Business visibility with Web services: making sense of your IT operations and of what they mean to you

Fabio Casati and Vijay Machiraju

Hewlett-Packard
1501 Page Mill Road, MS 1142
Palo Alto, CA, 94304
USA
{firstname.lastname}@hp.com

Abstract. Web services were born to enable application integration across corporations. However, with time, people are starting to realize that the Web services paradigm carries many implications and opens other possibilities. One of the most significant and least explored is the key role that Web services can play in application management. In this paper we show how, by using Web service technology, it is possible to achieve comprehensive visibility over the execution of IT operations and, perhaps even more importantly, how it is possible to assess the correlation between IT operations and metrics that are relevant from a business perspective. Establishing this correlation enables application management to become a tool for managing not only the IT infrastructure, but also the business.

1 Introduction

Web services are emerging as the novel paradigm for application integration and distributed computing. Their main contribution to date consists in providing a way to enable access to the local information system by means of standard Web protocols, thereby simplifying application integration across the Web. Furthermore, Web services standards have been designed from the start with the goal of enabling B2B interactions and therefore, although they are generally applicable and they are indeed often used for EAI, they are biased towards integration across organizations.

As Web service technology becomes more and more widespread and is used as the basis for many integration projects, it becomes apparent that the implications of the objective and of the approach taken by Web services are indeed very significant, and go beyond application integration and business process automation. In particular, Web services have the potential to help address old, largely unsolved problems in the *application management* domain and create new, important opportunities in terms of what can be managed and of what is the significance of application management from a business perspective.

So far, application management had to cope with many hard challenges. First, applications are heterogeneous - there are many varieties of them (e.g., SAP, Siebel, custom-developed applications, etc), they are created by many different vendors, and they run on various platforms. Second, there are no standards for instrumenting applications and collecting management information. The few that exist are either not widely used (e.g.,

ARM [14]), are not broadly applicable (e.g., JMX [17]), or are incompletely specified in the application domain (e.g., CIM [8]). Third, even if all of the applications in an enterprise are instrumented and data is collected, there is no simple way to get a holistic, end to end view of how the IT infrastructure is performing. As an example of this problem, consider an e-commerce application that is built out of three applications - a web server that hosts web pages, an application server that hosts business objects, and a database that hosts business data. All of those three applications could be instrumented to provide data, but what IT managers want to know is whether the e-commerce application is performing well on the whole, as opposed to separately managing the three systems.

By using Web services to create and integrate applications it becomes easier to solve the above problems. Web services push enterprises to wrap the functionality provided by all the underlying applications under a common service interface, described and accessed through standard languages and protocols. This service-oriented approach (also referred to as *service management* in the literature) opens the possibility of providing a comprehensive view of the applications to be managed, hiding the heterogeneity and complexity of the underlying applications and therefore considerably simplifying the task of the management system.

Another important implication of the Web services paradigm is that Web services are often used to expose coarse-grained operations and to make them available to business partners. For example, a chip manufacturer may offer operations used by PC vendors to order new chips (operation *purchaseChips*) and request their shipment to a specified location (operation *shipOrder*). The level of abstraction of these operations is significantly higher than those of conventional enterprise services (such as CORBA objects) and is directly related to the business operations: the way a Web service operates has an immediate effect on customers' satisfaction, since the Web service is what customers interact with. Therefore, by monitoring Web services we can get a sense of how well (or how poorly) we are meeting the customers' requirements. For example, a 3 days delay between the invocation of the *shipOrder* operation and its (possibly asynchronous) reply confirming the shipment means a 3 days delay in shipping the goods. This observation, as obvious as it is, changes the scope and the possibility of application management in a very significant way. Perhaps for the first time, it is in fact possible to develop technologies and tools that, by monitoring the IT operations (and specifically by monitoring Web services), can immediately assess the quality of the business transactions and, in general, can assess not only how the IT is performing, but also how the *business* is performing. Furthermore, these tools can be standard-based (since Web service technology is itself standard-based), and as such generally applicable.

In this paper we discuss how Web services enable this kind of holistic management. In particular, we discuss two aspects:

- visibility *of* Web services: how exposing application functionality as Web services enable the management system to collect IT- and business-relevant information from Web service execution data.
- visibility *through* Web services: how Web services enable comprehensive, IT-level visibility over the heterogeneous set of applications deployed by a company.

The aim of this paper is that of providing an overview of the concepts, while we refer the reader to other papers for the details. We will focus more on visibility of Web services, since this is the most innovative aspect, responsible for extending the notion of what application management means and what its scope is.

2 Visibility through Web services

Visibility through Web services uses Web services for what they have been designed (integration) and for their potential to reduce heterogeneity. In particular, Web services are used in this context to enable a management system to access all managed objects in a uniform fashion. Integrating management information is very important if we want to have a comprehensive visibility on the IT infrastructure: if network management, system management, and application management happen independently, there is no way one can understand the significance of an IT resource in assessing the overall quality of the IT operations. For example, assume that we have two entirely different management systems - one that does content placement (a content delivery network) and one that does resource allocation (e.g., a Grid or a Universal Data Center), both aimed at optimizing performances for the customers. These two systems should be coordinated by a larger entity that decides which is the right approach to improve performances. If these systems are independently managed, it becomes very hard to examine the correlation and interdependencies between the two systems, that can prove so crucial to identifying the problems (low performances) and the causes of such problems.

Web services can help in this scenario due to the fact that they are highly homogeneous and standardized. If the management functionality of the different systems are exposed as Web services, then a higher-level management entity can collect data from these systems using Web services standard, can process them, and can then send feedbacks back to the systems in order to execute actions that help improve the performances (Figure 1).

From a Web services perspective, there is nothing particularly innovative about this. It is just another use case that demonstrates the power of Web services in application integration. This is why we do not discuss this issue further, and we move instead to the complementary problem: that of managing the Web services themselves.

3 Visibility of Web services

The previous section has briefly shown how Web services enable the holistic management of enterprise applications. In the description above, “holistic” refers to the fact that, through Web services, it is possible to build a monitoring infrastructure that looks at all applications in a uniform manner, since the standardization imposed by Web services hides the heterogeneity from the management platform.

In this section we discuss instead visibility *of* Web services. This aspect is orthogonal with respect to the previous one in two ways. The first difference is that here Web services are not the instruments that enable the management platform to gain information on the managed applications, but are the actual object to be monitored and managed. The second difference is that while in management through Web services

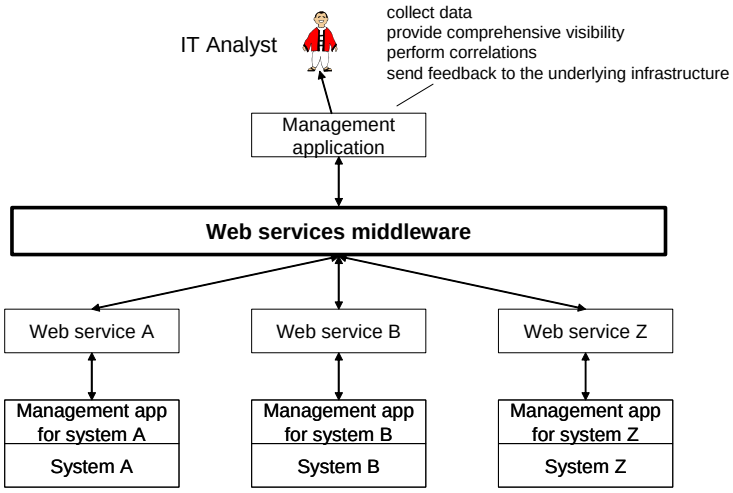

Fig. 1. Web services enable access to heterogeneous management platforms

we could observe many heterogeneous applications, but only from an IT perspective (i.e., we could measure and report IT metrics, such as performance or availability), in management of Web services we focus on visibility on one kind of application (Web services), but we can look at it with a very wide angle, covering both the IT and the business perspective. We have already mentioned in the introduction why this is possible: Web services are often used to support B2B integration, and are the entry points through which customers invoke the functionality offered by the provider. This implies that Web services are often coarse-grained and at a level of abstraction that is higher with respect to services in conventional (EAI) middleware, closer to the business level. Hence, the quality and efficiency of a Web service execution has a high correlation with the quality of service as perceived by the customer.

Note also that the coarse-grained nature of Web service is not only due to their orientation towards B2B applications. Even when they are used in EAI, Web services tend to be at a higher abstraction level than conventional services. This is because Web services carry performance penalties due to the need of transforming messages to and from XML and of packing/unpacking them into SOAP envelopes. Therefore, using Web services makes more sense when the operations they implement are complex and long-lasting, so that the relative impact of this performance penalty is smaller.

Now that we have motivated why Web services naturally enable visibility at the business level, we describe how this can be achieved. In particular, there are two problems that need to be solved: the first is how analysts can describe the business-level metrics that provide them with the information necessary to assess the quality of Web services execution from a business perspective. The second is related to how the data about Web service execution can be collected. We begin from the latter, as the kind of

data that is collected constrain what kind of (business) metrics can be defined on top of them.

3.1 Instrumentation hooks in Web services middleware

Speaking about management *of* Web services means that the objects that are instrumented and analyzed are the Web services themselves. Per se, Web services are "only" entry points to the local information systems, and therefore contain little or no business logic. The implementation is actually performed by the conventional middleware and by the back-end systems (e.g., an ERP or CRM application). So, what does management of Web services mean exactly? In terms of instrumentation, the difference between management of and through Web services concretely manifests itself in what is instrumented and what data is collected. In management through Web services the back-end objects are instrumented, and data about executions at the back end is then collected through Web services. In management of Web services, what is instrumented is the *Web services middleware*.

A thorough discussion of what Web services middleware is and what it includes is outside the scope of this paper. Briefly, in terms of infrastructure, the Web services middleware typically includes components such as (Figure 2):

- The *SOAP router*: it receives SOAP messages [9], parses them, and routes them to the appropriate back-end object (e.g., an EJB) through the middleware (e.g., an application server).
- The *conversation controller*: SOAP routers enable basic interactions among Web services. In many cases, however, the interaction is more complex and involves a set of message exchanges. We define a *conversation* as a set of interactions among two or more Web services. In general, a Web service may support some conversations while disallow others. We define a *business protocol* as the set of rules that specify which are the legal conversations that can occur between two or more Web services [2]. The conversation controller is a component that verifies that the conversation is compliant with the business protocol supported by the service. In addition, if each conversation is implemented by a different object, then the controller can also perform the function of dispatching messages to the appropriate back-end object (through the SOAP router) that is in charge of executing the conversation to which the messages belong.
- The *composition engine*: this is the runtime infrastructure that executes composite services (i.e., services whose implementation is specified by composing other services, typically through a graph-like language). Note that service composition represents an exception to the statement made above about Web services not implementing any business logic: service composition is the only case in which business logic is implemented at the Web service middleware level.
- The *name and directory server*: just like conventional middleware, Web services middleware also includes a directory that supports static and dynamic binding, often compliant with the UDDI specifications [1]. Such component can be internal to the company or it can be provided by a third party.

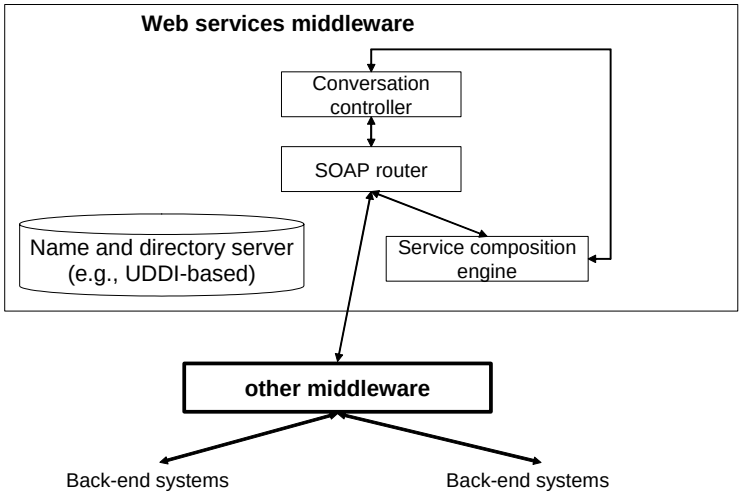

Fig. 2. Simplified view of a Web services middleware platform

Web services infrastructures are likely to be much more complex than this, especially as Web service technology progresses and new standards become accepted. For example, they will include transaction managers, sophisticated security features, and the like. However, the components described above suffice to illustrate the point we want to make in this paper.

Figure 3 shows the architecture of a typical Web services management system. In addition to interacting with other tiers of the information system, the key aspect of a Web services management system is that it also collects data from the Web services middleware components.

Specifically, the SOAP router has information about all the invocations, their timing, and whether they complete successfully or not. This information can be communicated to a management system using standards such as ARM. With that information, a management system (labeled as SOAP management in Figure 3) can monitor the performance of a Web service. Similar to an application management system, one can set response time thresholds and policies to automatically instantiate new objects when the response time of an operation goes beyond a threshold. Note that this functionality can also physically be embedded into the SOAP router itself instead of having a separate management system.

A conversation controller has information about the state of ongoing conversations, their timing, and their compliance with the business protocol. A management system (labeled conversation management in Figure 3) that receives this information can analyze conversation bottlenecks, manage their performance, and perform lifecycle operations such as suspend and resume on ongoing conversations. Once again, this management functionality can physically be embedded into the conversation controller itself.

The composition engine has information about the component Web services that have been invoked, the timing of the invocations in progress and of past invocations,

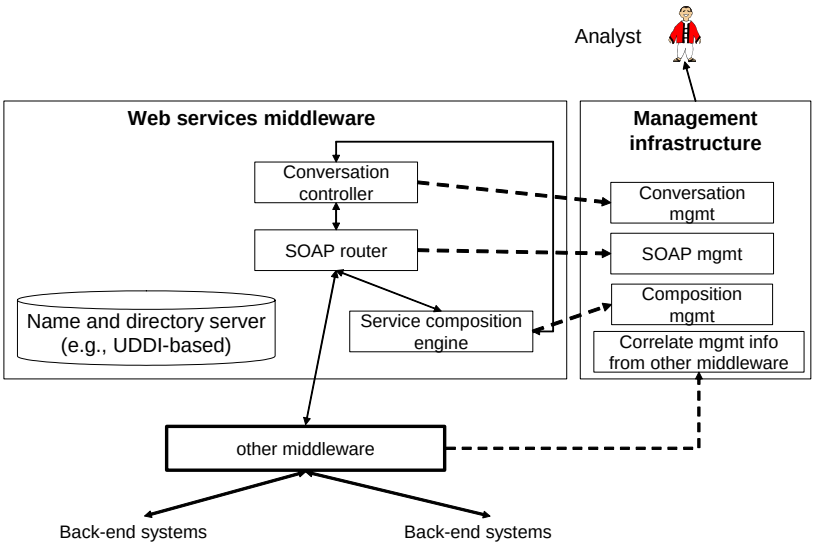

Fig. 3. Instrumentation and control of middleware components by a Web services management system

and the data exchanged between the composite service and its component services. Using this data, a management system (labeled composition management in Figure 3) can correlate the response times of a Web service's operations with the response times of its component Web services. One can make a judgment on whether a performance degradation in a Web service is caused by an internal failure or by a performance degradation in another Web service. A composition management system should also be able to analyze the composition model for bottlenecks, compare between two alternative service providers for executing a step in the composition, or automatically select one of the service providers depending on performance-based historical data analysis.

In addition to collecting data from the Web services tier, a Web services management system should also correlate that data with data collected from other tiers of the underlying application. Correlation techniques have been used in the past to componentize end-to-end response times into response times of individual tiers [3, 12]. Similar techniques can be extended to the Web services tier as well. With these extensions, a Web services management system will not only be able to report on metrics collected at the interface of the Web service, but also will be able to drill-down into lower tiers to analyze failures or diagnose performance, security, and configuration problems.

3.2 The role of standards

The techniques described above for instrumenting the infrastructure and collecting data are analogous to those of conventional middleware, and so is the nature of the analysis. After all, for what we have discussed so far, it does not make much of a difference

whether the component being instrumented is, for example, a Web service composition engine rather than a conventional workflow system. However, standardization makes things much easier in Web services with respect to conventional middleware. In fact, while the middleware hooks discussed in the previous section help in data collection, Web services standards help in better data interpretation and analysis. In Web services, the operational interfaces, business protocols, structure of exchanged messages, and composition models are all specified in standardized XML-based formats. From a management perspective, this makes the processing of the data collected much simpler than when the management system has to deal with heterogeneous representations. Furthermore, these specifications form the basis for a "management data model" that helps organize the collected instrumentation data. For example, a management system will know from a WSDL specification whether a certain operation is one-way or two-way, or what protocol bindings an operation uses [5]. Using this data model, a management system will be able to correctly interpret the contents of a message sent as part of that operation, since the structure of the message depends on the bindings used and the method of calculating the response time on an operation depends on whether it is synchronous or asynchronous. In case of asynchronous operations, then BPEL specifications, if present, and conversational information (e.g., conversation identifiers embedded into messages, possibly in compliance with the WS-Coordination standard [11]) can be used to establish the correlation between requests and replies. As another example, a management system may learn over time that a composite Web service (described in BPEL [6]) chooses the provider for executing a certain component among a restricted set of partners. Using that knowledge, the management system may be able to compute relative performance of the partners. As a final example, historical performance measurements accumulated by a management system about various steps in a composite service (along with those that are executed by invoking component Web services) can be augmented with the information in BPEL to predict the aggregate performance of the composite Web service [7].

3.3 Taking visibility to the business level

The previous sections have shown the potential for management systems that provide visibility into Web service executions. As interesting and as appealing as it is, this visibility is still at the IT level: the kind of analysis that can be performed is the same as the one described in management through Web services, although now we manage something different (the Web services) and we have the luxury of standardization that makes things much easier. We now gradually shift focus from IT-level visibility to business-level visibility.

In IT-level visibility, the metrics that analyst can see are often predefined. IT analysts are typically interested in performance, throughput, and number of invocations of a service. With a few exceptions, the semantics of these metrics is always the same regardless of the application that is managed. They can be provided out of the box by the management application. In business-level visibility, this is not the case. In fact, unlike IT metrics, business metrics are not predefined, and in particular their semantics is not predefined: although most analysts will be interested in metrics such as *quality*, *efficiency*, or *cost*, the semantics of these metrics differ from application to application,

and even from analyst to analyst. Hence, analysts must have a way to define metrics and their semantics.

Fortunately, the Web services middleware offers many abstractions and specifications that are ideally suited to be used as a basis for defining metrics and their semantics. There are essentially two factors that come into play with Web services and that contribute to this purpose. The first is related to the very nature of what Web services are meant to be: loosely-coupled, autonomous components described by means of standardized specifications. The implication of being loosely-coupled and autonomous is that services are described in a manner that is much more complete with respect to services in conventional middleware. Conventional services (e.g., CORBA objects) are basically described by their Interface Definition Language (IDL). Web service specifications are richer: the IDL itself (WSDL) is more complex and sophisticated than CORBA-like IDLs (for example it includes address and binding information), and un the near future, as Web services standard mature, it is likely to include non-functional attributes and policies associated to the service. These may range from information about the cost of the service to Quality of Service (QoS) guarantees. Furthermore, as we have seen, a Web service description may include the specification of the business protocols supported by the service. Finally, the standard-based nature of Web services and the service-oriented architecture they enable is likely to foster the use of service composition techniques, since standardization removes many of the hurdles that limited the widespread adoption of conventional composition technologies (a.k.a workflows). In other words, not only standardization efforts in the service composition space seem to be much more successful than analogous efforts in the workflow domain, but (perhaps more importantly) it is also likely that service composition techniques- unlike workflow techniques- will indeed be widely used in practice. In summary, what this mean is that many services will come with a detailed description of their interface, protocols, and composition.

The second factor is that Web services specifications are ideally positioned to enable analyst to define metrics that are related to how customers perceive the quality of the services delivered by a company and to correlate them with the internal executions of the Web service. In fact, interfaces and business protocols describe the external interaction of a service with its customers, while the composition describes the internal implementation (assuming that the service is composite). Therefore, the business protocol is the basis on top of which business metrics are defined, while service composition specifications will be used to perform the correlation between business metrics and internal implementation.

To see the problem more concretely, consider again the chip manufacturer Web service mentioned in the introduction. If customers need to execute a conversation with this Web service to achieve their goal, then the conversation is what the provider should care about, as the quality of the conversation is what the customer will base its judgment on. For example, the customer may consider a conversation as being of high quality and successful if it ends in a certain state (e.g., the *goodsDelivered* state), if the time distance between certain two messages exchanged in the same conversation (e.g., between the invocation of the *shipOrder* operation and its reply confirming the shipment) is below a certain threshold, and if the total discount offered by the provider (which can be

identified by looking at one of the parameters exchanged in one of the messages) is above a certain other threshold. Besides being what is perceived as high quality by the customer, the above condition can be part of the *Service Level Agreement* (SLAs) between the customer and the service provider. This means that the service provider, in order to assess the quality of the service delivered to the customer, will need to define a function that executes on top of the conversation data (that, as we have seen, is collected by the Web services management infrastructure) and test whether a certain condition (such as the one described above) holds. Since the conversations are described in a standard way by the service interface and business protocol, the analyst is aware of the operations offered by a service, of the possible states in which a conversation can be, and of the data exchanged with each message. Therefore, it is possible to define such functions by starting from the external specifications of the Web service[1]. The way these functions are specified and implemented depends on how the Web services standards and the execution data are mapped into the data model of the Web service management system, but the concepts are generally applicable. We refer the reader to [4] for more details about how the external quality of a service can be measured.

Another opportunity provided by Web services consists in correlating business metrics defined on top of the conversations with the internal execution of the Web service. This is related to the issue of correlating *business management* with *application management*. Business management involves managing to business metrics (e.g., revenue, number of completed orders, etc) and to business objectives (e.g., revenue targets, order targets, etc). Application management involves monitoring and managing an application from an IT perspective (e.g., performance and availability). Mapping these two domains has never been easy as application management data is too low level to be directly translated into information relevant at the business management level. The increasing use of business processes to implement and orchestrate Web services makes this gap smaller. Service level measurements such as operation response times, SLA violations, and comparisons between two supplier Web services are all much closer to business-level metrics than traditional application measurements. Furthermore, the management platform in Web services is aware that a certain process (composition) has been executed to support a certain conversation. Therefore, it is possible to identify characteristics of the composition that are common across, for example, all low-quality executions. This can be done by analyzing the data collected by the platform with simple statistical techniques or with data mining algorithms such as decision trees. Therefore, not only Web services allow visibility into business level metrics, but allows correlating external interactions with internal executions, to determine which are the causes that lead to executions that do not satisfy the customers' requirements. For example, the statistical analysis may determine that the execution does not satisfy the quality criteria described above whenever a certain shipper is used, or whenever the order is performed on a certain day. This information is immensely useful for the service provider in order to improve the quality of the services offered to its customers. This advantage of Web services is often overlooked but it might play a crucial role in practice.

[1] In the management literature, the term *Service Level Management* (SLM) is used to refer to management done at the interface of an application as opposed to management done at the level of application internals [16].

3.4 Potential future standards

This section discusses possible extensions to current Web services standards that can further facilitate the business-level visibility of an enterprise IT infrastructure using Web services.

- **Measurements of the state.** Web services interfaces such as WSDL express the operations exposed by a Web service to its clients. However, they are not used to expose state information or measurements to clients. The state of a Web service can be considered as a set of attributes, some of which are static and some of which are dynamic. Static attributes are those whose values do not change over time. For example, the port reference at which operations are made available is a relatively static attribute. Other attributes are dynamic and their values change every time they are queried. For example, the anticipated response time of an operation or the state of a conversation change from time to time and from client to client. In either case, a client would benefit from querying these attributes both before composition and during execution to get a better understanding of a Web service's quality of service.
- **Events.** When two or more Web services interact with each other, a problem in one can affect the service offered by another. For example, if the execution of an operation in one Web service depends on the results of an operation from a different Web service, then the performance of the latter has an effect on the performance of the former. In traditional distributed applications, unanticipated problems are notified by application components through events. A management system that listens to these events takes decisions on how to isolate the problems without propagating its effects to other components. Similar mechanisms are needed for Web services. It should be possible for one Web service to generate events that are intended for another Web service (or for another Web service's management system).
- **Policies.** Policies are a convenient mechanism to change the behavior of a system. Policies have been used to indicate what a management system might or might not do (authorization policies) and must or must not do (obligation policies) [15]. For example, a policy can be used to say that no more than five unsuccessful login attempts should be permitted on a system. Policies are also useful between interacting Web services. They can be used by a Web service to inform clients about the constraints under which it operates (e.g., to inform that operations will not be accessible during a particular set of maintenance hours or that all messages exchanged should be encrypted in a certain manner). Similarly, clients can indicate their preferences and constraints to a Web service using policies.
- **Service level agreements.** Another mechanism for expressing constraints (or desired quality guarantees) between Web services is through the use of SLAs. SLA monitoring and enforcement requires the SLAs themselves, the measurements needed to measure SLAs [13], and their outcomes to be exchanged between interacting clients and Web services. To certain extent, this is what the CPA in ebXML is intended to do, although ebXML does not explicitly mention non-functional properties. To cover the entire range of constraints on multi-party conversations, the SLAs must have the possibility of including quality of service information.

Standards such as ARM and SNMP that have been traditionally used for communicating the above forms of data structures between components in an intra-enterprise

application management scenario. However, they are insufficient to define the above interfaces and protocols in an inter-enterprise application management scenario. The reasons are as follows:

- These standards have been primarily designed with the assumption that all the available management information and controls can be exposed between an application and its management system. This is not true in the case of Web services that work across enterprise boundaries. Web services management standards have to deal with multi-party interactions while resolving the limited end-to-end visibility and control one has over the other's services.
- These standards have not been developed to work over the Internet or to manage applications as complex as Web services. For example, ARM is a simple Java/C API and requires both the sender and the receiver of events to be within the same process. SNMP is a management protocol that lacks security and can only handle exchange of simple name-value pairs of attributes. None of these standards are built on the top of Internet protocols or Web services standards.
- Since there are a large number of management standards and systems, every Web service in a collection of interacting Web services can potentially use different standards and management systems to accomplish its management. However, for the purpose of interoperability, Web services should be capable of managing their relationships with each other by exchanging measurements, events, policies, and SLAs in a standardized manner that hides the heterogeneity in internal management systems. This requires a richer set of Web services management standards.

Currently there are no Web services standards aimed at addressing these issues. However, some of them are slowly beginning to take shape (e.g., WS-Policy [10]). Once defined, these standards will improve the visibility a business has into its relationships with partners.

4 Conclusions

This paper has shown how Web service technology can be used to have a comprehensive visibility on the IT infrastructure of a company. On the one hand, management *through* Web services provide visibility at the IT level on all applications, thanks to the fact that Web services reduce heterogeneity. On the other hand, management *of* Web services allow analysts to achieve visibility at the business level. This is possible for two main reasons: the first is that Web services are at a higher level of abstractions, closer to the business. As such, Web service executions are strictly related to business operations. The second is that Web service descriptions are fairly rich, and include in particular the specifications of the conversations supported by a service as well as the composition logic used to implement the service. This is different from conventional middleware services, where conversations and compositions were not described, and certainly were not described in a standard way. We have shown how standardized description of conversations and compositions are the starting point for defining business-level metrics on Web services and for correlating such metrics with the internal execution of the service, thereby providing visibility at the business level and identification of the causes

that lead to low quality executions. Finally, we have emphasized the role that forthcoming standards may play in this area. In summary, Web services bring an entirely new perspective on the scope and capability of application management, a perspective that is likely to profoundly impact the way companies assess their IT infrastructure and its relation to the business.

References

1. T. Bellwood, L. Climent, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI Version 3.0, July 2002.
2. B. Benatallah, F. Casati, F. Toumain, and R. Hamadi. Conceptual modeling of web services conversations. In Johann Eder et al, editor, *CAiSE*, Lecture Notes in Computer Science. Springer, 2003.
3. Candle. *ETEWatch: End-to-end Response Time Monitoring*. http://www.candle.com.
4. F. Casati, E. Shan, U. Dayal, and M. Shan. Business-oriented management of web services. *Communications of the ACM*. To appear.
5. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. W3C Technical Note: Web Services Description Language (WSDL). Technical report, W3C, March 2002.
6. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services. version 1.1. Technical report, Mar. 2003.
7. D. Grigori et al. Business process intelligence. *Computers in Industry. Special Issue on Process/Workflow Mining*, 2003. To appear.
8. Distributed Management Task Force. *Common Information Model*. http://www.dmtf.org.
9. Don Box et al. *Simple Object Access Protocol (SOAP) 1.1*, May 2000. W3C Note. http://www.w3.org/TR/SOAP/.
10. Don Box et al. *Web Services Policy Framework (WS-Policy)*. IBM, Microsoft, BEA, SAP, Dec. 2002. http://www-106.ibm.com/developerworks/library/ws-polfram/.
11. F. Cabrera et al. *Web Services Coordination (WS-Coordination), 9 August 2002*. IBM, Microsoft, BEA. http://www.ibm.com/developerworks/library/ws-coor/.
12. Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat. EtE: Passive End-to-end Internet Service Performance Monitoring. In *Proceedings of the USENIX Annual Technical Conference*, Monterey, CA, USA, June 2002.
13. R. Hauck and H. Reiser. Monitoring quality of service across organizational boundaries. In *Trends in Distributed Systems: Towards a Universal Service Market. Proceedings of the third International IFIP/GI Working Conference, USM 2000*, Sept. 2000.
14. The Open Group. *Application Response Measurement. Version 1.2*. http://www.opengroup.org/management/arm.htm.
15. M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333, 1994.
16. R. Sturm, W. Morris, and M. Jander. *Foundations of Service Level Management*. Sams, 2000.
17. Sun Microsystems. *Java Management Extensions. Version 1.2.*, Dec. 2002.