Design Issues for Supportable Enterprise Web Architecture using Frameworks

Mehmet Bilgi, Halûk Gümüşkaya
Department of Computer Engineering, Fatih University, 34900 Büyükçekmece, İstanbul
mehmetbilgi@st.fatih.edu.tr, haluk@fatih.edu.tr

Abstract

This paper describes a Supportable Enterprise Web Architecture (SEWA) which is highly supportable, i.e. understandable, maintainable and scalable, and based on software engineering principles particularly object oriented design techniques. The development of SEWA emphasizes a roundtrip architectural modeling lifecycle. The lifecycle begins with the definition of a meta-architecture aimed at minimizing and managing software complexity. It then embraces various supportability metrics to ensure that the implementation conforms to the architectural design and that the resulting system is supportable. The architecture is based on basic and enterprise design patterns and on few new patterns specific to SEWA. This paper also describes best practices and design guidelines for building a real world enterprise web application based on SEWA using JSF, Spring and Hibernate as pluggable sample frameworks to evaluate the supportability and flexibility of the architecture.

1. Introduction

Applying software engineering principles, particularly object-oriented techniques, to the Web can be difficult. Many current Web technologies lend themselves to—or even encourage—bad practices. Scripting and server-page technologies can encourage cut-and-paste reuse, direct-to-database coding, and poor factoring [1].

The right combination of technologies, software engineering and object oriented design principles are increasingly important in web development. We applied best practices and design guidelines of software engineering and object-oriented techniques and a roundtrip modeling lifecycle to develop a supportable enterprise web architecture using a layered meta architecture approach.

We first chose J2EE as our reference software platform. Selecting a reference software platform, say the J2EE or .NET, as the starting point for a product or product line has strategic implications. The selection of one community over another has major cost implications. The J2EE community has much more of a low-cost, open source history than does the .NET community. J2EE was our selection for the starting point. There are many architectural choices and open source alternatives that can be used for enterprise application development on the J2EE platform. After selecting our reference platform, we defined a meta architecture, which we called Supportable Enterprise Web Architecture (SEWA). In order to evaluate the supportability and flexibility of this architecture, we developed an enterprise web application test bed, a simple product management system based on JSF, Spring and Hibernate as pluggable sample frameworks, each used for a different layer for our reference SEWA. Besides the test bed application, we used some modeling and analysis tools to evaluate the architecture.

This paper is structured as follows. The next section describes the concepts of software quality and process centered architecture from our meta architecture's point of view. In section 3, the principles and design guidelines in SEWA are presented. Some information is given about the JSF, Spring and Hibernate frameworks in section 4. Then our enterprise web application test bed is explained in detail in section 5. Finally, we present the test and analysis results of our reference meta architecture on the test bed in section 6.

2. Supportability as Software Quality Criteria and Process Centered Software Architecture

One of the major issues in software systems development today is quality. A quality attribute is a nonfunctional characteristic of a component or a system. Software quality is defined in IEEE 1061 [2] and it represents the degree to which software possesses a desired combination of quality attributes. Another standard, ISO/IEC 9126-1 [3], defines a software quality model. According to this definition, there are six categories of characteristics (functionality, reliability, usability, efficiency, maintainability, and portability), which are divided into subcharacteristics. Other simpler, widely accepted quality metrics for software systems are understandability, maintainability and scalability [4]. These three properties combined are frequently called the system's *supportability* features. We use this last quality metrics as the basic quality criteria for our meta architecture.

Supportability brings layering to a project. Layering is a very useful approach for an enterprise web project but it has drawbacks also. The most important of them is some performance reduction. There are certain types of applications, like real time applications, that can not resist this kind of a performance limitation. High supportability in enterprise web application projects developed by a large number of programmers and business people is more important than runtime efficiency.

The idea of predicting the quality of a software product from a *higher-level design description* is not a new one. In 1972, Parnas [5] described the use of *modularization* and *information hiding* as a means of high level system decomposition to improve flexibility and comprehensibility. In 1974, Stevens et al. [6] introduced the notions of module *cohesion* and *coupling* to evaluate alternatives for program decomposition. A software module is stable if cohesion (intra-module communication) is strong and coupling (inter-module interaction) is low. Good software architecture tries to maximize cohesion and minimize coupling.

One of the major design tasks in building distributed enterprise applications is to design good *software architecture*. During recent years, the notion of software architecture has emerged as the appropriate level for dealing with software quality. The software architecture of a system is defined as "the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" [7]. This definition focuses only on the *internal aspects* of a system and most of the software analysis methods and tools are based on it. Another definition establishes software architecture as "the structure of components in a program or system, their interrelationships, and the *principles and design guidelines* that control the design and evolution in time". This *process-centered definition* takes into account the presence of principles and guidelines in the architecture description. This second definition is more comprehensive and defines our architectural approach better in SEWA which has principles and guidelines in its roundtrip architectural modeling activities.

3. Design Principles in SEWA

3.1. Roundtrip Architectural Modeling

The development of an enterprise application must embrace a *roundtrip architectural modeling* lifecycle. Such a lifecycle begins with the selection of a reference software platform, such as J2EE, and definition of a *meta-architecture* aimed at minimizing and managing software complexity.

The SEWA design and conformance verification are performed with proactive and reactive approaches. Architectural design takes a proactive approach to managing dependencies. This is a forward-engineering approach – from design to implementation. The aim is to deliver a software design that minimizes dependencies by imposing an architectural solution on programmers.

Proactive approach is supported by the reactive approach that aims at measuring dependencies in implemented software. This is a reverse-engineering approach –from implementation to design. The

implementation may or may not conform to the desired architectural design. The purpose is to show in numbers how much the implemented system is worse than a good SEWA solution or another dependency-minimizing architecture. It then embraces various supportability metrics to ensure that the implementation conforms to the architectural design and that the resulting system is supportable. During the reverse engineering phase, the audits, metrics and architecture analysis results produced by modeling and analysis tools are used for conformance to the architecture.

3.2. SEWA Meta Architecture

The SEWA layered meta architecture is comparable to PCMEF [4, 8, 9] and the web framework presented in [1]. A layered architecture is a system containing multiple, strongly separated layers, with minimal dependencies between the layers. Such a system has good separation of concerns, meaning that we can deal with different areas of the application code in isolation, with minimal or no side effects in different layers. By separating the system's different pieces, we make the software supportable, adaptable so that we can easily change and enhance it as requirements change. The layers we are concerned with here include Presentation, Control, Mediator, Entity, and Foundation. With reference to the MVC (Model View Controller) framework, Presentation corresponds to MVC View, Control to Controller, and Entity to Model. Mediator and Foundation do not have MVC counterparts.

Presentation has classes that handle the user interface and assist in human-computer interactions. Control has program logic classes like searching for information in entity objects, asking the Mediator layer to bring entity objects to memory. Entity manages business objects currently in memory. Mediator mediates between entity and foundation subsystems to ensure that control gets access to business objects. It manages the memory cache and synchronizes the states of business objects between memory and the database. Foundation has classes that know how to talk to the database and produces SQL to read and modify the database.

The SEWA is based on basic and enterprise design patterns and on few new patterns specific to SEWA. The main sources of patterns for our architecture are GoF (Gang of Four) Patterns [10], Patterns of Enterprise Application Architecture [11] and Core J2EE Patterns [12].

3.3. Dependencies and Basic Design Principles

An object oriented software system has a set of intercommunicating objects. The allowed object communication paths, defined either statically (compile-time) or dynamically (run-time), determine the possible set of object dependencies. A necessary condition to understand a system behavior is to identify and measure all object dependencies [4]. A supportable system has dependency metrics.

Dependencies can be on classes, messages, events, and inheritance. The idea is to uncover all object dependencies in a system and make them explicit. The associations are established on all directly collaborating classes in compile-time data structures. The dynamic links formed at run-time and uncontrolled polymorphic behaviors are very difficult to control and create maintenance hassle. SEWA legitimizes run-time object communication using compile-time data structures and forbids muddy programming solutions utilizing just run-time programming structures.

Circular dependencies between layers, between packages and between classes within packages must be broken. Cycles should be resolved by creating a new package specifically to eliminate the cycle, and by forcing one of the communication paths in the cycle to communicate via interface [8].

The main dependency structure in SEWA is top-down. Objects in higher layers depend on objects in lower layers. Consequently, lower layers are more stable than higher layers. Upward dependencies are realized through loose coupling facilitated by interfaces, event processing, acquaintance package and similar techniques. Dependencies are only permitted between neighboring layers.

The upward notification principle in SEWA promotes low coupling in bottom-up communication between layers. This can be achieved by using asynchronous communication based on event processing. Objects in higher layers act as subscribers (observers) to state changes in lower layers. When an object (publisher) in a lower layer changes its state, it sends notifications to its subscribers. In response, subscribers can communicate with the publisher (now in the downward direction) so that their states are synchronized with the state of the publisher.

4. Frameworks for Web Applications: JSF, Spring and Hibernate

For the presentation layer, experience shows that the best practice is to choose an existing, proven Web application framework rather than designing and building a custom framework. We have several Web application frameworks to choose from, e.g., Struts, WebWork, and JSF. We used JSF [13] for SEWA because it defines suitable plug-in points for customization of behavior, like validation and conversion of user data to server-side objects. SEWA also benefits from JSF in the following important areas: declarative page navigation using rules and mapping user actions to server-side components and programmatically manipulating those components.

Spring organizes middle layer objects and handles plumbing for us [14, 15]. Spring can eliminate the proliferation of singletons and facilitates good object-oriented programming practices, e.g., programming to interfaces. Spring uses AOP (aspect-oriented programming) to deliver declarative transaction management without using an EJB container. This way, transaction management can be applied to any POJO (Plain Old Java Objects). Spring transaction management is not tied to JTA (Java Transaction API) and can work with different transaction strategies. Spring allows the dependencies of components on each other to be injected declaratively. This approach eliminates the need for manual configuration of components and unnecessary dependencies.

The foundation layer handles the data persistence with the relational database. Different approaches could be used to implement the foundation layer. Pure JDBC is the most flexible approach; however, low-level JDBC is cumbersome to work with, and bad JDBC code does not perform well. Another approach is use entity beans. An entity bean with container-managed persistence is an expensive way to isolate data-access code and handle O/R (object-relational) mapping data persistence. It is an application-server-centric approach. An entity bean does not tie the application to a particular type of database, but does tie the application to the EJB container. The last approach to implement the foundation layer is to use an O/R mapping framework. An O/R mapping framework takes an object-centric approach to implementing data persistence. An object-centric application is easy to develop and highly portable. Several frameworks exist under this domain—JDO (Java Data Objects), Hibernate, TopLink, and CocoBase are a few examples. We use Hibernate [16] in the implementation of SEWA to map the business objects to the relational database.

5. An Enterprise Web Application Test Bed

In order to evaluate the SEWA meta architecture and design principles, we developed an enterprise web application test bed, a simple product management system, based on JSF, Spring and Hibernate as pluggable frameworks, each used for a different layer for our reference SEWA [17].

We used Eclipse 3.0 [18] as a Java web application development and analysis platform. Borland Together for Eclipse 7.0 [19] was used as a modeling and analysis tool and IBM Structural Analysis for Java (SA4J) [20] was used for analysis, verification and validation purposes. The analysis results of these tools were used in the reverse engineering phase, and several refactoring techniques [21] were applied to improve the existing code. The test bed including the product management system and the development and analysis tools mentioned above are shown in Figure 1.

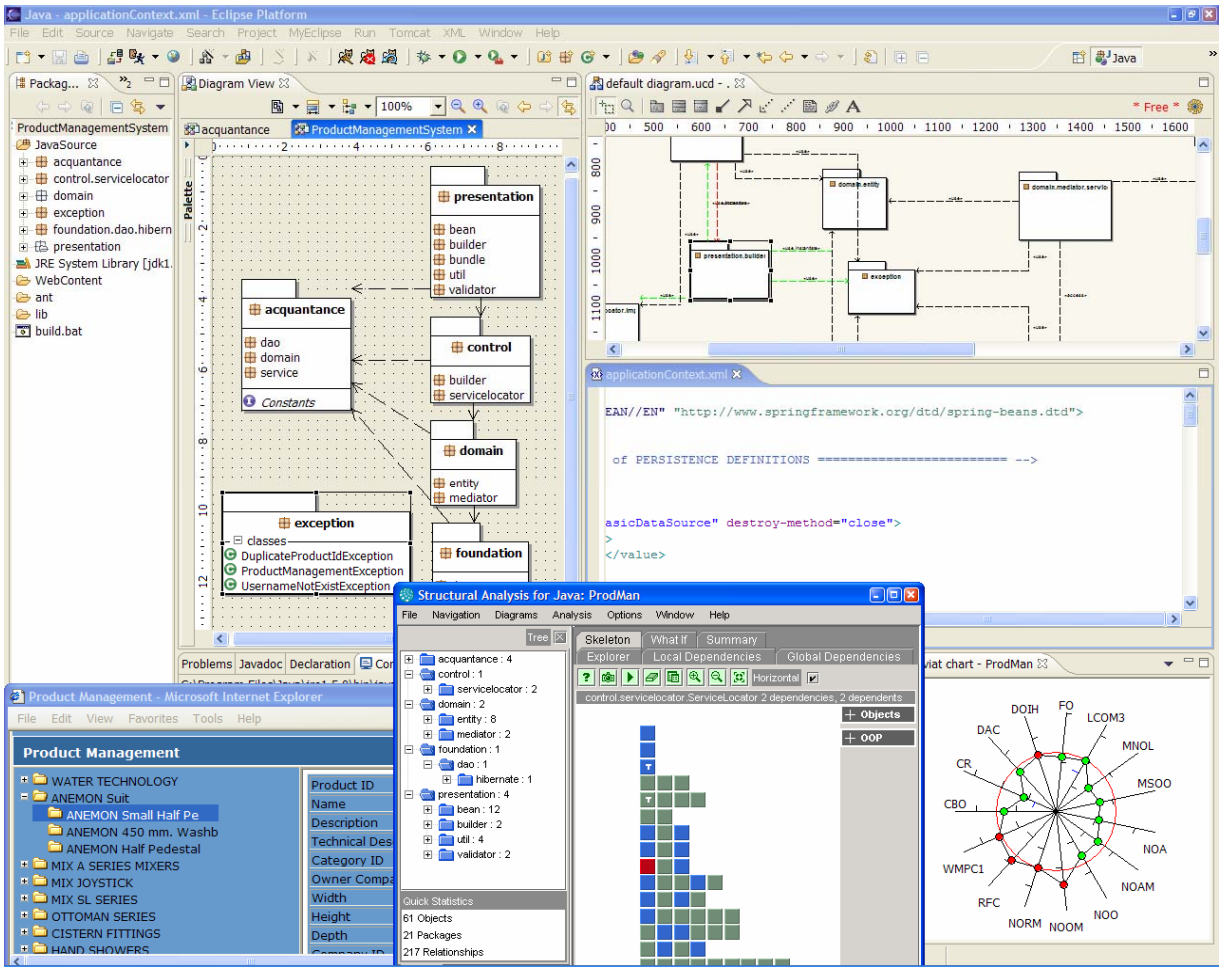


Figure 1. Enterprise web application test bed.

The product management system, as our test bed, was designed and implemented using the principles presented in this paper. The top level view seen from the highest level of the subsystem hierarchy and their dependencies are given in Figure 2. There are totally 15 subsystems. In the following subsections, the four main subsystems will be described in detail. The acquaintance and exception packages are not in the subsystem hierarchy. The acquaintance package keeps the layers loosely coupled with interfaces. Direct usage and instantiations of classes between layers are prevented by using these interfaces.

In the presentation layer, JSF beans delegate the user actions and input in the server side. JSF also allows programmatic control of the rendered components by backing beans. In **`presentation`** package, there are five sub-packages. **`Util`** contains utility classes that are related to presentation. **`Validator`** package is used for a custom tag that checks a select option to be in the rage of certain boundaries. **`Builder`** is responsible for conversion of domain objects and beans. **`Bundle`** contains the key-value mappings for a language to be used in the web pages. **`Bean`** package is responsible for the main presentation actions associated to system users. The tree component used in the application can be modified programmatically by the backing beans in the **`bean`** package. This leads to separation of code from page scripts, which results in better understandability and reduced maintenance costs.

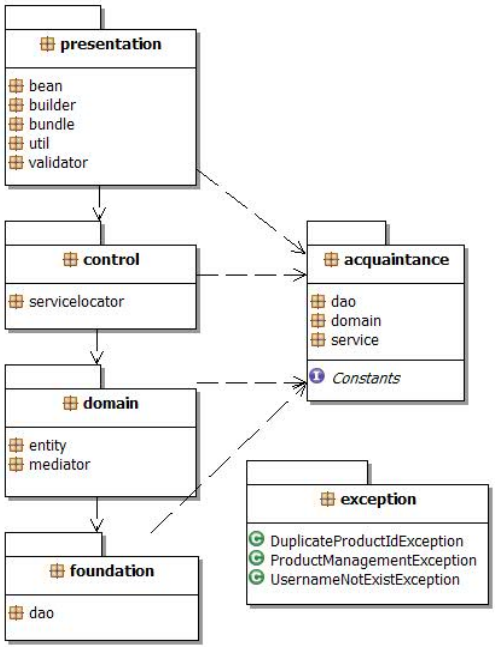


Figure 2. The top level view of subsystems and their dependencies.

The control layer consists of two parts. The first part is handled by JSF. JSF provides conversion and validation of input and actions and page navigation capabilities. These features can be considered as combination of input and application control [9]. For example, **`in faces-navigation.xml`** configuration file, the page navigation rules are defined by declaring two main conditions and the result. The conditions include: the source page of the request and a preset or calculated outcome. The Control includes the **`servicelocator`** sub-package as the second part. This package is responsible for locating the service beans from Spring's application context registry.

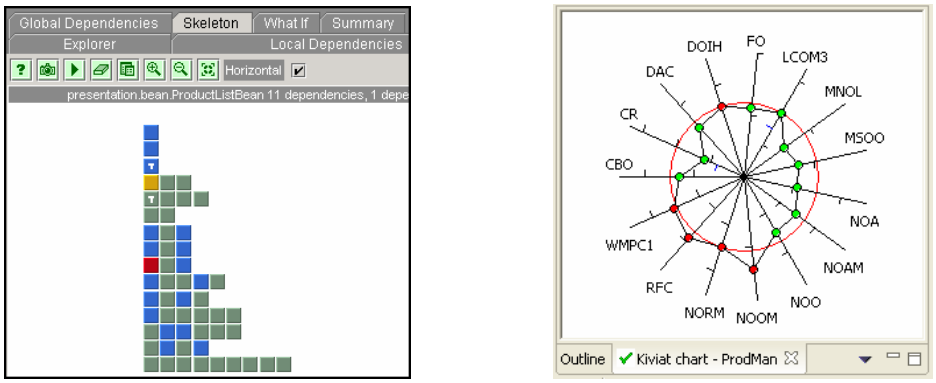
Spring keeps a registry of service objects with their names. A **`ServiceLocatorBean`** object looks up a necessary service by providing the service's name. This facility is very important for **`presentation`** to get necessary services.

`Domain` package contains two sub-packages: **`mediator`** and **`entity`**. Once the service references are gathered via Spring, they can deliver the application-wide services. These services may include basic CRUD operations and implemented in the **`mediator`** package which contains two services: **`UserService`** and **`ProductManagementService`**. When a business service is requested, a hand-over from **`mediator`** package to **`entity`** package occurs, at the exact time that the implemented logic shifts from application to business logic. **`mediator`** is also responsible for issuing data access requests to **`foundation`** package. These data access requests may be originated from both business related code (**`entity`** package) and non-business code (directly from **`control`** or **`mediator`** itself). In the **`entity`** package we have the business objects like **`Product`** and **`Category`**. These business objects implement the related business logic. They are also mapped to database tables.

`Foundation` package decouples the application and business services from the underlying data access strategy. The data access may be implemented via a single or multiple data base accesses. We used Hibernate to access a single data source. Spring provides a template to access database using Hibernate. Using this Spring template has several benefits, including logical conversions of exceptions thrown by Hibernate to **`RuntimeException`**s and elimination of session-related repeated code.

6. Analysis of the Web Application

The SEWA roundtrip software development cycle has an iterative process. That is, at the end of each iteration in our software development process, the resulting application code is analyzed to monitor the overall progress and architectural deviations. To find the architectural deviations in the testing application, SA4J was used as a dependency analysis tool and Together for Eclipse 7.0 was used for monitoring SEWA basic design metrics. We used the analysis results of these programs when comparing the implemented architecture with our reference SEWA. In Figure 3 (a), the skeleton analysis results of the system under test are given. SA4J produces a triangle-like shape for well engineered systems. For poorly engineered systems that allow direct usage of classes between layers, SA4J draws horizontal extensions at the upper levels of the shape.



(a) (b)

Figure 3. Skeleton analysis results using SA4J (a) and metric results visualized in Together (b).

SA4J has many useful dependency analysis features. "What If" scenario allows presenting the future effect of a possible change in one class on the overall system. However, Together is good at calculation of a wide range of audits and metrics and visualization of the results.

The audits and metrics feature of Together was used mainly for our basic metrics analysis. For basic design principles, like layering, coupling and cohesion, the important metrics include: lack of cohesion of methods (LCOM3), coupling between objects (CBO) and weighted methods per class (WMPC1) which is a measure of maintenance costs. Halstead effect together with WMPC1 gives an accurate vision of maintainability. All mentioned metrics for our testing system were found to be in the acceptable boundaries (dots in the red circle) as seen from the diagram given in Figure 3 (b). For situations for which metrics and analysis results don't satisfy the expectations, structural refactoring techniques are applied to the deviated subsystems (i.e. packages, classes) at the next iteration phase.

7. Summary

Supportable complex systems take the form of hierarchy and composition of objects. Intra-linkages of components must be stronger than inter-linkages. Dynamic links should be legalized as static associations. Complex enterprise applications that work are the result of simple systems that worked.

In this paper we introduced SEWA and its test bed application. The test bed was developed to apply the best practices of software engineering and object oriented design principles. The enterprise web application test bed is based on JSF, Spring and Hibernate, each used for a different layer for our reference SEWA. Besides the test bed application, we used some modeling and analysis tools to evaluate the architecture.

This paper also showed how to integrate different web technologies such as JSF, Spring and Hibernate, using a well defined SEWA meta architecture and design principles to build a real-world Web application. The combination of these three technologies provides a solid Web application development framework. By partitioning the whole Web application into layers and programming using interfaces, the technology used for each application layer can be replaced. For example, Struts can take the place of JSF for the presentation layer, and JDO or Toplink can replace Hibernate in the foundation layer.

References

[1] A. Knight, N. Dai, "Objects and the Web", *IEEE Software*, pp. 51-59, March/April 2002.

[2] IEEE Standard 1061-1992, *Standard for Software Quality Metrics Methodology*, 1992.

[3] ISO/IEC 9126-1, *Software Engineering - Product Quality - Part 1: Quality Model*, 2001.

[4] L. A. Maciaszek, B. L. Liong, "Designing Measurably-Supportable Systems", *Advanced Information Technologies for Management*, Research Papers No 986, ed. by E. Niedzielska, H. Dudycz, M. Dyczkowski, pp.120-149, 2003.

[5] D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of ACM*, vol. 15, no. 12, pp. 1053-1058, 1972.

[6] W.P. Stevens, G.J. Myers, L.L. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115-139, 1974.

[7] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 2nd Edition, Addison-Wesley, 2003.

[8] L. A., Maciaszek, "Roundtrip Architectural Modeling", *The Second Asia-Pacific Conference on Conceptual Modeling*, Newcastle, Australia, January 30 - February 4, 2005.

[9] L.A. Maciaszek, B.L Liong, *Practical Software Engineering*, Harlow England, Addison-Wesley, 2005.

[10] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[11] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

[12] D. Alur, J. Crupi, D. Malks, *Core J2EE Patterns*, 2nd Edition, Prentice Hall, 2003.

[13] Official JavaServer Faces site: http://java.sun.com/j2ee/javaserverfaces/index.jsp.

[14] Official Spring site: http://www.springframework.org/.

[15] R. Johnson's spring article: http://www.theserverside.com/articles/content/SpringFramework/article.html.

[16] Official Hibernate site: http://www.hibernate.org/.

[17] M. Bilgi, Y. Açıkgöz, M. E. Bodur, *Design of an Enterprise Web Architecture using Frameworks*, Senior Design Project, Fatih University, İstanbul, Turkey, June 2005.

[18] Eclipse program development platform web site: http://www.eclipse.org.

[19] Borland Together UML modeling tool web site: http://www.borland.com/together

[20] IBM Structural Analysis for Java tool web site: http://www.alphaworks.ibm.com/tech/sa4j.

[21] M. Fowler, *Refactoring, Improving the Design of Existing Code*, Addison-Wesley, 1999.