Appl. Math. Inf. Sci. 6 No. 2S pp. 479S-488S (2012)

Applied Mathematics & Information Sciences
An International Journal

An Architecture-Centered Method for Rapid Software Development

Chaoyu Lin[1] , Jyhjong Lin[2] and Weipang Yang[1]
[1] Department of Information Management, National Dong Hwa University, Hualien, Taiwan
[2] Department of Information Management, Ming Chuan University, Taoyuan, Taiwan
Corresponding author: Email: d9732010@ems.ndhu.edu.tw; jlin@mail.mcu.edu.tw; wpyang@mail.ndhu.edu.tw

Received Dec. 22, 2010; Revised Feb. 21, 2011; Accepted Apr. 23, 2011
Published online: 1 June 2012

Abstract: An architecture-centered method for rapid software development is presented in this paper. It basically follows the guidelines suggested by extreme programming that require highly expressive programming languages (i.e., Java) and CASE tools. As in extreme programming, this method addresses on rapid software development for small- or medium-sized projects. Further, for effective guidance on the development, it directs the construction of system components by imposing an architecture-based concept of layered specification and construction of these components through its activities. Since the method follows the guidelines suggested by extreme programming and supports effective guidance by a layered development of architectural components, team productivities can be greatly enhanced by less (but effective) overhead on specification work. The method uses UML and Petri nets as its modeling tool; for illustration, an example application is presented that specifies and directs the development of a software system with business-oriented Internet services.

Keywords: Software System, Development Method, Architecture-Centered, UML, Petri Nets.

1 Introduction

Properly identifying required activities and directing the completion of these activities for constructing relevant artifacts/deliverables are key issues for the successful development of a software system. For these needs, it has become a great deal of concerns for a software project team to impose a suitable development method because such a method can help to figure out required activities /artifacts and more importantly can provide information to assist on directing the completion /construction of these activities/artifacts. Although there are already plenty of software development methods in the literature and also as a common recognition there are no methods that are prefect to employ all sound features for effective development specification and guidance, a desired method that is suitable for a software development project can still be expected with the following features:(1) It can define the structural and dynamic aspects of the development work; (2) It is featured by proper mechanisms to support effective guidance on the development work; (3) The defined activities can be concerned in a leveled manner such that team members can participate in the execution by providing/monitoring respective information about the specification and completion of concerned activities/artifacts; and (4) It can support rapid development for small- or medium-sized projects by less but effective overhead on completing/constructing defined activities/ artifacts; automatic tools should also be generated to facilitate its practical applications.

As mentioned earlier, there are already plenty of software development methods in the literature. Among them, for instances, Waterfall model, Evolutionary Development, and Component-based Software Engineering are three traditional generic models [1-8] that are not mutually exclusive but often used together for development of large systems; Rational Unified Process [9,10] and Spiral

model [11,12] give an illustration that employs elements of these three models. Some formal approaches like B method and Cleanroom model can be found in [13-16] that emphasize on the mathematical specification for the software system and its mapping directly into an implementation. With respect to these design-oriented (more analysis and design work) models, many alternatives that criticize the overhead on design have been proposed (e.g., less or even no analysis and design work) such as Agile Development [17-19], Rapid Application Development (RAD) [20] and Extreme Programming (XP) [21,22] where code work (i.e., implementation, testing, and refactoring) are focused. Whereas such design- or code-oriented models try to get an extreme in very different spectrums, some compromised considerations can be found in [23,24] that employ simplified design and code work through rapid engineering ways where specific platforms and CASE tools are imposed on their development processes.

In general, these existing approaches provide sound mechanisms for development specification and guidance; some drawbacks with respect to the above desired features can still be found among them: (1) For the two kinds (design- or code-oriented) of models, each one gets an extreme in its concerning spectrums, so the advantages for one kind would become the opposites for the other; (2) For the comprised approach that takes advantage of the two extreme kinds, its less overhead on design work and associated rapid engineering way make it good for the rapid development for small- or medium- sized projects. However, such less design work on the other hand makes it lack sufficient mechanisms for supporting effective guidance on the development work; and (3) Among these existing approaches, they essentially focus on the specification and completion of defined activities /artifacts; in contrast, few considerations about the management of these tasks can be found in their statements. In our knowledge, however, such management issues should not be negligible since they play a critical role for the success of these tasks.

To address these deficiencies, we present in this paper an architecture-centered method that supports rapid software development for small- or medium-sized projects. This method in general follows the guidelines suggested by XP that require highly expressive programming languages (i.e., Java) and CASE tools. As in XP, this method asks less design work for speeding software development. However, for providing guidance on the development work, it focuses on the construction of system components by imposing an architecture-based concept of layered specification and construction of these components on its activities; this makes it easy to direct the development work by concerning the executions of these activities about their effects on these components for realizing desired user requirements. Since the method follows the guidelines in XP and supports effective guidance by a layered development of architectural components, team productivities can be greatly enhanced by less (but effective) overhead on specification work. Finally, the concept of developing architectural components in a layered manner assists also in the management issues since various activities can be concerned in a corresponding layered manner such that responsible team members can participate in the project by providing information or monitoring status about the specification and completion of these activities (e.g., determining and analyzing the status of these activities). For practical applications, the method uses UML [25,26] and extends Petri nets (PN) [27-29] as its modeling mechanisms, and, a prototype CASE tool is constructed to support the layered development of system components and the guidance and monitoring of the development work.

This paper is organized as follows. Section 2 presents the method and its corresponding models in UML and extended PN. For illustration, an example application is presented in Section 3 that specifies and directs the development of a Web software system with business-oriented services. Finally, Section 4 has the conclusions.

2 The Development Method

As shown in Figure 1, the method is architecture -based with six steps:

1. **use case identification,** described in an UML use case diagram, that clarifies user requirements to be satisfied in the system;
2. **conceptual components identification,** described in an UML robustness diagram, that identifies conceptual components with each one playing a specific role in the realization of a desired use case;
3. **components development order determination,** described in an extended PN diagram, that determines the specification and construction order of conceptual components through an iterative process of steps 4 – 6

according to a designated order of the three (Model, View, and Control) layers in the MVC architecture.

4. **architectural components specification,** described in two UML (class and sequence) diagrams, that imposes such architectural considerations as the MVC architecture and relevant supportive design patterns on conceptual components such that formal architectural components can be derived to effectively support the realization of the desired use case;
5. **system components design,** described in two (UML class and sequence) diagrams, that employs platform specific features into architectural components such that each resultant system component has a specific implementation code on the chosen platform and hence its construction can be easily achieved by a direct transformation from its design work;
6. **system components construction,** described in Java code, that implements and tests platform specific components for realizing the desired use case;

Note that steps 4 to 6 are an iterative process that proceeds the development of system components (via conceptual and architectural versions) under a prescribed order of MVC architectural layers and such a layered development process is particularly specified in an extended PN diagram that takes advantage of its formal semantics to support the guidance and monitoring of the development work.

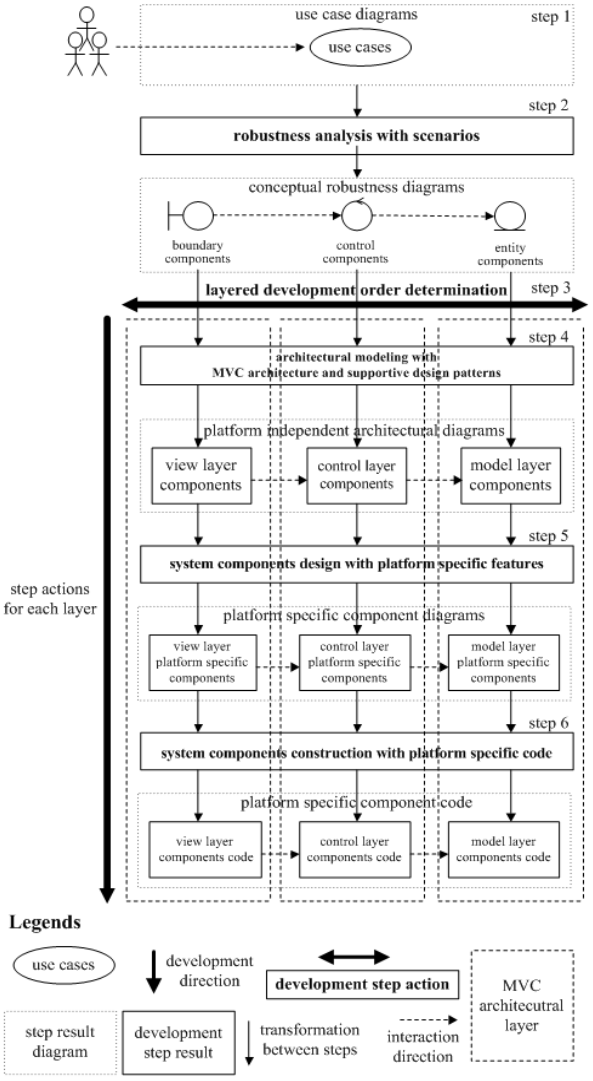


Figure 1: the architecture-centered development method

2.1 The use case identification

The first step is to identify user requirements of the system. As in many existing approaches, all desired user requirements can be identified by various techniques like interviewing, questionnaire, observation, etc. and the identified requirements are represented by means of use cases in a UML use case diagram. Note that the reader is referred to [25] for more detail about use cases and their representation in UML.

After identifying desired use cases, it is time to perform development work for constructing them in an incremental manner; that is, based on available resources and functional/non-functional considerations, these use cases can be constructed in an incremental plan where they are prescribed into a set of increments for design and construction into a controlled series of deliverable releases. The following describes the remaining steps for design and construction of each use case or increment.

2.2 The conceptual components identification

With a use case desired, its design work could begin by identifying an architectural partitioning of conceptual components that each plays a specific role in achieving this use case. In our best knowledge, clarifying conceptual components and their participant roles is a most suitable incentive to ensure the achievement of a desired use case [30,31]. For this purpose, robustness analysis with scenarios by [24] is used that first identifies possible scenarios to describe those behaviors expected in the use case. For description, each scenario is presented as a flow of events (or an equivalent UML activity diagram) where each event (or activity) represents a behavior needed to realize the use case and hence is composed of three parts expressed from left to right: an actor, an action that the actor takes, and entities on which the

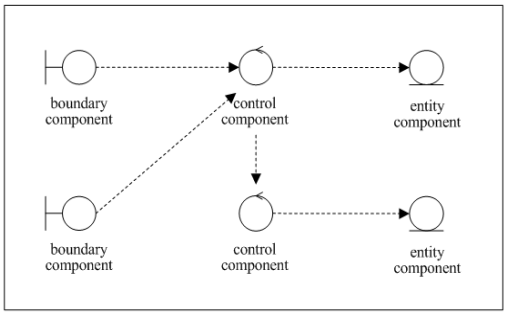


Figure 2: robustness diagram with three types of conceptual components

action is taken. Then, such a flow of events is traced in order to identify three (boundary, control, and entity) types of conceptual components that each plays a specific role in realizing the use case; for description, these components are described in an UML robustness diagram as shown in Figure 2 (The reader is referred to [24] for detail about the transformation from use cases into robustness diagrams).

2.3 The components development order determination

With conceptual components that each plays a specific role in the realization of a desired use case, it is good time to determine the specification and construction order of these conceptual components in a layered manner. This is because under a layered concept these conceptual components are categorized into three (boundary, control, and entity) types which match exactly those three respective layers in the most commonly recognized MVC architecture [32-35], their specification and construction can thus be proceed in such a layered manner. That is, conceptual components can be specified/ constructed through an iterative series of transformations into Java code according to some designated order of the three MVC (Model, View, and Control) layers. For example, components may be developed under such an order as V -> M -> C or M -> C -> V; in fact these orders can be determined by referencing the characteristics of the use cases being realized where each MVC layer addresses certain aspects of these characteristics (e.g., for interaction- or process- or data-oriented use cases, various development orders may be applied due to their different focuses).

The remaining steps 2.4 – 2.6 are therefore an iterative process for components in each MVC layer according to the designated order, and an extended PN diagram [36] is used to specify the process that supports particularly two advantages: (1) its architecture-centered design and layered development supports effective guidance on the development work (e.g., the design/code activities can be proceeded and directed under a designated order of their effects on system components toward the realization of desired use cases); (2) its layered development helps also the management of the development work by facilitating the participation of team members in a corresponding layered manner where these members may provide information about the executions of the design/code activities or their accessed components (e.g., the execution status of these activities or the versions of their accessed components) and then to monitor them by analyzing the PN process diagram based on its formal semantics (e.g., traversing the reachability graph derived from its behavioral semantics).

2.4 The architectural components specification

With conceptual components and their participant roles in achieving a desired use case, the next is to impose such architectural considerations as architectural frameworks and design patterns on these conceptual components to derive formal architectural ones that support the realization of the use case. In the context of architectural frameworks, many well-known approaches can be found such as MVC [32-35], MFC [37,38], and PCMEF [39]. Among them, nonetheless, MVC (Model, View, Control) is most commonly chosen for development of small- or medium-sized projects due to its well-recognized role-based concept, a classic use of separation of concerns in object-oriented design, and a most number of supportive design patterns [40]. Hence, MVC is applied herein in a layered manner such that components are allocated respectively at its three layers and developed layer by layer (e.g., View -> Control -> Model) according to some designated order. Note that with the chosen MVC architectural framework, supportive design patterns [33] can be used at each layer to enhance the effectiveness of these components on realizing the use case.

While architectural components are identified, two UML diagrams are used to specify these components and their behaviors: (1) a class diagram that describes these components and their dependent relationships; and (2) a sequence diagram that describes how these components interact with each other to achieve the use case. The following provide rules for the transformation from robustness diagrams into class and sequence ones:

(1) For each robustness diagram and its originating scenarios, two initial class and sequence diagrams

in the context of the MVC architecture can be derived by the following mapping rules:

rule	robustness diagram	class/sequence diagram in MVC
1	a boundary component in robustness diagram	a view component in class diagram
		a view component in sequence diagram
2	a control component in robustness diagram	a control component in class diagram
		a control component in sequence diagram
3	an entity component in robustness diagram	a model component in class diagram
		a model component in sequence diagram
4	an interaction from a boundary component to a control component in robustness diagram	a dependency from a view component to a control component in class diagram
		a call from a view component to a control component in sequence diagram
5	an interaction from a control component to an entity component in robustness diagram	a dependency from a control component to a model component in class diagram
		a call from a control component to a model component in sequence diagram
6	an interaction from an entity component to a control component in robustness diagram	a dependency from a model component to a control component in class diagram
		a return from a model component to a control component in sequence diagram
7	an interaction from a control component to a boundary component in robustness diagram	a dependency from a control component to a view component in class diagram
		a return from a control component to a view component in sequence diagram
rule	Scenarios	class/sequence diagram in MVC
8	an actor in scenarios	a view or control or model component in class diagram
9	an entity on which an action is taken in scenarios	a component or an attribute of a component or a parameter of an operation of a component at view or control or model layer in class diagram
10	an action that an actor takes on an entity in	an operation of the component derived from the actor in class diagram

(2) For these two class and sequence diagrams, consider supportive design patterns [33] and their applicability to determine if there are suitable design patterns to be imposed on either of the three MVC layers such that in collaboration with those architectural components in the class/sequence diagrams, the effectiveness of these architectural components on the realization of the use case can be enhanced. For instance, for flexibility in designing operational calls from view to control components or from control to model ones, a CMD (Command) design pattern may be imposed on the View and Control layers in MVC, while for flexibility in designing data access mechanisms, a DAO (Data Access Object) design pattern may be used in the Model layer.

2.5 The system components design

With architectural components that have platform independent features, the next step is to employ platform specific features into these components such that each resultant system component has a specific implementation code on the chosen platform and hence its construction can be easily achieved by a direct transformation from its design work. Since there are many existing platforms available for implementation and execution, for illustration Java JEE technologies [41] are used illustratively into architectural components to create JEE specific system components that support the realization of the use case (for illustration, Web services are provided). In Web software systems, functions are presented and accessed in Web pages where (1) (input/output/action types of) AWT or Swing widgets are allocated in containers (e.g., forms or tables) to provide interfaces between users and systems; (2) JSP or Servlet or EJB is used in system sites to provide services for users; and (3) Java Bean or EJB is used in system sites to support access and storage of databases. In our knowledge, these types of JEE components support well the implementation and execution of Web software systems and thus the following provide rules for the transformation from architectural components into JEE specific system ones:

rule	architectural components	JEE specific system components
1	a view component with a dependency from users in component diagram	input and action types of widgets in class diagram
		a Servlet or EJB component in component diagram
2	a view component with a dependency to users in component diagram	output type of widgets in class diagram
		a JSP or Servlet or EJB component in class diagram
3	a control component in component diagram	a Servlet or EJB component in class diagram
4	a model component in component diagram	a Java Bean or EJB component in class diagram
5	a call from a view component to a control component in sequence diagram	a call from an action widget to (an operation of) a Servlet or EJB component in sequence diagram
6	a call from a control component to a model component in sequence diagram	a call from a JSP or Servlet or EJB component to (an operation of) a Java Bean or EJB component in sequence diagram
7	a return from a model component to a control component in sequence diagram	a return from (an operation of) a Java Bean or EJB component to a JSP or Servlet or EJB component with an attribute for returned value
8	a return from a control component to a view component in sequence diagram	a return from (an operation of) a JSP or Servlet or EJB component to an output widget with an attribute for the returned value
9	an attribute of a component at view or control or model layer in component diagram	an attribute of the widget or JSP or Servlet or Java Bean or EJB component derived from the architectural component
10	a parameter of an operation of a component at view or control or model layer in component diagram	a parameter of an operation of the widget or JSP or Servlet or Java Bean or EJB component derived from the architectural component

2.6 The system components construction

With system components that have platform specific features, the next step is to implement these components in Java code that collaboratively provide the desired Web services. This is a trivial work because each system component has become a JEE one and its implementation can be easily achieved by constructing its code structure and executable structural/behavioral statements for providing the desired Web services.

3 The Illustration for Book Publishing

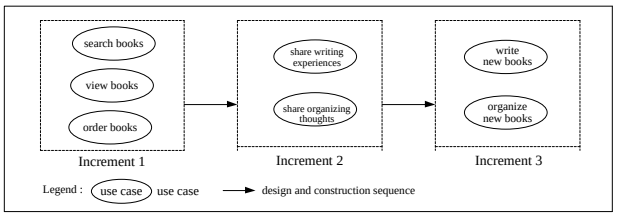


Figure 3: use cases and increments for book publishing

3.1 The use case identification for book publishing

As in many book publishing companies that offer various services such as (1) searching/viewing /ordering of books; (2) sharing of writing experiences or organizing thoughts; and (3) writing or organizing of new books, seven use cases can therefore be identified and prescribed into three increments for further design and construction as shown in Figure 3.

3.2 The conceptual components identification for book publishing

For each use case, its design work begins by identifying conceptual components that each plays a specific role in is achievement. For this purpose, robustness analysis with scenarios is used to identify three (boundary, control, and entity) types of components that each plays a specific role in realizing the use case. For our example, Figure 4 is a robustness diagram with conceptual components and relevant dependent relationships transformed from the 'share writing experiences' use case.

3.3 determine components development order for book publishing

With conceptual components identified, their development order is then determined in terms of some designated order of the three MVC layers. For the example, since the 'share writing experiences' use case focuses on the interactions among customers for sharing experiences where interaction and data models play critical roles for its realization, conceptual components in Figure 4 can thus be specified and constructed through an iterative process with the order View->Model->Control to address such characteristics. The steps 3.4 – 3.6 are therefore iterative for components in each MVC layer according to this order and an extended PN diagram in Figure 5 is used to specify this process where its analysis can be achieved by traversing its reachability graph as in Figure 6.

3.4 The architectural components specification for book publishing

The next step is to impose the MVC architectural framework and relevant supportive design patterns on conceptual components to derive formal architectural components that effectively support the realization of the desired use case. For the example, Figure 7 has class and sequence diagrams mapped from the robustness diagram in Figure 4 for realizing the 'share writing experiences' use case. In particular, while designing architectural components at each MVC layer, (1) for providing flexible operational calls, a CMD (Command) design pattern is imposed on the Control layer; and (2) for providing flexible data access mechanisms, a DAO (Data Access Object) design pattern is used in the Model layer.

3.5 The system components design for book publishing

With architectural components that have platform independent features, the next is to employ platform specific features into these components. For the example, Figure 8 has class and sequence diagrams transformed from Figure 7 for realizing the Scenario #1 (view experiences shared from other users) of the 'share writing experiences' use case. In particular, (1) Swing widgets (i.e., Text Field and Button) are allocated

Scenario #1 (view experiences shared from other users)
1) user inputs via *displaying HCI* specific topics about shared experiences
2) system receives the topics
3) system retrieves the shared experiences under the topics
4) system returns the shared experiences under the topics
5) user views via *displaying HCI* the shared experiences

Scenario #2 (respond comments about shared experiences)
1) user inserts via respond*ing HCI* comments about shared experiences under specific topics
2) system receives the comments
3) system saves the comments under specific topics
4) system returns a confirmation
5) user views via *responding HCI* the confirmation

Scenario #3 (insert experiences to be shared with other users)
1) user inserts via *insertion HCI* experiences under specific topics
2) system receives the experiences
3) system saves the experiences under specific topics
4) system returns a confirmation
5) user views via *insertion HCI* the confirmation

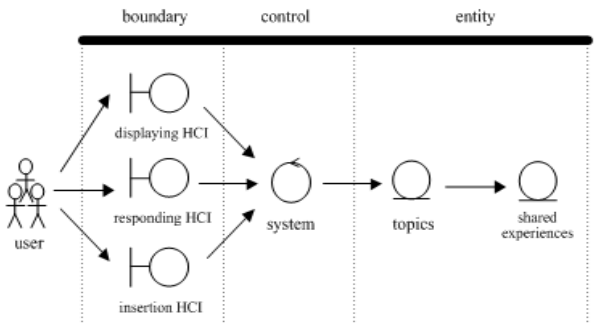


Figure 4: robustness diagram from robustness analysis with scenarios for realizing the 'share writing experiences' use case

in Forms to provide interfaces for users to insert topics about experiences; (2) Servlet and JSP (with Table/Text Field to hold and display shared experiences) is used at the Control layer to provide services for users; (3) Java Bean is used at the Model layer to support retrieval of shared experiences; (4) EJB is used at the model layer to support storage of shared experiences under specific topics and (5) these system components interact with each other via operational calls to provide the desired "view experiences shared from other users " of the 'share writing experiences' use case.

3.6 The system components construction for book publishing

With system components that have JEE features for implementation of Web services, the next step is to implement and test these components in Java code that collaboratively provide the desired Web services. This is a trivial work by constructing its code structure and executable structural and behavioral statements for providing these services.

4 Conclusions

In this paper, we present an architecture-centered method that supports rapid software development for small- or medium-sized projects. This method follows the guidelines in XP for modeling design diagrams and mapping into executable Java code. For providing development guidance, it employs a layered specification/ construction of system components based on the well recognized MVC architectural framework. To illustrate, it is applied to the development of a Web software system with services for book publishing. More specifically, the 'share writing experiences' use case is illustrated by first identifying its conceptual components and then specifying /constructing them through a View -> Model -> Control order.

With less overhead on specifying and constructing system components and effective guidance on the management of these activities, the method can support a rapid development for small- or medium- sized projects. Currently, a prototype CASE tool accompanied with the method is being constructed that will support the specification and construction of conceptual components at the three MVC layers respectively, via steps 3.4 – 3.6 with a series of transformations from conceptual specification to architectural specification to system design to code.

Finally, since the method supports the management of the development work by facilitating team members to provide information about the executions of the design/code works or their accessed components and then to monitor them by analyzing the extended PN diagram based on its formal semantics, we will therefore explore further its effects on the management of the development work where the provision and/or monitoring of information about the executions of the design/code works or their accessed components may be realized by certain useful Java documentation tools such as Annotations and Javadoc.

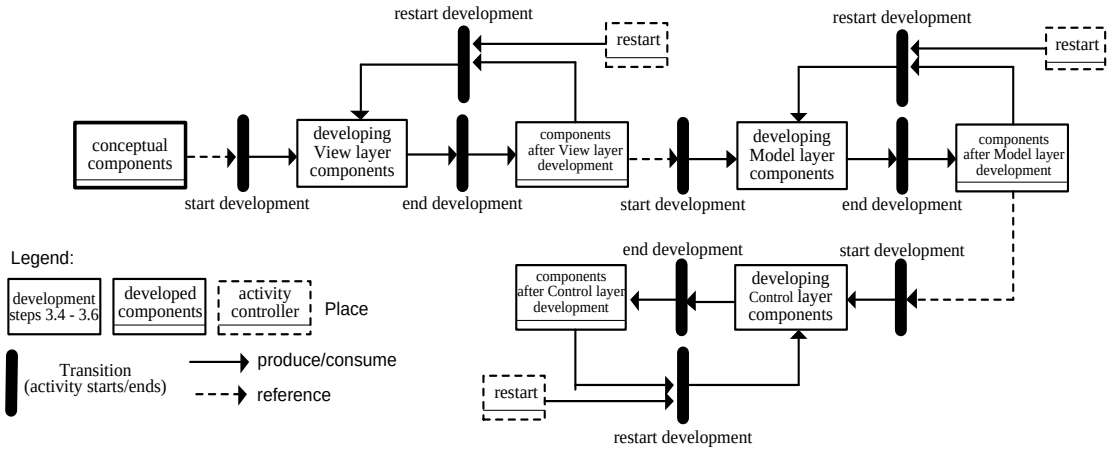
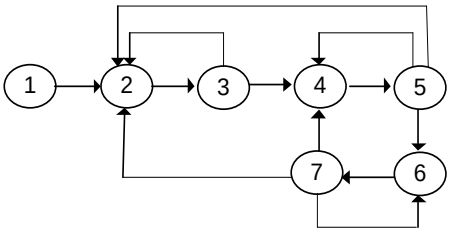


Figure 5: development process of layered components for use case - share writing experiences



Node 1 : before the development process
Node 2 : developing components at View layer
Node 3 : components at View layer identified
Node 4 : developing components at Model layer
Node 5 : components at Model layer identified
Node 6 : developing components at Control layer
Node 7 : components at Control layer identified

Figure 6: state reachability graph derived from the iterative development process in Figure 5

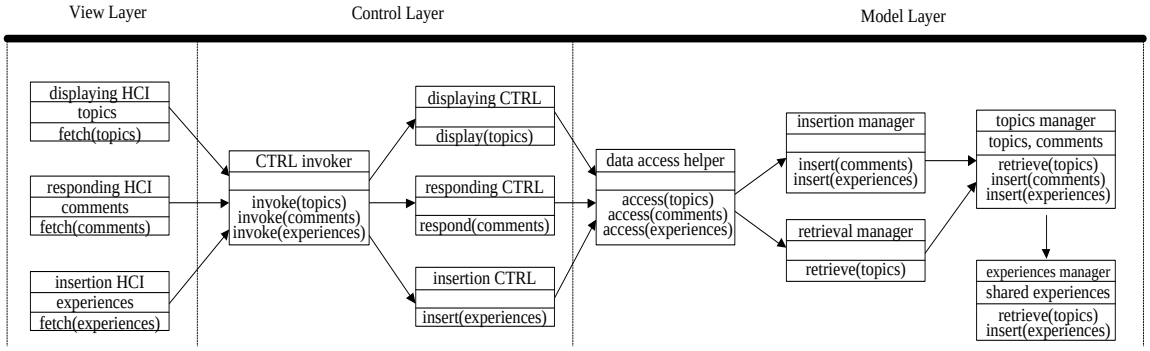


Figure 7.a: architectural class diagram from robustness diagram for realizing the 'share writing experiences' use

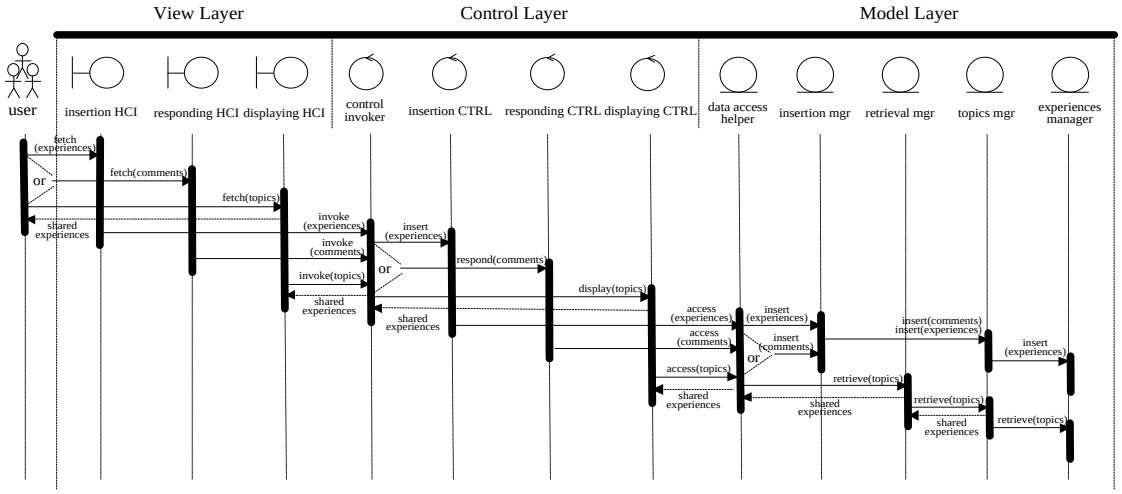


Figure 7.b: architectural sequence diagram from robustness diagram for realizing the 'share writing experiences'

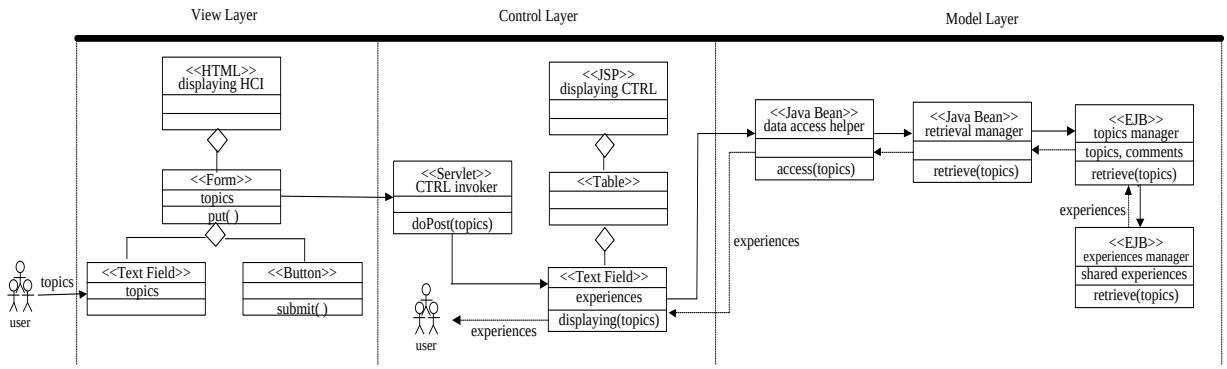


Figure 8.a: system class diagram from architectural one for viewing experiences shared from other users of the

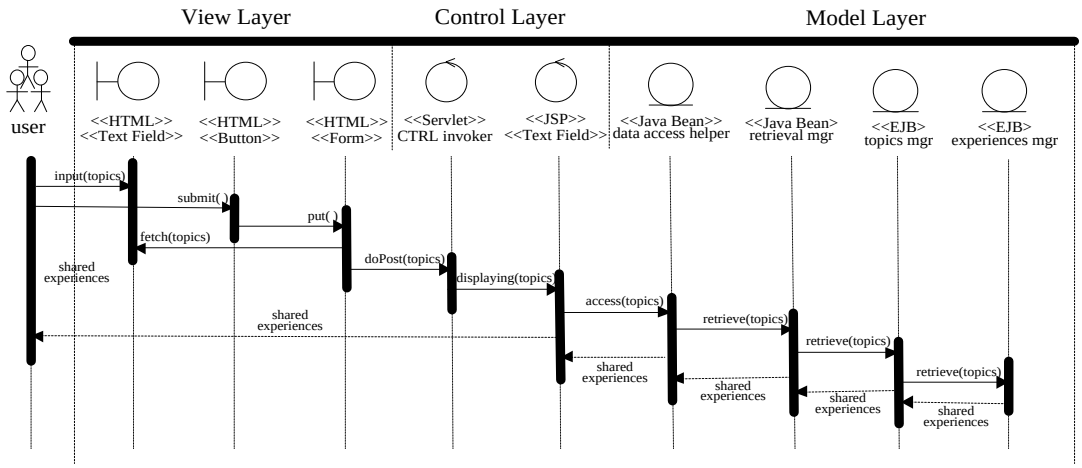


Figure 8.b: system sequence diagram for viewing experiences shared from other users of the 'share writing

References

[1] W. Royce, "Managing The Development of Large Software Systems: Concepts and Techniques," *Proc. of the 9th international conference on Software Engineering*, (1970), 328-338.

[2] D. Graham, Incremental Development and Delivery for Large Software Systems, *Proc. of IEE Colloquium on Software Prototyping and Evolutionary Development*, (1992), 2/1-2/9.

[3] J. Crinnion, The Evolutionary Development of Business Systems, *Proc. of IEE Colloquium on Software Prototyping and Evolutionary Development*, (1992), 3/1-3/11.

[4] J. Ning, Component-based Software Engineering, *Proc. of 5th IEEE International Symposium on Assessment of Software Tools and Technologies*, (1997), 34-43.

[5] J. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House (2000).

[6] X. Cai, et al., Component-based Software Engineering: Technologies, Development Frameworks, and QA Schemes, *Proc. of IEEE APSEC*, (2000), 372-379.

[7] A. Kleppe, et al., *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison Wesley (2003).

[8] X. Zhang, et al., The Research of The Component-based Software Engineering, *Proc. of 6th IEEE International Conference on Information Technology*, (2009), 1590-1591.

[9] P. Kruchten, *The Rational Unified Process*, Addison Wesley (1999).

[10] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison Wesley (2000).

[11] B. Boehm, A Spiral Model of Software Development and Enhancement, *IEEE Computer*, Vol. 21, No. 5, (1988), 61-72.

[12] C. Viravna, Lessons Learned from Applying The Spiral Model in The Software Requirements Analysis Phase, *Proc. of 3th IEEE International Symposium on Requirements Engineering*, (1997), 40.

[13] H. Mills, et al., Cleanroom Software Engineering, *IEEE Software*, Vol. 4, No. 5, (1987), 19-25.

[14] R. Linger, Cleanroom Process Model, *IEEE Software*, Vol. 11, No. 2, (1994), 50-58.

[15] J. Wordsworth, *Software Engineering with B*, Addison Wesley (1996).

[16] S. Prowell, et al., *Cleanroom Software Engineering: Technology and Process*, Addison Wesley (1999).

[17] P. Abrahamsson, et al., *Agile Software Development Methods*, ESPOO VTT Publications (2002).

[18] Agile, http://www.agilealliance.org/, accessed on Jan. 2011 (2011).

[19] R. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall (2003).

[20] J. Stapleton, *Dynamic Systems Development Method*, Addison Wesley (1997).

[21] L. Lindstrom and R. Jeffries, Extreme Programming and Agile Software Development Methodologies, *Information Systems Management*, Vol. 21, No. 3, (2004), 41-52.

[22] Extreme Programming: A Gentle Introduction, http://www.extremeprogramming.org/, accessed on January 2011 (2011).

[23] G. Armano and M. Marchesi, A rapid development process with UML, *ACM SIGAPP Applied Computing Review*, Vol. 8, No. 1, (2000), 4-11.

[24] J. Wu, An Extended MDA Method for User Interface Modeling and Transformation, *Proc. of 17th European Conference on Information Systems*, (2009), 1632-1642.

[25] J. Rumbaugh, et al., *The UML Reference Manual*, Addison Wesley (2004).

[26] G. Booch, et al., *The UML User Guide*, 2nd Edition, Addison Wesley (2005).
[27] J. Peterson, Petri Nets, *ACM Computer Surveys*, Vol. 9, No. 3, (1977), 223-252.
[28] J. Peterson, *Petri Net Theory and The Modeling of Systems*, Prentice Hall (1981).
[29] E. Yiannis, et al., Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets with Transition Enabling Function," *IEEE Transaction on Software Engineering*, Vol. 18, No. 3, (1992), 252-261.
[30] C. Hofmeister, et al., *Applied Software Architecture*, Addison Wesley (2000).
[31] P. Clements, et al., *Documenting Software Architectures: Views and Beyond*, Addison Wesley (2002).
[32] G. Krasner and S. Pope, A Cookbook for Using The MVC User Interface Paradigm in Smalltalk-80, *Journal of Object-Oriented Programming*, Vol. 1, No. 3, (1988), 26-49.
[33] E. Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley (1995).
[34] F. Buschmann, et al., *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley & Sons (1996).
[35] C. Larman, *Applying UML and Patterns*, Prentice Hall (2002).
[36] J. Lin and C. Yeh, An Object-Oriented Software Project Management Model, *The International Journal of Computers and Their Applications*, Vol. 10, No. 4, (2003), 247-262.
[37] Microsoft Corporation, *Microsoft Visual C++ Programming with MFC*, MS Press (1995).
[38] G. Shepherd, *MFC Internals*, Addison Wesley (1996).
[39] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison Wesley (2003).
[40] A. Leszek, et al., *Practical Software Engineering: A Case Study Approach*, Addison Wesley (2004).
[41] D. Kayal, *Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework*, APress (2008).

Chao-Yu Lin is a PhD candidate of the Department of Information Management at the National Dong Hwa University at Hualien in Taiwan. He received his M.S. degree in 1998 from the Department of Information Management at the Chaoyang University of Technology in Taiwan. Chao-Yu is a Senior Consultant, and he has more than ten years of work experience in Software Industry.

Jyhjong Lin is a full professor and the chair of the Department of Information Management at the Ming Chuan University in Taiwan. He received his Ph.D. degree in 1995 from the Computer Science Engineering Department at the University of Texas at Arlington in USA. His research interests include Software Engineering, Systems Architecture and Development, and Object-Oriented Techniques.

Wei-Pang Yang is the Dean of Academic Affairs of the National Dong Hwa University in Taiwan. He received his Ph.D. degree in 1984 from the Computer Engineering Department at the National Chiao Tung University in Taiwan. His research interests include database theory and application, information retrieval, data miming, digital library, and digital museum. Dr. Yang is a senior member of IEEE, and a member of ACM.