Pattern-Based Design of an Asynchronous Invocation Framework for Web Services

Uwe Zdun, Vienna University of Economics, Austria
Markus Voelter, Ingenieurbüro für Softwaretechnologie, Germany
Michael Kircher, Siemens AG, Germany

ABSTRACT

Asynchronous invocations are needed in the context of distributed object frameworks to prevent clients from blocking during remote invocations. Popular Web service frameworks offer only synchronous invocations (over HTTP). An alternative is messaging protocols but these implement a different communication paradigm. When client asynchrony is not supported, client developers have to build asynchronous invocations on top of the synchronous invocation facility. But this is tedious, error-prone, and might result in different remote invocation styles used within the same application. We present a number of patterns for asynchronous invocations and explain how these patterns can be used to build asynchronous invocation facilities for Web service frameworks. We exemplify this approach by explaining the design and implementation of an asynchronous invocation framework for Apache Axis.

Keywords: asynchronous invocations; patterns; remote objects; Web services.

INTRODUCTION

This article discusses asynchronous invocations in the context of Web services. Web services provide a standardized means of service-based, language independent and platform independent interoperation between different distributed software applications. The use of Web services on the World Wide Web is expanding rapidly as the need for application integration and interoperability grows (Booth et al., 2003).

Although there are many different kinds of distributed object frameworks that refer to the term Web services, a Web service can be described by a set of technical characteristics, including:

- The HTTP protocol family (Fielding et al., 1999) is used as the basic communication protocol.
- Data, invocations, and results are transferred in XML encoded formats, such as SOAP (Box et al., 2000) and WSDL (Christensen et al., 2001).
- Remotely offered services are invoked with a simple, stateless request/response scheme, and thus Web services are more often message-oriented than they are RPC-oriented.
- Many Web service frameworks are not limited to HTTP as transport protocol.
- The services are often implemented with different back-end providers (for instance, a Java class, an EJB component, a legacy system, etc.) and a model for integration of these back-ends is provided by the Web service framework.

Advantages of this approach to invoke remote objects (Voelter et al., 2002), or other kinds of service implementations such as procedures, are that Web services provide a means for interoperability in a heterogeneous environment. Basing the information exchange only on stateless message exchanges leads to a loose coupling of clients and servers. Web services are also relatively easy to use and understand due to simple APIs, and XML content is human-readable. The goals of Web services go beyond those of classical middleware frameworks, such as CORBA, DCOM, or RMI: they aim at standardized support for higher-level tasks such as service and process flow orchestration, enterprise application integration (EAI), and providing a “middleware for middleware” (Vinoski, 2003).

Current Web service frameworks have some liabilities associated. In the spirit of the original design ideas of XML (Bray et al., 1998) and XML-RPC (Winer, 1999) as the predecessor of today’s standard Web service message format SOAP, XML encoding was expected to enable simplicity and readability as a central advantage. However, today’s XML-based formats used in Web service frameworks, such as XML Namespaces, XML Schema, SOAP, and WSDL, are quite complex and thus not very easy to read and understand (by humans). In many cases, stateless communication as imposed by HTTP and SOAP causes some overheads because it may result in repeated transmission of the same data (for instance, for authentication or identifying the current session). Cai et al. (2002) provide detailed benchmarks comparing different encoding mechanism for Web services. This study leads to the following results: XML as a (string-based) transport format is bloated compared to more condensed (binary) transport formats. This results in larger messages, as well as a more extensive use of network bandwidth. This problem can be avoided by compressing XML data, but compression leads to an additional performance overhead. XML consists of strings for identifiers, attributes, and data elements. String parsing is more expensive in terms of processing power than parsing binary data.

Many Web service frameworks, such as Apache Axis (Apache Software Foundation, 2003), only allow for synchronous invocations (for synchronous protocols such as HTTP). That is, the client process (or thread) blocks until the response arrives. For client applications that have higher performance or scalability requirements, the sole use of blocking communication is usually a problem because latency and jitter make invocations unpredictable. In such cases we require the cli-
Related Content

Effective Service Composition in Large Scale Service Market: An Empirical Evidence Enhanced Approach
[www.igi-global.com/article/effective-service-composition-large-scale/64224?camid=4v1a](www.igi-global.com/article/effective-service-composition-large-scale/64224?camid=4v1a)

An Adaptive Access Control Model for Web Services
[www.igi-global.com/article/adaptive-access-control-model-web/3083?camid=4v1a](www.igi-global.com/article/adaptive-access-control-model-web/3083?camid=4v1a)
Facilitating Dynamic Service Compositions by Adaptable Service Connectors
[www.igi-global.com/article/facilitating-dynamic-service-compositions-adaptable/3075?camid=4v1a](www.igi-global.com/article/facilitating-dynamic-service-compositions-adaptable/3075?camid=4v1a)

Issues on the Compatibility of Web Service Contracts
[www.igi-global.com/chapter/issues-compatibility-web-service-contracts/59923?camid=4v1a](www.igi-global.com/chapter/issues-compatibility-web-service-contracts/59923?camid=4v1a)