

By Fabio Casati, Eric Shan, Umeshwar Dayal, and
Ming-Chien Shan

BUSINESS-ORIENTED MANAGEMENT OF WEB SERVICES

Using tools and abstractions for monitoring
and controlling Web services.

THE MAIN RESEARCH AND DEVELOPMENT FOCUS in Web services today is on basic middleware infrastructures. However, once technology consolidates and, consequently, a substantial number of Web services become available, the attention will shift from service deployment to service management. In fact, management complexity grows with the number and type of services deployed, and therefore companies will soon require tools that support and automate their management efforts [4]. The interest in this area is demonstrated by recent standardization proposals (such as the Open Management Interface [5] and the OASIS Web Services Distributed Management Technical Committee) and by the creation of large-scale research programs [7, 10].

The different facets of Web services management are discussed here, with specific focus on business-oriented management of Web services, which refers to assessing the impact of service execution from a business perspective and, con-



versely, to correcting and optimizing service executions based on business objectives. This is crucial for every corporation, as it aligns IT operations with business goals.

To characterize Web service management it is important to distinguish between management of Web services and management through Web services [1]. Management through Web services refers to leveraging Web services for managing heterogeneous and distributed systems, thanks to their ability of reducing heterogeneity through standardized interaction paradigms. By hiding the managed objects and the components of the management system behind Web service interfaces, the management system has a uniform way of receiving information and sending control stimuli. As an example, the Global Grid Forum has defined a framework for managing resources by defining services for allocation systems and for resources that are managed by allocation systems [6]. Management of Web services refers instead to the problem of monitoring and controlling the Web services themselves and their execution environment, to ensure they operate with the desired quality levels. Here, we focus on management of Web services, since it presents the most interesting challenges and opportunities.

Management of Web services can be classified based on its scope. We distinguish between infrastructure-level, application-level, and business-level (or business-oriented) management. Infrastructure-level management focuses on the Web services platform. The goal is to ensure the different components of the platform are operating with acceptable performances. This is typically achieved by means of agents that access the components' log files (including the error logs) and notify failures as well as performance degradations to a management console, that can in turn alert users when problems arise (see Figure 1). A different agent is typically needed for each component to be monitored. The console also periodically verifies these components are up and running, and restarts them in case of unwanted shutdowns.

Application-level management focuses on the Web services themselves. Specifically, the problem here consists of monitoring each Web service, to alert users in case of performance degradations as well as to proactively instantiate or destroy Web services to cope with

failures and changing load. Application-level management may also provide features such as provisioning (for example, preparing a service for execution or reserving resources) and access control. Application-level management is typically performed by placing interceptors within the path of the SOAP messages (see Figure 1), to log such information as when messages are sent and received or who the client is. In conjunction with information about the WSDL interfaces supported by a service, this enables the computation of, for example, how long an operation lasts and can therefore be used to

assess performances and provide users with monitoring and reporting information. Application-level management can also leverage infrastructure-level management to better identify the problems underlying low performances. Specifically, if users define the relationship between a Web service and the infrastructure on which the Web service relies, then the management system can analyze infrastructure-level issues in terms of the Web services they affect and, conversely, support drill downs from the analysis on a Web service to the analysis of the infrastructure supporting it.

If the management system also wants to control the service (for example, to set quality of service parameters), then the service must provide an interface that is known and accessible to the management system. To this end, either management systems prescribe an interface that services need to implement or, ideally, standardization consortia define interfaces that are implemented by many management systems and that services may choose to support if they want to be managed.

Infrastructure and application-level management are the natural consequence of the fact that Web services constitute an additional layer on top of the middleware stack [1], and traditional management approaches are extended to embrace this new layer. Although there are a few twists related to Web services, the problems are similar to those of traditional application and network management, and so are the solutions. This is why products for infrastructure and application-level management appeared very quickly

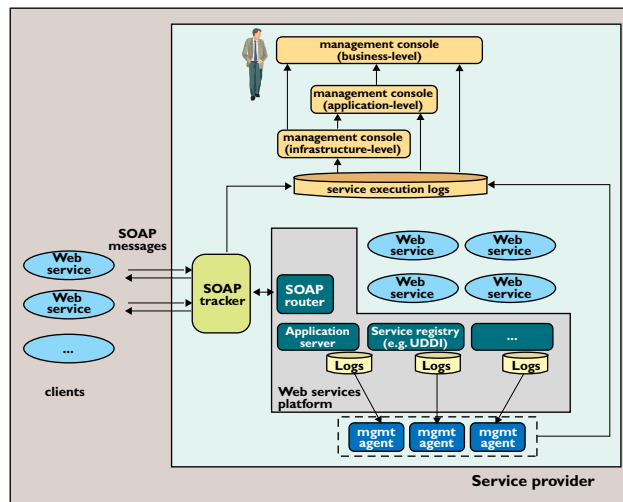


Figure 1. Application and infrastructure-level management.

(see, for example, [8, 9]). Instead, the real novelty that Web services bring to management lies in the need and opportunity for a business-level (or business-oriented) management.

In fact, with Web services, there is a closer relationship between IT and business operations. This is because Web services are typically at a higher level of abstraction with respect to traditional middleware objects (CORBA objects), and are often used to support business-to-business interactions. Therefore, the quality of service executions has a direct effect on the quality of the business transactions and on the relationships between customers and service providers. Monitoring service executions means monitoring business interactions with partners and, conversely, the service level agreements (SLAs) stipulated with the customer pose constraints on how services should operate. Furthermore, companies are under increasing pressure to measure the value of their IT investments and gain visibility into IT operations from a business perspective. Business-oriented management of Web services, discussed next, attempts to exploit these factors by using business metrics as the criteria based on which services are monitored and controlled.

A Holistic Approach to Web Service Management

Today, visibility of service executions from a business perspective is achieved by logging SOAP messages and related information (for example, SOAP header and body, as well as sender and receiver information) and in writing code that maps logged data into metrics meaningful from a business perspective (as opposed to application-level management, where metrics are mostly predefined and related to performance). For example, assume a company provides a service allowing customers to purchase PCs, and including in particular an operation called `order()` as part of its interface. A business manager may consider a purchase transaction to be successful if the `order()` operation returns in less than 30 seconds and has an output result of "accept." If this information can be determined from the SOAP data, then by querying such data it is possible to determine the success rate, which is what users care about in this example. Business-level infor-

mation can then be correlated with application-level and infrastructure-level management, to understand the application and infrastructure issues that contributed to unsuccessful operations.

This approach has many limitations: it requires a large development effort to implement the ad hoc code mapping execution data into business metrics (which varies based on the SOAP message content and on the goals of the analysis) and it suffers from performance problems whenever many real-time reports are needed. However, the most severe limitation is the lack of support for a holistic view of service interactions. What business users want is a complete picture of the external

quality of the interactions, as perceived by the customers, and its relationships with the way services are executed internally. To overcome these limitations, we developed a framework and a tool, collectively known as Web Service Manager (WSM), characterized by three main ingredients:

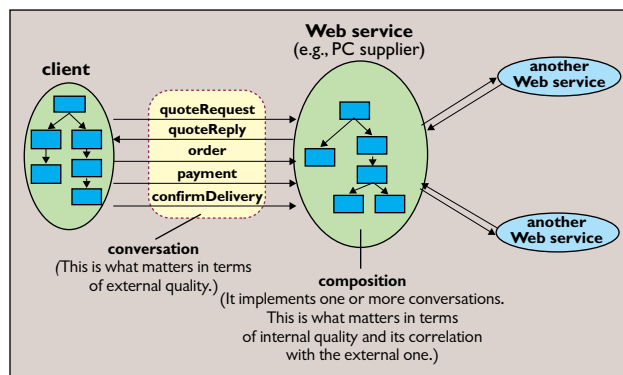


Figure 2. Conversations and compositions in Web services.

- A service model, based on existing standards;
- A metric model, facilitating the specification of properties of interest from a business perspective; and
- A development and runtime environment, enabling the definition and computation of business-level metrics and of how they are affected by service executions.

Service Model. Interactions between customers and providers often involve a conversation (a sequence of operation invocations). For example, to buy a computer, clients will first request the price, then submit an order, and finally make the payment. This fact has two important implications (see Figure 2) [1]. The first is that the business logic of a service includes the invocation of multiple operations. This business logic today is implemented in a third-generation language such as Java, but in the future it will likely be implemented by leveraging service composition technologies such as BPEL [2]. The second is that not all conversations lead to a correct interaction between services (an interaction that does not cause faults). For example, a PC-supplier service expects to receive first an order and then the payment, not vice versa. This means that to describe how to interact with them, services must not only specify their interfaces, but also the set of conversations they support, by using conversation definition

languages such as those described in [2, 3]. As Web service technology matures, increasing numbers of platforms will support both conversations and compositions.

To assess the external quality of Web services, conversation definitions (types) and executions (instances) are what companies really care about, since they are what characterize the interaction with clients. Therefore, in the general case, quality metrics and SLAs are defined on conversations, not operations. For example, a supplier may want to monitor how many PC-purchasing conversations meet a SLA stipulated with the customer, possibly defined in terms of time elapsed between order and delivery. This information can be computed if we have knowledge of the conversations executed by the parties and we have a way to identify messages belonging to the same conversation.

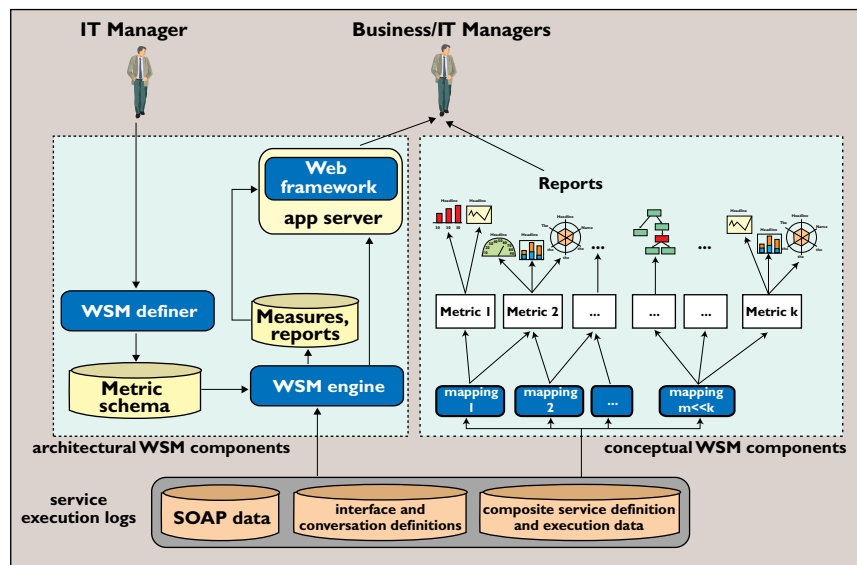
If in addition to conversations we can also monitor the internal composition logic executed to enact a conversation, then we can correlate the analysis of the external interaction with the internal implementation. The relevance of this combined analysis is that now we can understand how the internal execution affects the quality from a business perspective as well as what can be done internally to fulfill a certain business objective. Composition makes this possible since we now have visibility into the internal business logic of a Web service. For example, we can discover that certain quality goals are not met whenever the internal composition follows a certain path, or makes use of certain resources (for example, uses a certain gateway for credit card verification). Hence, by jointly analyzing compositions and conversations we can achieve a holistic management, from assessing external quality to identifying issues responsible for low quality executions, thereby providing an indication of how such problems can be addressed.

To leverage this opportunity, the WSM service model includes conversation and composition information, typically gathered from local service repositories and from a service composition engine (a platform that supports the specification and execution of composite services). WSM assumes conversations and compositions are specified in BPEL, although the concepts are generally applicable and it is

easy to map other data models into the WSM one. Indeed, in our experiments we used an internal prototype composition engine whose model differs from BPEL in the details, but is similar in the substance (as most process models are).

Metric Model. The WSM metric model allows users to map service execution data into information meaningful from a business perspective. The metric model is composed of three basic ingredients: mappings, metrics, and reports (see the conceptual framework in Figure 3).

A mapping is a parametric function that associates values to the elements being analyzed, based on the service model and on service execution data. Elements can be operations, interfaces, services, conversations, or compositions, both as types (composite service definitions) or as instances (composite service executions). For example, a function `timeBetweenOperations(op1, op2, threshold)` may be applied to conversation instances and associate, to each instance, a Boolean value denoting whether the time elapsed between the completion of operations `op1` and `op2` is greater than a threshold. In WSM, these functions are implemented in SQL since the execution logs are



stored in a relational database, but conceptually they can be specified in any language. WSM includes many built-in mapping functions, defined through our experience in using WSM. New mapping functions can be easily defined if built-in ones do not suffice. Mappings are the only place where coding is needed in WSM.

Metrics give meaning to the values computed by the mappings. Specifically, a metric defines a qualitative or quantitative property of the elements that users

Figure 3. Architectural and conceptual components of WSM

want to monitor. It is characterized by:

- The name of the metric (for example, “cost” or “SLA violation”);
- The data type (“numeric,” “Boolean,” or “taxonomy”);
- The type of elements to be measured (for example, “conversation instance”); and
- The computation logic, that is, the definition of how the metric should be computed.

The computation logic is specified by associating the metric to (at least) one mapping, and by specifying the value of the mapping parameters. For example, analysts can define a metric called SLA Violation of Boolean type, and define that its semantics for PC purchasing conversations are such that SLA violations occur when the time elapsed between the execution of operations order and confirmDelivery is more than two days. In this case the metric can be computed by reusing the function timeBetweenOperations, described earlier, with parameters (order, confirmDelivery, 2).

Reports represent different perspectives under which metrics can be analyzed, and are what is ultimately seen by business managers. By aggregating and correlating metric data in different ways, reports enable users to assess the business impact of service executions and identify the causes of poor quality. WSM reports can be defined without writing any code, since WSM has knowledge of the service model and of the properties of each metric. Therefore, it knows which statistics and correlations can be computed and how to do it. For example, once the SLA violation metric discussed earlier has been computed, WSM can immediately show basic statistics (such as SLA violations by week or by conversation type) as well as the relationship between SLA violations and those composition instances supporting the conversation instances in which the violation occurred. For example, users will be able to understand the relationships between SLA violations and the path taken in the internal execution, or the service providers involved in the execution, or any other metric defined at the composition instance level.

The rationale for having a three-level model based on mappings, metrics, and reports on top of the service model is due to code factorization (see Figure 3): typically, different metrics can be computed from the same mapping logic (that is, by processing and transforming service execution data in the same way), and different reports can be derived from the same metric. This modularization considerably simplifies report definition and maintenance: metrics and reports can often be defined without writing code, and if changes arise

MODULARIZATION
CONSIDERABLY SIMPLIFIES
REPORT DEFINITION AND
MAINTENANCE: METRICS
AND REPORTS CAN OFTEN
BE DEFINED WITHOUT
WRITING CODE, AND IF
CHANGES ARISE IN THE
UNDERLYING DATA
STRUCTURE, ONLY
MAPPINGS NEED TO BE
MODIFIED.



in the underlying data structure, only mappings need to be modified.

Development and Runtime Environment. Figure 3 shows the WSM architecture. In addition to agents extracting data from the components of the Web services platform, it includes:

- A development environment, enabling users to browse the set of operations, conversations, and compositions (WSDL and BPEL specifications) and define mappings, metrics, and reports on top of them.
- A metric computation engine, executing mapping functions on top of service execution data to compute metrics. Due to the set-oriented nature of SQL, one execution of a mapping is sufficient to compute all metrics based on that mapping, thereby reducing the computational effort with respect to traditional reporting tools, which execute one query per report.
- A framework for Web application development, allowing designers to deploy reporting applications at the click of a button, by automatically generating JSP pages that access the WSM API to retrieve reports. These pages can then be edited for cus-

tomization with any commercial Web development application.

Conclusion

We conclude by briefly summarizing and emphasizing our main themes. The first is that Web services are at a higher level of abstraction with respect to conventional middleware services, and therefore directly impact business-level metrics and need to be guided by them. The second is that to analyze Web services from a business perspective it is important to define and correlate metrics through interfaces, conversations, and compositions. The third is that simplicity and flexibility in managing the manager are crucial. In fact, metrics and reports are not “static,” and there is the frequent need to modify them to perfect the analysis and cope with changes. It is therefore crucial that management applications support these aspects through tools and abstractions such as the service and metric model of WSM. Our current research efforts aim at “closing the management loop,” that is, automatically controlling services based on the actual or predicted value of business metrics, in order to maximize user-defined utility functions. **□**

REFERENCES

1. Alonso, G., Casati, F., Kuno, H., and Machiraju, V. *Web Services: Concepts, Architectures, and Applications*. Springer Verlag, 2003.
2. Andrews, T. et al. *Business Process Execution Language for Web Services, Version 1.1*. May 2003.
3. Banerji, A. et al. *Web Services Conversation Language (WSCL) 1.0*. W3C Note, Mar. 2002.
4. Bloomberg, J. *Web Services Management: The Key to Service-Oriented Architectures*. ZapThink report, 2002.
5. Bullen, G. et al. *Open Management Interface. Version 1.0*. 2001; www.webmethods.com
6. Global Grid Forum. *Open Grid Services Architecture*; www.ggf.org/ogsa-wg.
7. IBM. *Research Program in Autonomic Computing*; www.research.ibm.com/autonomic/.
8. IBM. *IBM Tivoli Monitoring for Web Infrastructure*. 2002; www.ibm.com/software/tivoli/.
9. Hewlett-Packard. *HP OpenView Service Quality Manager*. Feb. 2003; www.managementsoftware.hp.com/.
10. Hewlett-Packard. *Web Services Management Engine*; www.managementsoftware.hp.com/.

FABIO CASATI (casati@hpl.hp.com) is a research scientist at Hewlett-Packard in Palo Alto, CA.

ERIC SHAN (eshan@hpl.hp.com) is a research intern at Hewlett-Packard in Palo Alto, CA.

UMESHWAR DAYAL (dayal@hpl.hp.com) is a lab director at Hewlett-Packard in Palo Alto, CA.

MING-CHIEN SHAN (mcshan@hpl.hp.com) is a program manager at Hewlett-Packard in Palo Alto, CA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 2003 ACM 0002-0782/03/1000 \$5.00