Enhancing Portal Design

Yuriy Taranovych
Technische Universität München, Germany

Michael Schermann
Technische Universität München, Germany

Andreas Schweiger
Technische Universität München, Germany

Helmut Krcmar
Technische Universität München, Germany

INTRODUCTION

In recent years, portals became more and more popular among organizations (Klaene, 2004). A portal provides a solution for aggregating content and applications from various information systems for presentation to the user (Linwood & Minter, 2004). Generally, portals pose three main architectural requirements (Linwood & Minter, 2004): as portals integrate heterogeneous content from various sources, a modularized architecture is necessary to allow maintainable portal systems. Second, portals require separating various concerns (Fowler, Rice, & Foemmel, 2002). For instance, the portal's user interface is supposed to display heterogeneous content consistently on various devices, whereas the back-end is supposed to syndicate content from various sources. Third, a consistent management and coordination of different information sources, portal elements, and other components is necessary for good portals design.

Based on these three characteristics of portals we investigate existing portal solutions (BEA WebLogic, IBM Websphere, Liferay Portal, eXo Platform, and JBoss Portal) to identify best practices in portal architectural design. In software engineering best practices are usually captured in patterns. The idea of using patterns for capturing best practices has been transferred from the fields of architecture and cognitive research to software engineering aiming at enhancing software development (Gamma, Helm, Johnson, & Vlissides, 1995) in terms of reusability or using established solutions. Furthermore, we identify patterns that are not used in the analyzed portals, but may significantly contribute to good architectural design.

Based on our analysis, we construct a portal pattern language to summarize existing best practices in portal architecture. Using the portal pattern language assists portal developers in evaluating specific design problems in the context of related problems. Thus, portal design decisions are made with an overall background of best practices in portal development.

The article is structured as follows: first we give an overview of patterns and their use in software development. Next, we present architectural design patterns that are being applied in portal development. Based on this, we construct the core set of a portal pattern language to support design decisions for portal architectures. The article closes with a summary and outlook on future research

SOFTWARE DESIGN PATTERNS

The original idea of patterns rose from Alexander's patterns in architecture (Alexander, Ishikawa, & Silverstein, 1977). Patterns describe a design problem and a general solution to it in a particular context. In that way the general solution can be adapted and thus reused in different settings. Hence patterns can be seen as best practices or accepted and proven solutions for recurring design problems. Patterns are captured experience of engineers or experts in a particular field. The software community adopted design patterns in the early eighties (Gamma et al., 1995).

A pattern generally comprises the following elements.

The context comprises causes which lead to the specific problem of the pattern as well as conditions under which the problem generally occurs. Hence, the context supports acquiring the relevance of a pattern. The problem section generally describes identified contradictions in the context of the pattern. Such aspects of pattern problems are usually called forces. The solution section of a pattern explains a proposal of how to solve the given problem by dissolving mentioned forces. Furthermore an illustration of possible side effects is given. The closing section of a pattern comprises references to related patterns (Alexander et al., 1977).

Software patterns are classified into three layers: architectural patterns, design patterns, and idioms (Coplien & Schmidt, 1995). Architectural patterns provide the highest level of abstraction in software patterns. They express fundamental structures and organization schemes for software systems. Architectural patterns provide a set of predefined subsystems, specify their relationships and include rules and guidelines for organizing the relationships among them (Shaw & Garlan, 1996). Design patterns suggest solutions by providing a collection of class or subsystem and define the relationship among them. Idioms describe how coding problems can be solved in particular programming languages.

We are focusing on supporting architectural design of portals and thus we analyze architectural patterns and develop the core of a pattern language in the remainder of the article.

ARCHITECTURAL PATTERNS FOR PORTAL DEVELOPMENT

The following sections describe architectural patterns which are used or can be reasonably used for the development of portals.

Layered Architecture Pattern

Context/Problem

Traditionally, developers start developing software with drawing a graphical user interface (GUI) and then writing blocks of code that execute application actions in response to user input (Yang, 2001). Many design methodologies start with the construction of a GUI, which often consolidates into a final system design. As a result, a program organized around GUI elements and user actions on those elements, with persistent data manipulation, application functionality, and display code completely interwoven (Yang, 2001).

Solution

To solve the problem, an application has to be put into different layers. This approach is considered to be beneficial, since it separates conceptually different issues. From an architectural point of view, the system is partitioned into a number of layers placed on top of one another. For example, services of layer n+1 consist mostly of the services provided by layer n or a combination of sublayers. There are numerous benefits of a layered architecture, including more freedom and increased flexibility. It leaves designers more responsibilities (Yang, 2001).

Applicability to Portals

Using the layered architecture pattern for portal development will allow fulfilling the separation of concerns requirement by putting conceptually different parts of the application into different layers. This pattern is recommended by for the development with BEA WebLogic portal (2005).

Model/View/Controller Pattern

Context/Problem

A mixture of code for data access and business logic presentation in applications can lead to several problems. Such applications are difficult to maintain because of interdependencies between all of the components. High coupling makes classes difficult or impossible to reuse, because they depend on so many other classes. Adding new data views often requires reimplementing or copying and pasting business logic code, which then requires maintenance in multiple places. Data access code suffers from the same problem (Yacoub & Ammar, 2003).

Solution

A proven way of solving the problem described above is to apply the model/view/controller (MVC) architectural pattern. The MVC pattern is originally designed for user interfaces in Smalltalk-80 (Krasner & Pope, 1988). The pattern consists of three parts: model, view, and controller. The model encapsulates the application logic and contains the functional core of the application. View components present information to the user. Different views present the model's information in a variety of ways. A view retrieves the current data values to be displayed from the model and put them on the screen. The controller describes how the user interface changes according to the user's input. The model does not need to take care of different needs of views, but simply notifies all registered views when the model is updated. In return, the views consult the model in order to get the relevant changes (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996; Yacoub et al., 2003).

Applicability to Portals

MVC is widespread in portal development. It could be identified in all investigated portals. This fact is not surprising as MVC makes its contribution to fulfilling all three architectural portal requirements discussed above. It provides a basis for modularization and separation of presentation (view) and functionality (model) as well as for coordination of them by means of a controller.