Association for Information Systems
AIS Electronic Library (AISeL)

ECIS 2012 Proceedings

European Conference on Information Systems (ECIS)

5-2-2012

A MDA-BASED DEVELOPMENT APPROACH FOR 3-TIERS APPLICATIONS

Shin-Shing Shin
National Taitung University

Jen-Her Wu
National Sun Yat-sen University

Ming-Che Hsieh
National Sun Yat-sen University

A MDA-BASED DEVELOPMENT APPROACH FOR 3-TIERED APPLICATIONS

Shin, Shin-Shing, Department of Information Science and Management Systems, National Taitung University, 684 Sec.1, Chung-Hua Road, Taitung, 950, Taiwan, shing@nttu.edu.tw

Wu, Jen-Her, Department of Information Management, National Sun Yat-sen University, 70 Lien-Hai Road, Kaohsiung, 804, Taiwan, jhwu@mis.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 presented a transformation approach of Model Driven Architecture for 3-tiered applications. Mapping rules from computation independent model to platform independent model and from platform independent model to platform specific model were developed. This approach was demonstrated using an on-line bookshop application. With this approach, the weak link of Model Driven Architecture, transformation from computation independent model to platform independent model, could be tackled, and 3-tiered applications could more systematically be analysed, designed, and generated and, thereby, increased system development productivity.

Keywords: Model Driven Architecture, Unified modelling language, Model transformation.

1. Introduction

Model Driven Architecture (MDA) is composed of model construction and model transformation. Over the past few years a considerable effort has been made in MDA (Bergmann et al. 2011, Khadka et al. 2011, van Amstel et al. 2011). However, gaps between computation independent model (CIM), platform independent model (PIM) and platform specific model (PSM) still exist. Currently, the study of MDA focuses almost exclusively on PIM-to-PSM transformations. Although a large number of studies have been on PIM-to-PSM transformations, little is known about CIM-to-PIM transformations probably because of its difficulty. This breaks the formal linkages between the specifications of CIMs, PIMs and PSMs, and reduces the usefulness of MDA.

The major difficulties include: (1) the semantic distance between CIMs and PIMs, (2) the semantic account of modelling techniques used in CIMs. Mapping effort depends on the semantic distance between two formalism notions (Caplat and Sourrouille 2005). Semantic distance is the degree to which concepts are related in meaning (Brooks 1995). Semantic mappings can be classified into inter-language mappings and intra-language mappings. When source and target models are expressed with the same formalism, the mapping is said to be intra-language; otherwise, it is inter-language. Inter-language mappings are more complex than intra-language mappings because inter-language mappings not only describe mapping actions going from a source model to a target model, but they also have to compare the two formalisms to determine the extent to which the semantics of the source model transfer to the target model. CIM-to-PIM transformation belongs to inter-language mappings because their modelling techniques are different. Considering semantic account of modelling techniques, CIMs express higher-level semantics using formalisms that are imprecise and informal such as use cases. The constructs in use cases are nearly semantics-free (Hailpern and Tarr 2006). Use cases lack sufficient semantics to support model refinement. The above difficulties motivate us to develop a CIM-to-PIM transformation approach. With this study, the weak link of MDA, CIM-to-PIM transformation, could be tackled.

In order to provide a complete transformation approach, PIM-to-PSM transformations were included. This study, therefore, developed a transformation approach for 3-tiered applications which defined CIM-to-PIM mapping rules and PIM-to-PSM mapping rules in terms of user-interface, business-logic and data tier. An on-line bookshop was used to demonstrate the feasibility of the approach. The paper proceeds as follows. The next section introduces the difficulty in CIM-to-PIM transformations and MDA, and reviews the category of transformation approaches. The transformation approach for 3-tiered applications is presented in Section 3. An on-line bookshop is presented in Section 4, followed by the conclusion and suggestions for future works in Section 5.

2. Literature review

This section briefly introduces Model Driven Architecture and the category of model transformation methodology.

2.1. Model Driven Architecture

Model Driven Architecture (MDA) was defined by the Object Management Group for software development. MDA regards systems development as model building and model transformation. This concept separates application modelling away from implementation details, and allows developers to build the model without the knowledge of implementation. This prevents design decisions from becoming intertwined with the application. It renders the application independent of its implementation, allowing it to be recombined with other technologies at some later time to create the whole system (Bézivin et al. 2004). A MDA framework for 3-tiered software architecture includes

three models in a hierarchical order: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM).

CIMs view systems from a computation independent viewpoint. This model is referred to as the domain model because it does not show the structural details of a system. CIMs bridge the gap between experts in the domain and experts in the design and the construction of artefacts, enabling them to collaborate on satisfying the requirements of a domain.

PIMs are a view of systems from the computation viewpoint. It's from a "what" view of CIMs into a "how" view of the world, but does not show details of the use of its platform. In designing object-oriented software, PIMs structure software around real-world problem space by identifying the objects and their relationships from computational viewpoint. PIMs are useful to identify objects in terms of three kinds of classes: boundary, control and entity, and clarify their relationships.

PSMs are a model of the same system specified by the PIM, but PSMs are a view of the system from the viewpoint of specific platforms. PSMs are derived from the PIM through combining the specifications in the PIM with the details that specify how the system uses particular platforms. PSMs represent target platforms with sufficient precision and finally are transformed into code.

2.2. Category of transformation approaches

MDA has been considered to be efficient in increasing software development productivity and reducing the software development cost and time to market. The above advantages are built on the assumption of the existence of a complete transformation mechanism. Many transformation approaches have been proposed to address this issue. These approaches are classified into three categories: (1) Direct model manipulation, (2) Intermediate representation, and (3) Transformation language support (Sendall & Kozaczynski 2003).

Direct model manipulation contains transformation logics that can access and manipulate the internal model representation of models. 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 be Java codes that access APIs provided by Java Metadata Interface (JMI) to conduct model transformations. An example in this category is Jamda (Czarnecki & Helsen, 2003), which provides a set of classes representing UML models and several APIs for manipulating those models. However Jamda does not support the Meta-Object Facility of Object Management Group which dominates this domain. The advantage of this category is that developers can use their preferred language to define transformation logics. This could reduce the difficulty and complexity of constructing mapping rules. The disadvantage is that APIs lack high-level abstractions in specifying transformations (Liu, 2005) and the transformation function may be limited by the APIs (Mens & Van Gorp, 2005).

Intermediate representation conducts model transformations through a standard intermediate form, such as XML Meta data Interchange (XMI). Taking PIM-to-PSM transformations as an example, a PIM can be transformed into an intermediate form, XMI document. Then, a PSM can be generated from the XMI document. Usually, the transformation is achieved by stand-alone tools which are loosely coupled with the modelling tools, such as Extensible Stylesheet Language Transformations (XSLT). XSLT allows the manipulation of XML files to accomplish transformations from a source model to a target model. However, manual implementation of model transformations in XSLT quickly leads to a non-maintainable situation because of the verbosity and poor readability of XMI and XSLT.

Transformation language support uses a specific language to specify model transformations. The most famous one in this category is graph-based transformation approaches, whose mapping rules are expressed in a graph pattern containing left-hand side (LHS) and right-hand side (RHS). Once the LHS pattern of a mapping rule is matched in a source model, the rule is fired to replace the LHS pattern of the source model by the RHS pattern in its place. Graph-based transformation approaches seem to be a natural solution for specifying model transformations because the modelling languages provided by Object Management Group, such as activity diagrams and class diagrams, are graphic

representation. This approach has even been considered the most potential solution (Sendall & Kozaczynski 2003). One example is shown in Fig. 2 and Fig. 3. This approach complies with the principle of divide-and-conquer, which decomposes a complex transformation into several smaller parts that can be conquered individually and then reintegrated into the whole one. It increases the readability, modularity and maintainability of the transformation mechanism. Additionally, the visualization of graphic transformation makes system developers more comfortable in using this approach. One example is Graph Rewriting and Transformation language (GreAT). GreAT diagrammatically specifies mapping rules using class diagrams (Agrawal et al., 2003). GreAT focuses on PIM-to-PSM transformations, while this study tried to develop graph-based CIM-to-PIM transformations.

3. The transformation mechanism for 3-tiered applications

This study developed a transformation approach for 3-tiered applications, as shown in Fig. 1. This approach consisted of three steps: (1) creating the CIM of systems using use case diagrams and class diagrams; (2) developing CIM-to-PIM mapping rules and generating PIMs from CIMs by these rules; (3) developing PIM-to-PSM mapping rules and generating PSMs from PIMs by these rules. PSMs were separated into three distinct tiers: user-interface, business-logic, and data. The user-interface tier defined user interfaces for interaction. The business-logic tier contained domain logics, and aimed to respond to the request from the user-interface tier and invoke changes on the data tier. The data tier was the representation of the information on which the application operates. This study targeted HTML and JavaScript in user-interface tier, JSP and Java in business-logic tier, and Oracle database in data tier. PSMs could be further transformed into real executable program. Transformation from PSM to Code model is trivial. This study focused on CIMs, PIMs, PSMs and the transformation between them, especially the CIM-to-PIM transformation.

This approach obeyed the mechanism of mapping rule and mark developed by OMG for MDA. Mapping rules provide specification for transforming a source pattern into a target pattern. Mark is used to indicate which rule will be applied to a source pattern and bind values in the matched part of the source pattern to the appropriate slots in the generated model. In this usage a target pattern can be thought of as templates for generating a target model, and the use of mark as a way of binding the template parameters. Input to a transformation is a marked source model and mapping rules. Fig. 2 shows a simple example that transforms an activity diagram into a class diagram by rule R1 and R2 in Table 1 in a nested procedure. The source pattern contains the action “Browsing Book List” with a note. Developers first mark the pattern with rule R2, and conduct this transformation. A boundary class corresponding to the note is first generated by rule R2. A dependency is then generated by rule R2; meanwhile rule R2 invokes rule R1 to generate a control class corresponding the action “Browsing Book List”, and the dependency link the control class and the boundary class. These rules were originally graph-based. Because of space limitation this study only shows the explanation part of these mapping rules. Taking the rule R2 in Table 1 as an example, the complete rule is shown in Fig. 3, which was graphically represented and followed by explanation. Mapping rules for generating PIMs and PSMs are described as follows.

3.1. Platform Independent Model

This stage develops CIM-to-PIM mapping rules to refine the initial objects identified in CIMs and constructs PIMs using class diagrams by these rules. Classes in PIMs are classified into entity, boundary and control class. Entity classes represent data stored in a database. Boundary classes represent user interfaces and are triggered by users to communicate with control classes. Control classes capture application logic and act as a bridge between boundary classes and entity classes. This classification corresponds to the three tires of PSMs, making the following PIM-to-PSM transformation easier. This study preliminarily established eleven CIM-to-PIM mapping rules as shown in Table 1.

The mapping rules R1, R2 and R6 define transformation for control class, boundary class and entity class respectively. The mapping rules R3, R4, R5 and R7 define transformation for dependency relationships. The mapping rules R9, R10 and R11 define transformation for association relationships.

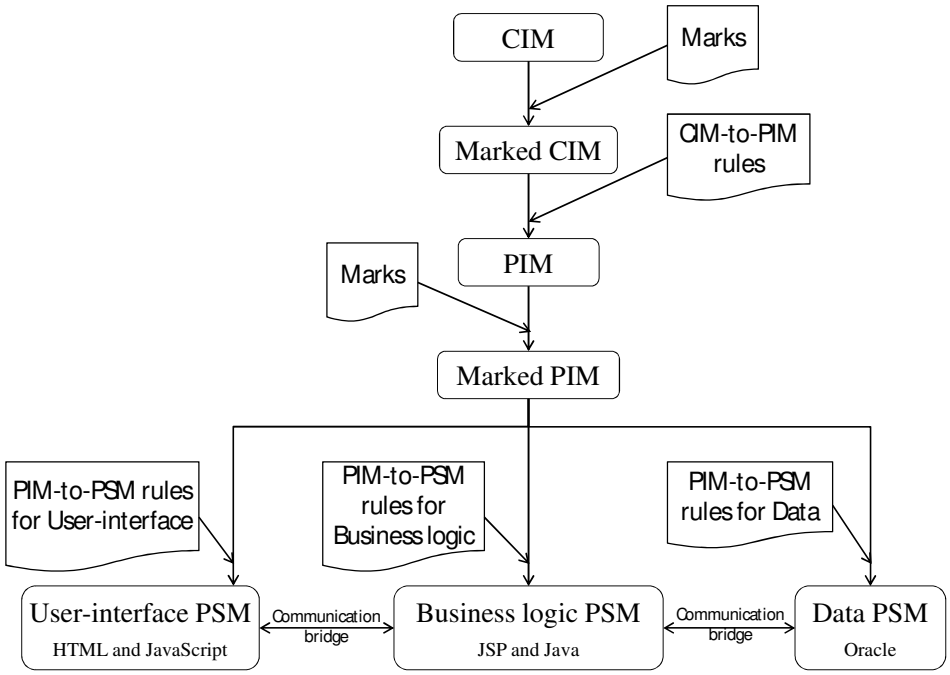


Figure 1. Transformation Framework for 3-tiered Applications.

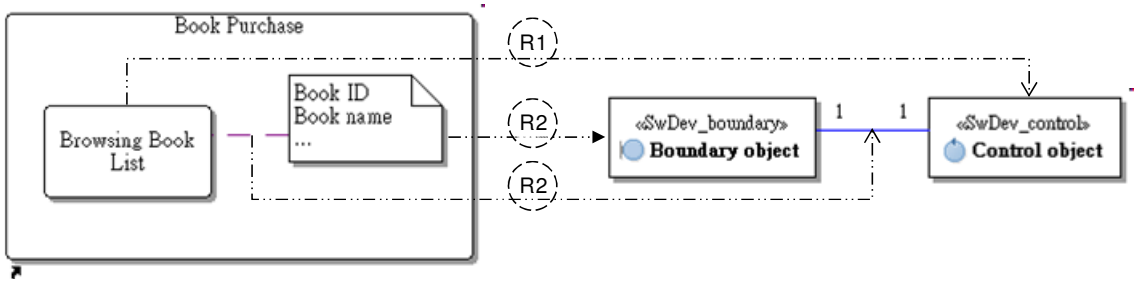
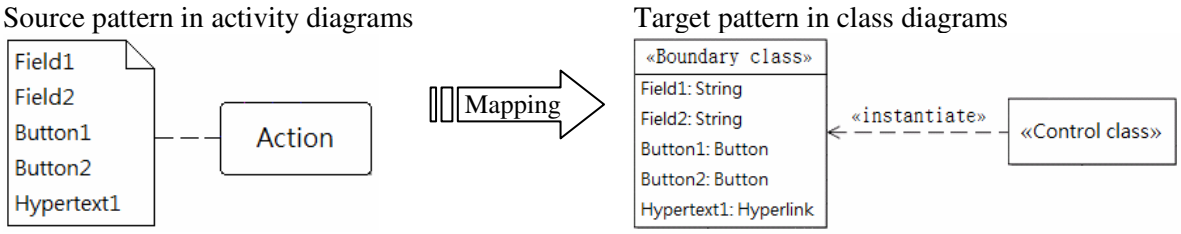


Figure 2. Transformation example.

No.	Mapping rules
R1.	An action in an activity diagram generates a control class in a class diagram.
R2.	When a note attached to an action in an activity diagram is used to present information about user interfaces, the note generates a boundary class in a class diagram. The association between the note and the action generates a dependency by the control class, generated by the action (rule 1), on the boundary class.
R3.	In an activity diagram, when a note of an action is used to present information about user interfaces and some information is used to trigger an action such as a button and hypertext, a dependency by the button (or hypertext) on the control class, generated by the triggered action, is generated.
R4.	When a note of an action in an activity diagram contains persistent information, the note generates a persistent entity class and a dependency by the control class on the persistent entity class. The control class is generated from the action according to rule 1.
R5.	When a note of an action in an activity diagram contains non-persistent information, the note generates a non-persistent entity class and a dependency by the control class on the

	non-persistent entity class. The control class is generated from the action according to rule 1.
R6.	An object node with a note in an activity diagram generates a persistent or non-persistent entity class.
R7.	A control flow from an action to an object node generates a dependency by the control class, generated by the action, on the entity class, generated by the object node.
R8.	When a class is generated several times by different rules, the classes should be merged into one.
R9.	In bi-directional relationships, when an instance of either class is connected to many instances of the other class, a many-many association is generated. When an attribute is a function of both classes, not just of one class, one association class with the attribute is generated.
R10.	In bi-directional relationships, when each instance of the second class is connected to at most one instance in the first class, a many-one association is created.
R11.	In bi-directional relationships of persistent entity classes, when each instance of either class is related to at most one instance of the other class, a one-one association is created.

Table 1. CIM-to-PIM mapping rules



Explanation:

When a note attached to an action in an activity diagram is used to present information about user interfaces, the note generates a boundary class in a class diagram. The association between the note and the action generates a dependency by the control class, generated by the action (rule 1), on the boundary class.

Figure 3. A graph-based transformation example from activity diagrams to class diagrams.

3.2. Platform Specific Model

This stage develops PIM-to-PSM transformations based on the architecture developed by Hennicker & Koch (2001) for 3-tiered applications. The major components in the architecture include (1) components: entity classes, server pages, client pages, forms and framesets; (2) relationships among these components: association, aggregation, composition, generalization and dependency. The section first briefly introduces these components and relationships, and then introduces PIM-to-PSM mapping rules in terms of user-interface, business-logic and data tier.

Entity classes may be divided into two types: permanent and non-permanent. Permanent entity classes often map to database tables that hold the information needed to "outlive" use case execution. Non-permanent entity classes "die" when the use case ends, such as search results. Server pages represent a dynamic web page whose content is assembled on the server as they are requested. Typically, server pages contain scripts that are executed by the server to interact with server resources, such as databases, business logic components, external systems, and so on. Client pages are an HTML-formatted web page with a mix of data, presentation, and even logic. Client pages may contain scripts that are rendered by client browsers. Client pages can interact with other client or server pages through associations. Forms are a collection of standard input elements in a client page. Forms accept input from users and submit it to a server page for processing. Framesets are a container of multiple client pages. A client page can be divided into smaller frames. Framesets support nested structure, and therefore the contents of a frame may be a client page or another frameset.

Association is used to describe the situation that a class refers to an object, or several objects of another class. Aggregation is the typical whole/part relationship, and is considered as a stronger version of the association. Aggregation is used to indicate that a class actually owns but may share objects of other classes. The lifetime of the 'part' isn't controlled by the 'whole'. That is, the 'whole' doesn't take the direct responsibility for creating or destroying the 'part'. Composition is a whole/part relationship too. It's exactly like Aggregation except that the lifetime of the 'part' is controlled by the 'whole'. That is, the 'whole' takes the direct responsibility for creating or destroying the 'part'. Dependency defines the situation that a client class requires the presence of the supplier class for its correct implementation or functioning. The client class can call an operation from the supplier class or instantiate an object of supplier class. Generalization relation is the equivalent of an inheritance relationship in object-oriented terms (an "is-a" relationship). The transformation procedure is described as the following.

3.2.1. User-interface

Components in user-interface tiers could be generated from the boundary classes of PIMs. User-interfaces generally contain many interactive components, such as list and radio-button. Many widgets have been developed to facilitate user-interface design. Designers usually prefer not developing user-interface components in house, but using widgets available on the market. A user-interface function may be implemented using different widgets which have the same functionality. The amount of widgets is big and mapping rules for widgets is intuitional. Therefore, this study did not develop mapping rules for widgets. Four rules in Table 2 were preliminarily developed for components mentioned above: client pages, forms, targets and framesets.

No	Mapping rules
P1.	A frameset is composed of targets. While a frameset is used, a client page is displayed on a target of the frameset.
P2.	Elements in a form in a client page are modeled separately as a «Form» class that has a composition association with the client page and a dependency relationship by the «Form» class on the server page class.
P3.	When a PSM targets the HTML technique, a boundary class in a class diagram (PIM) generates a client page in a class diagram (PSM). An attribute of the boundary class generates an attribute in the client page with the corresponding data type of the HTML technique.
P4.	A dependency relationship in a class diagram (PIM) generates a dependency relationship and a method in a class diagram (PSM). The method responds to the call.

Table 2. PIM-to-PSM mapping rules for the user-interface tier

3.2.2. Business-logic

Most business-logic of systems have been identified in PIMs and recorded in control classes. Therefore, most components of business-logic tiers could be generated from the control classes of PIMs. In addition, some control classes could be generated from non-persistent entity classes, for instance of the rule B6 in Table 3. Entity classes are classified into persistency and non-persistency. Persistent entity classes have data that must be "remembered" when the system is restarted. Non-persistent entity classes do not have persistent data and are usually instantiated as needed during system execution. Further, every attribute in PIMs could generate an attribute in PSMs. Attributes are classified into simple attributes and complex attributes. Simple attributes use primitive data types, such as integer, float, and double, while complex attributes use reference data types, such as object class. A simple attribute in PIMs could be transformed into an attribute in PSMs with the corresponding type of specific technology. A complex attribute in PIMs could be transformed into an attribute referring to the class of the complex attribute. With the above discussion, this study developed six mapping rules as shown in Table 3.

3.2.3. Data

Data PSMs are used to present information which has to be stored in terms of specific database technology. There are two kinds of entity classes as mentioned in Section 3.2.2. Data PSMs focus on mapping rules for persistent entity classes. A persistent entity class could be transformed into a table to keep the data. Furthermore, a simple attribute could be transformed into a column of the table with the corresponding data type of Oracle, while a complex attribute could be transformed into a foreign key referring to the table of the complex attribute. The mapping rules are presented in Table 4.

No	Mapping rules
B1.	A control class in a PIM generates a server page in a PSM class diagram.
B2.	Every attribute of a control class in a PIM could be transformed into an attribute of the corresponding control class of the PSM.
B3.	Every data type of a control class in a PIM could be transformed into the corresponding data type of the control class of the PSM.
B4.	Two operations are generated for each attribute of control classes of the PSM. One is set operation which is used to set attribute value, and the other one is get operation which is used to get attribute value.
B5.	Every operation of a control class in a PIM could be transformed into an operation of the corresponding control class of the PSM.
B6.	A non-persistent entity class in a PIM generates a JavaBean in a PSM class diagram. An attribute of the non-persistent entity class generates an attribute of the JavaBean with the corresponding data type of the specific technique of PSM.
B7.	An association between a control class and an entity class in a PIM could generate a dependency from the server page to the table in a class diagram.

Table 3. PIM-to-PSM mapping rules for the business-logic tier

No	Mapping rules
D1.	When a PSM targets the relational database technique, a persistent entity class in a class diagram (PIM) generates a table in a class diagram (PSM). An attribute of the entity class generates a field in the table with the corresponding data type of the relational database technique.
D2.	A key attribute of a persistent class could be transformed into a primary field of the relation schema based on the method of mapping data type mentioned above.
D3.	On the basis of the method of mapping data type mentioned above, a complex attribute of a persistent class could be transformed into a field of the relation schema which acts as a foreign key maintaining the association relationship with the referenced schema.
D4.	When a PSM targets the relational database technique, an association class in a class diagram (PIM) generates a table in a class diagram (PSM). An attribute of the association class generates a field in the table with the corresponding data type of the relational database technique.
D5.	An association path with an association class (class C) between two persistent entity classes (class A and class B) in a class diagram (PIM) generates three tables and two associations in a class diagram (PSM). Tables A, B and C are generated from the classes A, B and C, respectively. The two associations include a one-to-many association from table A to table C and a one-to-many association from table B to table C.
D6.	An association path without an association class between two persistent entity classes in a class diagram (PIM) generates an association between the two tables with the same multiplicity in a

class diagram (PSM). These two tables are generated from these two persistent entity classes.

Table 4. PIM-to-PSM mapping rules for data tires

4. The illustrative example

This study took an on-line bookshop application as an example to demonstrate the feasibility of this transformation approach. Because of the research limitation, only the purchase-book function of the application was used to explain the process. This study created a use case, an activity diagram and three drawings for the purchase-book function. The activity diagram is shown in Fig. 4. The scenario begins at "browse books." While customers are interested in an item, they can review the details of this item, specify quantity and then put this item into a cart. If they want to buy other items, they can go back to "browse books." Or, they can check the content of the cart and choose one of the following actions: browse again, delete items, reset the quantities, or check out. After confirming the shopping cart, an order and invoice will be created. The ordered items will be sent to the customers with an invoice.

4.1. Platform Independent Model

On the basis of the mapping and mark mechanism of the transformation framework (Fig. 1), some patterns in the activity diagram (Fig. 4) was marked with mapping rules in Table 1. After firing the mapping rules, the PIM (Fig. 5) of the purchase-book function was generated from the marked activity diagram. The used mapping rules are labelled in Fig. 5. For instance of the control classes, "Browse Books", "Display Details, Specify Quantity, Add Item to Cart", and "Display Cart", are labelled with R1, meaning that they were generated according to rule R1. Three boundary classes ("Catalog Page", "Book Details Page", and "Shopping Cart Page") were generated according to the rule R2. The "PrePage" button of the "Catalog" (note) triggers the "Browse Books" (action) in the activity diagram. According to the rule R3, a dependency was created by the "PrePage" button on the "Browse Books" (Control class) in the PIM.

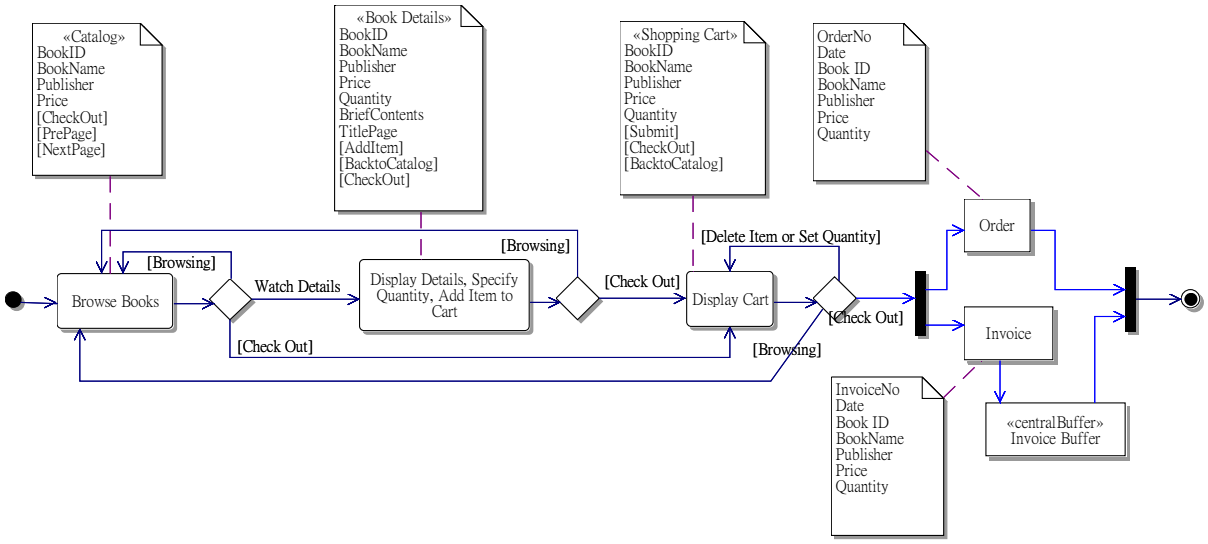


Figure 4. Activity diagram of the purchase-book function

4.2. Platform Specific Model

The PIM generated in Section 4.1 was marked with the mapping rules established in Section 3.2 and then transformed into the PSM (Fig. 6) including user-interface, business-logic and data tier. Some transformation examples are explained as follows.

In the user-interface tier, the boundary classes, “Catalog Page”, “Book Details Page” and “Shopping Cart Page”, in the PIM were transformed into three client pages having the same name with the corresponding boundary classes according to the rule P3. The “Book Details Page” and “Shopping Cart Page” in the PIM receive input from users and submit it to a server page for processing. According to the rule P2, the “Book Details Form” and “Shopping Cart Form” were created; two composition relationships were created: one between “Book Details Page” and “Book Details Form” and the other one between “Shopping Cart Page” and “Shopping Cart Form;” two dependency relationships were created: one from “Book Details Form” to “Display Cart” and the other one from “Shopping Cart Form” to “Display Cart.” According to the rule P1, a one-to-many composition relationship between “Target” and the “Frameset” was created. And, the “Target” was associated with the three client pages, “Catalog Page”, “Book Details Page” and “Shopping Cart Page.”

In the business-logic tier, three control classes, “Browse Books”, “Display Details, Specify Quantity, Add Item to Cart”, and “Display Cart” in the PIM were transformed into three server pages having the same name with the corresponding control classes according to the rule B1. According the rule B7, two dependencies were created: one from “Browse Books” to “Book” and the other one from “Display Details, Specify Quantity, Add Item to Cart” to “Book.” According to the rule B6, the non-persistent entity class, “Cart Items”, was transformed into a Java Bean, “Cart Items”, in the PSM.

In the data tier, three persistent entity classes, “Book”, “Order” and “Invoice”, in the PIM were transformed into three tables with the same name according to the rule D1. Two association classes, “Invoice Item” and “Order Items”, in the PIM were transformed into two tables with the same name according to the rule D4. The many-to-many association between the “Book” and the “Order” was transformed into two one-to-many associations: one between the “Book” and the “Order Items” and the other one between the “Order Items” and “Order” according to the rule D5.

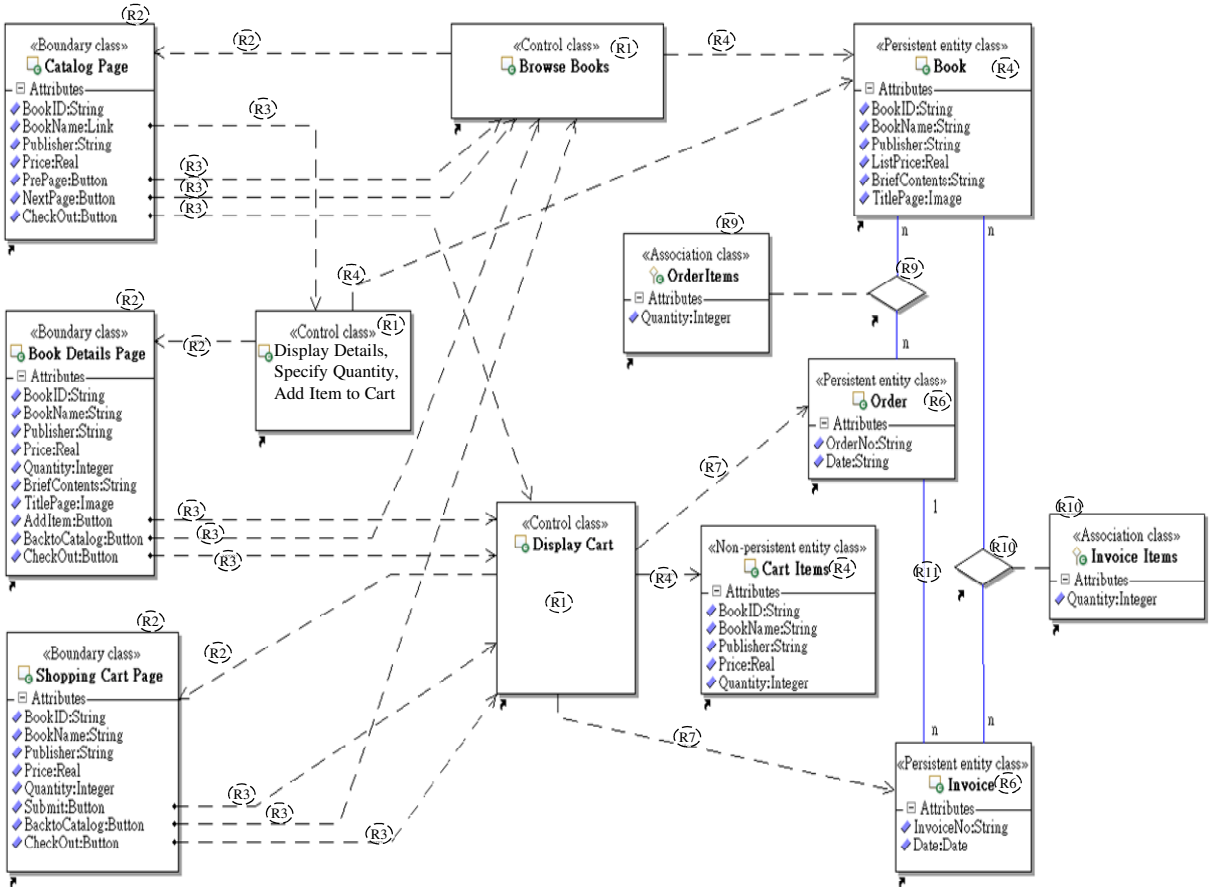


Figure 5. PIM of the purchase-book

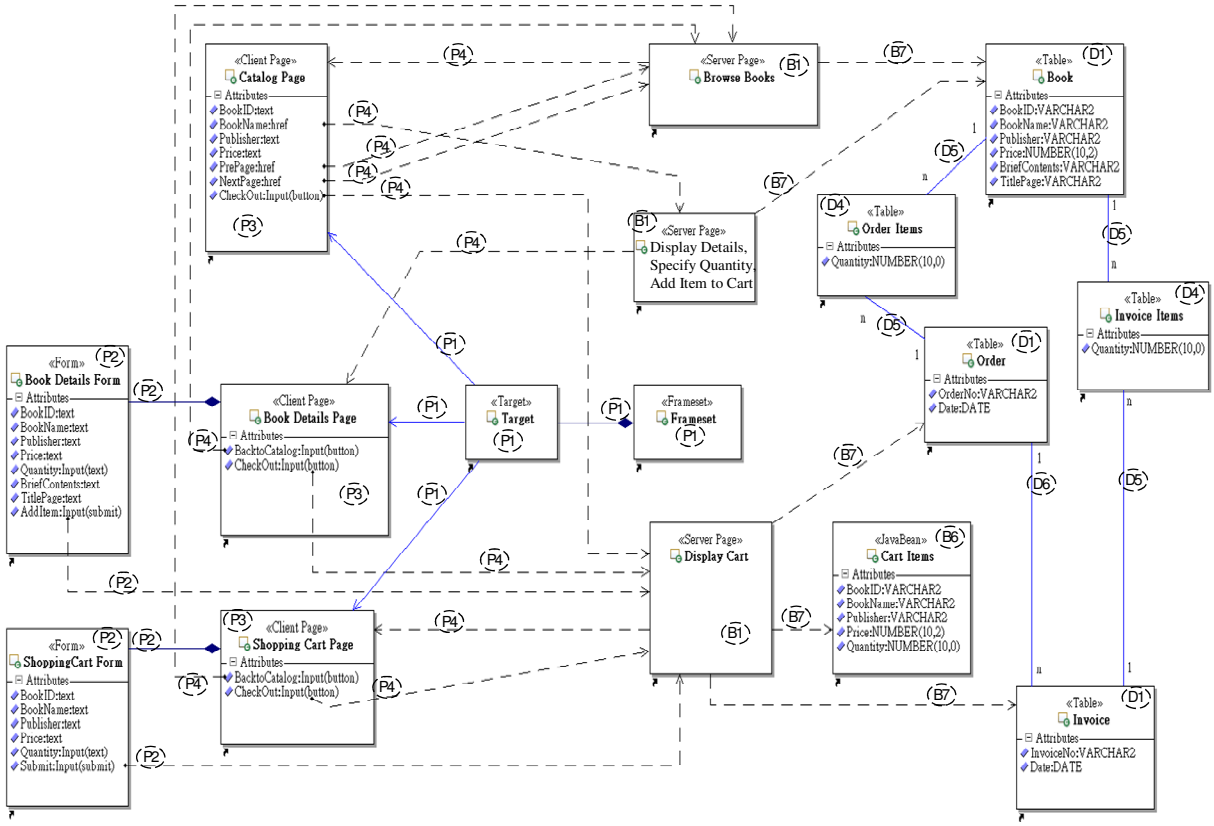


Figure 6. PSM of the purchase-book

5. Conclusions and discussion

The study of MDA focuses almost exclusively on PIM-to-PSM transformations. This study developed a graph-based transformation approach for 3-tiered applications. This approach defined modelling languages used for representing CIMs, PIMs and PSMs, and CIM-to-PIM and PIM-to-PSM mapping rules. An on-line bookshop was built to demonstrate the feasibility of the approach. This approach could be more systematically conduct CIM-to-PIM transformations, and therefore alleviates the gap between CIM and PIM. Besides, this study provides a whole picture for generating 3-tiered applications from CIM to PIM and then to PSM. The PSM targeted HTML and JavaScript, JSP and Java, and Oracle database in user-interface, business-logic and data tier respectively.

The contribution of this study is two-fold. First, the result of this study provided an insight for understanding and reducing the transformation gap between CIM and PIM. Second, this study provides a graph-based transformation mechanism for generating 3-tiered applications from CIM to PIM and then to PSM. With this approach, 3-tiered applications can be more easily and systematically analysed, designed, and generated. This work is the beginning of a line of MDA transformation approach. Future research directions are abundant. Currently, developers have to manually choose appropriate mapping rules and conduct transformations. Tools with the capability of automatically identifying potential mapping rules should be developed. Then, MDA transformations would be easier and more efficient.

Acknowledgements

This research was supported by the National Science Council of Taiwan under grant #NSC 99-2221-E-143-003

References

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

Bergmann, G., Boronat, A., Heckel, R., Torrini, P., Ráth, I. and Varró, D. (2011). Advances in Model Transformations by Graph Transformation: Specification, Execution and Analysis. Lecture Notes in Computer Science, 6582, 561-584.

Bézivin, J., Hammoudi, S., Lopes, D. and Jouault, F. (2004). Applying MDA approach for web service platform. In Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference – EDOC 2004, Monterey, California, USA.

Brooks, T.A. (1995). People, words and perceptions: A phenomenological investigation of textuality. Journal of the American Society for Information Science, 46 (2), 103-115.

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

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

Hailpern, B. and Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. IBM Systems Journal, 45 (3), 451-461.

Hennicker, R. and Koch, N. (2001). Modeling the user interface of web applications with UML. Workshop of the pUML-Group held together with the «UML»2001 on Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists (GI), Toronto, Canada, pp. 158-172

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

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

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