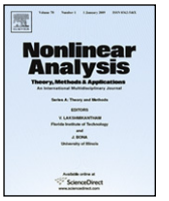
Nonlinear Analysis 71 (2009) e1633–e1637

Contents lists available at ScienceDirect

Nonlinear Analysis

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



Defining methodologies for developing J2EE web-based information systems

Askar S. Boranbayev *

Eurasian National University, 5 Munaitpassov Street, Astana, 010008, Kazakhstan

ARTICLE INFO

Keywords:
J2EE
Framework
Web development

ABSTRACT

This paper describes the concepts behind the developed framework for Java-based projects and describes how it can be used for IT projects. The developed framework was created because many common design and development tasks are being repeated in different ways, and are not always consistent with best practices. We have identified common application concerns and delivered design patterns and tools that represent effective solutions. The developed framework delivers: (1) an application development stack starting from the user interface to data integration; (2) an architecture, infrastructure and associated techniques for enhancing and leveraging some other frameworks. The proposed architecture defines a development methodology, which is designed to assist in custom development projects.

1. Introduction

Software tools and packages for the nonlinear analysis of complex spatial–dynamic systems increasingly use Web-based networking platforms for the implementation of their user interface, scientific analysis, distribution of simulation results, and information exchange among scientists. The real-time Web-based access to nonlinear analysis simulation software becomes a critical part of many applied systems. The intensive technological change in networking hardware and software [1] provides more freedom of choices than in the past [2]. Therefore, the rational selection and development of the Web platform is of increasing importance for the whole area of nonlinear analysis and its numerous applications. The current stage of Web development is characterized by the emergence of a significant number of open source frameworks. Frameworks shift the focus of Web development to a higher level, allowing the reuse of basic functionality and thus increasing the productivity of development.

In some cases, open source frameworks do not provide a solution to common problems. For this reason, developers build their own development framework on top of an open source framework. The aim of this paper is to describe a developed Java-based framework that leverages open source frameworks and assists in developing Web-based applications. By analyzing some existing open source frameworks, this paper presents a new architecture, infrastructure and associated techniques for enhancing and leveraging some other frameworks. The proposed architecture defines its own development methodology, which is designed to assist in custom development projects and integration projects.

There are common application design concerns which are often used across projects. Even across unique functional requirements, there are commonly occurring patterns of use cases, which lend themselves to design and development reuse. This paper describes a "customized" framework, which had been developed in an effort to identify such common application concerns and identify design patterns that can be used by the developers. This framework, which we will refer to as the developed XYZ framework, provides a set of patterns and tools that were built on industry best practices, tailored to common application concerns. It provides an application development stack, from presentation to integration layers.

* Corresponding address: 5 Munaitpassov Street, 010008 Astana, Kazakhstan. Tel.: +7 7172342105.
E-mail address: aboranba@yahoo.com.

0362-546X/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.na.2009.02.002

This paper articulates these application concerns and the patterns, tools and industry best practices. The developed XYZ framework can be customized to various projects' needs. It was developed and configured based on various frameworks and tools such as Struts, Spring, Hibernate and JUnit.

2. Major technologies of the developed framework

2.1. Layers and separation of code and configuration

Web applications have various design concerns such as presentation, business logic, data access and security. A separation of design concerns into distinct code layers has several advantages such as: ease of maintenance, the ability to implement good design patterns, and the ability to select specialized tools and techniques for specific concerns. Separating a project into layers can result in dependencies between those layers. For example, a single-use case involving simple data entry and inquiry usually must integrate presentation, business logic and data access together to deliver required functionality [3]. Therefore, there must be a well defined strategy to manage the dependency relationships. The developed XYZ framework combines design patterns, reusable code and configuration files to make this as easy as possible. This framework uses Spring's Inversion of Control to manage dependencies. The Spring Framework [4] provides a way to tie together the objects that make up an application. It accomplishes this goal with the Spring Application Context, which is a strategy for managing dependencies between objects. Spring uses *dependency injection* and *method interception* techniques described below.

The code that we write is dependent on the objects it uses. It is responsible for creating these objects. This may result in tight coupling, but we would prefer that our code be loosely coupled. Dependency injection is a technique which helps us to accomplish this. Dependency injection is a form of Inversion of Control (IoC) [5]. When applications use dependency injection, the code becomes much cleaner and easier to follow. It is also loosely coupled, allowing for easier configuration and testing. The XYZ framework uses several Spring application context files to define dependencies between the layers. Method interception is a concept of Aspect Oriented Programming (AOP) [6]. Spring AOP implements method interception through JDK dynamic proxies. The XYZ framework uses Spring AOP to manage concerns such as transaction management and performance monitoring.

The developed XYZ framework consists of two distinct parts: *code* and *configuration*. Code resides in a particular application layer and focuses on a particular piece of the application solution. This could be interacting with a database, or presenting data to the screen. Configuration glues the various layers of the application together. Separating configuration from code allows us to manage configuration independently, giving us the flexibility of applying different configurations to the same code base. For example, a Data Access Objects (DAO) implementation knows that it is using JDBC to connect to a database through a data source, but it does not know anything about the implementation of that data source. It may come from a Java Naming and Directory Interface (JNDI) context or be derived from a driver manager. It may point to the remote database or a local database. Regardless of where the data source comes from, the DAO implementation will operate on the data source in the same manner. Likewise, a Service object may depend on a DAO, but it does not know whether the DAO is implemented via Hibernate, straight JDBC, or a Web service. The service object interacts with the DAO in the same manner, regardless of the DAO's implementation.

Spring gives us a way to manage our application's entire configuration through a Spring application context, defined by a set of XML files. We could define the application context in one file. However, by defining it in groups of smaller files, we can simplify configuration management. A logical set of such application context files which forms a complete application configuration is called a configuration set.

During the development of Java-based enterprise applications the standard configuration is where a framework's configuration set uses external resources such as data sources and JNDI resources. This type of configuration sometimes can create problems with: (1) An incomplete database that has not yet been loaded. Developers may want to test the display of certain types of data, but if the underlying database has not yet been completed, they will not be able to do this. (2) Services or DAOs that may not have been developed yet. Integrating with unfinished services or DAOs may halt development.

These issues decrease productivity. The developed XYZ framework has separated its configuration from its code, we can use an alternate configuration set targeted specifically towards development. This relieves us from worrying about the availability of external systems, which are irrelevant to solving immediate development needs.

The developed XYZ framework defines two configuration sets: *default* and *standalone*. We can also customize our application by adding additional configuration sets based on our project needs. The *default* configuration set connects to the development database using the DataSource defined in JNDI. It uses fully developed application services and DAOs. The *standalone* configuration set is the most flexible environment for development. This configuration set: (1) connects to either a locally installed database or the development database using a DriverManagerDataSource; (2) uses Spring's DataSourceTransactionManager for local transaction management; (3) uses fully developed application Services and DAOs; and (4) fully wired Spring application contexts can be run and tested entirely outside of the application server.

The developed XYZ framework is configured by its application context. The application context may be defined in one or more XML files. A configuration set is a set of XML files that define one application context. The configuration set consists of two parts: service and Web. The service part defines services, DAOs, and resources for the service and integration layers. The Web part defines components for the presentation layer. A configuration set cannot be complete without both of these parts.

The developed XYZ framework configuration sets are grouped together by what Spring calls a bean reference context defined in the files *beanRefContext.xml* and *applicationContextMapping.properties*. The *beanRefContext.xml* file defines the service part of all configuration sets. This file is located in the src/config directory of the service project. Application context files shared between configuration sets are also located in this directory. In addition, each configuration set has its own subdirectory, which contains files specific to it. Services and DAOs, for instance, are shared between configuration sets, while supporting services (like data sources) belong in the subdirectories. XML files define the Spring beans in this application by using the ⟨bean⟩ tag. A Spring bean is a Java object created and initialized by the application context.

2.2. Classes and dependencies

Using the developed XYZ framework, the following code and configuration artifacts will be typically required to develop a user interface screen: (a) Action, ActionForm classes and validation.xml entries; (b) service interface and implementation class; (c) DAO interface and implementation class; (d) dependency management between all of the above. When starting development for a use case, we must be aware of the need of all these classes and their dependencies upfront.

2.3. Testing technology

Testing should be an integral part of the development process. For applications built using the developed XYZ framework, *unit testing* means testing methods of a single class in the service or integration layer. Presentation layer artifacts (Action classes) are not taken up for unit testing. The purpose of this test is to ensure that the behavior encapsulated by the class works as expected when testing integration with other components. Unit tests in applications, developed using the developed XYZ framework, are based on the JUnit framework [7]. Unlike unit tests, *integration tests* do require code dependencies to be available. The purpose of this test is to ensure that the integration between different classes (developed by different developers) works as expected. During the *Functional Testing* process, the focus is on testing functionality of the application by using data to depict different scenarios. Functional testing typically involves testing classes in the Service layer with different data. It can also be performed by testing the user interface layer and by using real dependencies.

In order to perform different types of testing, the application being developed must be testable. Let us list some of the basic characteristics of a testable application. (1) Ease of developing unit and integration tests. We should be able to unit test without necessarily using data sources, or queues. Also, we should be able to mock dependencies of code under test. (2) Ease of simulating various test scenarios for functional testing. (3) Ease of re-running all tests repeatedly over the life cycle of the application. (4) Clean separation of testing code from application code.

A well structured application that separates design concerns such as presentation, services and data access is important for designing testable applications. Application coding starts with getters, setters, variables etc., which are then integrated to provide the required solution. Unit tests are a fundamental building block for any testing approach. The developed XYZ framework's design facilitates development of testable applications by: providing testing template classes that help to create unit tests; and also by enabling easy configuration of application to adapt to testing needs. Unit tests can be run like any JUnit test. The specially developed default "build script" provides a task to run unit tests. This task can be called when generating an EAR file for deployment or can be run separately.

2.4. Web presentation design

The developed XYZ framework uses the Struts framework concepts and JavaScript to implement presentation concerns and provides additional features that can be extended for use in projects. When using the Struts framework for development, at first we set up the Action Servlet in web.xml; then we set up configuration, action mappings, form beans and local forwards in the struts-config.xml; and finally we set up validation rules in validation.xml.

This approach has been changed in applications that were built using the developed XYZ framework, such that developers must not directly edit the struts-config.xml or validation.xml. Instead we specify this information as XDoclet annotations in the Action and ActionForm classes directly. This information is then transferred into the struts-config.xml and validation.xml by running the Ant script.

There are two types of validations that are required: *data format validation* and *business logic validation*. Data format validations are best done in the presentation layer, while business logic validations are best done in the services layer. Business logic validation errors, which occur in the service layer, should be handled by throwing custom exceptions.

The following are design goals for the presentation layer: (1) There is only one Action class ascending one ActionForm per JSP. A single Web page must all be handled in a single Action class. (2) Dependencies and validation rules are specified using XDoclet annotations. (3) Developers should avoid or minimize the use of session objects, because it hinders scalability.

The XYZ framework provides a default template Action class that contains a solution to the above mentioned design goals for this layer. The following are typical code artifacts required for developing a Web page: (1) Create a new JSP with a default hidden field called "actionType" for use in handling user actions that are expected to occur on the page. (2) Create a new Action class that extends this template Action class. We must specify dependencies on ActionForm specific methods that handle user actions represented by values in hidden field "actionType". After that we can declare permissions that are

required in order to access this Action class. This is done in the Spring configuration files. (3) Create a new ActionForm class and specify validation rules as required using XDoclet annotations.

Once the JSP, Action and ActionForm are created, it is necessary to run the Ant script to regenerate the "struts-config.xml" file.

2.5. Database access

Applications built with the developed XYZ framework support the use of direct JDBC and Hibernate framework to persist in sending data to relational databases. The application is configured with the necessary Spring context files. DAOs that use direct JDBC must extend from Spring framework's JdbcDaoSupport.java class. Similarly, DAOs that use Hibernate must extend from Spring framework's HibernateDaoSupport.java class.

2.6. Configuration through annotation

The developed XYZ framework uses the Spring framework to maintain dependencies between code artifacts in applications. Some of these dependencies (e.g. between Action and ActionForm) are configured in "struts-config.xml", while some others (e.g. between Service and DAO) are configured in Spring application context files (applicationContext.xml). In a team environment these configuration files are shared by developers. That is why version conflicts on these configuration files might occur. The developed XYZ framework provides a new efficient approach that uses special annotations to specify all of these dependencies. By using these annotations, configurations become simpler and conflicting changes to configuration artifacts are avoided.

3. Services of the developed framework

The developed XYZ framework promotes the use of Plain-Old-Java-Objects (POJOs) to implement business logic. Business logic must be declared as interfaces. All service implementations must implement one or more business interfaces. It is recommended that the service layer throws custom business exceptions when there are business rule validation errors. The developed XYZ framework adopts a declarative transaction management approach based on the Spring framework [8]. This is implemented using Aspect Oriented Programming (AOP).

The developed XYZ framework promotes a good practice of separating the deployment interface (contract with service consumers) from the service interface (contract representing application business logic). *Deployment interface* is a Java interface that represents the service exposed in the WSDL. The class that implements this interface always must delegate requests to the class that implements the service interface. This ensures that all business logic is maintained at one place in the correct layer. *Service interface* is a Java interface that represents business logic. In most cases the deployment interface will contain a subset of methods from the service interface.

Apache Axis 1.2.4 Web service framework is the current standard for Web services. When developing Web services, there are two distinct approaches [9]: *contract first* and *contract last*. The difference between *contract first* and *contract last* approaches lies in whether the WSDL is created first or whether it is generated from code.

Contract first approach is a good practice for developing Web services, when the service consumer and provider are external vendors who might implement the Web service using different technologies from each other (they may use.NET instead of Java) [10].

4. Middle layer integration

There are several techniques for integrating with the external resources such as databases and Web service. The developed XYZ framework uses these techniques in a logical layer called the 'integration' layer. The design goals for this layer are: (1) Access to databases using JDBC or Hibernate should be encapsulated in Data Access Objects (DAO). (2) Consuming Web services should be as simple as possible. (3) All conversion of external data formats into application domain objects should be restricted to this layer. (4) Unit testing classes in this layer should be simple and easy to do.

The developed XYZ framework supports the use of Hibernate and direct JDBC calls to access relational databases. It is recommended to use Spring framework's template classes: JdbcTemplate and HibernateTemplate. When using direct JDBC to access relational databases, it is recommended that the application's DAOs extend from Spring framework's JdbcDaoSupport. The JdbcTemplate class manages resources used in accessing databases (for example PreparedStatement). The developed XYZ framework inserts the data source into DAOs using application configuration files. When using Hibernate to access relational databases, the Hibernate SessionFactory is injected into the application DAOs through application configuration files.

5. Development life cycle

The developed XYZ framework's emphasis on structure enables clear definition of roles and their interactions in a development team. Three roles are described below. An interaction between such roles is critical to the successful completion of application development. (a) *Front end developers* are focused on developing JSPs, Action/ActionForm classes and exposing Web services. (b) *Services developers* are focused on developing application services and integrating the different parts of the application used by these services. (c) *Integration developers* are focused on developing integration artifacts such as DAOs, or consuming Web services.

One of the basic problems in development is how to develop and integrate code when its dependent components are not ready or available. The developed XYZ framework resolves this problem by providing a structure to declaratively inject "mock objects" and also replace "mock objects" with real objects as the development life cycle progresses. This is possible due to the ability to configure our application using different configuration sets. The framework enables teams to make testing an integral part of the development process. This is made possible by writing and running JUnit tests. The framework is focused on testing application services and their dependencies. Applications are deployed in a single Enterprise Archive (EAR) file. The developed Ant script generates this EAR file and can be run manually or periodically through a scheduler. It is recommended to run all the JUnit tests before creating the EAR for deployment.

6. Conclusion

In this paper, the author has provided an overview of the developed J2EE framework. The author has addressed important architecture topics, technologies and development steps that one should consider in a J2EE project. The information is taken from real-world experiences, and is intended to help developers build J2EE systems, and design their own custom frameworks. This, however, is just the tip of the iceberg, as no short paper could describe in detail J2EE's potential impact on scientific and enterprise applications and, especially, on Web-based simulation software for nonlinear analysis.

Acknowledgements

The author is grateful to his Ph.D. advisor Professor Yuri Yatsenko (from Houston Baptist University) for his advice.

References

[1] N. Hritonenko, Yu. Yatsenko, Creative destruction of computing systems: Analysis and modeling, Journal of Supercomputing 38 (2006) 143–154.
[2] Yu. Yatsenko, N. Hritonenko, Network economics and optimal replacement of age-structured IT capital, Mathematical Methods of Operations Research 65 (2007) 483–497.
[3] A.S. Boranbayev, Reference architecture for web applications, Reports of the National Academy of Science of the Republic of Kazakhstan 5 (2007) 18–26.
[4] The Spring Framework official website: http://www.springframework.org/.
[5] Martin Fowler discusses details of the dependency injection pattern and how spring injects dependencies: http://www.martinfowler.com/.
[6] Spring reference manual's chapter on aspect oriented programming with spring: http://static.springframework.org/spring/docs/2.0.x/reference/aop.html#aop-introduction-defn.
[7] JUnit framework: http://www.junit.org/.
[8] Spring Framework official website - Chapter 9, Transaction management: http://static.springframework.org/spring/docs/2.0.x/reference/transaction.html.
[9] A.S. Boranbayev, Optimal methods for java web services, News of the National Academy of Science of the Republic of Kazakhstan 5 (2007) 38–43.
[10] Spring Framework official website. Chapter 2: http://static.springframework.org/spring-ws/sites/1.5/reference/html/why-contract-first.html.