Chapter 14
Building Web Services with Time Requirements

Nuno Laranjeiro
University of Coimbra, Portugal

Marco Vieira
University of Coimbra, Portugal

Henrique Madeira
University of Coimbra, Portugal

ABSTRACT
Developing web services with timing requirements is a difficult task, as existing technology does not provide standard mechanisms to support real-time execution, or even to detect and predict timing violations. However, in business-critical environments, an operation that does not conclude on due time may be completely useless, and may result in service abandonment, reputation, or monetary losses. This chapter presents a framework that allows deploying web services with temporal failure detection and prediction capabilities. Detection is based on timing restrictions defined at execution time and historical data is used for failure prediction according to prediction modules. Additional modules can be added to the framework to provide more advanced failure detection and prediction capabilities. The framework enables providers to easily develop and deploy time-aware web services, with the failure detection code decoupled from the application logic, and allows consumers to express their timeliness requirements.

INTRODUCTION
Web services offer a clear interface connecting providers and consumers and are more and more considered adequate strategic vehicles for the exchange and distribution of data. Web service consumers and providers interact by means of SOAP messages, using a protocol that, jointly with WSDL and UDDI, is the core of the web services technology (Curbera et al., 2002).

The development of web services that can handle timeliness requirements is a complex task. In fact, current technology, development models, and programming tools for web services do not offer simple support to assure timeliness attributes during execution. Despite the fact that some
Building Web Services with Time Requirements

Transactional models offer support for detecting operations that take longer than the desired time (Elmagarmid, 1992), they frequently require a huge programming effort. In reality, programmers must choose the proper middleware (which includes a transaction manager that must be adequate to the deployment environment specificities), create extra code to manage the transactions, state their attributes, and implement the support for timing requirements. Actually, transactions are well fit to support typical transactional behaviors, but they are not a simple mechanism to create and deploy time-aware services. Thus, transactions support for timing failures detection is poor and, additionally, they do not provide support for timing failures prediction.

Although techniques and tools to easily create time-aware web services are nowadays lacking, there has been a steady increase on the number of real applications having to satisfy this kind of requirements. Frequently, programmers handle these issues by implementing improvised solutions to support timing failures (obviously, this can be costly and failure prone). Time, as a concept, has been completely missing from standard web services development models. Furthermore, relevant characteristics such as the detection and prediction of temporal failures have been completely disregarded, particularly if we consider that services are typically deployed over unreliable and open environments. In these conditions, services typically exhibit high or very unpredictable execution times.

High execution times are frequently linked with the serialization procedure required in each web service invocation. In this process, a high amount of protocol information has to be sent in each message, even if the message’s contents is very small (i.e., the SOAP protocol involves the use of a large quantity of data to encapsulate the application data to be transmitted). This serialization procedure is very important, particularly considering that one web service can also act as a client of another web service, thus doubling the end-to-end serialization effort. Unpredictable execution times are mostly associated with the use of unreliable, sometimes not fast, transport channels (i.e., the Internet) for simple client to server communication and also server-to-server communication. Such features make it difficult for programmers to deal with timeliness requirements.

Considering timing requirements, two outcomes are possible from a web service execution: either the service can produce an answer on time, or not. In any case the client must wait for the execution to finish or for the violation of the deadline (in this case, some timing failure detection mechanism must be developed). Despite this, frequently it is possible to forecast the occurrence of timing failures ahead of time. Indeed, execution history can be frequently used to predict, with a certain confidence degree, if a response will be obtained on time. This is not only of utmost importance for client applications, that can use alternative services or retry, but also for servers. These, in turn, can use historical information to adequately administer the resources that are allocated to each operation (e.g., an operation that is predicted to be useless can be aborted or executed under a degraded mode).

In this chapter we discuss the problem of timing requirements in web services. Besides defining the concepts, we present an extensible server-side framework that offers detection and prediction functionalities (wsTFDP: Web Services Timing Failures Detection and Prediction). This framework provides a ready to use timing failure detection mechanism that, based on transparent code instrumentation, is able to collect historical data that can be used for prediction. The framework also provides a prediction component that implements Dijkstra’s shortest path computation algorithm, which makes use of historical execution time values to determine if execution can terminate on time. This framework can be easily extended with multiple components that implement different prediction algorithms. By using wsTFDP, developers are able to easily plug in their...
Related Content

A Survey of Trust Use and Modeling in Real Online Systems
www.igi-global.com/chapter/survey-trust-use-modeling-real/30453?camid=4v1a

A Unified Classification Ecosystem for Auctions
www.igi-global.com/chapter/unified-classification-ecosystem-auctions/64150?camid=4v1a

Enhancing User Experience in Financial Services
www.igi-global.com/article/enhancing-user-experience-in-financial-services/115862?camid=4v1a

Machine Learning for Adaptive Planning
www.igi-global.com/chapter/machine-learning-adaptive-planning/24460?camid=4v1a