Information and Software Technology 53 (2011) 307–316

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Pattern-based framework for modularized software development and evolution robustness

Chih-Hung Chang [a,*], Chih-Wei Lu [a], Pao-Ann Hsiung [b,1]

[a] Department of Information Management, Hsiuping Institute of Technology, No. 11, Gongye Rd., Dali City, Taichung County, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, National Chung Cheng University, No. 168, University Rd., Min-Hsiung, Chia-Yi, Taiwan, ROC

ARTICLE INFO

Article history:
Available online 23 November 2010

Keywords:
Software standardization and integration
Evolution robustness
Design pattern
Framework

ABSTRACT

Context: Software development is now facing much more challenges than ever before due to the intrinsic high complexity and the increasing demands of the quick-service-ready paradigm.
Objective: As the developers are now called for more quality software systems from the industries, there is insufficient guidance from the methodologies and standards of software engineering that can provide assistance to the rapid development of qualified business software.
Method: In this work, we discuss the advantages of the pattern-based software development. We verify the benefits using a pattern-based software framework called OS2F, and a corresponding system design architecture that is intended for the rapid development of web applications.
Results: The objective of the framework/architecture is that, through software patterns, developers should be able to separate the work of system development from the business rules so as to reduce the problems caused by a developer's lack of business experiences.
Conclusion: Through a suitable pattern-based software framework, the quality of the product can thus be enhanced, software development time and cost decreased, and software evolution robustness improved.
Crown Copyright © 2010 Published by Elsevier B.V. All rights reserved.

1. New demand of modern software and old challenge to legacy system

Software systems now face much more challenges than ever before due to the intrinsic high complexity of systems and the increasing demands from organizations. Moreover, customers now demand shorter and shorter time-to-market. For instance, due to the development and maturation of WWW and Java [42] technologies in the recent years, it becomes a noticeable trend that many services are now web-based or applications are all internet-related. New requirements emerge that modern software projects, like web-based applications, can no more be satisfied by just delivering qualified packages to fulfill the first-round requirements of the contract. As a result, rapid development with rigid discipline seems to become one of the significant characteristics under the ignited spot lights.

In contrast, such a quick-down time-to-market or rapid development demand does not mean that the corresponding system's life time should be unexpectedly shortened. Once a software system is adopted and put on the fire line, it becomes a "legacy" [38]. Organizations rely on these legacy systems to provide critical information and support daily business operations. In fact, software systems are now apparently business-critical for modern industries. Companies rely more and more on the services of software, and sometimes trivial failures of the services could cause serious consequences to damage business activities and company properties. Thus any change to a legacy system would be treated chill and criticized. However, from the point that a business system is arranged onto the line and starts running, software has to be changed for necessary fixing/improvement issues or hardware/environment shifts. Software evolution is an unavoidable phase of the software life cycle. Enterprises have no choice but to invest manpower and money in software maintenance and evolution which, later proves itself, is regularly much more costly than the precedent development work.

Even many legacy systems were designed in a traditional way with an inflexible architecture, it is a hard choice for the organizations either to step in and fix it or to give up the legacy investment and start a new one. Due to the rapid development of web applications, in case of improper representations or designs, they would be very difficult to understand and maintain. That makes the activities of software maintenance or software evolution to become costly. For the necessary evolution, it seems to leave very little choices to the customers, thus encouraging users to abandon their legacy systems. Such a circumstance just likely leads the customers

* Corresponding author. Tel.: +886 4 24961123x3112; fax: +886 4 24961100x3000.
E-mail addresses: chchang@hit.edu.tw (C.-H. Chang), cwlu@hit.edu.tw (C.-W. Lu), pahsiung@cs.ccu.edu.tw (P.-A. Hsiung).
[1] Tel.: +886 5 272 0411x33119.

0950-5849/$ - see front matter Crown Copyright © 2010 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2010.11.006

to meet dead ends at both sides. Consequently, the risk along with the cost grow bigger and bigger and finally go into a vicious circle.

Probing into the causes of the problem, traditionally the objective of software analysis and design of the software development activities is to find out and match up the user requirements. Under the pressure of quick-down time-to-market, if there is no suitable toolset or methodology, developer cannot analyze and design the system carefully. Mostly programmers would only have time to finish the system with meeting the user's first-round requirements. Meanwhile, software systems usually involve teamwork. However, most systems are developed in an ad hoc manner with very limited standard enforcement, which makes the software development and evolution even more difficult and costly.

To deal with the problem, one of the most likely solutions is to extend software development lifecycle by not only meeting with the first-round user requirements, but also by taking care of the flexibility and scalability for software evolution robustness seriously from the beginning. To develop a flexible software system, adopting a practical software standardization and integration is the essential key, and accordingly, a solid and easy software architecture for low-cost evolution should be the major factor to meet the needs for a modern software project.

Regarding the modern software trend of fast-development and evolution robustness, many standardized software methods and mechanisms have been proposed and advocated to improve software productivity and to reduce the cost of maintenance. These include object-oriented (OO) technologies, standardization, and frameworks and so on. For object-oriented technologies, the property of inheritance allows software components to be reused, which obviously can reduce the cost of software development, such as Unified Modeling Language (UML) in object-oriented design (OOD) and component-based approaches in object-oriented programming (OOP). For standardization, software standards, such as design patterns (DPs), and commonly accepted standard mechanisms have been proposed and advocated to improve software productivity and reduce the high cost of software. Design patterns provide a clear concept of design structure by describing the relationships of inheritance and reference between components of the system. However, programmers usually focus on code reuse while ignoring design reuse. In short, these developing concepts may facilitate the rapid development for new internet-related projects like web applications. However, these very useful tools and concepts still lack systematic organizations.

In this paper, we introduce a software pattern-based framework, the Open Source Software Framework (OS2F), as well as, the corresponding design architecture, which is intended for rapid development of web applications. The objective of the framework/architecture is that, through software patterns, developers should be able to separate the work of development from the business rules so as to reduce the problems caused by the developer's lack of business experience. With modularization and standardization, it becomes easy for a system's evolution robustness.

This paper is organized as follows. Section 2 gives an overview about related software technologies. The pattern-based software evolution architecture will be discussed in Section 3. Section 4 shows the experiments and the verification of the proposed approach. The conclusion is given in Section 5.

2. Background and related technologies for software standardization and integration

2.1. Object-oriented technologies

Traditional software development usually leads to custom-made systems, which have the significant advantage of being optimally adapted to the user's business models and therefore fulfill most of the requirements of the in-house proprietary knowledge or practices. But it also has severe disadvantage of high cost and the limitation and dependence of individual expertise, which would cause noticeable difficulty of maintenance and evolution. Thus the necessity of object-oriented technology is revealed.

Object-oriented (OO) technology is a modern software methodology, which organizes data/code in ways that "echo" how things appear, behave, and interact with each other in the real world. OO technologies greatly influence software development and maintenance through faster development, cost saving and quality improvement [37,45].

In OO technologies, software component [38] is a portion of software design/service that can actually be deployed, and be treated as an isolated part of a system. It is different from a software object which almost would not be sold, bought, interchanged, or deployed across different projects. Another way in which component differs from object is that software components usually involve more design issues such as different implementation technologies, more complex functions, or even various languages/designs with specific flow controls, i.e. design patterns. Although "component" and "object" are usually used interchangeably, when we talk about reuse, we have to deal with them separately.

2.2. XML

XML [43] is the standard language supported by the World Wide Web Consortium (W3C). XML has many useful features, such as application neutrality (vender independence), user extensibility, the ability to represent arbitrary and complex information, a validation scheme of data structure, and is human readable.

The design goals for XML [8] are:

- XML shall be straightforwardly usable over the internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

XML is also found in certain forms when making data representations in different domain, such as ebXML [17] (Electronic Business using eXtensible Markup Language). It is a modular suite of specifications that enables enterprises to conduct business over the internet. VoiceXML [14,43] covers voice dialogs, speech synthesis, speech recognition, telephony call control for voice browsers and other requirements for interactive voice response applications, including use by people with hearing or speaking impairments. MathML [43] is a set of low-level specifications for describing mathematics as a basis for machine to machine communication. XML provides both platforms for representation and integration of development and application in the lifecycle of software engineering.

2.3. Design patterns

In various domain applications, there are certain characteristic problems inside the design activity that occur repeatedly. It is obviously important if we can make such knowledge explicit and pub-

licly available to other practitioners. The most important advantage of doing so is that inexperienced designers can gain access to a library of techniques that are immediately applicable, thus making a good design last longer. This is especially critical for the process of developing expertise in the relevant design technique. By the inheritance of object-oriented design technology, design patterns have become the major way of codifying the ways in which expert designers tackle and solve particular commonly occurring problems in design.

Design patterns (DPs) [16,18,27] are a series of familiar usages and constructions utilized throughout system design. Design patterns allow rapid coding of certain components by following certain patterns of steps. This can improve the documentation and maintenance of existing systems by providing an explicit specification of class, object interactions and their underlying intents. One of the main purposes of design patterns is to help software engineers to understand the common characteristics of software objects/components in specialized domain.

Patterns facilitate reuse of well-established solutions when known problems are encountered. They support higher abstraction levels than traditional object-oriented individual classes and instances [16]. Alexander et al. [1] discussed the pattern: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

Most design patterns contain generic structures/behaviors, which need to be adapted to different applications. Each application may involve more than one design pattern at different levels of the system. Current adaptation and integration of design patterns are implemented manually. Cooper [13] showed how to use DPs in object-oriented programming. Nguyen [34] implemented data structures, like binary search tree and linked list, using design patterns. Riehle [36] used class diagrams and role diagrams to help designers focus on the collaborations and distribution of objects using DPs.

2.4. Framework

Framework is another remarkable software technology that embraces both component reuse and design reuse. Any domain software developed on existing architecture will have high reusability if designed with a framework in mind [33]. Framework [7,31,39] is a set of abstract and concrete classes that collaborate with each other, and contain their interface definition of instances [10]. These classes are constructed as reusable components for some domain. Designers can inherit from these instances of classes to create new ones. Basically framework provides an environment to support all activities involved in software development.

However, when we attempt to create a composite of different frameworks we may run into some problems. Mattsson and Bosch [30] proposed to apply existing design patterns as a solution to these problems.

Applying framework and design patterns to develop systems have increased our work efficiency. Framework is usually used on problems within a specific domain, while design patterns are used to solve general problems. If we can find the design pattern that fits our requirement, we can apply design patterns to any application domain, making design patterns more general than framework.

Jacobsen et al. [23] proposed that if we can use patterns in the analysis, design, and implementation phases of system development, we can develop system that follow the characteristics of design patterns, which can reduce the effort of checking consistency between the phases.

In recent years, web applications as well as the related requirements have grown quickly. Many software framework technologies that can be applied to corresponding web applications were proposed, such as *Spring framework* [24]. It is an open source framework aiming to address and reduce the complexity of developing an enterprise application, which will be discussed more lately.

2.5. Struts

Struts [3] is a part of the Apache Software Foundation project, under Jakarta. Struts is a flexible Control Layer based on standard technologies such as Java Servlets, JavaBeans, ResourceBundles, and XML. The framework helps programmers create an extensible development environment for your application, based on published standards and proven design patterns.

Struts is based on a concept known as *MVC*. The term "MVC" originated with the SmallTalk Model–View–Controller framework. Under MVC, an application is seen as three distinct parts. The problem domain is represented as the *Model*. The output to the users is the *View*, and the input from the users is represented as *Controller*. Struts is composed of a group of collaboration classes/components, and the JSP tag lib servlet.

The struts framework provides several components that make up the Control Layer of a MVC-style application. These include controller components, servlets, developer-defined request handlers, and several supporting objects.

The Struts taglib component provides direct support for the View layer of an MVC application. Some of these tags access the control-layer objects. Other taglibs, including JSTL, can be used with the framework. Other presentation technologies, like Velocity Templates and XSLT can also be used with the framework.

The model layer in an MVC application is often project-specific. The framework is designed to make it easy to access the business-end of your application, but leaves that part of the programming to other products, such as JDBC, Enterprise Java Beans, etc.

2.6. Spring

Spring [24] is a layered Java/J2EE application framework, based on code published in *Expert One-on-One J2EE Design and Development* by Johnson [25].

As Johnson described [26], Spring is a powerful framework that solves many common problems in J2EE. Spring provides a consistent way of managing business objects, and encourages good practices such as programming to interfaces, rather than classes. The architectural basis of Spring is an Inversion of Control (IoC) container based around the use of JavaBean properties. However, this is only a part of the overall picture: Spring is unique in that it uses its IoC container as the basic building block in a comprehensive solution that addresses all architectural tiers.

Spring provides the following features:

- A unique data access abstraction, including a simple and productive JDBC framework that greatly improves productivity and reduces the likelihood of errors. Spring's data access architecture can also be integrated with TopLink, Hibernate, JDO and other O/R mapping solutions.
- A unique transaction management abstraction, which enables a consistent programming model over a variety of underlying transaction technologies, such as JTA or JDBC.
- An AOP framework written in standard Java, which provides declarative transaction management and other enterprise services to be applied to POJOs or – if you wish – the ability to implement your own custom aspects. This framework is power-

ful enough to enable many applications to dispense with the complexity of EJB, while enjoying key services traditionally associated with EJB.
- A powerful and flexible MVC web framework that is integrated into the overall IoC container.

Spring consists of six well-defined modules, as shown in Fig. 1. Each module (or component) of Spring can be standalone, or be implemented with one or more other modules. The module functions are described as follows.

(1) Core packaging: The Core provides the basic functions of the Spring framework. The main component of core container is BeanFactory, which is the realization of the factory model. BeanFactory uses Inversion of Control (IoC) to make application configuration, dependence norms and the actual application code separate.
(2) AOP: the AOP module provides implementation of aspect-oriented programming that fits with the AOP Alliance, such as method-interceptors and point cuts. According to the configuration manager function, Spring AOP module integrates aspect-oriented programming into the Spring framework. Therefore, it is easy to make any component in the Spring framework to support AOP. Besides, Spring AOP module can provide transaction management services so we do not need to be dependent on EJB components. The transaction management can be integrated with applications easily.
(3) DAO: The JDBC-abstraction layer provides a useful exception control framework. It can be used to manage exception handling and the output messages from databases built by different vendors. Exception control frameworks simplify error handling, and decreases the amount of exception code (for example, opening and closing the connection) that must be written. Spring-oriented JDBC DAO complies with the common abnormal level DAO framework.
(4) ORM: The ORM package provides integration layers for popular object-relational mapping APIs, including JPA [41], JDO [40], Hibernate, SQL map and iBatis [4]. All of these follow the Spring general norms and exception control frameworks.
(5) Web: The advantage of the Spring MVC Web framework is that the view layer (JPS, HTML, or PDF) can be changed easily and efficiently according to IoC and AOP functions. Furthermore, you can integrate them with your favorite Web framework.

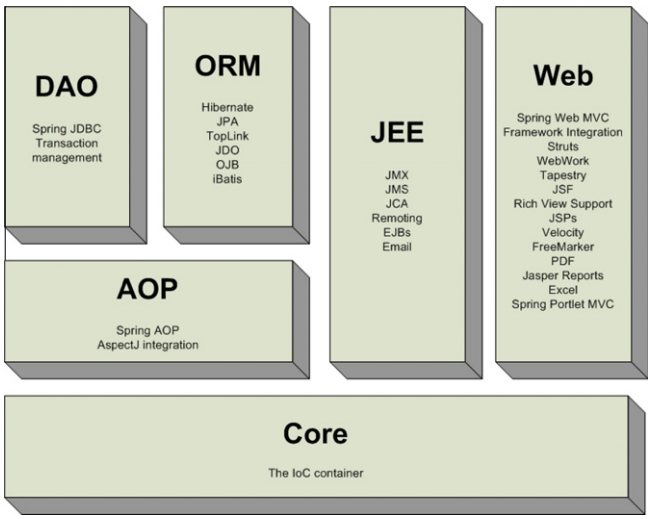

Fig. 1. The architecture of Spring framework.

(6) JEE: the JEE module contains JMX, JMS, JCS, EJB, Email and several enterprise services. It uses abstract classes to encapsulate the services. It hides the complex details of system logic and thus makes it easily usable by programmers.

2.7. Hibernate

Hibernate [21] is a solution for Object/Relational Mapping (ORM) [20]. Hibernate is object/relational persistence and query service. Hibernate lets users develop persistent classes following object-oriented idiom – including association, inheritance, polymorphism, composition, and collections. In brief, the relationship between objects and objects in Java is mapped to a relationship between tables and tables. Hibernate provides an automatic conversion solution in the program process.

The features of Hibernate [21] are as following:

- Natural programming model – Hibernate supports natural OO idiom; inheritance, polymorphism, composition and the Java collections framework.
- Support for fine-grained object models – a rich variety of mappings for collections and dependent objects.
- No build-time bytecode enhancement – there's no extra code generation or bytecode processing steps in your build procedure.
- Extreme scalability – Hibernate is extremely performant, has a dual-layer cache architecture, and may be used in a cluster.
- The query options – Hibernate addresses both sides of the problem; not only how to get objects into the database, but also how to get them out again.
- Support for "conversations" – Hibernate supports both long-lived persistence contexts, detach/reattach of objects, and takes care of optimistic locking automatically.
- Free/open source – Hibernate is licensed under the LGPL (Lesser GNU Public License).

3. Pattern-based framework — a feasible solution for software evolution robustness

Computer networks and the Internet related business activities have become the main medium for enterprise-level software deployment. The network operating environment now greatly stretches the range of scalability, from a few users' activities to millions of simultaneous interactions. Scalability means "no change in software and guarantee quality of service" as the number of users and connections increases indefinitely. A scalable system should be robust, reliable, flexible, and adaptable to changing conditions, which is a key character of modern web applications. It includes the following features:

- Openness – able to integrate with various platforms and systems.
- Extensibility – easy to extend its interface and applications.
- Maintainability – evolutionary with software/hardware technologies.
- Modularity – easy to plug in or be integrated with other components.
- Distribution – deployable to the Internet world.

As the new trend emerges, old big headstone is still unneglect. In the past, companies have already spent a lot of money on business software systems. Many of these old systems are still business-critical. These legacy systems [38] need to be maintained and evolved due to many factors, including error correction, requirements change, business rules change, structural re-organization, etc. A fundamental problem in maintaining and evolving

legacy systems is to understand the subject system [32]. Evolving legacy systems to scalable systems is expected that can greatly reduce the software cost and increase the performance of the essential systems. However, poor consideration from the beginning for evolution robustness limits the effectiveness and efficiency of changes to the legacy systems. For long term consideration, by trading-off the fast-down new systems and the legacy values, a proper framework, invoking efficiency pattern based technologies, of software development from scratch for evolution robustness is needed.

3.1. Advantages of pattern-based framework

Design patterns (DPs) [16] have integrated successful standard design practices and expert experiences into a set of components that exhibit known behaviors with better structures. Patterns facilitate reuse of well-established solutions when known problems are encountered. They support higher abstraction levels of reuse than traditional object-oriented individual classes and instances.

Design patterns usually provide a possible way to deal with non-functional requirements since they provide solutions to satisfy functional requirements, as well as, "better" solutions to meet non-functional requirements [44].

Polymorphism is one of the design properties in QMOOD [5], defined as the ability to substitute objects whose interfaces match for one another at run-time. A system with polymorphism possesses better flexibility, extensibility and effectiveness.

Fig. 2 is an example of observer design pattern. In the left of the figure is the original structure, and the right is the new structure obtained by applying the Observer pattern. The subject object calls the Observer object directly. After the transformation, two hierarchies are created, concrete observers are abstracted as a more general class Observer, and the polymorphic methods update is added. The prosperities of hierarchy, abstraction and polymorphism are all positive, indicating the Observer pattern is an improver of the three design properties.

Most polymorphism-improver design patterns are also hierarchy-improver and abstraction-improver. Hierarchy represents the generalization–specialization concept in a design, when properly applied, can enhance a system's functionality. Abstraction also represents the generalization–specification concept, but focus on the average number of classes from which a class inherits information. The Decorator pattern in Fig. 3 looks like a hierarchy-improver. Fig. 3a is the original structure and Fig. 3b is the new structure obtained by applying the Decorator pattern. After adapting the Decorator pattern, the inheritance of system class hierarchies shown in the class diagram is applied to extend functionality, which may produce an explosion of subclasses to support every combination. Such a design may have high degree of abstraction. A system with abstraction may have better extensibility and effectiveness. When we apply polymorphism to a pattern's transformation, a new hierarchy is established and the abstraction increases consequentially.

A pattern with hierarchical structure does not imply it is a hierarchy- or abstraction-improver. For example, the *Decorator* pattern looks like a hierarchy-improver. It is also to provide a flexible structure for extending responsibilities without defining too many subclasses. Decorator is actually a complexity-improver since it can reduce the number of classes and methods.

Fig. 4 illustrates the structure of the Adapter pattern. Using the adapter pattern, we can increase the reusability of the system. Programmers can use a class that class a method through an interface, and does not implement. As Gamma et al. [16] described, the Adapter pattern is applied when use the Adapter pattern when:

- User wants to use an existing class, and its interface does not match the one user's need.
- User wants to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that do not necessarily have compatible interfaces, and
- user needs to use several existing subclasses, but it is impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.

As discussed in Hsueh et al. approach [22], after system adapting pattern, the quality of system will be improved.

In most researches on pattern-based engineering, DPs play a partial supporting role during the development process. Due to a lack of global consideration, DPs at present can only offer limited assistance within system re-engineering. We suggest that DPs not only should be treated as an assistant wizard on the data-level, code-level, and design-level, but also they should be extended to the architecture-level. For instance, the Model/View/Controller (MVC) triad of classes [28] is used to build user interfaces in Smalltalk-80. The MVC model can be referred to as a high-level DP to handle the similar problem domain dealing with multiple views applications.

In short, design patterns integrate successful standard design practices and expert experiences into a set of components that exhibit known behaviors, but with better structures. Design patterns are considered to be one of the most forward-looking methods for

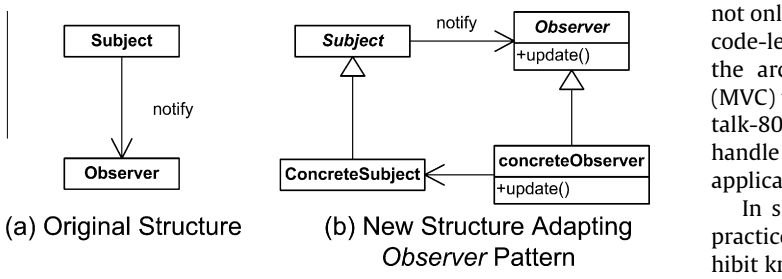

Fig. 2. An example of *Observer* pattern.

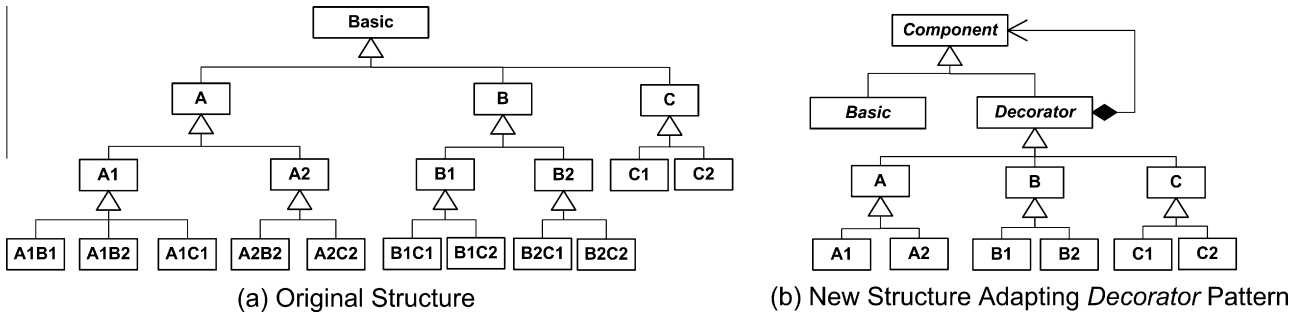

Fig. 3. An example of *Decorator* pattern.

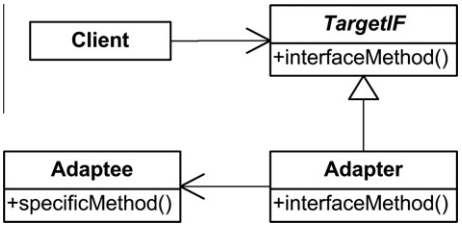

Fig. 4. The structure of Adapter pattern.

modern system analysis and design [29]. Design patterns aim to make it easier for designers to reuse well-known and successful designs and architecture from expert experience. It also helps programmers to choose design alternatives that make a system reusable and avoid alternatives that compromise reusability.

3.2. Problems of pattern-based framework

While verifying the usage of pattern-based framework, several practical problems should be addressed in advance. In our previous research [11], some problems of pattern-based re-engineering were discussed. To promote the application into pattern-based framework, problems should be clarified for pattern-based software development and evolution.

3.2.1. The representation and classification of patterns

Properly documenting, representing and classifying design patterns can improve the effectiveness of their usages. In 1992, Coad [12] stated seven patterns applied in OOA and OOD. Gamma et al. [16] collected 23 design patterns and categorized these patterns into three classes: *Creational Patterns*, *Structural Patterns* and *Behavioral Patterns*. Their works provide a very handy reference book for software designers, particularly for the beginners.

3.2.2. The retrieval of patterns

The effectiveness of retrieving design patterns is strongly related to their representation and classification. Unfortunately, not many researches have been focused on this area. The retrieval of design pattern involves the selection of a set of components, which is much more complicated than the traditional software components retrieval. This is another issue which we do not discuss in this paper.

3.2.3. The adaptation and integration of patterns

Most of design patterns contain generic structures/behavior that need to be adapted to different applications. Each application may involve more than one design pattern at different levels of the system. Each application may involve more than one design pattern at different levels of the system. Most of approaches to solve the problems of the adaptation and integration of design patterns are manual approaches.

3.2.4. The higher-level patterns

In most researches for pattern-based engineering, DPs only play a partial supporting role in the development process. It is suggested that DPs not only should be treated as an assistant wizard on the data-level, code-level, and design-level, but also should be extended to the architecture-level. For instance, the Model/View/Controller (MVC) triad of classes is used to build user interfaces in Smalltalk-80 [28]. The MVC model is a framework that contains high-level DPs that handle the similar problem domain dealing with multiple views applications.

As the advantages and problems discussed above, adapting proper architecture/framework to facilitate and integrate the usage of patterns to a key issue for system fast-development and evolution robustness. To verify the feasibility and capability of the concept, we design a prototype of the pattern-based framework with some common patterns, which is applied to the domain of web-based applications development. The design and the verification are discussed in the following section.

4. OS2F, an experiment of pattern-based framework design and verification

In this section, we should verify the benefits of pattern-based framework through a series of experimental studies. We has designed a pattern-based framework, named the Open Source Software Framework (OS2F), which is a system framework as well as a design architecture based on the concept and open source technologies described in Section 2.

4.1. System framework of OS2F

As the description in Section 2, Struts is a powerful framework to help programmers create a development environment for various functional modules extension. Hibernate is an ORM solution for database access. Spring plays a role of coordinator between user-end and system-end message exchanges. OS2F is proposed to integrate these open source technologies, as shown in Fig. 5.

Practically, OS2F Presentation Layer is responsible for the services of user interface interactions, which on the back-end are controlled by the Control Layer. Business Layer then supports the integration of the system functional logics. At last, in Data Access Layer, Hibernate ORM offers more efficient access to the database. We use Struts in Presentation Layer and Control Layer to provide *View* and *Control* features of MVC system pattern into the user interface implementation. The corresponding *Model* feature is afterward adopted with the following open source integration framework which is provided by Spring in the Business Layer. With Spring's IoC decoupling feature of the execution of a certain task from implementation, developers can finally focus more on the

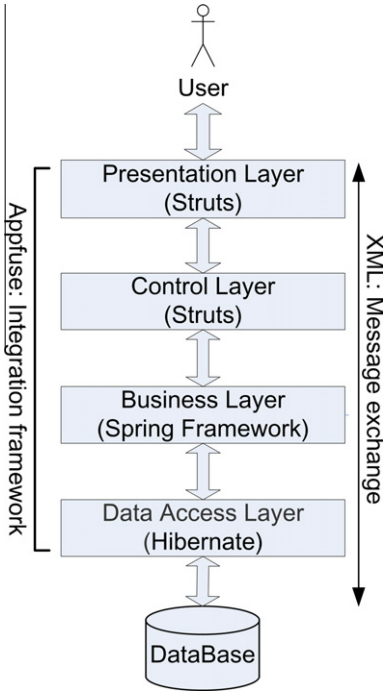

Fig. 5. The system framework of OS2F.

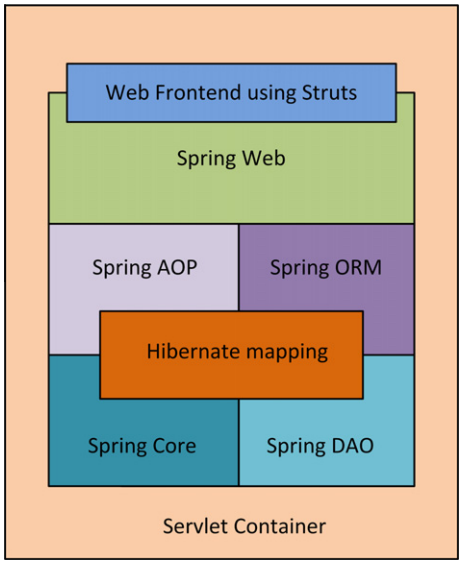

Fig. 6. System integration model of OS2F.

work of control and flow design, rather than on the complex of business rules and the variety of programming languages. We use XML as the representation for message exchange and objects mapping among different layers. For implementation practices, we employ appfuse [35] to complete the task for the open source technologies integration.

Fig. 6 shows OS2F with another viewpoint of system integration model. The Spring framework serves as core of the architecture. It consists of AOP, ORM, DAO and Core from the Spring infrastructure class libraries. It is a limpid architecture with a bridge-like feature between the front-end web page design and the corresponding back-end Hibernate information exchange to database. This framework is now implemented and operated under a Servlet Container, the most common being Apache Tomcat [2]. Other well-known Servlet containers may be adopted as needed by a user.

According to the Spring framework characters of Bean Factory and IoC (described in Section 2.6), OS2F to separate work of development from business rules. The implementation deals only with the business logic (and DAO interaction). It does not have knowledge of infrastructure services.

The Spring BeanFactory represents a class that manages the life cycle of singleton services (among other things). These services are injected with dependencies using Inversion of Control (IoC). Services that depend on external resources are "told" what those dependencies are instead of "getting" them using the Service Locator pattern. The injection of dependents is achieved either via a constructor or setter injection. The JavaBean that represents the service has a "set" method that accepts the dependant (setter injection) or it has a constructor that accepts the dependent object

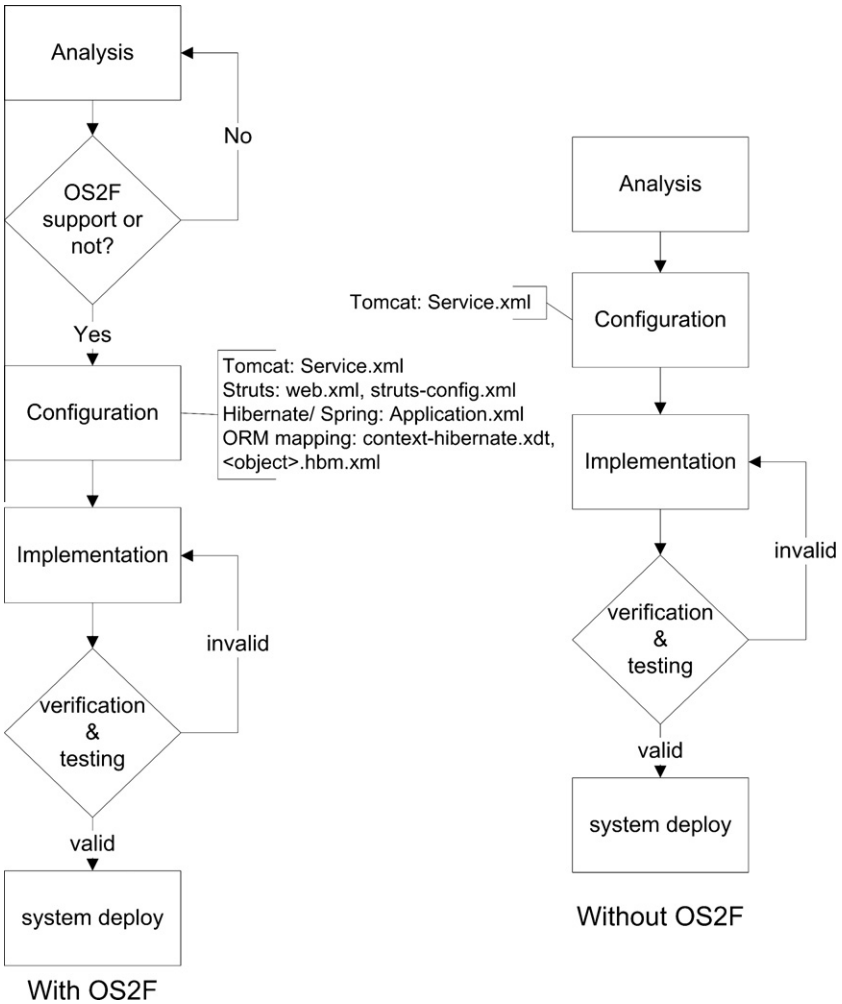

Fig. 7. System implementation process.

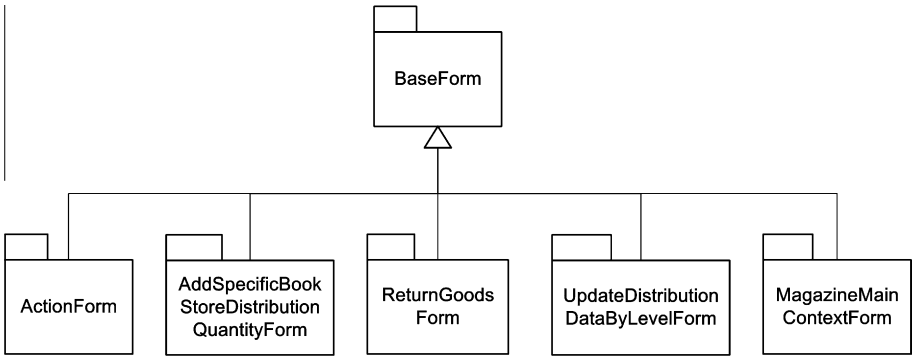

Fig. 8. MVC UI view of Project-A.

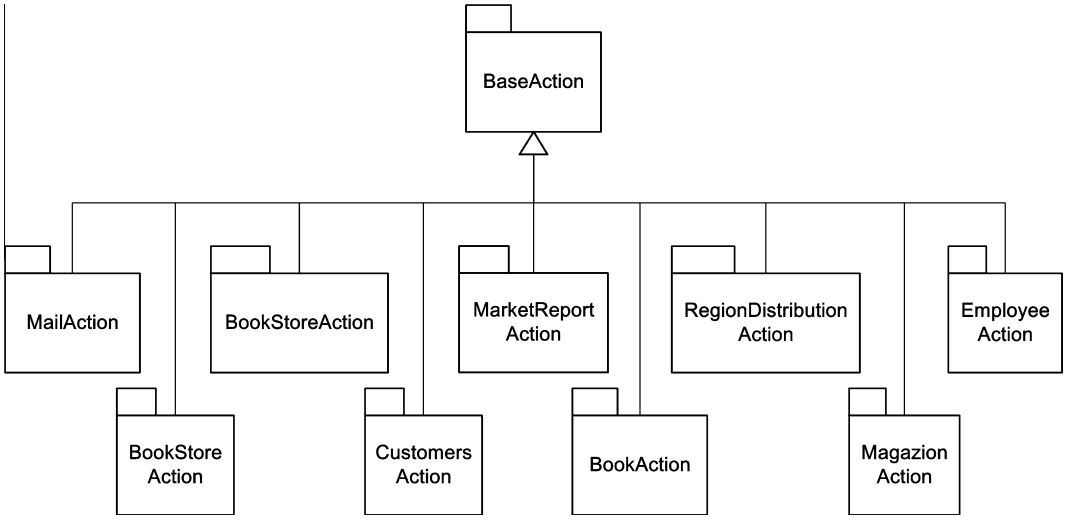

Fig. 9. MVC control view of Project-A.

Table 1
The system developing cost with/without OS2F.

	Case-A1	Case-A2	Case-B1	Case-B2
Manpower	2	2	6	6
Line of code	28,700	32,500	86,500	98,700
Days	87	98	57	88

Table 2
A comparison between Struts and JSP in the experiment.

	Struts	JSP
Collaborative development	4.5	1.5
Maintenance	4.6	1.5
Reusability	4.7	1.4
The time of study	1.5	4.3

(constructor injection). Using an XML configuration file, the service object is configured with its dependent objects using either of these two mechanisms. There can be one or more bean factories in an application. Bean factories can be overlaid, such that there is one for production but it is overlaid by the one for development. The bean factory is accessed by the Presentation Layer to access services offered by the application.

For OS2F, there are four design patterns in our approaches, *MVC* Pattern, *Observer* Pattern, *Decorator* and *Adapter* Pattern. The *MVC* pattern provides rich functionality for building robust Web Applications and it is available as a separate module in the Distribution. The *Observer* Pattern is also known as a publisher and subscriber design pattern. This pattern is useful when we have one publisher and many subscribers (one-to-many) that are interested in the publisher's state or messages. Additionally, interested subscribers have the ability to register and unregister as they please. Lastly, subscribers are notified of the publisher's messages automatically (that is, by no effort of their own). *Decorator* and *Adapter* Pattern including Spring AOP, Dynamic Proxy and Aspect oriented which those patterns by providing another way of thinking about program structure.

4.2. Experiment

In this section, we demonstrate the related experiments to validate the usability and effectiveness of OS2F. The process flows of the experiment are shown in Fig. 7. The left part of the figure shows the processes of the experimented projects which adopt OS2F; on the right it shows the processes of the projects in traditional ways.

In order to validate the usability and effectiveness of OS2F, we apply OS2F for two real business projects development and implementation. The first project is an ERP system (a book logistic system, called Project-A later). Project-A is a medium-size case consisting of 33 tables. The second project is a GLM system (a Glo-

bal Logistics Management system for a textile factory, called Project-B later). Project-B comes to a comparative larger case with about 100 tables. To reveal the difference by adopting OS2F, each project had been deployed to two separate development teams. Team-1 would take advantage of OS2F and team-2 on the other hand would do the work in that traditional way. Figs. 8 and 9 are the MVC UI view and Control view from Case-A1's works that go with Project-A with OS2F, respectively.

Both of these projects' database servers are Oracle 10 g, and the web servers are Tomcat 5. The logging service we use is log4j [19], the test case generate tool we used is JUnit API [6], and the integration development environment for both the projects is Eclipse 3 [15].

The requirements of the projects are proposed in advance. The following steps describing details of the processes for the experiment procedures recommended are also proposed to the experiment teams so they can join and be observed from the scratch as same as possible.

Step 1: Following the requirements proposed, analysis and design system using UML.
Step 2: Setup the web server configuration.
Step 3: Implement project applications with/without adopting OS2F as cases assignment.
Step 4: System verification and validation.

Table 1 shows the costs estimation of this experiment's result. Case-A1 and Case-B1 adopt OS2F to develop the systems. On the other hand Case-A2 and Case-B2 develop systems without OS2F. To understand the service/benefit of OS2F, we collected the three kinds of questionnaires of eight project members; include comparison between Struts and JSP, comparison between Hibernate and JDBC, and comparison between OS2F and traditional web architecture. Each item is given a score of 5–1, where five is excellent, four is good, three is fair, two is improvement needed, and one is poor.

Table 2 is a score comparison between Struts 1.3.8 and JSP 2.0, which refers to the work and response of the relevant UI development. Table 3 is a comparison result between Hibernate 3 and JDBC 3.0, which refers to the difference about the database access designs. Table 4 shows the comprehensive advantage/difference between the processes of OS2F and non-OS2F developments.

In these experiments, we use OS2F, as well as, non-OS2F approaches to implement web applications. The performance in Data Access Layer with Hibernate technique is obviously much easier than the work with JDBC/ODBC. Meanwhile, according the benefit of Hibernate, the source code and database are separated by object and HQL (Hibernate Query Language) [9]. It is not surprising that if programmers need to modify database structure later, they only need to modify hibernate mapping configure of corresponding hbm.xml, the source code needs not be modify and it makes software evolution much easier and more efficient. The similar scenarios happen in Presentation Layer, Control Layer and Business Layers are similar to data access layer. According to Struts and Spring, the system structure is more modularized. These properties make the target web application easier to collaborative development, maintenance, and integration. The development/maintenance cost can be therefore reduced.

Table 3
A comparison between Hibernate 3 and JDBC 3.0 in the experiment.

	Hibernate	JDBC
Resource cost	2.5	3.5
Performance	4.4	1.6
Development time	4.7	2.5
The time of study	1.5	4.4

Table 4
A comparison between OS2F and non-OS2F developments.

	OS2F	Non-OS2F
Scalability	4.7	1.6
Modularization	4.6	1.7
Collaborative development	4.7	1.5
Maintenance	4.6	1.3
Integration	4.5	1.5
Flexibility	4.8	1.5
Performance	3.9	1.4
Reusability	4.7	1.4
Internationalization	4.8	1.6
Database resource cost	2.5	3.8
Data access performance	4.0	2.2

5. Conclusion

Web applications of late have become high-spotted. Responding to this need, a host of solutions are emerging, such as Struts, Spring framework, and Hibernate, but most of them only benefit users partially. In this article we discuss our proposal and the corresponding experiences with the pattern-based framework OS2F which integrates Struts, Spring framework, and Hibernate, as well as, how OS2F can be used to make a web application more maintainable and better performing.

In this paper, we discuss the advantages of pattern-based framework, and we use a pattern-based framework design to verify the benefits of pattern. Software development using pattern-based layers and modular architecture is helpful to improve system flexibility and which is also easy to software evolution. In our experiments, system development using pattern, that improves the issues of system abstraction, hierarchy, and complexity. We integrate various specific open source applications and frameworks into a moderate framework OS2F, and a corresponding 4-layers software architecture, to show how can a pattern-based architecture improve software quality and system development/evolution efficiency. As the experiment in Section 4, pattern-based framework could really simplify the management of software hierarchy and therefore decrease the cost of the software. We also develop a GLM system and integrate it with a legacy client–server system application to verify the feasibility of services integration evolution. At the meantime, thanks to the clear-defined layers, all objects could be created with the same rules; this standardization is helpful while involving teamwork and collaborative development.

In the cases, we have also experimented on the usability and estimated the effectiveness of OS2F. While developing a large web application similar to our cases with OS2F architecture, we could accomplish a well-organized system and reduce overall cost, especially the upcoming evolution processes. Besides, the quality of software development is guaranteed due to the rules and guidelines from the promised pattern-based frameworks.

In our up going works, a comprehensive toolset with the pattern-based framework based on the experiment of OS2F should be designed to get more verification in practical and sized project.

Acknowledgment

This research was supported in part by National Science Council, Taiwan ROC, under Grant No. NSC97-2218-E-164-001.

References

[1] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S.A. Angel, A pattern language, Oxford University Press, New York, 1977.
[2] Apache Software Foundation, Apache Tomcat, Apache Software Foundation. <http://tomcat.apache.org/> (accessed 12.05.07).

[3] Apache Software Foundation, Apache Struts, Apache Software Foundation. <http://struts.apache.org/1.3.9/userGuide/index.html> (accessed 15.07.07).
[4] Apache Software Foundation, iBATIS, Apache Software Foundation. <http://ibatis.apache.org/> (accessed 20.04.09).
[5] J. Bansiya, C.G. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering 28 (1) (2002) 4–17.
[6] K. Beck, E. Gamma, D. Saff, JUnit 4.1. <http://www.junit.org/index.htm> (accessed 12.03.07).
[7] B.H.L. Betlem, R.M. van Aggele, J. Bosch, J.E. Rijnsdorp, An object-oriented framework for process operation, Technical Report, Department of Chemical Technology, University of Twente, 1995.
[8] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan, Extensible Markup Language (XML) 1.1, second ed., W3C, September 2006. <http://www.w3.org/TR/2006/REC-xml11-20060816/> (accessed 12.04.07).
[9] H. Chauhan, Introducing HQL: the object-oriented query language from Hibernate, WebMediaBrands. <http://www.developer.com/java/ent/article.php/3322131> (accessed 20.04.09).
[10] D.J. Chen, T.K. Chen, An experimental study of using reusable software design frameworks to achieve software reuse, Journal of Object-Oriented Programming 7 (2) (1994) 56–67.
[11] C.W. Chu, C.W. Lu, C.H. Chang, Y.C. Chung, Pattern-based Re-engineering, Handbook on Software Engineering and Knowledge Engineering, vol. I, World Scientific Publishing, River Edge, NJ, 2001. pp. 767–786.
[12] P. Coad, Object-oriented patterns, Communication of ACM 35 (9) (1992) 152–159.
[13] J.W. Cooper, Using design patterns, Communication of ACM 41 (6) (1998) 65–68.
[14] J. Ding, Y. Huang, C.W. Chu, Video database techniques and video-on-demand, Handbook of Distributed Multimedia Databases: Techniques and Application, The Idea Group Publishing, Hershey, PA, 2001.
[15] Eclipse Foundation, Eclipse, The Eclipse Foundation. <http://www.eclipse.org/> (accessed 12.04.07).
[16] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading MA, 1995.
[17] R.J. Glushko, J.M. Tenenbaum, B. Meltzer, An XML framework for agent-based e-commerce, Communications of the ACM 42 (3) (1999) 106–114.
[18] M. Grand, Patterns in Java, second ed., vol. 1, Wiley Publishing, Indianapolis, 2002.
[19] C. Gülcü, Short introduction to log4j, Apache Software Foundation. <http://logging.apache.org/log4j/1.2/manual.html> (accessed 20.04.09).
[20] T. Halpin, Object Role Modeling (ORM). <http://www.orm.net/> (accessed 20.04.09).
[21] Hibernate, Red Hat Middleware. <http://www.hibernate.org/> (accessed 12.04.07).
[22] N.L. Hsueh, P.H. Chu, C.W. Chu, A quantitative approach for evaluating the quality of design patterns, Journal of Systems and Software 81 (2008) 1430–1439.
[23] E.E. Jacobsen, B.B. Kristensen, P. Nowack, Patterns in the analysis, design and implementation of frameworks, in: Proceedings of the 21st International Computer Software and Applications Conference, IEEE Computer Society Press, Silver Spring, MD, 1997, p. 344.
[24] R. Johnson, et al., The spring framework – reference documentation, Interface21. <http://www.springframework.org/docs/reference/index.html> (accessed 30.04.07).
[25] R. Johnson, Expert One-on-One J2EE Design and Development, Wrox Press, Hoboken, NJ, 2002.
[26] R. Johnson, Introduction to the spring framework, May 2005. <http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework> (accessed 02.10.07).
[27] R.E. Johnson, B. Foote, Designing reusable class, Journal of Object-Oriented Programming 1 (2) (1988) 22–35.
[28] G.E. Krasner, S.T. Pope, A cookbook for using the model-view controller user interface paradigm in smalltalk-80, Journal of Object-Oriented Programming 1 (3) (1988) 26–49.
[29] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice-Hall International, Englewood Cliffs NJ, 1997.
[30] M. Mattsson, J. Bosch, Framework composition: problems, causes and solutions, in: Proceedings Technology of Object-Oriented Languages and Systems, Interactive Software Engineering, 1997, pp. 203–214.
[31] M. Mattsson, Object-oriented frameworks, Licentiate Thesis, Department of Computer Science, Lund University, 1996.
[32] A.V. Mayrhauser, A.M. Vans, Program understanding: models and experiments, Advances in Computers 40 (1995) 1–38.
[33] S. Moser, O. Nierstrasz, The effect of object-oriented frameworks on developer productivity, IEEE Computer 29 (9) (1996) 45–51.
[34] D. Nguyen, Design patterns for data structure, in: Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education, ACM Press, 1998, pp. 336–340.
[35] M. Raible, Appfuse Confluence. <http://appfuse.org/display/APF/Home> (accessed 21.08.08).
[36] D. Riehle, Describing and composing patterns using role diagrams, in: Proceedings of the 1996 Ubilab Conference, 1996, pp.137–152.
[37] D.C. Rine, Supporting reuse with object technology, IEEE Computer 30 (10) (1997) 43–45.
[38] I. Sommerville, Software Engineering, sixth ed., Addison-Wesley Publishing Co., Inc., Wokingham, England, 2001.
[39] S. Sparks, K. Benner, C. Faris, Managing object-oriented framework reuse, IEEE Computer 29 (9) (1996) 52–61.
[40] Sun microsystems, Java Data Objects (JDO), Sun Microsystems. <http://java.sun.com/jdo/> (accessed 20.04.09).
[41] Sun Microsystems, Java Persistence API, Sun Microsystems. <http://java.sun.com/javaee/technologies/persistence.jsp> (accessed 20.04.09).
[42] Sun Microsystems, The Source for Java Developers, Sun Developer Network, Sun Microsystems. <http://java.sun.com> (accessed 12.08.07).
[43] W3C, Extensible Markup Language (XML), World Wide Web Consortium. <http://www.w3.org/xml> (accessed 01.12.06).
[44] T. Winn, P. Calder, Is this a pattern, IEEE Software 19 (1) (2002) 59–66.
[45] R.J. Wirfs-Brock, R.E. Johnson, Surveying current research in object-oriented design, Communications of the ACM 33 (9) (1990) 105–124.