Chapter VII

Applying Patterns for Reengineering to the Web

Uwe Zdun, Vienna University of Economics, Austria

ABSTRACT

This chapter examines the use of patterns for reengineering legacy systems to the Web. Today reengineering existing (legacy) systems to the Web is a typical software maintenance task. In such projects developers integrate a Web representation with the legacy system’s application programming interface (API) and its responses. Often, the same information is provided to other channels than HTTP and in other formats than HTML as well, and the old (legacy) interfaces are still supported. Add-on services such as security or logging are required. Performance and scalability of the Web application might be crucial. To resolve these issues, many different concepts and frameworks have to be well understood, especially legacy system wrapping, connection handling, remoting, service abstraction, adaptation techniques, dynamic content generation, and others. In this chapter, we present patterns from different sources that resolve these issues. We integrate them to a pattern language operating in the context of reengineering to the Web, and present pattern variants and examples in this context.

INTRODUCTION

Many existing software systems have to be migrated to the Web. That means that the legacy system gets an interactive Web interface, typically in addition to its existing interfaces. This Web interface forwards Web requests to the legacy system, decorates the responses of the system with HTML markup, and sends the results to the Web client. Inputs are handled via the Web browser, say, using HTML links and forms. For small applications, building a Web interface is quite simple; however, for larger, existing systems there are some typical issues, including:
high flexibility and adaptability of the Web interface,
reuse of the presentation logic,
high performance,
preserving states and sessions,
user management, and
serving multiple channels.

That means that implementing a Web interface for a larger, existing legacy system is a non-trivial task that is likely to be underestimated. This underestimation, in turn, results in unexpected maintenance costs and development times. However, there are many successful projects that have avoided the common pitfalls.

Software patterns are a way to convey knowledge from successful solutions. To tackle the issues named above, we present patterns from various pattern languages and pattern catalogs that can be applied in the context of reengineering to the Web. In particular, we discuss connection and invocation handling, legacy system wrapping and adaptation, dealing with asynchronous invocations, concurrency handling, service and channel abstraction, session management, and dynamic content generation and conversion. The patterns are originally presented in different contexts; in this chapter we present them in variants specific for reengineering to the Web. This way we build up a pattern language for reengineering to the Web, consisting of patterns already documented elsewhere in related, yet different, contexts.

The remainder of this chapter is structured as follows. In the next section we give a brief problem outline, and then we briefly explain the concepts “pattern” and “pattern language”. Next, we discuss connection and invocation handling in Web applications that connect a legacy system to the Web. Then we introduce solutions for legacy system wrapping, adaptation, and decoration. In the following section we discuss how to provide the services of a legacy system to more than one channel. Also we illustrate how to provide sessions and state in Web applications despite the HTTP protocol being stateless. Next we discuss content generation, representation, and conversion for the Web. The following section deals with the integration of typical add-on services, such as security or logging. Finally, we give an overview and discussion of the pattern language that we have explained incrementally in the preceding sections, provide a case study, and conclude.

**PROBLEM OUTLINE**

In the context of reengineering to the Web, a legacy application gets an (additional) interactive Web interface that decorates the outputs of the system with HTML markup and translates the (e.g., form-based) inputs via the Web browser into the (legacy) system’s APIs. At first glance, this seems to be a conceptually straightforward effort, though it might turn out to be a lot of work for larger systems.

In Figure 1 we can see a simplistic three-tier architecture for interactive, Web-based applications. A Web user agent, such as a browser, communicates with a Web server. The Web server “understands” that certain requests have to be handled interactively. Thus, it forwards the request and all its information to another module, thread, or process that translates the HTTP request to the legacy system’s APIs. An HTML decorator
Traditional Data Oriented versus Process Oriented Reengineering of Legacy Systems
[www.igi-global.com/chapter/traditional-data-oriented-versus-process/63244?camid=4v1a](www.igi-global.com/chapter/traditional-data-oriented-versus-process/63244?camid=4v1a)

The Importance of Data Within Contemporary CRM
[www.igi-global.com/chapter/importance-data-within-contemporary-crm/44183?camid=4v1a](www.igi-global.com/chapter/importance-data-within-contemporary-crm/44183?camid=4v1a)