Complete Separation of the 3 Tiers

Divide and Conquer

Oliver Rutishauser

Complete Separation of the 3 Tiers

Divide and Conquer

Oliver Rutishauser

This book is for sale at http://leanpub.com/summer-mvc

This version was published on 2013-03-16

Tweet This Book!

Please help Oliver Rutishauser by spreading the word about this book on Twitter!

The suggested hashtag for this book is #Object-oriented web application development, web application architecture and framework, three-tiered application, tier-to-tier interface, interface and data definitions, API, OOP, Java, Java interface, composite beans, delivered beans, common types, fron.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#Object-orientedwebapplicationdevelopment,webapplicationarchitectureandframework
three-tieredapplication,tier-to-tierinterface,interfaceanddatadefinitions,API,OOP,Java,Javainterface,
compositebeans,deliveredbeans,commontypes,fron

Contents

ABSTRACT

Most Java applications, including web based ones, follow the 3-tier architecture. Although Java provides standard tools for tier-to-tier interfaces, the separation of the tiers is usually not perfect. E.g. the database interface, JDBC, assumes that SQL statements are issued from the application server. Similarly, in web based Java applications, HTML code is assumed to be produced by servlets. In terms of syntax, this turns Java source code into mixtures of languages: Java and SQL, Java and HTML. These language mixtures are difficult to read, modify, and maintain.

In this paper we examine criteria and methods to achieve a good separation of the 3 tiers and propose a technique to provide a clean separation. Our proposed technique requires an explicit Interface and Data Definitions. These allow isolation of the back-end, application server, and front-end development. The Definitions also enable application design in terms of aggregated data structures. As a result significant amounts of auxiliary code can be generated from the Definitions, enabling the developers to concentrate on the business logic. By and large the proposed approach greatly facilitates development and maintenance, and overall improves the quality of the products.

DESIGN

Having analyzed common problems and popular solutions we offer the following 3-tier architecture. The Front End is a web based client or a stand alone Java client. The Business Logic is implemented on a web server, for instance as a number of servlets or C++ CGI executables. The Back End is an SQL database, such as MySQL or Oracle. We will use XML over HTTP as a connection between the application server and the front end; and JDBC as a connection between the application server and the database. These are our preferences; so far â€“ nothing new.

But in addition we will completely separate the three tiers by defining the front and back interfaces in a concise and language/layer independent form of XML clauses. These API definitions become the complete and final agreements between the GUI, server, and database. Ideally, such APIs will mean the same things for the developers of different layers, but in different terms. Thus, a web client developer is interested in what CGI parameters the web server expects and what XML data the server will issue back; but he is not interested in the internal representation of these parameters or data on the server side. Likewise, the database developer is interested in which stored procedures with which signatures he needs to implement, but he is hardly interested in which wrapper classes will be used to access his stored procedures and manage connections. Finally the application server developer will find convenient to use beans and wrapper classes with setter and getter methods for processing aggregated data, issuing screens and accessing stored procedures.

We claim our tool can support the interfaces as described above. The implementation is based on the idea that the API definitions can be used to generate servlet stubs and skeletons, screen classes, stored procedure wrapper classes, and aggregated data beans used throughout the whole application. The generated auxiliary classes will guarantee that the front end, application server, and the back end are consistent and can work together.

THE SUMMER MVC

The Summer MVC tool is provided as a main Ant script, a code generator implemented as a number of XSL files, and a library of auxiliary run time Java classes. To use the tool, the application data structures and tier interfaces should be defined in application API files. Then the generator is invoked to generate application common beans, screens and database wrapper classes. Next, the generated classes are compiled and packed into a Java archive. After that the beans and interface classes become available for the application server Java project.

All the steps, including compilation of Java project and deployment of project classes along with a library of the generated common beans, interfaces, and auxiliary Java classes, can be done by executing just one Ant target. The tool can be easily integrated into Java development IDE, such as JDeveloper or Eclipse.

1. A Simple Example

Regrettably, the limited size of this paper does not allow producing a detailed example. Fortunately, this is not necessary either, because we have already discussed all the components in the previous sections. Here we will just outline the recommended development of an application. It starts with defining aggregated application data structures, such as Customer, Personal Information and Address (see Section 3.1.) Then the interface definition should be created to enumerate and define all possible requests from the GUI to the application server and from the application server to the database stored procedures (see Sections 3.7, 3.13, 3.15, and 3.11.) A simple application can process just one request, e.g. to find a customer. The GUI passes the desired name or phone number to the application server as described in section 3.14. The trivial request handler involves no specific business logic, but just forwards the input parameters to the corresponding stored procedure and sends the result back. Interface definitions for this case include one request handler, and one stored procedure signature.

From this point, the GUI, application server, and database developers can work independently. A standalone GUI can be written in Java and will be able to call the defined request handlers on the server side as described in Section 3.9 and 3.10. A web-based GUI can be entirely written in a markup language as described in "OOML: Structured Approach to Web Development" by Francisco and Sadikov (2008.) In both cases the application server is realized as a set of generated servlets corresponding to the defined request handlers. The business logic is implemented in Java and will reside in the middle tier, see Section 3.8. The application server will use a set of convenient wrapper classes to access the stored procedures. The stored procedures involved will be implemented in SQL, as described in Section 3.11.

2. Projects

We have successfully applied the Summer MVC to a number of existing projects mostly involved in self and agent based customer care web applications. The tool allowed us to make the projects more understandable, and the code more readable. This significantly reduced development and maintenance time and cost.

3. Evolution

As stated above the Summer Framework does not require a new project or a one time switch from an old technology. On the contrary, the tool can be gradually applied to an existing project. Among the few projects that have utilized the tool, only one was implemented with completely new code. All the others were existing projects and we had no luxury to switch a whole project at once. So we had to apply the Summer MVC to just those parts of the projects where new features or modifications were required. When following this approach, it took months to completely switch the projects to the introduced technology. As each feature was ported, the applications became more comprehensible, maintainable, and reliable.

The Summer Framework itself also underwent some evolution, but all the changes were to fix minor bugs, and to introduce a couple of new, more convenient functions. And the best thing was that fixing a bug just in one place, removed that bug from all relevant functions in all our projects implemented with the tool.

CONCLUSIONS

We offer a language independent way to define interfaces for the whole project. After that, different languages and tools can be used, each for its layer. This is different from the prevalent understanding of interfaces as tunnels to act at another layer in terms of that layer.

```
connection.prepareStatement("SELECT C1 FROM T1");
```

```
System.out.println("<TABLE><TR><TD>...");
```

In these examples Java code literally includes statements in foreign languages (SQL and HTML, correspondingly.) From our viewpoint here is no due separation of the tiers.

The proper separation of the tiers means that they can each be written in a different programming language (say, HTML, Java, and SQL,) but there should never be any mixture of the languages, or any mixture of the code belonging to different tiers in one file.

In this article we have provided an example of a GUI that is written entirely in a markup language, of an application server that is written in Java, and a back end written in SQL. Our example relies on the introduced tool, Summer MVC, facilitating development of a three tiered application, based a strict interface and data definitions.

REFERENCES

1. APACHE. 2004. EJB Control Developer's Guide. http://beehive.apache.org/docs/1.0/system-controls/ejb/ejbDoc.html The Apache Software Foundation.
2. FALKNER, Jayson; JONES, Kevin. 2004. Servlets and JavaServer Pages; the J2EE Web Tier. Servlets. Addison-Wesley, UK.
3. FRANCISCO, John; SADIKOV, Victor. 2008. OOML: Structural Approach to Web Development. 18th International World Wide Web Conference, under review.
4. JAGLALE, J☒r☒me. 2008. Spring MVC Fast Tutorial. http://maestric.com/en/doc/java/spring Maestric Web Home, Vancouver, Canada.
5. SHEIL, Humphrey. 2001. To EJB, or not to EJB? http://www.javaworld.com/javaworld/jw-12-2001/jw-1207-yesnoejb.html Network World, Inc.
6. SKONNARD, Aaron. 2003. Understanding WSDL. http://msdn2.microsoft.com/en-us/library/ms996486.aspx Microsoft Corp.
7. WIKIPEDIA. 2008. The Free Encyclopedia. Enterprise JavaBeans. http://en.wikipedia.org/wiki/Enterprise_-Java_Bean Wikimedia Foundation, Inc.