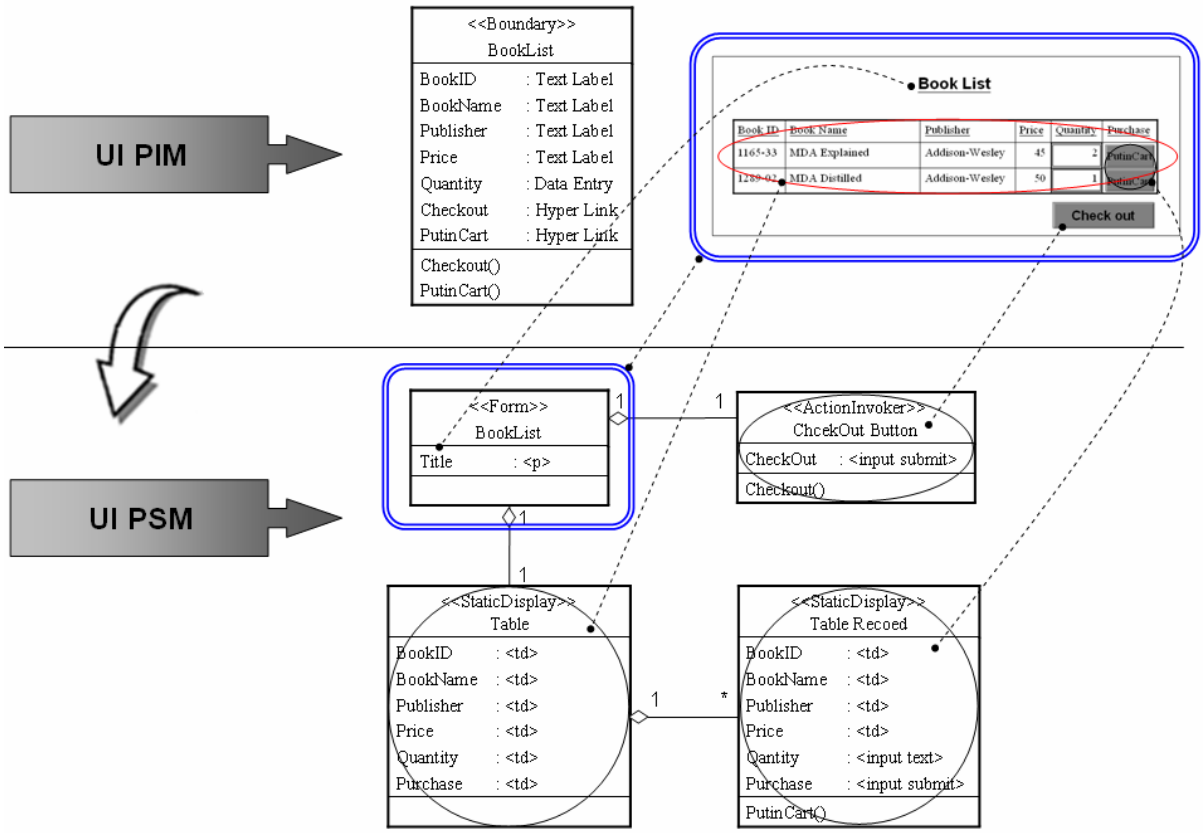


Figure 3. The transformation of BookList from PIM to PSM

Rule	Pattern in PSM class diagram	Code
1	Form <<Form>> Form_Name Title :	1. <%@ page language="java" %> 2. <%@ page session="true" %> 3. <% Vector DataFromControlObject = (Vector) session.getAttribute("Attribute1"); %> 4. <html> 5. <head> 6. <meta http-equiv="Content-Type" content="text/html; charset=big5"> 7. <title>Form_Name</title> 8. </head> 9. <body> 10. Title 11. </body> 12. </html>
2	Button <<ActionInvoker>> Button_Name Parameter1 : <input submit> Communication_Bridge()	1. <form name="Button_Name" action="Communication_Bridge" method="POST"> 2. <input type="submit" value="Parameter1"> 3. </form>

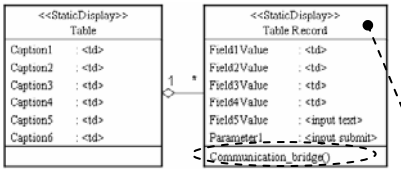
3	Table 	1. <table border="1" width="631"> 2. <tr> 3. <td width="194">Caption1</td> 4. <td width="194">Caption2</td> 5. <td width="81">Caption3</td> 6. <td width="93">Caption4</td> 7. <td width="57">Caption5</td> 8. <td width="47">Caption6</td> 9. </tr> 10. <% for (int i=0; i < DataFromControlObject.size() ; i++) 11. {DataType DataList = (DataType) DataFromControlObject.elementAt(i); 12. %> 13. <tr> 14. <td><%= DataList.getField1Value() %> </td> 15. <td><%= DataList.getField2Value() %> </td> 16. <td><%= DataList.getField3Value() %> </td> 17. <td><%= DataList.getField4Value() %> </td> 18. <td><input type= "text" name= "Field5Value"> </td> 19. <td> 20. <form name="TableRecord" action="Communication_bridge" method="POST"> 21. <input type="submit" value=" Parameter1 "> 22. </form> 23. </td> 24. </tr> 25. <% 26. } 27. %> 28. </table>

Table 4. The transformation patterns from a PSM class diagram to JSP code.

4 CONCLUSIONS AND FUTURE WORK

This study presented a methodology that integrates the MDA, UML, robustness analysis and object-oriented programming concepts to specify user requirements, construct the PIM, and then transform it into user interface PSM and to user interface code based on the 3-tiers JSP architecture. This methodology includes four major levels: CIM, PIM, PSM, and JSP code modeling. A case with prototype system is used to demonstrate the feasibility of this methodology.

The contribution of this paper is two-fold. First, the proposed methodology provides a complete procedure for developing a web-based 3-tiers JSP application including CIM modeling, PIM modeling, and PIM to UI PSM and to UI code transformations. These results provide a greater insight for understanding and reducing the gap among PIM, UI PSM, and UI code. Second, the proposed methodology provides a mechanism to automate the transformation among PIM to UI code and the communication bridge between UI code and application code to increase the system development efficiency.

With this methodology, the web-based, 3-tier JSP applications can more easily and systematically be analyzed, designed, and generated. This work is the beginning of a line of MDA transformation methodology. Future research directions are abundant. For instance, a complete transformation methodology which includes the PIM to relation schema, Java class diagram, and widget class diagram is worthy to perform. Other issues may include developing the software system to support the transformation automation.

Acknowledgement: This research was partially supported by the National Science Council of Taiwan under grant # NSC 95-2221-E-276 -001 and NSC 94-2416-H-110-017.

References

Agrawal, A., G. Karsai and F. Shi (2003). A UML-based Graph Transformation Approach for Implementing Domain-Specific Model Transformations, Technical report, (ISIS), Vanderbilt University, Nashville, TN.

Bézivin, J., S. Hammoudi, D. Lopes and F. Jouault (2004). Applying MDA Approach for Web Service Platform. In Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference – EDOC 2004, pp. 20-24, IEEE Press, Monterey, California, USA.

Caplat, G. and Sourrouille, J.-L. (2005). Model mapping using formalism extensions. IEEE software, 22 (2), 44-51.

Czarnecki, K. and S. Helsen (2003). Classification of model transformation approaches. In Proceedings of OOPSLA 2003 Workshop on Generative Techniques in the Context of MDA, pp. 33-50, Anaheim, CA, USA.

da Silva, P.P. and Paton, N.W. (2003). User interface modeling in UMLi. IEEE software, 20 (4), 62-69.

Duddy, K., A. Gerber, M. Lawley, K. Raymond and J. Steel (2003). Model transformation: A declarative, reusable patterns approach. In Proceedings of the 7th International IEEE Conference on Enterprise Distributed Object Computing (EDOC), pp. 174-195, IEEE Press, Brisbane, Qld., Australia.

Kleppe, A., J. Warmer and W. Bast (2003). MDA Explained-The Model Driven Architecture: Practice and Promise. Addison-Wesley, Boston.

Koehler, J., Hauser, R., Sendall, S. and Wahler, M. (2005). Declarative techniques for model-driven business process integration. IBM Systems Journal, 44 (1), 47-65.

Liu, J., K. He, B. Li, C. He and P. Liang (2005). A Transformation Definition Metamodel for Model Transformation. In Proceedings of the International Conference on Information Technology: Coding and Computing, pp. v-xiv, Las Vegas, Nevada, USA.

Mahmood, S., Lai, R., Soo Kim, Y., Hong Kim, J., Cheon Park, S. and Suk Oh, H. (2005). A survey of component based system quality assurance and assessment. Information and Software Technology, 47 (10), 693-707.

Mellor, J.S., K. Scott, A. Uhl and D. Weise (2004). MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, Boston.

Mens, T., and P. Van Gorp (2005). A taxonomy of model transformations. In Proceedings of International Workshop on Graph and Model Transformation (GraMoT), Tallinn, Estonia.

Miller, J. and J. Mukerji (2003). MDA Guide Version 1.0.1. Eds. Object Management Group

Rosenberg, D. and K. Scott (2001). Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example. Addison-Wesley, Boston.

Sendall, S. and Kozaczynski, W. (2003). Model transformation: the heart and soul of model-driven software development. IEEE Software, 20 (5), 42-45.

Uhl, A. (2003). Model Driven Architecture Is Ready for Prime Time. IEEE Software, 20 (5), 70-72.

Weis, T., Ulbrich, A. and Geihs, K. (2003). Model Metamorphosis. IEEE Software, 20 (5), 46-51.

Wu, J.H., Huang, Y.C. and Shin, S.S. (2005). Object-Oriented Analysis and Design: Transformation from Class Diagram to Relational Table and Application Template. Journal of Internet Technology, 6 (4), 453-461.

Zhou, J. and T. Stålhane (2004). A Framework for Early Robustness Assessment, Software Engineering and Applications (SEA' 04), MIT Cambridge, MA, USA, 64-69.