Mockup-driven Fast-prototyping Methodology for Web Requirements Engineering

Jia Zhang
Chicago Technology Partners Inc.
Chicago, IL 60615
jiazhangchicago@yahoo.com

Carl K. Chang
Department of Computer Science
Iowa State University
Ames, IA 50011
chang@cs.iastate.edu

Jen-Yao Chung
IBM T.J. Watson Research
Yorktown Heights, New York 10598
jychung@us.ibm.com

Abstract

Web application development differs from the development of traditional software in several significant ways; therefore requirements engineering for web applications entails new demands accordingly. This paper proposes an extreme web requirements engineering - mockup-driven fast-prototyping methodology to help elicit and finalize system requirements, as well as facilitate adjustment to quickly changing user requirements typical to web applications. Supporting the inclusion of customer feedback early in the development process, this strategy minimizes the risk of wasting valuable development efforts because of ambiguous or incomplete specifications. Real-life experiences of the use of the methodology in industry are reported as examples.

1. Introduction

Due to the distinctive features of web application different from traditional software [3], requirements engineering (RE) faces challenges in several significant ways. First is the insufficient requirements specification, since web application development usually starts from ill-structured and vague requirements. Second is the gap between the requirements and the hypermedia design on web browsers [9]. Third is the complexity management. Embracing computing and networking technologies, the development of a web application can be a very complex and costly task. Fourth is the variety of user groups. Web applications need to serve for much more inclusive user base, even for unknown users [3]. Fifth is the traceability. Web applications constantly change their requirements so as to meet the volatile demands from the market. Sixth is the fast release. Web applications usually request short development cycle in order to sustain the competition from others. Seventh is the request of integration with the latest web technology, as it is imperative to evolve quickly changing technology driven by competition [3]. Some of these challenges are not unique characteristics exhibited only by web applications. However, web applications necessitate all these demands to be fulfilled in one application; therefore how to effectively and efficiently elicit and validate frequently changing requirements starting from an ambiguous goal remains an indispensable challenge.

Although the last decade has witnessed numerous of comprehensive notations, models, and methodologies that have been conducted in RE, little attention has been paid to the methodologies coping with the requirements elicitation and finalization of web applications [1]. Our goal is to synergistically apply well-tried concepts in RE to web applications, in order to provide an efficient methodology. In this paper we propose a mockup-driven fast-prototyping methodology (MODFM) in order to help elicit and finalize system requirements, as well as facilitate adjustment to quickly changing user requirements typical to web applications. Seamlessly integrating with the most recent web technologies, MODFM guarantees to deliver running web application prototypes to incrementally elicit, validate, and finalize user requirements early and consistently; consequently reduce the cost of nearly inevitable changes to the business rules, programming environment, or software design.

The rest of this paper is organized as follows. In Section 2, the related work is discussed. In Section 3, we introduce a web application architecture. In Section 4, we present web system generator. In Section 5, we propose our mockup-driven fast-prototyping methodology. In Section 6, we discuss the experiments. In Section 7, we make a conclusion.

2. Related Work

There have been a number of researches conducted in RE. Among the variety of research achievements, four of the most promising RE approaches can be taken into account for RE for web applications: the fast-prototyping technique, structured analysis approach, use case methodology, and architecture-driven requirements engineering. Szekely defines the fast-prototyping concept as constructing a small-scale version of a complicated system in order to acquire critical knowledge required to build a full system [10]. This prototype not only provides to clients a complete picture what the final product will

Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)
0730-3157/03 $ 17.00 © 2003 IEEE

be, but also facilitates requirements validation, elicitation, and revision. Since its inception in 1970s, structured analysis has been considered as a powerful and natural approach to analyze complicated software application requirements [11]. Use case approach [5], originally proposed by Jacobson and colleagues, utilizes a scenario-driven mechanism and has been extensively adopted and widely considered as the most popular requirements elicitation technique in industry [6]. Software Architecture Based Requirements Engineering (SABRE) [2] aims at bridging between software architecture and RE, which core concept is that SA helps RE and RE helps firm up SA as a high-level solution.

These four techniques provide some clear guidelines and best practices to RE: prototype construction, functional decomposition, as well as structure-oriented and scenario-centered requirements elicitation. At this moment, however, it appears that little attention in these techniques has been paid to cope with the specific characteristics of web applications [8]. An additional limitation is that those methods may or may not integrate easily with current web technologies. Here we seek to incorporate these approaches and apply to web applications in order to provide efficient support for the end-to-end process of generating and elicitation of web requirements.

Our work integrates with and builds on top of a variety of state-of-the-art web technologies. J2EE [7] technology defines a platform to simplify enterprise development and deployment; the detailed information can be found from http://java.sun.com/j2ee. We chose J2EE due to the fact that it has been extensively considered as the de facto standard of web engineering [3].

3. Formalized Web Application Architecture

One principle idea of SABRE [2] is to conduct RE in the frame of appropriate software architecture. To apply this concept, we first propose a formalized software architecture for J2EE-oriented web applications. This architecture is based on our investigation in J2EE-related technologies, and the first author's empirical experiences as an architect designing twelve industrial web applications in the last three years, ranging from e-University suite, e-Hospital system, e-Payment system and web services, to authentication system. As illustrated in Figure 1, a web application system can be organized as a collection of modules, running in the environment of application server [7] with database configured and set up. Each module can be in turn divided into a list of interacted mini-modules. As shown in Figure 1, the arrows between mini-modules exhibit navigational relationship between them typical of web applications. For example, a user can navigate to mini-module #2 from mini-module #1. Figure 1 illuminates that each mini-module can be implemented as a tiny application consists of fourteen components and exhibits a two-tier MVC [4] model [12]. These components can be integrated into the front-end tier and the back-end tier. The front-end tier exhibits a MVC [4] model: every JSP page represents a view, together with a form bean contains the contents of the JSP page; a servlet acts as the controller; and two action classes–a (pre-, post-) action pair–act as the model. Pre-action prepares the contents for the JSP view; while post- action gathers user input from the JSP page and performs some operations. Capturing the essential scheme underlying of each web page, this formalization clarifies the navigation flow and enables automatic code generation that we will discuss in the later section.

The back-end tier exhibits another MVC model: service component exposes the back-end to the front-tier; EJB-related [7] components implement the application

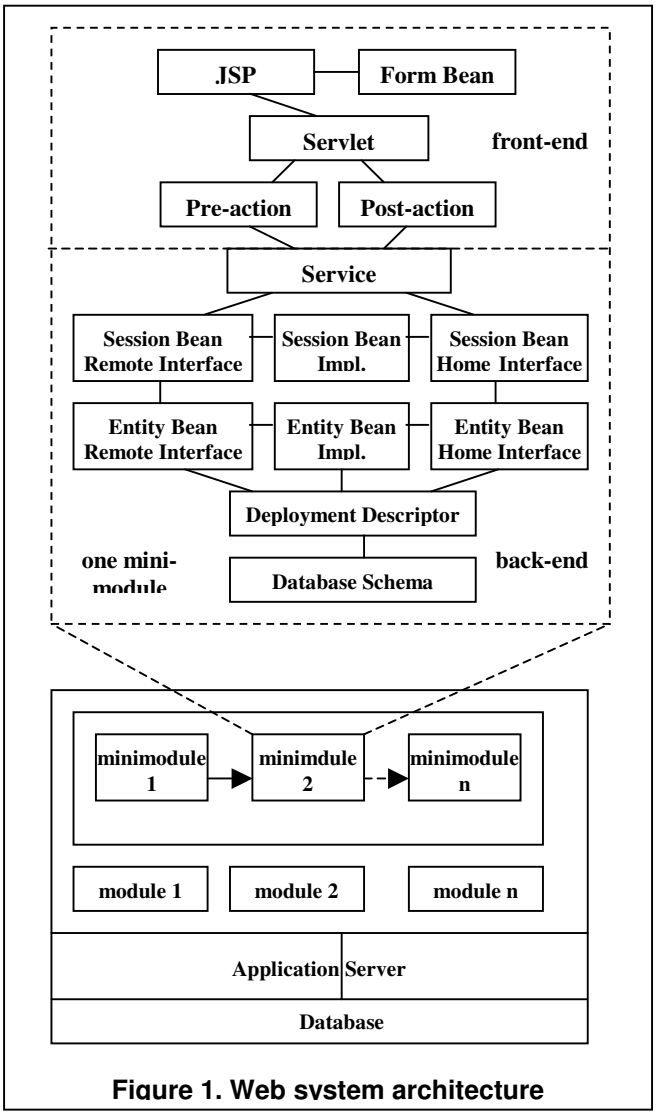

Figure 1. Web system architecture

model; and the database acts as the data storage. As shown in Figure 1, a typical EJB-related implementation consists of seven sub-components: session bean remote interface, session bean home interface, session implementation, entity bean remote interface, entity bean home interface, entity implementation, and deployment descriptor.

This two-tier MVC architecture clearly identifies an object-oriented and component-layered structure for web applications. According to this architecture, any module in a web application can be realized by the composition of JSP pages, form beans, pre-actions, post-actions, service methods, EJB components, and database schemas. As a result, automatic code generation becomes highly practical.

4. Web System Generator (WSG)

Our structured architecture makes it reasonable and feasible to apply automatic code generation for web application development. Our previous research yields a J2EE-oriented web system generator (WSG), which contains two related code generators: web code generator (WGenerator) and menu system generator (MSG). We enhance WSG so as to support requirements elicitation. The detailed information about WSG can be found elsewhere [12]; here we just summarize WSG and discuss our enhancement. Based on the sound understanding of J2EE technology and the author's ample industrial experiences, WGenerator provides a complete set of templates for each of the fourteen components specified in Figure 1. As a result, one merely needs to provide data models and some configuration criteria in XML files; and WGenerator will generate all related code for each model from front-end to back-end. With WGenerator generating a running unit piece, menu system generator (MSG) acts as its complement in order to glue together unit pieces to become a running system. User defines hierarchical menu system in a XML file: not only features of each menu item, but also both navigational relationship and spatial display relationship between menu items.

```
<WGenerator>
 <OBJ NAME="ECheckPaymentProfile" TNAME="echeck_profile">
   <ATTR ATNAME="Key" ATYPE="String" UNIQ="YES"/>
   <ATTR ATNAME="HolderName" ATYPE="String" VALID="YES"/>
   <ATTR ATNAME="AccountType" ATYPE="String" VALID="YES"/>
   <ATTR ATNAME="RoutingNum" ATYPE="String" VALID="YES"/>
    <ATTR ATNAME="AccountNum" ATYPE="String" VALID="YES"/>
   <ATTR ATNAME="EmailAdd" ATYPE="String" VALID="YES"
          COMMENTS="Has to be valid email address"/>
   <ATTR ATNAME="DayPhone" ATYPE="String" VALID="YES"/>
   <ATTR ATNAME="NightPhone" ATYPE="String" VALID="YES"/>
   <ATTR ATNAME="AddressLine1" ATYPE="String" UNIQ="NO"
  COMMENTS="1. if the address is a domestic US address:
                    a) address line 1 should not be empty;
                    b) address line 2 is optional;
                    c) city, state, zip code can not be empty;
                    d) zip code should be format of either "ddddd" or "ddddd-
dddd", while each "d" represents a digit;
                  2. if the address is a foreigh address:
                    a) address line 1 should not be empty;
                    b) address line 2 is optional;
                    c) city can not be empty;
                    d) state, zip code are optional;"/>
   <ATTR ATNAME="AddressLine2" ATYPE="String" UNIQ="NO"/>
    <ATTR ATNAME="City" ATYPE="String" UNIQ="NO"/>
   <ATTR ATNAME="State" ATYPE="String" UNIQ="NO"/>
   <ATTR ATNAME="Country" ATYPE="String" UNIQ="NO"/>
   <ATTR ATNAME="ZipCode" ATYPE="String" UNIQ="NO"/>
 </OBJ>
</WGenerator>
```

Figure 2. data file ECheckPaymentProfile.xml

We enhanced WSG in order to support requirements elicitation. Figure 2 illustrates an enhanced data file of a user's payment profile of electronic check. After the user defines his payment profile, he can select to use the profile to pay his bills without re-typing all related information. For every item of the corresponding data model, one can define its name and data type. *ECheckPaymentProfile* contains fourteen data items: holder name, account type, routing number, account number, email address, day time phone number, evening phone number, address line 1, address line 2, city, state, country, and zip code. Key is used to store the primary key internally. The detailed information about the specification rules for each item can be found in [12]. Attention should be paid here that we provide the ability for users to specify comments on data items. As shown in Figure 2, two data items are associated with comments: email address and address line 1. Comments of the former one say that the email address has to be valid email address; comments of the latter item declare the validation rules for address.

We as well enhance the JSP page generation accordingly. If one data item is associated with comments, a clickable link with word "Comments" will be shown beside of the data item on the generated page. If user click on the link, a window will popup displaying the comments defined in the corresponding data file. Taking Figure 2 as an example, Figure 3 is its corresponding generated JSP page. We can see that there are two links beside of email address and address line 1 respectively; each link is in blue color. Figure 3 also illuminates that the window popped up when the link beside of address line 1 is clicked. This window contains the exact information defined in Figure 2 as comments of data item address line 1, which is the validation rule for address.

This enhancement facilitates WSG to server for requirements elicitation. In the process of web requirements engineering, it is normally sufficient for business rules to be defined in informal language or even natural language. This enhancement enables business rules to be associated with corresponding data items on the web page, so that not only clients can review the rules, but also

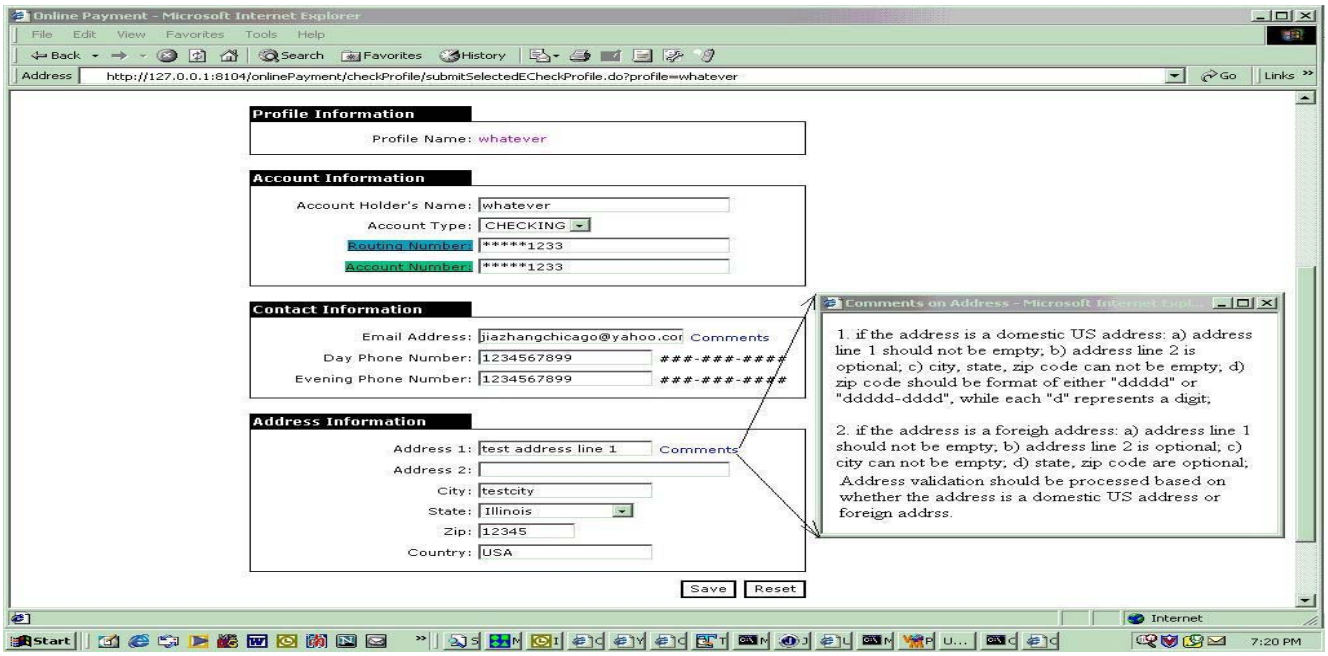

Figure 3. Generated JSP page for Figure 2

developers can implement these rules and replace the comments accordingly.

5. Mockup-driven Fast-prototyping Methodology (MODFM)

On the basis of our formalized web system architecture and enhanced web system generator, we propose a mockup-driven fast-prototyping methodology (MODFM) serving for web requirements engineering. Applying fast-prototyping concept [10] in web applications, a mockup refers to a running, navigable, partial or full-sized model of a web application, used for requirements elicitation, validation, and finalization. We should notice that MODFM is applicable to a web application if the application satisfies the following four assumptions. First, the application is a typical web application, which is navigable through a set of web pages. Second, the application can be decomposed into modules exhibited by a hierarchical menu system. In other words, all the web pages can be organized into a menu system. Third, the project at least starts with a very high-level functional description of the system. Fourth, the hierarchy of the menu system is no greater than three. In addition, we suppose that a test server has already been set up so that all mockups can be delivered to the test server, while clients can test the mockups and provide feedback remotely.

The essential tenet of MODFM is that, always use running mockup system to elicit and validate user

Mockup-Driven Fast-prototyping Methodology Algorithm

1. Functionally decompose the system into menus. Every menu item is identified as one web page. It can be as simple as a "Hello world" page.
2. Generate a mockup containing only menu system and dummy pages into a released version.
3. Deliver the mockup into test server. Go back to step 1 if client would like some changes; otherwise go to step 4.
4. Iterate on every menu item step 4 through 7. If finished, go to Step 8.
5. Educe scenarios of each menu item page. Identify a list of pages, gather page information, and record the business logic between pages.
6. Generate a mockup, with business logic displayed on every page as written document.
7. Deliver the mockup to client. Go back to step 5 if client would like some changes; otherwise jump to step 4.
8. Deliver the mockup to developers.

requirements. This iterative process applies a top-down approach to decompose system functionalities to web pages while simultaneously generating navigable running mockup system, with functionalities organized by menu system. The "Mockup-driven Fast-prototyping Methodology Algorithm" side bar summarizes this iterative process. A module acts as the container for a list of web pages that exhibit high cohesion and low external coupling. In a typical web application, a module is normally realized by a menu item. Functional decomposition tasks consist of identifying menu items and sub-items, and organizing them in a menu system. Scenario analysis tasks consist of identifying web pages, finding out the navigational relationship among pages, and allocating pages to the appropriate menu item. Mockup construction then occurs bottom up.

5.1. Menu system identification

The first step of MODFM is to functionally decompose a system into a menu system, where structured analysis approach [11] is applied. MODFM treats the entire system as a single abstract module representing the system's highest-level functionality. Applying use case methodology [5], scenario analysis refines system-level requirements; and helps to identify high-level menu items within each functional module. One of the projects where MODFM is actually being utilized for the requirements analysis is a typical web application - an e-Payment system. E-Payment system is an on-line payment system that facilitates users to pay the bill and review bill presentment from the Internet. Figure 4 illustrates the original menu system identified. As this example shows, we identify six menu items at the e-Payment system level: message board, user preference, payment preference, electronic bill presentment, electronic bill payment, and bill history. Secondly, a mockup will be generated containing only menu system and dummy pages into a released version. When the mockup is delivered into the test server in step 3, user feedback can be gathered quickly. If clients would like some changes, we would go back to the step 1 to revise the menu system, and regenerate the mockup. This iteration will repeat until clients are satisfied with the menu system, and then procedure will move forward to the next step.

5.2. Iterative requirements elicitation

The following steps are going to be iterated on every menu item. For each menu item, discussions need to be conducted with clients in order to find out possible scenarios. Based on the scenario, a list of pages can be identified accordingly; and the navigational relationships between pages can also be identified. Figure 5 is an

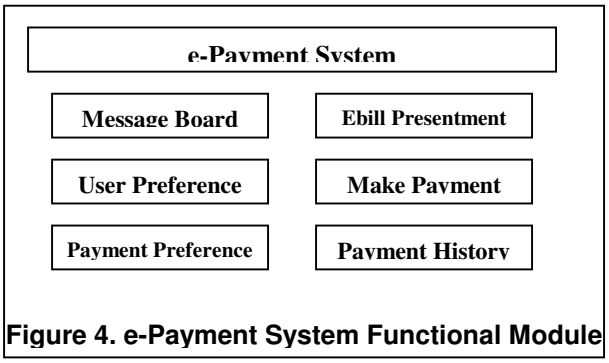

Figure 4. e-Payment System Functional Module

example of the pages and their relationships identified for the module payment preference in Figure 4 above. As shown in Figure 5, five pages are identified. The entry point of the payment profile module is the page payment profile list. Users can click on the list and delete some profile items if desired. Users can also select one profile to edit the information. Two pages are provided serving for the credit card editing and electronic check editing respectively; and the control will be sent back to the page of the list of profiles. Users can as well choose to add new credit card profiles or e-check profiles. Two pages serve for these purposes respectively; and the control will be transferred back to the page of the updated list of profile.

For every web page identified, analysts discuss with users in order to specify a list of data items that the page will display. Taking e-check payment profile page from Figure 5 as an example, users are required to provide fourteen items of information, which can be summarized by analysts in data file illustrated in Figure 2. We can see that the business rules are recorded with the page, associated with corresponding data item. We already gave an example in the previous section. After data items are identified for a web page and recorded in data file, we can invoke our WSG to generate a running mockup, with business logic displayed on every page as links beside the corresponding data items and be able to popup in another window as written document. This mockup can be

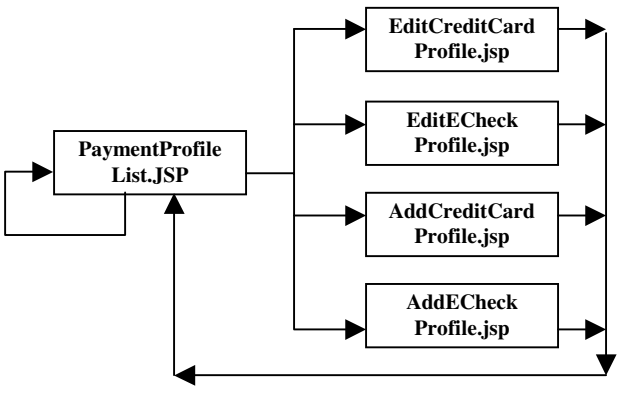

Figure 5. Web page identified for payment profile

delivered to clients for feedback. If clients would like some changes; analysts would change the data file, and regenerate the mockup, therefore repeating the process; otherwise one can move on to the next page. After finishing all pages, the mockup can be delivered to developers for the business logic development.

6. Experiments

We have tested the MODFM on three industrial web applications. The first one is an e-Hospital service suite that allows users to customize a new hospital web site at the run time, as well as provides to hospitals with web-based capabilities such as managing accounts, administrations, notifications, etc. The second one is an e-University administrative system that support functionalities such as student records, admissions, financial aid, financial services, registration, payment management, faculty, etc. The third one is an e-Payment system that is an on-line electronic payment system supporting functionalities such as both e-Check and credit card payment, bill presentment as both paper bill and PDF format, bill loading, and payment history presentment, etc. The detailed case study will be reported elsewhere. We believe that MODFM possesses great potential in software industry in web application area. In our experiences, MODFM promises more direct requirements elicitation from running mockups, more client satisfaction due to the fact that what clients see is what they will get, more efficient human resource usage, since no technical resources are necessary in mockup-driven requirements elicitation.

7. Conclusion

MODFM improves the efficiency of requirements elicitation for web applications. It is accomplished by utilizing formalized J2EE-oriented architecture and web system generator. MODFM guarantees to deliver running web applications to incrementally elicit, validate, and finalize user requirements early and consistently; consequently reduce the cost of nearly inevitable changes to the business rules, programming environment, or software design. Many of these practices have been part of conventional wisdom for years, such as fast-prototyping, use case analysis, and SABRE; but rethinking their interaction is the value of MODFM.

The efficiency and effectiveness of MODFM highly rely on our architectural model and web system generator and these two techniques are J2EE-oriented. Hence, if the web technology changes significantly, these two techniques need to be upgraded accordingly. However, we believe that the concept and approach of MODFM is extensively applicable; therefore it can be fully reused by future web applications as long as corresponding architecture-oriented web system generator is plugged in.

8. References

[1] D. Bolchini and P. Paolini, "Capturing Web Application Requirements through Goal-Oriented Analysis", The 5th Workshop on Requirements Engineering, Valencia, Spain, Nov. 11-12, 2002.

[2] C.K. Chang, J.C. Huang, S. Hua, and A.K. Combelles, "Function-class Decomposition: A Hybrid Software Engineering Method", *IEEE Computer*, Dec. 2001, pp. 87-93.

[3] Y. Deshpande and S. Hansen, "Web Engineering: Creating a Discipline among Disciplines", *IEEE Multimedia*, Apr.-Jun. 2001, pp. 82-87.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison Wesley 1994.

[5] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object Oriented Software Engineering – A Use Case Driven Approach,* Addison-Wesley, 1992.

[6] W.J. Lee, S.D. Cha, and Y.R. Kwon, "Integration and Analysis of Use Cases using Modular Petri Nets in Requirements Engineering", *IEEE Transactions on Software Engineering*, vol. 24, no. 12, Dec. 1998, pp. 1115-1130.

[7] http://java.sun.com/j2ee.

[8] J. Nawrocki, M. Jasinski, B. Walter, and A. Wojciechowski, "Extreme Programming Modified: Embrace Requirements Engineering Practices", IEEE Joint International Conference on Requirements Engineering (RE'02), Essen, Germany, Sep. 9-13, 2002, pp. 303-310.

[9]. N. Nüell, D. Schwabe, and P. Vilain, "Modeling Interactions and Navigation in Web Applications", Proceedings of the World Wide Web and Conceptual Modeling'00 Workshop, ER'00 Conference, Springer, Salt Lake City, 2000.

[10] P. Szekely, "User Interface Prototyping: Tools and Techniques", *USC/Information Sciences Institute,* 1994.

[11] E. Yourdon, *Modern Structured Analysis*, Yourdon Press, Upper Saddle River, N.J., 1989.

[12] J. Zhang and J.Y. Chung, "Mockup-driven Fast-prototyping Methodology", *Software-Practice and Experience,* to appear.