A WEB-BASED TIME BOOKING FRAMEWORK

Liping Zhao, Abdelgadir Ibrahim
Department of Computation, UMIST, Manchester, U.K.

Keywords: Framework, pattern, product family, web-based application

Abstract: A framework can be viewed as a design schema from which application systems derive. Application systems of a framework are called instantiations of the framework and collectively form a product family. The article describes the development of a web-based time booking framework. The development consists of three main stages: meta level design, framework level design and application instantiation. The article illustrates, through the development of a car hiring application system, that once the framework is in place, instantiating an application system only requires a few simple steps.

1 INTRODUCTION

Many business and service providers require that *assets*, whether they are tangible such as conference rooms or intangible ones such as doctor appointments, be booked for a particular date and time prior to their use in order to prevent overload and undesirable time clashes. For the purpose of this article a *booking system* is defined as a set of mechanisms and procedures together with the supporting tools, whether in paper or electronic form, that enable a business to provide its customers with a specific service at a pre-agreed date and time. In this article, the terms *asset* and *service* are used interchangeably to refer to the booked entity, and the same applies to business and service provider.

The booking procedure may vary from one business to another and may also vary according to the characteristics of the customer or the booked asset. Some business for example only accepts bookings from registered customers e.g. a health care surgery. Others may impose some restrictions on the assets that a customer is allowed to book. A car hiring company, for example, may employ a policy that prevent customers with certain number of license penalty points from hiring cars of a certain value. Yet other business may demand that bookings be made at least a specific number of days in advance for managerial purposes. These collectively are termed *business rules* and they provide the operational behaviour of the booking system.

Although the business rules for time booking may vary from one application to another, most time booking applications share, albeit possible small variations, a common set of procedures for asset booking. These involve a customer contacting a service provider and requesting a specific service. Based on the service request, the provider then presents the customer with a list of possible times at which the service can be offered. Once the customer has selected the desired time, the provider then confirms the agreed time either verbally or in written form. Figure 1 illustrates this common booking process.

The identification of the common booking process in Figure 1 suggests that it make sense to develop a time booking framework for different booking applications. Developers utilising this framework can then extend this common functionality as appropriate for their specific application requirements.

Framework development is more difficult and expensive than normal application development (Durham, Johnson, 1996). In order to be useful, a framework must be simple and easy for a developer to understand and learn while at the same time be functionally rich so that it can be used quickly and easily without much effort by developers. Furthermore, a framework must be sufficiently flexible so that they can be customised for the individual application requirements. Hence a framework should only be developed in situations where applications share a common set of requirements. Application systems of a framework are called *instantiations* of the framework and collectively, they are referred to as *a product family* (Parnas, 1976). All the members in a product family are similar in their generic functionality, but each member addresses a different business need.

This article illustrates the development of a web-based framework for time booking applications. It describes the three main design stages, which are meta level design, framework level design and application instantiation. The article illustrates, through the development of a car hiring application system, that once the framework is in place, instantiating an application system only requires a few simple steps.

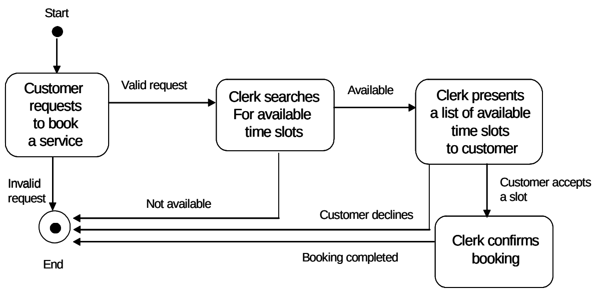


Figure 1: Common booking process

2 FRAMEWORK DEVELOPMENT

We have adopted the classic three-tier architecture for the framework development and the *Model 2* (PetSchulat, 2001) approach as our server side technology (Figure 2). *Model 2* combines *Java Server Pages* (JSPs), *Java Servlets* and *Java Beans* (Govoni, 1991), (Hall, 2002), which fit into the *Model-View-Controller* (MVC) architecture (Reenskaug, 1996). In *Model 2, Java Beans* act as the model, containing time booking functionality and requirements. *Java Servlets* act as the controllers, receiving users' input and serving their requests before forwarding control to the appropriate *JSP* for presentation. The controllers also make available to the *JSP* environment all the required beans from which the *JSPs* can extract the results for presentation to users. The *JSPs* play the role of the *view* which provides the page layout for the user interface. By dividing the application server into the model, the views and the controllers, the framework developers can work independently on different parts of the application.

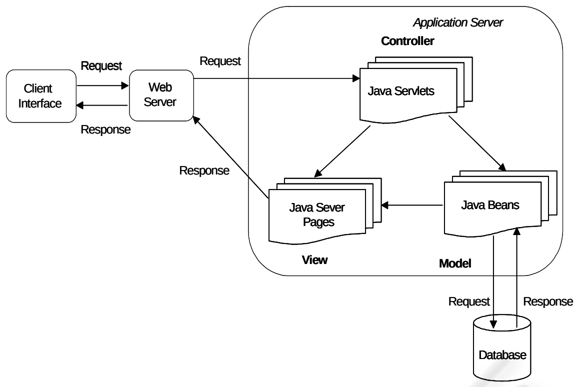


Figure 2: The framework architecture

For space reason, this paper focuses on the development of Java Beans (the model) as shown in Figure 2. We divide framework development into the three levels. At the top level, or meta-level, we design the framework's schema, which outlines the arrangement of the framework components. At this level, a framework component is a grouping of a set of classes. The framework schema is similar to the relational database schema in that it is generally applicable to a broad range of frameworks. In the middle level, or framework level, we design the classes and interfaces of a particular framework. A framework thus can be viewed as an instance of the framework schema. A framework corresponds to a relational table description. At the bottom level, we derive a particular application system by extending the framework.

2.1 Meta Level Design

The model is the heart of the framework, which contains time booking information, functionality and requirements. The model is also the gateway to the database which stores all the persistent booking information. The meta level of the model consists of two components: *Application Services* and *Java Beans* (Figure 3). *Java Beans* are a set of Java classes which have three main responsibilities: (1) to represent business requirements, (2) to serve as a mapping between the database and other framework components, and (3) to manipulate the database. These *Java Bean* classes are called *Data Beans* to reflect the nature of their responsibility. The Application Services component has two major responsibilities: (1) to interact with *Data Bean* classes and (2) to provide other framework components with all the booking related functions. The Application Services interacts with a data bean in the following manner:

– An application service component first creates and populates a bean with the content of a database record. Thereafter, all changes to that record are made to the specific bean, which then propagates to the database.

– A data bean holds the content of a database record created by a booking service object and knows how to update, insert or delete that record. The framework makes changes to a record here instead of directly manipulating the database.

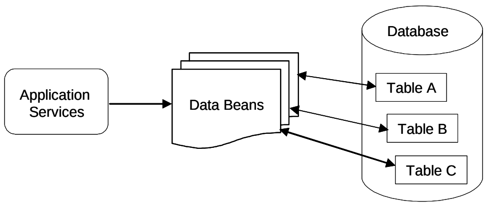
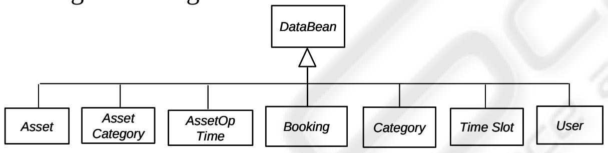


Figure 3: The schema of the framework model

2.2 Framework Level Design

The framework level design concerns designing the classes and interfaces of the booking framework, which consists of the following steps.

Step 1. Designing data bean classes. The data bean classes represent business requirements and map these requirements onto a set of database tables. The challenge is to decide how to divide the business requirements into appropriate classes to achieve a common maximum subset of requirements. To help us to make the decision, we have used the E-R modelling and relational database normalization techniques. The result of this design is given in Figure 4.



Bean Class	Description
Asset	Represent an asset record and related operations
AssetCategory	Represent an asset category record and related operations
AssetOpTime	Represent an operational hour record and related operations
Booking	Represent a booking record and related operations
Category	Represent a category record and related operations
TimeSlot	Represent a time slot record and related operations
User	Represent a user record and related operations

Figure 4: Java bean classes for booking requirements

In Figure 4, all the classes are abstract classes. *DataBean* is the interface to all the Bean classes. This reduces the coupling between other framework components and the Bean classes and increases the framework's extendibility. The application developer can extend or add a new Bean class through the interface. Similarly, this design reduces the coupling between the framework components and the database tables. Hence, if changes are to be made to a database table such as renaming or adding a new column, only the relevant bean may need to be changed. Furthermore, the relational-object mapping in the Bean classes simplifies other framework components as they can now interact with the relational database in an object-oriented fashion.

To maximize the flexibility and extensibility of these Beans, we have used the Factory Method pattern for instantiating Bean objects and the Template Method pattern (Gamma, Helm, Johnson, Vlissides, 1995) for implementing operations.

Step 2. Designing Application Services classes. As stated in Section 2.1, the Application Services component holds the responsibility of interacting with the Data Beans and providing all the booking related functions. We first design an *Application Services* interface for other framework components, such as the controllers. We then derive a Booking Services interface as the main entry point for all the booking related functions. Figure 5 illustrates this design.

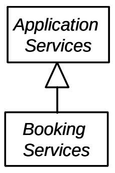


Figure 5: Booking services containing booking functions

2.3 Framework Instantiation

To develop a car hiring booking system, we take the following steps:

1. Add the following vehicle specific fields to the ASSETS table:
   REGNO; MODEL; INSURANCE; FUEL; MILEAGE; PRICING; ADDITONALINFO; DEPOSIT; SEATING; TONNAGE
2. Create a Bean class *Vehicle* which extends class *Asset* with attributes and methods that handle the above additional fields.
3. Create a *VehicleBeanFactory* class which extends *BookingBeanFactory* and overrides the appropriate methods to return an asset of type *Vehicle* instead of the default *DefaultAssetImpl*.

4. Create a *VehicleBookingServices* class which extends the *BookingServices* class and overrides method createBeanFactory() to return a *VehicleBeanFactory* instead of the default *BookingBeanFactory*.
5. Modify the appropriate *JSP* to include vehicle specific fields.

Figure 6 illustrates the framework design structure.

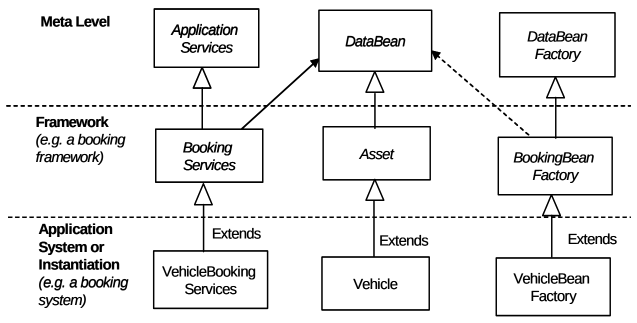


Figure 6: The three levels of framework design

3 CONCLUSION

Framework design takes place at the three levels. The top level is the schema or meta design, which identifies and arranges the groupings of the framework components. A framework schema should be generally applicable to a variety of frameworks. The middle level design is specific to a framework, concerning the mapping of user requirements onto the classes, the design of interfaces and functionality. The middle level design is perhaps the most challenging one as it needs to consider how the framework can cope with the future change and what flexibility should be built into the framework so that changes can be made without major redesign of the framework.

The bottom level design is to generate a specific application system, which involves extending the framework classes and overriding the class methods. Suffice it to say that the main purpose of frameworks is to reduce the time and cost of design at this level by facilitating reuse of both design and implementation at the levels above.

The article has demonstrated that the booking framework can be easily extended to implement an application system. Although framework development incurs higher costs, the costs will be paid off when the framework is used to create application systems, which only requires a few simple steps.

ACKNOWLEDGEMENTS

We appreciate the valuable comments from the reviewers.

REFERENCES

Durham, A., Johnson, R. (1996). A Framework for Run-time Systems and its Visual Programming Language. *In Proc. Of OOPSLA '96, Object-Oriented Programming Systems, Languages, and Applications. San Jose, CA.*

Gamma, E., Helm, R., Johnson, R. Vlissides, J. (1995). *Design Patterns, Elements of Reusable Object-Oriented Software,* Reading, MA: Addison-Wesley.

Govoni, D. (1999). *Java application frameworks*, John Wiley & Sons.

Hall, M. (2002). *More servlets and Java server pages*, Sun Microsystems Inc.

Parnas, D. L. (1976). On the design and development of program families, *IEEE Transactions on Software Engineering*, vol. 2, no. 1.

PetSchulat, S. (2001). JSP or Servlets - which architecture is right for you? Java Report, no. 6.

Reenskaug, T. (1996). Working With Objects, USA: Manning.