What Can Web Services Bring To Integrated Management?

Aiko Pras
University of Twente, The Netherlands

Jean-Philippe Martin-Flatin
NetExpert, Switzerland

Since the turn of the millennium, Web services have pervaded the middleware industry, despite their technical limitations, their ongoing standardization, the resulting lack of stable standards, the different meanings of the term *service* to different people, and the fact that marketing forces have blurred the technical reality that hides behind this term. What is so special about Web services and their underlying Service-Oriented Architecture (SOA)?

As a major technology on the software market, Web services deserve to be investigated from an integrated management perspective. What does this middleware technology bring to management applications? Is it yet another way of doing exactly the same thing? Or does it enable management software architects to open up uncharted territories for building a new generation of management solutions? Can it help us manage networks, systems and services with unprecedented flexibility, robustness and/or scalability?

In this chapter, we first present the technical motivation for shifting toward Web services. In Sections 2, 3 and 4, we describe different facets of this middleware technology: its architecture, its protocols, and the main standards that make up the so-called *Web services stack*. In Sections 5 and 6, we show that Web services can be used in integrated management in an evolutionary manner. The example studied here is network monitoring. We propose a fine-grained Web service mapping of SNMP operations and managed objects, and show that the same management tasks can be performed with this new middleware to implement network monitoring. In Section 6, we go back to the rationale behind SOA and compare it with fine-grained Web services. This leads us to propose another perspective, more revolutionary, where services are coarse-grained, wrapping up autonomous programs rather than getting and setting managed objects. We analyze the challenges posed by these coarse-grained Web services to management applications, and the changes that they require in today's management habits. Finally, we give examples showing how management platforms could gradually migrate toward SOA in the future.

This paper has been published in the Elsevier Handbook of Network and System Administration, editors: J. Bergstra and M. Burgess.
It should be cited as follows: Pras, A. and Martin-Flatin, J.P.: *What Can Web Services Bring To Integrated Management?* In: Handbook of Network and System Administration. Elsevier, Amsterdam. ISBN 978-0-444-52198-9, 2007

1 Motivation

The charm of novelty should not obliterate the fact that it is unwise to change a working solution. Before jumping on the bandwagon of Web services, supposedly the latest and greatest technology in the middleware market, let us review the current state of affairs in integrated management and investigate whether we need a new solution at all.

1.1 Command Line Interface

Our first observation is that today's management of networks, systems and services still relies to a great extent on protocols and technologies that were developed several decades ago. For instance, the most frequently used management tool still seems to be the Command Line Interface (CLI), which was introduced in the early days of computing (and later networking) to manage computer systems and network devices. In the Internet world, network operators still often rely on this rather crude interface to configure their equipment. The CLI is not only used interactively (in *attended mode*) when Network Operations Center (NOC) staff log into remote systems to manually monitor and modify their operation, but also in *unattended mode* when management scripts automatically connect to remote machines (e.g., using `expect` [47]) to check their configuration and operation and possibly alter them. In large organizations, these scripts are essential to manage and control the Information Technology (IT) infrastructure in a cost-effective and reliable manner.

Unfortunately, different CLIs use different syntaxes: there is no widely adopted standard. Not only do CLIs vary from vendor to vendor, but also from product line to product line for a given vendor, and sometimes even between versions of a given product. Scripts are therefore highly customized and equipment specific, and considerable investments are needed to maintain them as networks, systems and services evolve and come to support new features.

1.2 SNMP Protocol

In the 1970s and early 1980s, when large networks began to be built all over the world, this interoperability issue became increasingly serious. In the 1980s, the International Organization for Standardization (ISO) and the Internet Engineering Task Force (IETF) proposed to address it by defining standard management protocols.

To date, the most successful of these protocols has been the IETF's Simple Network Management Protocol (SNMP), which is accompanied by a standard describing how to specify managed objects—the Structure of Management Information (SMI)—and a document defining standard managed objects organized in a tree structure—the Management Information Base (MIB-II). As the latter defines only a small number of managed objects, additional MIB modules soon appeared.

The first versions of these SNMP standards were developed in the late 1980s and became IETF standards in 1990 (they are now known as SNMPv1 and SMIv1 [85]). Soon afterward, they were implemented in many products and tools, and used on a very large scale, much larger than expected. In view of this success, the original plan to eventually migrate to ISO management protocols was abandoned.

Instead, the IETF decided to create a second version of the SNMP standards. This version should have better information modeling capabilities, improved performance and, most importantly, stronger security.

The work on information modeling progressed slowly but smoothly through the IETF standard track; SMIv2, the new version of the Structure of Management Information, became a proposed standard in 1993, a draft standard in 1996, and a standard in 1999 [56]. In addition, a new Packet Data Unit (PDU) was defined to retrieve bulk data (`get-bulk`).

Unfortunately, the original developers of SNMPv2 disagreed on the security model and failed to find a compromise. In 1994, two competing versions of SNMPv2 were standardized (SNMPv2p and SNMPv2u), which utterly confused the network management market. In 1996, a compromise was standardized (SNMPv2c) [85] with only trivial support for security ("community" string, i.e. the clear-text identification scheme implemented in SNMPv1). SNMPv2c brought several improvements to SNMPv1 and was widely adopted, but it did not address the market's main expectation: the support for professional-grade security.

In 1996, a new IETF Working Group was created, with different people, to address security issues in SNMP. This resulted in SNMPv3, which progressed slowly through the standard track and eventually became an IETF standard in 2003.

By the time this secure version of SNMP was released, the market had lost confidence in the IETF's ability to make SNMP evolve in a timely manner to meet its growing demands. To date, two decades after work on SNMP began in 1987, there is still no standard mechanism in SNMP for distributing management across multiple managers in a portable way... As a result, management platforms have to use proprietary "extensions" of SNMP to manage large networks.

In last resort, a new IETF group was formed in 2001, called the Evolution of SNMP (EoS) Working Group, to resume standardization work on SNMP. But it failed to build some momentum and the group was eventually dismantled in 2003, without any achievement.

Waiting for a New Solution

SNMPv1 was undoubtedly a great success, but it failed to evolve [79]. The frequent use of SNMP to manage today's networks should not hide the fact that many organizations also use the CLI and *ad hoc* scripts to manage their network infrastructure. This not only defeats the point of an open and standard management protocol, but also prevents the wide use of policy-based management, because CLI-based scripts alter the configuration settings supposedly enforced by Policy Decisions Points (PDPs, i.e. policy managers in IETF jargon). Today, the market is clearly waiting for a new solution for network management.

SNMP never encountered much success in systems and service management. Since its inception, service management has been dominated by proprietary solutions; it still is. The situation was similar in systems management until the early 2000s. Since then, the Web-Based Enterprise Management (WBEM) management architecture of the Distributed Management Task Force (DMTF) has built some momentum and is increasingly adopted, notably in storage area networks. The DMTF has focused mostly on information modeling so far. The communication model of WBEM can be seen as an early version of SOAP.

For large enterprises and corporations, the main problem to date is the integration of network, systems and service management. These businesses generally run several management platforms in parallel, each dedicated to a management plane or a technology (e.g., one for network management, another for systems management, a third one for service management, a fourth one for the trouble-ticket management system, and a fifth one for managing MPLS-based VPNs[1]). In these environments, management platforms are either independent (they do not communicate) or integrated in an *ad hoc* manner by systems integrators (e.g., events from different platforms are translated and forwarded to a single event correlator). For many years, the market has been waiting for a technology that would facilitate the integration of these platforms. Researchers have been working on these issues for years, but turning prototypes into reliable and durable products that can be deployed on a very large scale still poses major issues. The market is ripe now for a new technology that could make this integration easier.

1.3 From Specific To Standard Technologies

For many years, the IETF encouraged the engineers involved in its Working Groups to devise a taylor-made solution for each problem. This led to a plethora of protocols. SNMP was one of them: as network bandwidth, Central Processing Unit (CPU) power and memory were precious few resources in the late 1980s, their usage had to be fine-tuned so as not to waste anything.

Since then, the context has changed [53]. First, the market has learned the hard way what it costs to hire, train and retain highly specialized engineers. Second, considerably more network and computing resources are available today: there is no need anymore for saving every possible bit exchanged between distant machines. Third, software production constraints are not the same: in industry, a short time-to-market and reasonable development costs are considerably more important than an efficient use of equipment resources.

For integrated management, the main implications of this evolution are twofold. First, it is now widely accepted that using dedicated protocols for exchanging management information does not make sense anymore: wherever possible, we should reuse existing and general-purpose transfer protocols (pipes) and standard ways of invoking service primitives. Second, instead of using specific technologies for building distributed management applications, we should strive to use standard middleware.

Two platform-independent middleware technologies encountered a large success in the 1990s: the Common Object Request Broker Architecture (CORBA) [67] and Java 2 Enterprise Edition (J2EE) [13]; the latter is also known as Enterprise Java Beans (EJBs). Both made there way to some management platforms for distributing management across several machines, but none of them had a major impact on the integrated management market. Why should Web services do any better?

1. MPLS stands for Multi-Protocol Label Switching, VPN for Virtual Private Network.

1.4 WEB SERVICES

The main difference between Web services and previous types of middleware is that the former support a loose form of integration (*loose coupling*) of the different pieces of a distributed application (*macro-components*[1]), whereas the latter rely on their tight integration (*tight coupling*).

The macro-components of a loosely integrated application can work very differently internally: they just need to be able to share interfaces and communicate via these interfaces. What happens behind each interface is assumed to be a local matter. Conversely, the macro-components of a tightly integrated application are internally homogeneous. They are all CORBA objects, or EJBs, etc.

The concept of loose integration is not new. Not only that, but nothing in CORBA (or J2EE) prevents application designers from supporting loose integration, where CORBA objects (or EJBs) serve as interfaces to entire applications; indeed, examples of wrapper CORBA objects (or EJBs) are described in the literature, and some can be found in real-life. But their presence is marginal in the market because this is not how the object-oriented middleware industry has developed.

What is new with Web services is that loose integration is almost mandatory, because Web services are grossly inefficient for engineering the internals of an application. Reengineering a CORBA application with Web services operating at the object level would be absurd: performance would collapse. What makes sense is to wrap up an existing CORBA application with one or several Web service(s), to be able to invoke it from a non-CORBA application.

What is so great about loose integration?

First, customers love it because it preserves their legacy systems, which allows them to cut on reengineering costs. In the past, people working in distributed computing and enterprise application integration had the choice between proprietary middleware (e.g., .NET or DCOM, the Distributed Object Component Model) or interoperable middleware (e.g., CORBA or J2EE). These different types of middleware all assumed a tight integration of software; everything had to be an object: a CORBA object, an EJB, a DCOM component, etc. Using such middleware thus required either application reengineering or brand new developments. Only the wealthiest could afford this on a large scale: stock markets, telecommunication operators and service providers prior to the Internet bubble burst, etc. With Web services, legacy systems are no longer considered a problem that requires an expensive reengineering solution. On the contrary, they are a given, they are part of the solution. When distributed computing and enterprise application integration make use of Web services, not everything needs to be a Web service.

Second, vendors also love loose integration because it allows them to secure lucrative niche markets. Externally, each macro-component of the distributed application looks like a Web service. But internally, it can be partially or entirely proprietary. Interoperability is only assured by interfaces: vendors need not be entirely compatible with other vendors. The

1. In Section 6.2, “Loose Coupling”, we will present examples of macro-components for management applications.

business model promoted by this approach to distributed computing and enterprise application integration is much closer to vendors' wishes than CORBA and J2EE.

In addition to loose integration, Web services present a number of generic advantages that pertain to middleware, eXtensible Markup Language (XML) or the use of well-known technologies [1][53]. For instance, they can be used with many programming languages and many development platforms, and they are included in all major operating systems. Even calling Web services from a Microsoft Excel spreadsheet is easy, and the idea of building simple management scripts within spreadsheets can be appealing [43]. Last but not least, as there are many tools and many skilled developers in this area, implementing Web services-based management applications is usually easier and less expensive than developing SNMP-based applications.

2 Introduction to Web Services

Let us now delve into the technical intricacies of Web services. This technology is defined in a large and growing collection of documents, collectively known as the *Web services stack*, specified by several consortia. In this section, we present the big picture behind Web services, describe the three basic building blocks (SOAP, WSDL and UDDI) and summarize the architecture defined by the World-Wide Web Consortium (W3C). In the next two sections, we will present a selection of standards that are also part of the Web services stack and deal with more advanced features: management, security, transactions, etc.

2.1 Communication Patterns and Roles

Web services implement the Producer-Consumer design pattern [31]. When a Web service consumer binds to a Web service provider (see Figure 1), the consumer sends a request to the provider and later receives a response from this provider. This communication is generally synchronous: even though SOAP messages are one-way, SOAP is often used for synchronous Remote Procedure Calls (RPCs), so the application running the Web service consumer must block and wait until it receives a response [44]. Asynchronous versions of SOAP also exist [73] but are less frequent.

The first difficulty here is that the consumer usually knows the name of the service (e.g., it can be hard-coded in the application), but ignores how to contact the provider for that service. This indirection makes it possible to decouple the concepts of logical service and physical service. For instance, a given service can be supported by provider P1 today and provider P2 tomorrow, in a transparent manner for the applications. But this flexibility comes at a price: it requires a mechanism to enable the consumer to go from the name of a service to the provider for that service.

With Web services, the solution to this problem is illustrated by Figure 1. First, the consumer contacts a service description repository and requests a specific service. Second, the repository returns a handle to the service provider. Third, the consumer can now directly bind to the provider and send a Web service request. Fourth, the provider sends back a Web service response.

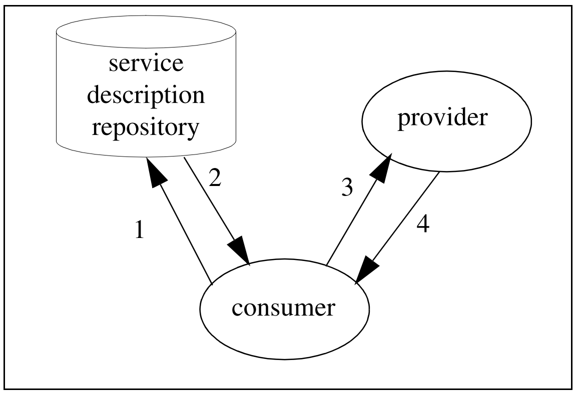


Figure 1: Finding a service offered by just one provider

The second difficulty is that a given service may be supported not by one provider at a time, as before, but by multiple providers in parallel. For instance, different providers may support a variable Quality of Service (QoS) [93], or charge different prices for the same service.

In this case, the consumer typically uses a broker instead of a repository. First, the consumer contacts a trusted broker and requests the best offering for a given service, with parameters that specify what “best” means. Second, the broker builds an ordered list of offers (by contacting repositories, providers or other brokers, or by using locally cached information). Third, the consumer selects an offer and sends a Web service request to that provider. Fourth, the provider sends back a Web service response.

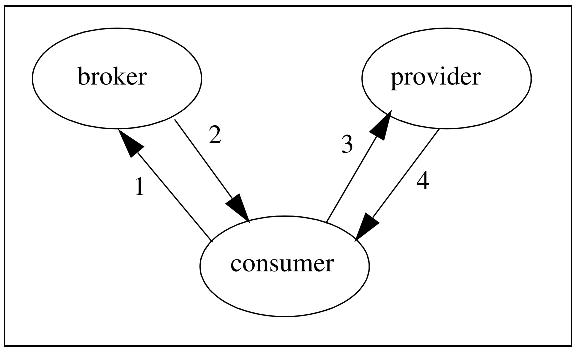


Figure 2: Finding a service offered by multiple providers

2.2 SOAP PROTOCOL

SOAP [32][33] is a protocol that allows Web applications to exchange structured information in a distributed environment. It can be viewed as a simple mechanism for turning service invocations into XML messages, transferring these messages across the network, and translating them into service invocations at the receiving end [1].

SOAP can be used on top of a variety of transfer protocols: HyperText Transfer Protocol (HTTP) [29], Blocks Extensible Exchange Protocol (BEEP) [68], etc. The SOAP binding used by most applications to date specifies how to carry a SOAP message within an HTTP entity-body; with this binding, SOAP can be viewed as a way to structure XML data in an HTTP pipe between two applications running on distant machines.

A SOAP message is an *envelope* that contains a *header* and a *body*. The header is optional and carries metadata; the body is mandatory and includes the actual application payload. Both the header and the body may be split into *blocks* that can be encoded differently. A SOAP message is used for one-way transmission between a *SOAP sender* and a *SOAP receiver*, possibly via *SOAP intermediaries*. Multiple SOAP messages can be combined by applications to support more complex interaction patterns such as request-response. SOAP also specifies how to make RPCs using XML.

In its original definition, the SOAP acronym stood for Simple Object Access Protocol. In the version standardized by the W3C, SOAP is no longer an acronym.

2.3 WEB SERVICES DESCRIPTION LANGUAGE

The Web Services Description Language (WSDL) [16][17] is a standard XML language for describing Web services. Each Web service is specified by a *WSDL description* that separates the description of the abstract functionality offered by a service (*type* and *interface* components) from concrete details of the service description (*binding* and *service* components); the latter defines how and where this functionality is offered. Multiple WSDL descriptions can be published in a single WSDL file, which is sometimes called a *WSDL repository.*

The current version of the standard is WSDL 2.0, released in March 2006. The previous version, WSDL 1.1, was standardized in March 2001. Both are widely used today. These two versions of WSDL are quite different. For instance, the *message* component was made obsolete in WSDL 2.0. The *port type* component in WSDL 1.1 evolved into the *interface* component in WSDL 2.0. The *port* component in WSDL 1.1 was renamed *endpoint* in WSDL 2.0. In Section 5.4, we will present the different components of a WSDL description while we study a detailed example.

If we compare Web services with CORBA, the WSDL language corresponds to the CORBA Interface Definition Language (IDL); a WSDL repository is similar to an Interface Repository in CORBA; applications can discover Web services in a WSDL repository and invoke them dynamically, just as CORBA applications can discover an object interface on the fly and invoke it using the Dynamic Invocation Interface. However, the underlying development paradigm is quite different. CORBA requires a developer to create an interface before implementing clients and servers that match this interface, whereas WSDL descriptions may be provided after the initial creation of the service. Moreover, it is not necessary to store all WSDL descriptions in a designated repository (whereas it is mandatory in CORBA): the Web service provider may also choose to serve a WSDL description at the physical location where the service is offered.

2.4 UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION

Universal Description, Discovery and Integration (UDDI) [87][88] is a standard technology for looking up Web services in a registry. A UDDI registry offers "a standard mechanism to classify, catalog and manage Web services, so that they can be discovered and consumed" [21]. Here are typical scenarios for using a UDDI registry [21]:

- Find Web services based on an abstract interface definition.
- Determine the security and transport protocols supported by a given Web service.
- Search for services based on keywords.
- Cache technical information about a Web service, and update this cached information at run-time.

Three versions of UDDI have been specified to date. UDDIv1 is now considered historic. The market is migrating from UDDIv2 [6] to UDDIv3 [21]; both are currently used. The main novelties in UDDIv3 include the support for private UDDI registries [1] and registry interaction and versioning [87].

The XML schema that underlies UDDI registries consists of six elements [21]:

- *businessEntity* describes an organization that provides Web services; this information is similar to the yellow pages of a telephone directory: name, description, contact people, etc.;
- *businessService* describes a collection of related Web services offered by an organization described by a `businessEntity`; this information is similar to the taxonomic entries found in the white pages of a telephone directory;
- *bindingTemplate* provides the technical information necessary to use a given Web service advertised in a `businessService`; it includes either the access point (e.g., a Uniform Resource Locator, URL) of this Web service or an indirection mechanism that leads to the access point; conceptually, this information is similar to the green pages of a telephone directory;
- *tModel* (technical model) describes a reusable concept: a protocol used by several Web services, a specification, a namespace, etc.; references to the `tModel`'s that represent these concepts are placed in a `bindingTemplate`; as a result, `tModel`'s can be reused by multiple `bindingTemplate`'s;
- *publisherAssertion* describes a relationship between a `businessEntity` and another `businessEntity`; this is particularly useful when a group of `businessEntity`'s represent a community whose members would like to publish their relationships in a UDDI registry (e.g., corporations with their subsidiaries, or industry consortia with their members);
- *subscription* enables clients to register their interest in receiving information about changes made in a UDDI registry. These changes can be scoped based on preferences provided by the client.

There are well-known problems with UDDI [1][54]: performance, scalability, taxonomies, etc. In practice, this technology is not widely used for publishing and discovering Web services; when it is used, it is often "extended" with proprietary enhancements. Web service discovery is still a very active research area, particularly in the field of semantic Web services (see Section 3.7).

2.5 Web Services Architecture

The Web Services Architecture (WSA) was an attempt by the W3C to "lay the conceptual foundation for establishing interoperable Web services" [9]. This document defines the concept of Web service, architectural models (the message-oriented model, the service-oriented model, the resource-oriented model and the policy model), relationships that are reminiscent of Unified Modeling Language (UML) relationships, and a hodgepodge of concepts grouped under the name "stakeholder's perspectives": SOA, discovery, security, Peer to Peer (P2P), reliability, etc.). Management is quickly mentioned in Section 3.9 of [9], but the W3C seems to have paid little attention to this issue thus far.

A more interesting attempt to define an architecture for Web services can be found in Section 5.3 of [1].

3 Advanced Web Services

3.1 Standardization

Because interoperability is crucial to Web services, their standardization has been of key importance since their inception. So far, two consortia have been particularly active in this field: the W3C and the Organization for the Advancement of Structured Information Standards (OASIS). More recently, other industrial consortia have worked in this field and produced standards or so-called "best practices", including the Web Services Interoperability Organization (WS-I), the Open Grid Forum (OGF), the Distributed Management Task Force (DMTF) and Parlay.

3.2 Interoperability

In April 2006, WS-I specified the Basic Profile 1.1 [4], a set of implementation guidelines to help people build interoperable Web services. This document contains "clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability". The main focus is on SOAP messaging and service description. In particular, the guidelines for using SOAP over HTTP are important. A number of Web services standards refer to this Basic Profile.

3.3 Composition

A *composite Web service* is a group of Web services that collectively offer some functionality. Each Web service taking part in a given composite Web service is known as a *participant*. Each participant can itself be a composite Web service, which can lead to a nested structure of services of arbitrary depth. The process of bundling Web services together to form a composite Web service is known as *service composition*. A Web service that is not composite is called *atomic* (or *basic* [1]).

There are many ways of specifying composite Web services, expressing constraints between participants, and controlling their synchronization and execution. They are usually grouped into two categories: static and dynamic. Static composition is the simplest form of composition: composite Web services are defined in advance, once and for all.

A generally more useful, but also considerably more complex, form of composition is the dynamic composition of Web services. In this case, Web services are composed at run-time: the participants of a composite Web service are chosen dynamically based on a variety of concerns: availability, load-balancing, cost, QoS, etc. The main challenge here is to build composite Web services semi-automatically or in a fully automated manner. Different techniques have been proposed in the literature [78]; they borrow heavily from other existing and well-established fields (e.g., workflow systems or artificial intelligence planning).

An interesting issue in Web service composition is how to compute the QoS of a composite Web service when we know the QoS offered by each participant Web service [71]. Complex composite services can lead to situations where the error bars become ridiculously large when simple multiplicative techniques are used; reducing these error bars is non-trivial.

Whether a Web service is atomic, statically composed or dynamically composed is transparent to its consumers. Service composition is a key characteristic of Web services: their success in e-business owes a great deal to service composition.

3.4 Orchestration and Choreography

For many years, the terms *orchestration* and *choreography* were used interchangeably in the Web services community. There is still no consensus on their exact meaning. The W3C proposes to distinguish them as follows [34].

Orchestration

In the realm of Web services, orchestration is about controlling the execution of a composite Web service, viewed as a business process. It specifies how to control, from a central point, the execution and synchronization of all the participant Web services (e.g., by triggering the execution of Web service WS2 when Web service WS1 completes).

Orchestration is usually implemented by an executable process that interacts with all Web services involved. Orchestration languages make it possible to define the order in which the participant Web services should be executed, and to express the relationships between Web services in a workflow manner.

The most famous orchestration language to date is probably WS-BPEL (Web Service Business Process Execution Language) [2], an OASIS draft standard formerly known as BPEL4WS (Business Process Execution Language for Web Services) [22] and already widely used. The main objectives of WS-BPEL are to describe process interfaces for business protocols and define executable process models.

CHOREOGRAPHY

Choreography is a declarative way of defining how participants in a collaboration (e.g., a composite Web service) should work together, from a global viewpoint, to achieve a common business goal. The objective here is to define a common observable behavior by indicating in a contract where information exchanges occur, what rules govern the ordering of messages exchanged between participants, what constraints are imposed on these message exchanges, and when the jointly agreed ordering rules are satisfied [45].

The need for a high-level contract, independent of execution and implementations, stems from the fact that enterprises are often reluctant to delegate control of their business processes to their partners when they engage in collaborations. Choreographies enable partners to agree on the rules of collaboration for a composite Web service without specifying how each participant Web service should work. Each participant is then free to implement its own portion of the choreography as determined by the global view [45]. For instance, one participant may use WS-BPEL for orchestrating business processes, while another participant may use J2EE.

The choreography language standardized by the W3C is called the Web Services Choreography Description Language (WS-CDL) [45]. With this language, it is supposedly easy to determine whether each local implementation is compliant with the global view, without knowing the internals of these local implementations.

3.5 TRANSACTIONS

Classic transactions are short-lived operations that exhibit the four ACID properties: Atomicity, Consistency, Isolation and Durability. These properties are fulfilled by transactional systems using a coordinator and a two-phase protocol called *two-phase commit*. During the first phase, the coordinator contacts each participant in the transaction and asks them to make local changes in a durable manner, so that they can either be rolled back (i.e., cancelled) or committed (i.e., confirmed) later. During the second phase, we have two options. If a failure occurred on any of the participants during phase one, the coordinator sends an abort to each participant and all changes are rolled back locally by the participants; the transaction then fails. Otherwise, the coordinator sends a commit to each participant and all changes are committed locally by the participants; the transaction completes successfully. This two-phase protocol is blocking: once they have completed phase 1 successfully, the participants block until the coordinator sends them a commit (phase 2).

This *modus operandi* is not applicable to Web services [48][70]. First, Web services are not necessarily blocking (e.g., when they rely on an asynchronous implementation of SOAP). Second, in the realm of Web services, transactions are usually long-lived; using the above-mentioned protocol may cause resources to remain unavailable for other transactions over extended periods of time, thereby impeding concurrency. Third, when composite Web services are used across multiple organizations, security policies often prevent external entities from hard-locking a local database (allowing it would open the door to denial-of-service attacks, for instance). Hence another mechanism is needed.

Several techniques were devised in the past [48] to release early the resources allocated by a long-lived transaction, and allow other transactions to run in the meantime. In case of subsequent failure, compensation mechanisms make it possible to bring the transactional system to the desired state (e.g., a payment by credit card can be reimbursed). However, these compensations do not guarantee all ACID properties. This is the approach generally adopted for Web services [1]. Another possibility is to subdivide long-lived transactions into independent short-lived transactions that can run independently as classic atomic transactions.

Detailing the support for transactions in a Web service environment would require an entire book. The OASIS consortium is the main standardization actor in this arena. It adopted an approach similar to Papazoglou's [70], where transactions are part of a bigger framework that defines business protocols, coordination protocols, transactions, orchestration of business processes, choreography, etc. Several building blocks have already been standardized: the business transaction protocol (WS-BTP), atomic transactions (WS-Atomic-Transaction), business activities (WS-BusinessActivity), coordination protocols (WS-Coordination), transactions (WS-Transaction), etc. These standards are still evolving and should not be considered stable yet. Readers interested in learning more about transactions in the context of Web services are referred to the Web site of the OASIS Web Services Transaction Technical Committee [65].

3.6 SECURITY

WS-Security specifies how to secure the SOAP protocol. By supporting end-to-end application-level security, it nicely complements HTTPS, the secure version of HTTP, which can only secure communication on a hop-by-hop basis (with HTTPS, each intermediary can decrypt and re-encrypt the HTTP message) [1]. Whereas the entire payload is encrypted with HTTPS, WS-Security makes it possible to encrypt only one block of the SOAP body (e.g., bank details).

WS-Security was standardized by OASIS in 2004 (version 1.0) and upgraded in 2006 (version 1.1). It consists of a main document, known as SOAP Message Security, and several companion documents.

SOAP Message Security 1.1 [60] specifies SOAP extensions for sending security tokens as part of a SOAP message, guaranteeing message integrity (payload) and ensuring message confidentiality. This specification makes it possible to secure Web services with various security models: Public Key Infrastructure (PKI), Kerberos, Secure Sockets Layer (SSL), etc. *Per se*, it does not provide a complete security solution for Web services; instead, "it can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and security technologies" [60].

In version 1.1, there are six companion documents: Username Token Profile (to identify the requestor by using a username, and optionally authenticate it with a password), X.509 Certificate Token Profile (to support X.509 certificates), the SAML Token Profile (to use Security Assertion Markup Language assertions as security tokens in SOAP headers), the Kerberos Token Profile (to use Kerberos tokens and tickets), the REL Token Profile (to use the ISO Rights Expression Language for licenses), and the SwA Profile (to secure SOAP messages with Attachments).

3.7 SEMANTIC WEB SERVICES

In 2001, Berners-Lee launched the Semantic Web with a powerful statement that has become famous: "The Semantic Web will bring structure to the meaningful content of Web pages" [7]. The idea was to add more metadata to Web pages (using XML) to enable software agents to parse them automatically and "understand" their hierarchically structured contents (containment hierarchy), as opposed to seeing only flat structures and keywords out of context. The main building blocks defined so far for the Semantic Web are the Resource Description Framework (RDF) [51] and the Web Ontology Language (OWL) [58].

In the realm of Web services, this vision translated into the concept of semantic Web services [59]. Whereas Web services focus mostly on interoperability, semantic Web services complement interoperability with automation, dynamic service discovery and dynamic service matching. Semantic Web services address the variety of representations (e.g., different names for the same concept, or different signatures for semantically equivalent operations) by using automated or semi-automated ontology-matching mechanisms [69]. This makes it possible, for instance, to compose Web services at run-time with increased flexibility. OWL-S (Web Ontology Language for Services) [52] is probably the most famous language to date for specifying semantic Web services.

4 WEB SERVICES FOR MANAGEMENT, MANAGEMENT OF WEB SERVICES

In this section, we review the main standards pertaining to (i) the use of Web services for managing networks, systems and services, and (ii) the management of Web services.

4.1 OASIS: WEB SERVICES FOR DISTRIBUTED MANAGEMENT (WSDM)

Unlike the W3C, OASIS has already paid much attention to management issues in the Web services arena. In 2005, the Web Services Distributed Management (WSDM) Technical Committee [63] standardized version 1.0 of a group of specifications collectively called WSDM. In August 2006, version 1.1 was released. To date, these standards are probably the most relevant to the integrated management community as far as Web services are concerned. Let us describe the main ones.

Management Using Web Services (MUWS) [11][12] is a two-part series of specifications that define how to use Web services in distributed systems management. First, MUWS defines a terminology and roles. As prescribed by WSDL and WS-Addressing [10], a Web service is viewed as an aggregate of *endpoints*. Each endpoint binds a Web service interface (described by a WSDL `portType` element) to an address (URL). Each interface describes the messages that can be exchanged and their format [91]. A *manageable resource* is a managed entity (e.g., a network device or a system). A *manageability endpoint* is bound to a manageable resource and can be accessed by a Web service consumer. It acts as a gateway between Web services and a given manageable resource. It can interact with the manageable resource either directly or via an agent (e.g., an SNMP agent). A *manageability consumer* is a Web service consumer, a Web-enabled managing entity (e.g., a management application running on a remote network management system). After discovering Web service endpoints, it can exchange SOAP messages with these endpoints to request management information (monitoring in polling mode), to subscribe to events (e.g., if a PC needs to report that the temperature of its mother board is too high), or to configure the manageable resource associated with a manageability endpoint [11].

Second, MUWS defines some rules that govern communication between manageability endpoints and consumers (i.e., between managing and managed entities). To discover the manageability endpoint that enables a given manageable resource to be accessed by a Web service, a manageability consumer must first retrieve an Endpoint Reference [10]; next, it can optionally retrieve a WSDL file, a policy, etc. Once the discovery phase is over, the manageability consumer (managing entity) can exchange messages with a manageability endpoint (managed entity) by using information found in the Endpoint Reference [11].

Third, MUWS specifies messaging formats to achieve interoperability among multiple implementations of MUWS (e.g., when management is distributed hierarchically over different domains [82] controlled by different managing entities) [12].

Management of Web Services (MOWS) [91] specifies how to manage Web services. It defines the concept of Manageability Reference (which enables a manageability consumer to discover all the manageability endpoints of a manageable resource), and a list of manageability capabilities (metrics for measuring the use and performance of Web services, operational state of a managed Web service, processing state of a request to a managed Web service, etc.) [91].

4.2 DMTF: WS-MANAGEMENT

WS-Management [23] was standardized by the DMTF in April 2006 to promote Web services-based interoperability between management applications and managed resources. It is independent of the WBEM management architecture and the Common Information Model (CIM).

First, WS-Management defines a terminology and roles. A *managed resource* is a managed entity (e.g., a PC or a service). A *resource class* is the information model of a managed resource; it defines the representation of management operations and properties. A *resource instance* is an instance of a resource class (e.g., a CIM object).

Second, WS-Management specifies mechanisms (i) to get, put, create, and delete resource instances; (ii) to enumerate the contents of containers and collections (e.g., logs); (iii) to subscribe to events sent by managed resources; and (iv) to execute specific management methods with strongly typed parameters [23].

The approach promoted by WS-Management corresponds to what we call fine-grained Web services in this chapter. In Section 5, we will see Web services (this time in the SNMP world) that similarly perform a `get` on an Object IDentifier (OID), i.e. a managed object.

The relationship between WSDM and WS-Management is not crystal clear. They do overlap but the scope of WSDM is much wider than that of WS-Management, so they are not really competitors (at least not for now). It seems that WS-Management could eventually provide a migration path for exchanging CIM objects between managing and managed entities, from the DMTF's pre-Web services and pre-SOAP "CIM operations over HTTP" to a Web services-based communication model for WBEM. In this case, WS-Management would only target the WBEM community, not the Web services community at large. This is however mere speculation and remains to be seen.

4.3 OASIS: WS-RESOURCE

In 2001, the Global Grid Forum (GGF), now known as the OGF, began working on the modeling of Grid resources. This led to the creation of the Open Grid Services Infrastructure (OGSI) Working Group. In 2002 and 2003, the concepts of Grid services and Web services gradually converged, as the specifics of Grids were separated from features that belonged in mainstream standardization. A number of GGF group members got involved in W3C and OASIS groups, and several ideas that had emerged within the GGF percolated to these groups. GGF focused on its core business (Grid-specific activities) and more generic standardization efforts moved to other standards bodies.

In early 2004, a group of vendors issued a draft specification for modelling and accessing persistent resources using Web services: Web Services Resource Framework (WSRF) version 1.0. It was inspired by OGSI and played an important role in the convergence of Web and Grid services. Shortly afterward, this standardization effort was transferred to OASIS, which created the WSRF Technical Committee [64] to deal with it. This led to the phasing out of OGSI and its replacement by a series of documents collectively known as WSRF. In April 2006, version 1.2 of WSRF was released by OASIS. It includes WS-Resource [30], WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup and WS-BaseFaults (for more details, see Banks [5]).

4.4 PARLAY-X

The multi-vendor consortium called Parlay has defined a number of open service-provisioning Application Programming Interfaces (APIs) to control various aspects of Intelligent Networks. At the same time the Open Service Access (OSA) group of the 3rd Generation Partnership Project (3GPP) was working on similar APIs that allowed applications to control a Universal Mobile Telecommunications System (UMTS) network. These two groups joined their activities and now common Parlay/OSA API standards exist. These APIs require detailed knowledge of the Intelligent Networks, however, which means that only a limited number of experts are able to use these APIs. The Parlay group therefore defined some easier to use APIs, at a higher level of abstraction. These APIs, called Parlay-X, are defined in terms of Web services. Examples of such services include “connect A to B”, “give status of X (on/off)”, “send SMS[1]”, “give location of mobile handset” and “recharge pre-paid card”. The users of a Parlay-X server do not need to know the details of the underlying network; the Parlay-X server translates the Web services calls into Parlay/OSA calls that are far more difficult to understand. It is interesting to note that Web services technologies were applied by the Parlay group long before the IETF or the Internet Research Task Force (IRTF) even started to think about these technologies.

1. Short Message System.

4.5 IETF AND IRTF-NMRG

In the summer of 2002, the Internet Architecture Board (IAB) organized a one-off Network Management Workshop to discuss future technologies for Internet management. One of the conclusions of this workshop was that time had come to investigate alternative network management technologies, in particularly those that take advantage of the XML technology [79].

Web services are a specific form of XML technology. The 11th meeting of the IRTF Network Management Research Group (NMRG) [38], which was organized three months after the IAB workshop, therefore discussed the possible shift toward XML and Web services-based management. Although many questions were raised during that meeting, the most interesting question was that of performance; several attendees expressed their concern that the anticipated high demands of Web services on network and agent resources would hinder, or even prohibit, the application of this technology in the field of network management. At that time, no studies were known that compared the performance of Web services to that of SNMP. Most attendees therefore preferred to take a simpler approach not based on Web services (e.g., JUNOScript for Juniper). The idea behind JUNOScript is to provide management applications access to the agent's management data, using a lightweight RPC mechanism encoded in XML. As opposed to SNMP, JUNOScript uses a connection-oriented transport mechanism (e.g., `ssh` or `telnet`). An advantage of JUNOScript is that related management interactions can be grouped into sessions, which makes locking and recovery relatively simple. In addition, contrary to SNMP, management information is not limited in size and can be exchanged reliably. JUNOScript provided the basis for a new IETF Working Group called NetConf. Details about NetConf can be found in an another chapter of this book.

Within the IRTF-NMRG, several researchers continued to study how to use Web services in network, systems and service management. Some of the results of this work are presented in the next section.

5 FINE-GRAINED WEB SERVICES FOR INTEGRATED MANAGEMENT

At first sight, an intuitive approach to introduce Web services in network management would be to map existing management operations (e.g., the SNMP operations `get`, `set` and `inform`) onto Web services. Is this efficient? Is this the way Web services should be used?

5.1 FOUR APPROACHES

Mapping the current habits in SNMP-based network management onto Web services leads to four possible approaches, depending on the genericity and the transparency chosen [83]. *Genericity* characterizes whether the Web service is generic (e.g., a Web service that performs a `get` operation for an OID passed as parameter) or specific to an OID (e.g., `getIfInOctets`). *Transparency* is related to the parameters of the Web service, which can be either defined at the WSDL level or kept as an opaque string that is defined as part of a higher-level XML schema; this will be referred to as Web services that have either transparent or non-transparent parameters. The four resulting forms of Web service are depicted in Figure 3.

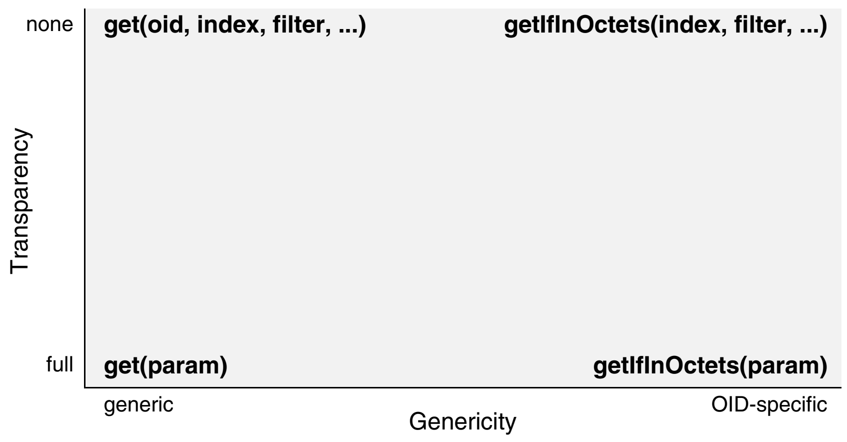


Figure 3: Fine-grained Web services: transparency vs. genericity

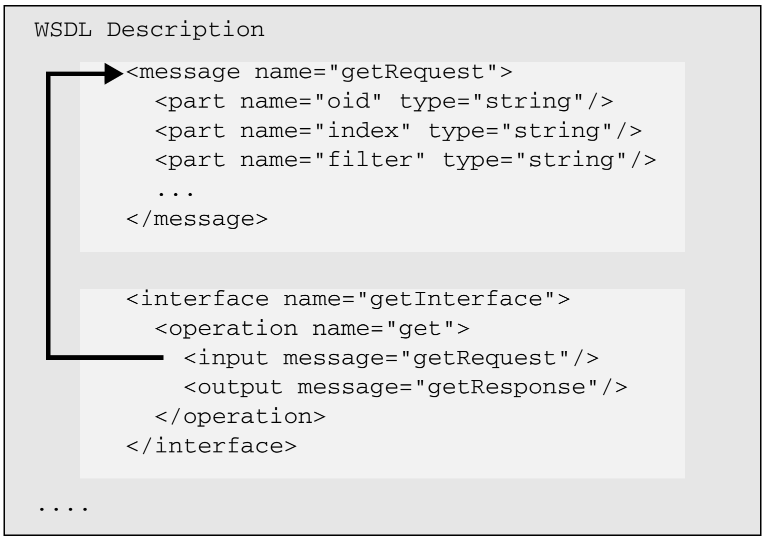
5.2 Parameter Transparency

Web service operations are defined as part of the abstract interface of WSDL descriptions. An operation may define the exchange of a single message, such as a `trap`, or multiple messages, such as `getRequest` and `getResponse`. For each message, the parameters are defined in so-called `<part>` elements. One approach consists in passing multiple parameters and defining the syntax of each parameter in a separate `<part>` element. An example of this is a `getRequest` message, which defines three parameters: `oid` (object identifier), `index` and `filter`. This example of non-transparent parameters is showed in Figure 4; note that for the sake of simplicity, all parts are of type `string` in this example.

Another approach is to define just a single parameter, and leave the interpretation of this parameter to the (manager-agent) application. In this case, only a single `<part>` element is needed. The syntax of the parameter is of type `string`, which means that the parameter is transparently conveyed by the WSDL layer and no checking is performed at that layer (see Figure 5).

An advantage of having parameter transparency is that a clear separation is made between the management information and the protocol that exchanges this information. In this case, it is possible to change the structure of the management information without altering the definition of management operations. From a standardization perspective, this is a good selling point. However, this is not a mandatory feature as WSDL supports other constructs (e.g., the `import` statement) to divide a specification over several documents, which can then be processed independently along the standardization track.

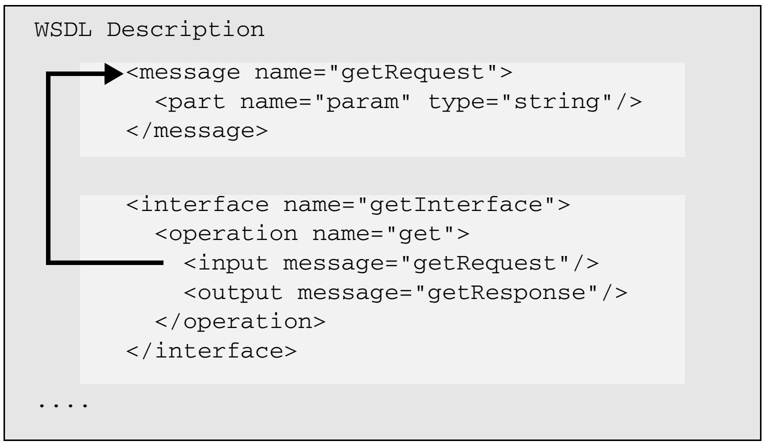
In case of transparent parameters, the application can exchange management data in the form of XML documents. In this case, an XML parser is needed to extract the parameters



```
WSDL Description
  <message name="getRequest">
    <part name="oid" type="string"/>
    <part name="index" type="string"/>
    <part name="filter" type="string"/>
    ...
  </message>

  <interface name="getInterface">
    <operation name="get">
      <input message="getRequest"/>
      <output message="getResponse"/>
    </operation>
  </interface>
....
```

Figure 4: Operation with non-transparent parameters



```
WSDL Description
  <message name="getRequest">
    <part name="param" type="string"/>
  </message>

  <interface name="getInterface">
    <operation name="get">
      <input message="getRequest"/>
      <output message="getResponse"/>
    </operation>
  </interface>
....
```

Figure 5: Operation with transparent parameters

from the document. When the application receives such a document, XPath [20] expressions may be used to select the required information. On the manager side, special functionality is needed to create such XML documents. As specific parsing is needed in the application, transparent parameters should only be used by experienced users who need this flexibility. For a PC user, in his home environment, who wants to include some management information in an Excel spreadsheet, this is too complicated. In such a case, a simple approach is required in which the user does not need to parse XML documents with tools like XPath and XQuery. Instead, parsing and checking should be performed at the WSDL layer, thus by the spreadsheet itself. Non-transparent parameters are therefore better suited for simple management applications, as found in home or small-enterprise environments.

5.3 GENERICITY

Another approach is to vary the genericity of operations, as illustrated by this example. Assume a managed system provides host-specific information such as `SysLocation` and `SysUptime`, as well as statistical information for the network interfaces such as `IfInOctets`, `IfOutOctets`, `IfInErrors` and `IfOutErrors`. As the system can have multiple interfaces, the four interface-related OIDs can be provided multiple times. Figure 6 shows this information in the form of a containment tree.

To retrieve this information, management operations need to be defined. If these operations are defined at a low level of genericity, a dedicated operation is required for each variable. The resulting operations then look like `getSysUptime` or `getIfOutOctets` (see Figure 6). Note that the operations for retrieving network interface information, need to have a parameter supplied to identify the interface.

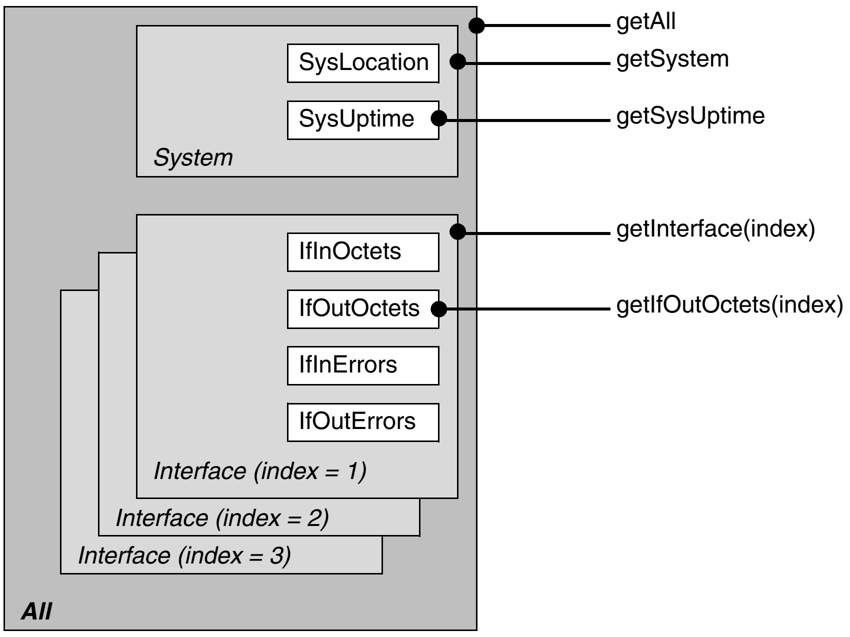


Figure 6: Containment diagram

Management operations can also be defined at a higher level of genericity. For example, a `getSystem` operation can be defined to retrieve `SysLocation` as well as `SysUptime`, and a `getInterface(index)` operation to retrieve all management information for a specific interface. If we push this rationale even further, a single `getAll` operation can retrieve all information contained in the managed system.

An important characteristic of this approach is that the naming of the operations precisely defines the functionality. Therefore it is very easy for the manager to determine which operation to call to retrieve certain information. Note that the number of parameters associated with each operation may be small: no OID is needed because the name of the operation already identifies the object. Only in case multiple instances of the same object class exist (e.g., the `Interface` object in Figure 6) does the manager have to provide an instance identifier.

Like in other management architectures such as OSI Management (the Open Systems Interconnection management architecture devised by the ISO) [35] or TMN (the Telecommunications Management Network devised by the International Telecommunication Union, ITU) [35], the containment hierarchy can be presented as a tree (see Figure 7). Fine-grained operations can be used to retrieve the leaves of the tree, and coarse-grained operations to retrieve the leaves as well as their higher-level nodes. In SNMP, conversely, only the leaves of the MIB trees are accessible. In general, however, it is not possible to select in a single operation objects from different trees; if the manager wants to retrieve, for example, the OIDs `System` and `IfInErrors`, two separate SNMP operations are needed.

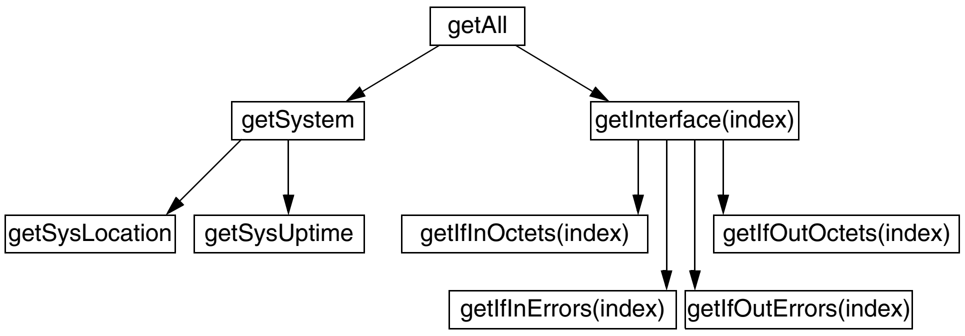


Figure 7: Containment tree

To restrict an operation to specific objects, it is possible to use Web services in combination with filtering. Theoretically, filtering allows usage of `getAll` to retrieve every possible variable, provided that an appropriate filter (and possible index) is given. In combination with filtering, `getAll` can thus be considered an extreme form of genericity, useful for all kinds of task. In practice, however, filtering can be expensive. When it is performed on the agent side, it may consume too many CPU cycles and memory. When filtering is performed on the manager side, bandwidth may be wasted and the management platform may become overloaded. Although filtering is very flexible and powerful, it should be used with care.

5.4 EXAMPLE OF WEB SERVICES-BASED MANAGEMENT

To illustrate the use of Web services for integrated management, let us now study an example of Web services-based network monitoring system. The example is about the SNMP Interface table (`ifTable`), which is part of the Interfaces Group MIB (`IF-MIB`) [57]. The `ifTable` provides information on the operation and use of all network interfaces available in the managed entity. These interfaces can be physical interfaces (e.g., Ethernet and Wireless Local Area Network cards) or virtual interfaces (e.g., tunnels or the loop-back interface). For each interface, the table includes a separate row; the number of rows is therefore equal to the number of interfaces. The table consists of 22 columns and includes information such as the interface index (`ifIndex`), the interface description (`ifDescr`), the interface type (`ifType`), etc. Figure 8 shows a summary of this table.

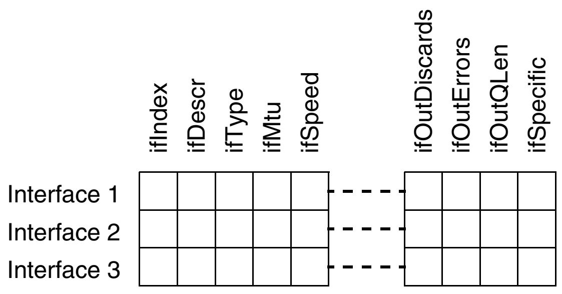


Figure 8: Interface Table

Before we can come up with a WSDL description for retrieving the Interface table, we must choose the genericity of the WSDL operations. Three levels of genericity are possible, ranging from high genericity to low genericity.

At the highest level of genericity, a single operation can be defined to retrieve all objects of the `ifTable`. We call this Web service operation `getIfTable`.

Instead of retrieving the entire `ifTable` in a single call, it is also possible to define Web service operations that retrieve only a single row or a single column. The operation that retrieves a single row of the Interface table is called `getIfRow`. This operation gets all information related to a single network interface and requires as parameter the identifier of that interface; a possible choice for this parameter is the value of `ifIndex`. The operation that retrieves a single column is called `getIfColumn`. To identify which column should be retrieved, a parameter is needed that can take values such as `ifIndex`, `ifDescr`, `ifType`, etc. Alternatively, it is possible to define separate operations per column; in that case, the resulting operations would be `getIfIndex`, `getIfDescr`, `getIfType`, etc.

Finally, at the lowest level of genericity, separate operations are defined to retrieve each individual `ifTable` object. This approach is somehow comparable to the approach used by the SNMP `get` operation (although the latter allows multiple objects to be retrieved in a single message). Depending on the required parameter transparency, again two choices are possible. The first possibility is to have a generic `getIfCell` operation; this operation requires as input parameters an interface identifier to select the row (interface), as well as a parameter to select the column (e.g., `ifIndex`, `ifDescr` or `ifType`). The second possibility is to have special operations for each column; these operations require as input parameter the identifier of the required interface. An example of such an operation is `getIfType(interface1).`

Figure 9 summarizes the choices that can be made to retrieve table information; they range from high genericity (`getIfTable`) to low genericity (`getIfCell`). Note that this figure does not show the choices that are possible with respect to parameter transparency.

Let us now discuss the main parts of the WSDL descriptions of these operations. These descriptions are a mix of WSDL 2.0 and WSDL 1.1, as WSDL was not yet finalized when

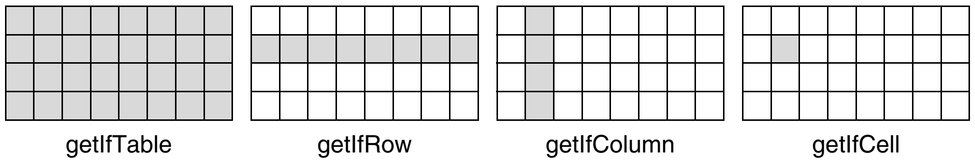


Figure 9: Different choices to retrieve table objects

this work was performed. In the gSOAP toolkit that we used [25], the WSDL descriptions consist of the following (container) elements:

- The <types> element (WSDL 2.0), to define new data types.
- The <message> elements (WSDL 1.1), to specify the data that belongs to each message.
- The <interface> element (WSDL 2.0), to combine one or more messages and ease the generation of programming stubs.
- The <binding> element (WSDL 2.0), to associate the interface element with a transport protocol.
- The <service> element (WSDL 2.0), to associate the interface element with a web address (URL).

THE <TYPES> ELEMENT

XML supports two types for defining elements: `<simpleType>` and `<complexType>`. A `<simpleType>` element contains a single value of a predefined form (e.g., an integer or a string). A `<complexType>` element groups other elements (e.g., an address element can contain street, postal code, and city sub-elements).

The WSDL descriptions of our examples use `<complexType>` elements to group certain objects within the `ifTable`. Which object are taken together depends on the genericity of the Web service. For example, all columns of the `ifTable` can be grouped together, yielding a single `<complexType>` that can be used to retrieve entire `ifTable` rows. This complex type is called `ifEntry` in our example (see Figure 10), and can be used by the `getIfTable` as well as `getIfRow` operations. Note that for the `getIfRow` operation, the `<sequence>` element should be left out, as only one row can be retrieved at a time.

THE <MESSAGE> ELEMENTS

The `<message>` elements are used in WSDL 1.1 to describe the information that is exchanged between a Web service producer and its consumers. There are `<message>` elements for request (input) and response (output) messages. These messages consist of zero or more `<part>` elements. In request messages, the `<part>` elements represent the parameters of the Web service. In response messages, these elements describe the response data.

The `getIfTable` Web service supports a single operation: retrieving the complete table. Figure 11 shows the `<message>` elements for this operation. The figure shows a request message that contains a “community” string; this string is used for authentication purposes in SNMPv1 or SNMPv2c. The response message contains an element of type `ifEntry`, which was defined in Figure 10, as well as an integer representing the number of table rows.

```
<types>
  <complexType name="ifEntry">
    <sequence>
      <element name="ifIndex" type="xsd:unsignedInt"/>
      <element name="ifDescr" type="xsd:string"/>
      <element name="ifType" type="xsd:unsignedInt"/>
      <element name="ifMtu" type="xsd:unsignedInt"/>
      <element name="ifSpeed" type="xsd:unsignedInt"/>
      <element name="ifPhysAddress" type="xsd:string"/>
      <element name="ifAdminStatus" type="xsd:unsignedInt"/>
      <element name="ifOperStatus" type="xsd:unsignedInt"/>
      <element name="ifInOctets" type="xsd:unsignedInt"/>
      <element name="ifInUcastPkts" type="xsd:unsignedInt"/>
      <element name="ifInDiscards" type="xsd:unsignedInt"/>
      <element name="ifInErrors" type="xsd:unsignedInt"/>
      <element name="ifOutOctets" type="xsd:unsignedInt"/>
      <element name="ifOutUcastPkts" type="xsd:unsignedInt"/>
      <element name="ifOutDiscards" type="xsd:unsignedInt"/>
      <element name="ifOutErrors" type="xsd:unsignedInt"/>
      <element name="ifOutQLen" type="xsd:unsignedInt"/>
      <element name="ifSpecific" type="xsd:string"/>
    </sequence>
  </complexType>
  ...
</types>
```

Figure 10: The ifEntry type

```
<message name="getIfTableRequest">
  <part name="community" type="xsd:string"/>
</message>

<message name="getIfTableResponse">
  <part name="sizeTable" type="xsd:int"/>
  <part name="ifEntry" type="utMon:ifEntry"/>
</message>
```

Figure 11: Message definitions for getIfTable

All other Web services in this section support multiple operations. The `getIfRow` Web service can be defined to support two operations: retrieve a specified row, and retrieve a list of valid row index numbers (which correspond to the network interfaces in the agent system). The `getIfColumn` and `getIfCell` Web services can be defined in terms of operations (one per `ifTable` column).

Figure 12 shows the messages for two of the operations used by the `getIfCell` Web service. The request messages have an index element for referencing the correct cell.

```
<message name="getIfIndexRequest">
  <part name="index" type="xsd:unsignedInt"/>
  <part name="community" type="xsd:string"/>
</message>

<message name="getIfIndexResponse">
  <part name="ifIndex" type="xsd:unsignedInt"/>
</message>

<message name="getIfDescrRequest">
  <part name="index" type="xsd:unsignedInt"/>
  <part name="community" type="xsd:string"/>
</message>

<message name="getIfDescrResponse">
  <part name="ifDescr" type="xsd:string"/>
</message>
```

Figure 12: Message definitions for getIfCell

THE <INTERFACE> ELEMENT

An `<interface>` element defines the operations that are supported by the Web service. Each operation consists of one or more messages; the `<interface>` element therefore groups the previously defined `<message>` elements into `<operation>` elements. In addition, the `<interface>` element may define `<description>` elements. In general, four types of operation exist: *one way*, *request-response*, *solicit response* and *notification*. In our example, they are all of the *request-response* type. Figure 13 shows the `<interface>` element for the `getIfTable` Web service.

```
<interface name="getIfTableServicePortType">
  <operation name="getIfTable">
    <documentation>function utMon__getIfTable</documentation>
    <input message="tns:getIfTableRequest"/>
    <output message="tns:getIfTableResponse"/>
  </operation>
</interface>
```

Figure 13: Interface definition for getIfTable

The other Web services support multiple operations. Figure 14 shows part of the `<interface>` element for the `getIfColumn` Web service.

```
<interface name="getIfColumnServiceInterface">
  <operation name="getIfIndex">
    <documentation>function utMon__getIfIndex</documentation>
    <input message="tns:getIfIndexRequest"/>
    <output message="tns:uIntArray"/>
  </operation>
  <operation name="getIfDescr">
    <documentation>function utMon__getIfDescr</documentation>
    <input message="tns:getIfDescrRequest"/>
    <output message="tns:stringArray"/>
  </operation>
  <operation name="getIfType">
    <documentation>function utMon__getIfType</documentation>
    <input message="tns:getIfTypeRequest"/>
    <output message="tns:uIntArray"/>
  </operation>
  ...
</interface>
```

Figure 14; Interface definition for getIfColumn

THE <BINDING> ELEMENT

A `<binding>` element specifies which protocol is used to transport the Web service information, and how this information is encoded. Similar to the `<interface>` element, it also includes the operations that are supported by the Web service, as well as the request and response messages associated with each operation. In principle, there are many choices for the transport protocol; in practice, SOAP over HTTP is by far the most popular choice. Figure 15 shows part of the `<binding>` element for the `getIfColumn` Web service.

```
<binding name="getIfColumnServiceBinding"
      type="tns:getIfColumnServiceInterface">
  <SOAP:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>

<operation name="getIfIndex">
  <SOAP:operation soapAction=""/>
  <input>
    <SOAP:body use="encoded" namespace="urn:utMon"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
    <SOAP:body use="encoded" namespace="urn:utMon"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
</operation>

<operation name="getIfDescr">
  <SOAP:operation soapAction=""/>
  <input>
    <SOAP:body use="encoded" namespace="urn:utMon"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
    <SOAP:body use="encoded" namespace="urn:utMon"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
</operation>
...
</binding>
```

Figure 15: Binding definition for getIfColumn

THE <SERVICE> ELEMENT

The <service> element is used to give the Web service a name and specify its location (a URL). Figure 16 shows this element for the example of the `getIfRow` Web service. The name is `getIfRowService`, the location is `http://yourhost.com/` and the transport protocol is defined by the `getIfRowServiceBinding`.

```
<service name="getIfRowService">
  <documentation>getIfRow service</documentation>
  <endpoint name="getIfRowService"
     binding="tns:getIfRowServiceBinding">
    <SOAP:address location="http://yourhost.com/"/>
  </endpoint>
</service>
```

Figure 16: Service element definition for getIfRow

5.5 PERFORMANCE

One of the main arguments against using Web services in network management is that the performance of Web services is inferior to that of SNMP. We show in this section that this argument is invalid. Although there are scenarios in which SNMP provides better performance, there are other scenarios in which Web services perform better. In general, the following statements can be made:

- In case a single managed object should be retrieved, SNMP is more efficient than Web services.
- In case many objects should be retrieved, Web services may be more efficient than SNMP.
- The encoding and decoding of SNMP PDUs or WSDL messages is less expensive (in terms of processing time) than retrieving management data from within the kernel. The choice between Basic Encoding Rules (BER) or XML encoding is therefore not the main factor that determines performance.
- Performance primarily depends on the quality of the implementation. Good implementations perform considerably better than poor implementations. This holds for SNMP as well as Web services implementations.

In the literature, several studies can be found on the performance of Web services-based management. Choi and Hong published several papers in which they discuss the design of XML-to-SNMP gateways. As part of their research, the performance of XML and SNMP-based management software was investigated [18][19]. To determine bandwidth, they measured the XML traffic on one side of the gateway, and the corresponding SNMP traffic on the other side. In their test setup, separate SNMP messages were generated for every managed object that had to be retrieved; the possibility to request multiple managed objects via a single SNMP message was not part of their test scenario. The authors also measured the latency and resource (CPU and memory) usage of the gateway. They concluded that, for their test setup, XML performed better than SNMP.

Whereas the previous authors compared the performance of SNMP to XML, Neisse and Granville focused on SNMP and Web services [61]. As the previous authors, they implemented a gateway and measured traffic on both sides of this gateway. Two gateway translation schemes were used: protocol level and managed object level. In the first scheme, a direct mapping exists between every SNMP and Web services message. In the second scheme, a single, high-level Web service operation (e.g., `getIfTable`) maps onto multiple SNMP messages. These authors also investigated the performance impact of using HTTPS and `zlib` compression [94]. The Web services toolkit was NuSOAP. For protocol-level translation, they concluded that Web services always require substantial more bandwidth than SNMP. For translation at the managed-object level, they found that Web services perform better than SNMP if many managed objects were retrieved at a time.

Another interesting study on the performance of Web services and SNMP was conducted by Pavlou *et al.* [75][76]. Their study is much broader than a performance study and also includes CORBA-based approaches. They consider the retrieval of Transmission Control Protocol (TCP) MIB variables and measure bandwidth and round-trip time. Unlike in the previous two studies, no gateways are used, but a dedicated Web services agent was implemented using the Systinet WASP toolkit [90]. The performance of this agent was compared to a Net-SNMP agent [62]. The authors neither investigated alternate SNMP agents, nor the effect of fetching actual management data in the system. They concluded that Web services causes more overhead than SNMP.

In the remainder of this section, we discuss a study performed in 2004 at University of Twente by one of the authors of this chapter; more details can be found in [77]. For this study, measurements were performed on a Pentium 800 MHz PC running Debian Linux. The PC was connected via a 100 Mbit/s Ethernet card. Four Web services prototypes were build: one for retrieving single OIDs of the `ifTable`, one for retrieving `ifTable` rows, one for retrieving `ifTable` columns and one for retrieving the entire `ifTable` (see Section 5.4). To retrieve management data from the system, we decided to use the same code for the SNMP and Web services prototypes. In this way, differences between the measurements would only be caused by differences in SNMP and Web services handling. For this reason, the Web services prototypes incorporated parts of the Net-SNMP open-source package. All SNMP-specific code was removed and replaced by Web services-specific code. This code was generated using the gSOAP toolkit version 2.3.8 [25][26], which turned out to be quite efficient. It generates a dedicated skeleton and does not use generic XML parsers such as the Document Object Model (DOM) or the Simple API for XML (SAX). For compression, we used `zlib` [94]. The performance of the Web services prototypes was compared to more than twenty SNMP implementations, including agents on end-systems (Net-SNMP, Microsoft Windows XP agent, NuDesign and SNMP Research's CIA agent) as well as agents within routers and switches (Cisco, Cabletron, HP, IBM and Nortel).

BANDWIDTH USAGE

One of the main performance parameters is bandwidth usage. For SNMP, it can be derived analytically as the PDU format of SNMP is standardized. For Web services, there were no standards when we conducted this study, so we could not derive analytical formulae that would give lower and upper bounds for the bandwidth needed to retrieve `ifTable` data. In fact, the required bandwidth depends on the specific WSDL description, which may be different from case to case. This section therefore only discusses the bandwidth requirements of the prototype that fetches the entire `ifTable` in a single interaction; the WSDL structure of that prototype was presented in Section 5.4.

Figure 17 shows the bandwidth requirements of SNMPv1 and Web services as a function of the size of the `ifTable`. Bandwidth usage is determined at the application (SNMP/SOAP) layer; at the IP layer, SNMP requires 56 additional octets, whereas Web services require between 273 and 715 octets, depending on the number of retrieved managed objects and the compression mechanism applied. In case of SNMPv3, an additional 40 to 250 octets are needed.

Figure 17 illustrates that SNMP consumes far less bandwidth than Web services, particularly in cases where only a few managed objects should be retrieved. But, even in cases where a large number of objects should be retrieved, SNMP remains two (`get`) to four (`getBulk`) times better than Web services. The situation changes, however, when compression is used. Figure 17 shows that, for Web services, compression reduces bandwidth consumption by a factor 2 when only a single object is retrieved; if 50 objects are retrieved, by a factor 4; for 150 objects, by a factor 8.

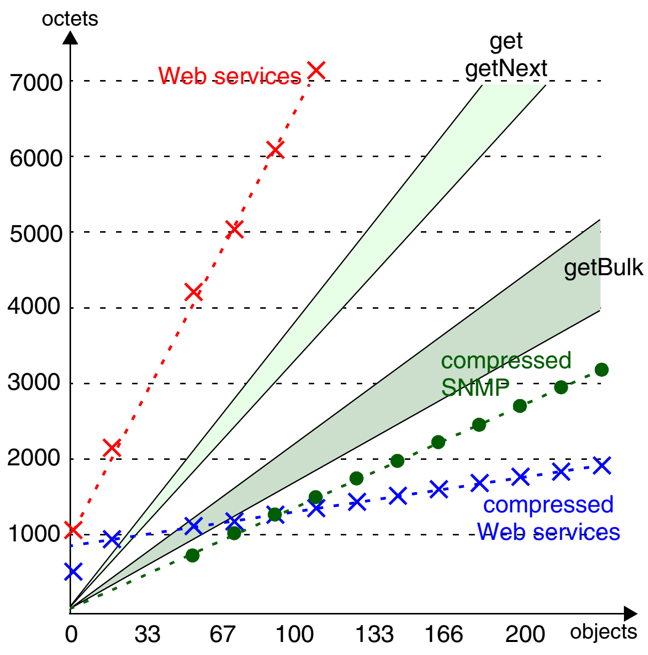


Figure 17: Bandwidth usage of SNMP and Web Services

The threshold where compressed Web services begin to outperform SNMP is 70 managed objects. This number is not ridiculously high: it is easily reached when interface data needs to be retrieved for hundreds of users, for example in case of access switches or Digital Subscribe Line Access Multiplexers (DSLAMs). Figure 17 also shows that the bandwidth consumption of compressed SNMP, as defined by the IETF EoS Working Group, is approximately 75% lower than non-compressed SNMPv1. This form of SNMP compression is known under the name ObjectID Prefix Compression (OPC).

CPU TIME

Figure 18 shows the amount of CPU time needed for coding (decoding plus the subsequent encoding) an SNMP message (encoded with BER) or Web services (encoded in XML) encoded messages. As we can see, the different curves can be approximated by piecewise linear functions. For an SNMP message carrying a single managed object, the BER coding time is roughly 0.06 ms. A similar XML-encoded message requires 0.44 ms, which is seven times more. Coding time increases if the number of managed objects in the message increases. Coding an SNMP message that contains 216 objects requires 0.8 ms; coding a similar Web services message requires 2.5 ms. We can therefore conclude that XML encoding requires 3 to 7 times more CPU time than BER encoding. Figure 18 also shows the effect of compression: compared to XML coding, Web services compression turns out to be quite expensive (by a factor 3 to 5). In cases where bandwidth is cheap and CPU time expensive, Web services messages should not be compressed.

To determine how much coding contributes to the complete message handling time, the time to retrieve the actual data from the system was also measured. For `ifTable`, this

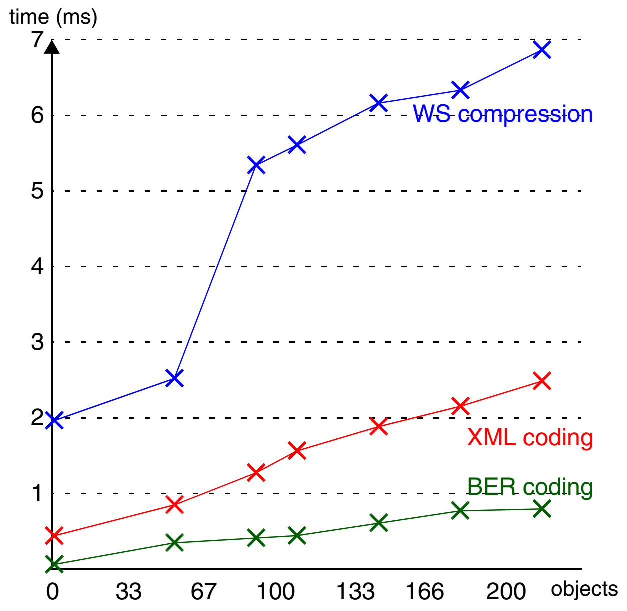


Figure 18: CPU time for coding and compression

retrieval requires, among other things, a system call. We found out that retrieving data is relatively expensive; fetching the value of a single object usually takes between 1.2 and 2.0 ms. This time is considerably larger than the time needed for coding.

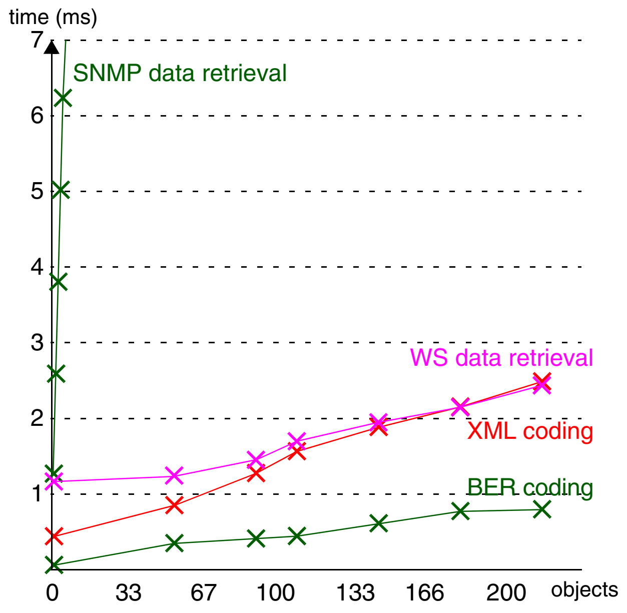


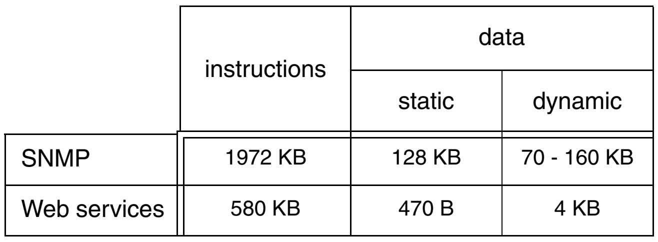
Figure 19: CPU time for coding and data retrieval

The results are depicted in Figure 19; to facilitate the comparison, the BER and XML coding times are mentioned as well. The figure shows that data retrieval times for Net-SNMP increase quite fast; five managed objects already require more than 6 ms; for retrieving 54 managed objects, 65 ms were needed; and retrieving 270 managed objects from the operating system took 500 ms! These staggering times can only be explained by the fact that Net-SNMP performs a new system call every time it finds a new `ifTable` object within the `get` or `getBulk` request; Net-SNMP does not implement caching. The conclusion must therefore be that data retrieval is far more expensive than BER encoding; CPU usage is not determined by the protocol handling, but by the quality of the implementation.

MEMORY USAGE

Memory is needed for holding program instructions ("instruction memory") and data ("data memory"). For data memory, a distinction needs to be made between static and dynamic memory. Static memory is allocated at the start of the program and remains constant throughout the program lifetime. Dynamic memory is allocated after a request is received and released after the response is transmitted. If multiple requests arrive simultaneously, dynamic memory is allocated multiple times.

Figure 20 shows the memory requirements of Net-SNMP and our Web services prototype. We see that Net-SNMP requires roughly three times as much instruction memory than the Web services prototype. Net-SNMP requires 20 to 40 times more data memory, depending on the number of objects contained in the request. These large numbers, much larger than their Web services counterparts, should not be misinterpreted: the functionality of Net-SNMP is much richer than that of our Web services prototype. For instance, Net-SNMP supports three different protocol versions, includes encryption, authentication and access control, and is written in a platform-independent way.



	instructions	data	
		static	dynamic
SNMP	1972 KB	128 KB	70 - 160 KB
Web services	580 KB	470 B	4 KB

Figure 20: Memory requirements

ROUND-TRIP TIME

In the previous subsections, the performance of the `getIfTable` Web service was compared to that of Net-SNMP. This subsection compares the performance of this Web services prototype to other SNMP implementations. As we could not add code to existing agents code for measuring BER encoding and data retrieval times, we measured instead the round-trip time, as a function of the number of retrieved managed objects. This time can be measured outside the agent and thus does not require any agent modification.

	1	22	66	270
WS	1,7	2,6	10,3	36,5
WS-Comp	3,3	4,3	5,6	11,8
SNMP-1	1,0	2,5		
SNMP-2	1,0	2,7		
SNMP-3	1,1	3,2	6,9	14,9
SNMP-4	2,0	15,2		
SNMP-5	3,7	18,3		
SNMP-6	1,3	18,9	53,0	
SNMP-7	2,9	58,0	167,0	695,0
SNMP-8	3,1	58,2	168,8	700,0
SNMP-9	3,5	62,0	183,8	767,0
SNMP-10	5,0	80,9		
SNMP-11	12,0	86,9	244,3	
SNMP-12	12,4	257,9		

Figure 21: Round-trip time (in ms)

Figure 21 shows the results for 12 different SNMP agents, as well as the `getIfTable` Web service prototype. It is interesting to note that the round-trip time for normal Web services increases faster than that of compressed Web services. For most SNMP measurements, we used `getBulk` with max-repetition values of 1, 22, 66 and 270; for agents that do not support `getBulk`, a single `get` request was issued, asking for 1, 22, 66 or 270 objects. Not all agents were able to deal with very large size messages; the size of the response message carrying 270 managed objects is such that, on an Ethernet Local Area Network, the message must be fragmented by the IP protocol. As the hardware for the various agents varied, the round-trip times showed in Figure 21 should be used with great care and only considered purely indicative. Under slightly different conditions, these times may be quite different. For example, the addition of a second Ethernet card may have a severe impact on round-trip times. Moreover, these times can change if managed objects other than `ifTable` OIDs are retrieved simultaneously. In addition, the first SNMP interaction often takes considerably more time than subsequent interactions.

Despite these remarks, the overall trend is clear. With most of the SNMP agents that we tested, the round-trip time heavily depends on the number of managed objects that are retrieved. It seems that several SNMP agents would benefit from some form of caching; after more than 15 years of experience in SNMP agent implementation, there is still room for performance improvements. From a latency point of view, the choice between BER and XML encoding does not seem to be of great importance. Our Web services prototype performs reasonably well and, for numerous managed objects, even outperforms several commercial SNMP agents.

6 Toward Coarse-Grained Web Services for Integrated management

In the previous section, Web services were used in SNMP-based network management as getters and setters of fine-grained MIB OIDs. Similarly, in a WBEM environment, they could be used as getters and setters of CIM object properties, or they could be mapped to fine-grained CIM operations. This is the granularity at which Web services-enabled management platforms generally operate today. For instance, HP's Web Service Management Framework [14] specifies how to manipulate fine-grained managed objects with Web services. But is this the way Web services are meant to be used as management application middleware?

Web services did not come out of the blue, context free: they were devised with SOA in mind. SOA is an IT architectural style, to use Fielding's terminology [28], and Web services implement it. In this section, we go back to the rationale behind SOA, consider management applications from this perspective, and show that Web services could be used differently in integrated management. We illustrate this claim with examples of coarse-grained Web services for integrated management.

6.1 Service-Oriented Architecture

Defining SOA poses two problems. First, there is no single and widely shared definition. No one clearly defined this concept before it began pervading the software industry. This concept grew and spread by hearsay in the software engineering community, as a conceptualization of what Web services were trying to achieve. The definitions proposed by the W3C and OASIS came late, and despite the credit of these organizations in the community, none of their definitions are widely accepted.

Second, since 2004–2005, SOA has turned into a buzzword in the hands of marketing people. Today, there are almost as many definitions of this acronym as vendors in the software industry. There are also countless examples of software tools and packages that have been re-branded "SOA-enabled" without supporting all the features that engineers generally associate with this acronym.

In an attempt to distinguish between a stable technical meaning and ever-changing marketing blurb, some authors use the term Service-Oriented Computing (SOC) [71][80] to refer, among engineers, to the distributed computing paradigm, leaving SOA to hype. Other authors consider that SOC is a distributed computing paradigm whereas SOA is a software architecture; the subtle difference between the two is not always clear. In this chapter, we consider these two acronyms synonymous and use SOA in its technical sense.

So, what is SOA about? Because no one "owns" its definition, we reviewed the literature and browsed a number of Web sites to form an opinion.

In WSA [9] (defined in Section 2.5), the W3C defines SOA as "a form of distributed systems architecture" that is "typically characterized" (note the implied absence of consensus) by the following properties:

- *logical view:* the service is an abstract view of actual programs, databases, business processes, etc.;
- *message orientation:* the service is defined in terms of messages exchanged between consumers and providers;
- *description orientation:* the service is described by machine-parsable metadata to facilitate its automated discovery;
- *granularity:* services generally use a small number of operations with relatively large and complex messages;
- *network orientation:* services are often used over a network, but this is not mandatory;
- *platform neutral:* messages are sent in a standard format and delivered through interfaces that are independent of any platform.

In August 2006, OASIS released its Reference Model for SOA (SOA-RM) [50]. This specification was long awaited by the industry. Unfortunately, this version 1.0 is quite verbose and rather unclear, and often proposes overly abstract definitions; we hope that OASIS will solve these teething problems in subsequent versions. Difficult to read as it is, this document still clarifies three important points:

- "SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains";
- a service is accessed via an interface;
- a service is opaque: its implementation is hidden from the service consumer.

Channabasavaiah *et al.* [15] are more practical and define SOA as "an application architecture within which all functions are defined as independent services with well-defined invokable interfaces, which can be called in defined sequences to form business processes". They highlight three aspects:

- all functions are defined as services;
- all services are independent and operate as black boxes;
- consumers just see interfaces; they ignore whether the services are local or remote, what interconnection scheme or protocol is used, etc.

IBM's online introduction to SOA [37] lists the following characteristics:

- SOA is an IT architectural style;
- integrated services can be local or geographically distant;
- services can assemble themselves on demand; they "coalesce to accomplish a specific business task, enabling (an organization) to adapt to changing conditions and requirements, in some cases even without human intervention"; coalescence refers to automated service composition at run-time;
- services are self-contained (i.e. opaque) and have well-defined interfaces to let consumers know how to interact with them;
- with SOA, the macro-components of an application are loosely coupled; this loose coupling enables the combination of services into diverse applications;
- all interactions between service consumers and providers are carried out based on service contracts.

Erl [27] identifies four core properties and four optional properties for SOA:

- services are autonomous (core);
- loose coupling (core);
- abstraction: the underlying logic is invisible to service consumers (core);
- services share a formal contract (core);
- services are reusable;
- services are composable;
- services are stateless;
- services are discoverable.

O'Brien *et al.* [66] complement this list of properties with the following two:

- *Services have a network-addressable interface:* "Service (consumers) must be able to invoke a service across the network. When a service (consumer) and service provider are on the same machine, it may be possible to access the service through a local interface and not through the network. However, the service must also support remote requests."
- *Services are location transparent:* "Service (consumers) do not have to access a service using its absolute network address. (They) dynamically discover the location of a service looking up a registry. This feature allows services to move from one location to another without affecting the (consumers)."

Singh and Huhns [80] characterize SOA as follows:

- *loose coupling*;
- *implementation neutrality:* what matters is the interface; implementation details are hidden; in particular, services do not depend on programming languages;
- *flexible configurability:* services are bound to one another late in the process;
- *long lifetime:* interacting entities must be able to cope with long-lived associations; services should live "long enough to be discovered, to be relied upon, and to engender trust in their behavior";
- *coarse granularity:* actions and interactions between participants should not be modeled at a detailed level;
- *teams:* participants should be autonomous; instead of a "participant commanding its partners", distributed computing in open systems should be "a matter of business partners working as a team".

For these authors, the main benefits of SOA are fourfold [80]:

- services "provide higher-level abstractions for organizing applications in large-scale, open environments";
- these abstractions are standardized;
- standards "make it possible to develop general-purpose tools to manage the entire system lifecycle, including design, development, debugging, monitoring";
- standards feed other standards.

All these definitions (and many others not mentioned here) draw a fairly consistent picture. SOA is a software paradigm, an IT architectural style for distributing computing with the following properties:

- *Loose coupling:* In SOA, distributed applications make use of loosely coupled services.
- *Coarse-grained:* Services have a coarse granularity, they fulfill some functionality on their own.
- *Autonomous:* Services are stateless and can be used by many applications or other services.
- *Interfaces and contracts:* Services are opaque entities accessed via interfaces on the basis of a high-level contract; implementation details are hidden.
- *Network transparency:* Services can be local or remotely accessed via the network. The location of the service provide may change over time.
- *Multiple owners:* Services can be owned by different organizations.
- *Service discovery:* Services are discovered dynamically, possibly automatically, at run-time; service providers are not hard-coded in applications.
- *Service composition:* Services can be composed transparently, possibly dynamically at run-time.

There is also a large consensus about three things. First, Web services are just one technology implementing SOA. Other technologies may come in the future that could implement SOA differently, perhaps more efficiently. For instance, one could imagine that SOA be implemented by a new variant of CORBA that would (i) define services as high-level CORBA objects wrapping up CORBA and non-CORBA legacy applications, (ii) use the Internet Inter-ORB Protocol (IIOP) instead of SOAP and HTTP to make service consumers and providers communicate, (iii) advertise services in the form of interfaces (expressed in CORBA IDL) in standard external registries (as opposed to CORBA-specific registries), (iv) enable these services to be discovered and composed dynamically at run-time using semantic metadata, and (v) solve the poor interoperability and the performance problems of multi-vendor inter-ORB communication.

Second, nothing in SOA is new. For instance, interfaces that make it possible to access opaque entities were already defined in Open Distributed Processing [39]. What is new is that we now have the technology to make it work: “SOAs are garnering interest, not because they are a novel approach, but because Web services are becoming sufficiently mature for developers to realize and reevaluate their potential. The evolution of Web services is at a point where designers can start to see how to implement true SOAs. They can abstract a service enough to enable its dynamic and automatic selection, and Web services are finally offering a technology that is rich and flexible enough to make SOAs a reality” [49].

Third, a key difference between Web services and distributed object middleware is that Web services are not objects that need other objects for doing something useful: they are autonomous applications fulfilling some functionality on their own. As a result, Web services should not be regarded as yet another type of object-oriented middleware competing with CORBA or J2EE [89]: they are supposed to operate at a higher level of abstraction.

It should be noted that this coarse-grained approach to middleware poses problems to the software industry, because hordes of engineers are used to thinking in terms of tightly coupled objects, and a plethora of tools work at the object level [89]. Even some academics struggle to adjust their mindframe [8].

6.2 Comparison between SOA and Current Practice

Now that we better understand the SOA approach to distributed computing, let us compare, one by one, the principles that underpin Web services with the current practice in integrated management. By "practice", we mean the management platforms that enterprises actually use to manage their networks, systems and services, as opposed to the latest commercial offerings from vendors who may claim to support SOA.

Applications Make Use of Services

SOA and integrated management both have the concept of *service*; they give it slightly different but reasonably consistent meanings. For both, we have low-level services that abstract certain parts of the IT infrastructure (e.g., a network service for an MPLS-based VPN) and high-level services that are user-centered and achieve a specific business goal (e.g., a business service for online book ordering).

A Web service does not necessarily pertain to Service-Level Management (SLM): it may be a building block for fulfilling network management tasks, or systems management tasks. In this case, the SOA concept of service corresponds roughly to the concept of management task in integrated management.

In integrated management, a high-level service embodies a contract between an end-user (a person) and a service provider (a vendor). In TMN, a network service is captured by the Service Management Layer (SML), while the contract between a user and a vendor belongs to the Business Management Layer [35]. The SML is where network and business services are tied together. In SOA, the service contract binds a service consumer to a service producer.

Both in SOA and integrated management, services can be characterized by a Service-Level Agreement (SLA), which captures the end-users' perception of whether the IT infrastructure works well for achieving a specific business goal. High-level services serve as intermediaries between business views and IT views; they enable end-users to abstract away the infrastructure and consider it as a black box.

Loose Coupling

Integrated management platforms use both tight and loose forms of integration. We can distinguish five cases.

First, communication between managed entities (agents) and managing entities (managers) is loosely coupled. It relies on standard management protocols (e.g., SNMP) that hide implementation details by manipulating abstractions of real-life resources known as managed objects.

Second, the *macro-components* of a monitoring application (event correlator, event filter, event aggregator, data analyzer, data filter, data aggregator, data poller, topology discovery engine, network map manager, etc. [53]) are tightly integrated. They are provided by a single vendor. Communication between these macro-components is proprietary, using all sorts of programming tricks to boost performance (e.g., the maximum number of events that can be processed per time unit).

Third, when management is distributed over multiple domains (e.g., for scalability reasons in large enterprises, or budget reasons in corporations with financially independent subsidiaries), manager-to-manager communication is proprietary and relies on tight coupling in the SNMP world; in the WBEM world, it is usually loosely coupled and based on standard management protocols. Note that exchanging management information with external third parties requires sophisticated firewalls in the case of tight coupling, and simpler firewalls in the case of loose coupling.

Fourth, communication between the monitoring application and the data repository is sometimes loosely coupled, sometimes tightly coupled. The data repository is logically unique but can be physically distributed; it consists typically of a relational database for monitoring data and a Lightweight Directory Access Protocol (LDAP) repository for configuration data; sometimes it may also include an object base, or an XML repository, or a knowledge base, etc. An example of loose coupling[1] is when the monitoring application accesses a relational database via a standard interface such as Open Data Base Connectivity (ODBC) using a standard query language such as the Structured Query Language (SQL); the actual data repository used underneath (Oracle database, Sybase database, etc.) is then transparent to the application. An example of tight coupling is when the application uses a proprietary query language (e.g., a so-called "enhanced" version of SQL, with proprietary extensions to the language). Vendor lock-in always implies tight coupling.

Fifth, in large enterprises or corporations, people not only use monitoring applications, but also data mining applications that perform trend analysis, retrospective pattern analysis, etc. Examples include Enterprise Resource Planning (ERP) to anticipate the saturation of resources, security analysis to search for attack patterns, event analysis to identify the important cases that need to be fed to a case-based reasoning engine, etc. Communication between the data repository and these data mining applications is usually loosely coupled: people use standard interfaces and protocols for exchanging data because these applications are generally provided by third-party vendors. For performance reasons, data mining applications may occasionally be tightly coupled to the data repository.

SERVICES ARE COARSE-GRAINED

How do the SOA concepts of "fine-grained services" and "coarse-grained services" translate into integrated management? In the SOA approach to distributed computing, we saw that applications make use of services, which isolate independent pieces of the application that can be reused elsewhere (e.g., by other applications or other services). Some of these services are conceptually at a high level of abstraction (e.g., "Balance the load of virtual server VS among the physical servers PS1, PS2, PS3 and PS4"), others at a low level (e.g., "Set the administrative status of CPU C to `down`"). Functionality is thus split between the application and reusable services of varying levels of abstraction.

In integrated management, this corresponds to management tasks in the functional model [53]. Functionality is split into management tasks, some of which are low level, some of which are high level. Some of these functions are generic (e.g., see the standard TMN

1. Note that we use "loose coupling" in a flexible manner here, compared to SOA: by using SQL, the application cannot interact with a generic data repository, but only with a relational database.

management functions [41]) and can in principle be separated from the application. The common practice in telecommunications is that management applications are based on CORBA and implement a mixture of OSI Management, TMN and SNMP. Generic functions are accessed via CORBA interfaces as independent low-level services, which can be considered coarse-grained because they do not simply operate at the managed object level. In enterprises, however, the situation is very different: today's management platforms are based on SNMP or WBEM (which do not define standard generic functions), not on OSI Management or TMN; they are typically implemented in C, C++, Java or C#, and based neither on CORBA nor on J2EE. As a result, they do not handle separate services.

In short, in the enterprise world, management tasks are not implemented in the form of independent coarse-grained services. This is a major difference with SOA.

From outside, management applications appear to operate at only two scales: the managed object level (when interacting with managed entities), and the management application level (the monitoring application globally supports some functionality as a big black box). There are no intermediaries that could easily be mapped to Web services.

Internally, management applications actually operate at more scales. When they are coded in an object-oriented programming language, these scales are: managed object, object, software component, thread, process, and complete application. With procedural programming languages, these scales are: managed object, function, thread, process, and complete application.

Services are autonomous

We saw in Section 1.2 that large enterprises generally run several management platforms in parallel. These applications are autonomous in the SOA sense, but they are not packaged as services. They usually do not communicate directly (there are exceptions), but they all (at least supposedly) send meaningful events to the same event correlator, which is in charge of integrating management for the entire enterprise. In some cases, this logically unique event correlator can be physically distributed [54].

Network management platforms (e.g., HP Openview and IBM Tivoli) often accept equipment-specific graphical front-ends (so-called "plug-ins") provided by third-party vendors. These extensions can be seen as autonomous services that are statically discovered and integrated.

Interfaces and Contracts

Integrated management applications work with two types of interface: interfaces to their internal components, and interfaces to managed entities (managed objects). The former do not hide implementation details (on the contrary, these components can expect other components to support programming language-specific features such as Java exceptions, for instance), but the latter do (managed objects are standard and hide the idiosyncrasies of the underlying managed resource).

In SOA, services accessed via interfaces form the basis for a contract. In integrated management, contracts are captured in the form of SLAs.

NETWORK TRANSPARENCY

In SOA, network transparency means that services can be local or remote. This transparency has no equivalent in management platforms. For obvious confidentiality and robustness reasons, no one wants a monitoring application to discover automatically, and later use, an event correlator provided by an unknown organization on the opposite side of the planet!

MULTIPLE OWNERS

In integrated management, the monitoring and data mining applications are generally run by the same organization, in which case there is only one owner. In large organizations, these applications can be distributed over a hierarchy of domains; different management applications are then run by different teams, possibly attached to different profit centers; but they all really belong to the same organization, the same owner from an SOA standpoint.

When organizations outsource partially or totally the management of their IT infrastructure and/or the management of their services, there are clearly multiple owners. Relationships between them are controlled by business contracts, and not just Web services contracts.

Extranets and subsidiaries are typical examples where we have different owners. In an extranet, a corporation works closely with a large set of subcontractors and/or retailers. This requires the IT infrastructures of different owners to be tightly coupled. The situation is similar in corporations that have financially independent subsidiaries. Although the latter are different enterprises, different owners, they have strong incentives for cooperating as if they were a single organization, a single owner in SOA parlance.

SERVICE DISCOVERY

In today's integrated management applications, only very simple forms of service discovery are generally implemented, primarily for retrieving configuration settings. In systems management, LDAP repositories are commonly used.

SERVICE COMPOSITION

Integrated management platforms generally found in businesses today do not use dynamic service composition at run-time. Management tasks are composed statically in the monitoring application or the data mining applications, and cast in iron in static code.

6.3 ANALYSIS

The first outcome of our comparison is that the management platforms deployed in real life are far from implementing SOA natively. Instead of simply using Web services to exchange managed objects or events, as suggested by recent standardization activities such as WSDM or WS-Management, we should rethink (i.e., re-architect and re-design) management applications to make them SOA compliant. This is a classic mission for software architects.

The main differences between SOA and today's practice in integrated management are (i) the absence of autonomous coarse-grained management tasks in management platforms, (ii) the tight coupling of the monitoring application (most if not all of its macro-components communicate via proprietary mechanisms), and (iii) the quasi absence of service discovery (except for retrieving configuration settings).

A fourth difference, the lack of dynamic service composition at run-time, is less of a problem because, strictly speaking, service composition is not a requirement in SOA: it is simply desirable, and the odds are that when the first three problems are solved, people will find ways of composing these new services.

Reconciling the first three differences identified above requires modifying not only the way management applications are structured, but also the management architectures that underpin them (notably SNMP and WBEM).

Changes in the Management Architecture

In integrated management, it is customary to decompose a *management architecture* (also known as *management framework*) into four models [30]: (i) an organizational model, which defines the different entities involved in a management application and their roles; (ii) an information model, which specifies how resources are modeled in the management domain; (iii) a communication model, which defines how to transfer data (e.g., a communication pipe and service primitives for exchanging data across this pipe); and (iv) a functional model, which defines functions (management tasks) that do real work; these functions can be fine-grained (e.g., a generic class for managing time series) or coarse-grained (e.g., an event correlator). Migrating to SOA requires changes in all but one of these models.

First, the *communication model* needs to migrate to SOAP. This migration would be straightforward for WBEM, whose communication model is close to SOAP. For SNMP, as we saw in Section 5, it would imply the adoption of another standard communication pipe (typically HTTP, possibly another one) and new communication primitives (SOAP RPCs).

Second, SOA challenges the *organizational model* of the management architectures used to date: SNMP, WBEM, TMN, OSI Management, etc. The management roles (defined in the manager-agent paradigm [74]) need to be revisited to make integrated management comply with SOA. First and foremost, the concept of coarse-grained autonomous service needs to be introduced. Migrating to SOA is also an invitation to stop interpreting the manager-agent paradigm as meaning "the manager should do everything and the agent nothing", as already advocated by Management by Delegation [92] in the 1990s but never quite enforced.

As far as the *functional model* is concerned, SOA encourages a new breakdown of management tasks among all actors involved. Instead of bringing all monitoring data and all events to the monitoring application and doing all the processing there, in a large monolithic application, SOA suggests to define intermediary tasks and wrap them as independent and reusable services that can run on different machines. This too is somewhat reminiscent of Management by Delegation [92], the Script MIB [46] in SNMP or the Command Sequencer [40] in OSI Management, except that Web services should be full-fledged applications, not just small scripts with a few lines of code.

Changes in the Management Applications

Assuming that the main management architectures used to date (SNMP and WBEM) are updated, what should be changed in management applications to make them implement autonomous coarse-grained services, loosely coupled services and service discovery?

The first issue (autonomous coarse-grained services) is about delegation granularity in a management application, i.e. how this application is subdivided into management tasks and how these tasks are allotted to different entities for execution. A few years ago, one of the authors of this chapter used organization theory to identify four criteria for classifying management architectures: semantic richness of the information model, degree of granularity, degree of automation of management, and degree of specification of a task [53]. Based on that, we defined an “enhanced” taxonomy of network and systems management paradigms. For the delegation granularity, four scales were proposed in our taxonomy: no delegation, delegation by domain, delegation by microtask and delegation by macrotask. Fine-grained Web services correspond to microtasks; coarse-grained Web services correspond to macrotasks. To be SOA compliant, management applications should therefore support a new form of delegation: macrotasks.

More specifically, we propose that management applications use Web services at three scales, three levels of granularity. First, *simple management tasks* are slightly coarser-grained than Web services operating at the managed-object level. Instead of merely getting or setting an OID, they can execute small programs on a remote managed entity. Their execution is instantaneous. The Web service providers have a low footprint on the machine where they run. Second, *elaborate management tasks* operate at a higher level of granularity and consist of programs that do something significant. Their execution is not instantaneous and may require important CPU and memory resources. Third, *macro-components* (see examples in Section 6.2, “Loose Coupling”) communicate via Web services instead of proprietary mechanisms. This last change makes it easy to distribute them across several machines, which complements domain-based distributed hierarchical management.

Once we have autonomous coarse-grained services, the second issue (loosely coupled services) is easy to deal with: management applications just need to use Web services and SOAP for communicating with these services. This guarantees a loose coupling of the management tasks.

To address the third issue (service discovery), we propose to let managing and managed entities discover one another at run-time, using Web services. This enables managed entities to be associated dynamically with a domain, instead of being configured once and for all. This change is also in line with recent standardization activities (e.g., WSDM).

6.4 How To Make Management Platforms SOA Compliant

Let us now go through a series of examples that illustrate how an SOA-enabled management platform could work.

TYPE 1: SIMPLE MANAGEMENT TASKS AS WEB SERVICES

Two examples of simple management tasks are the invocation of non-trivial actions and the execution of simple scripts.

Example 1: Remote execution of a non-trivial action

Non-trivial actions can be seen as RPCs bundled with a bit of code. The reboot of a remote device is a good example of action that can be invoked by a management application as a Web service delegated to a third party (neither the manager, nor the agent in SNMP parlance). In this Web service, we first try to do a clean ("graceful") reboot by setting an OID or invoking an operation on a CIM object. If it does not work, we then trigger a power-cycle of the machine. This operation can translate into a number of finer-grained operations, depending on the target machine to be rebooted. For instance, to power-cycle a remote PC that is not responding to any query anymore, we can look for an electromagnet that controls its power. If we find one (e.g., in a configuration database or a registry), we then set an OID or invoke a CIM operation on this electromagnet, depending on the management architecture that it supports, to force the power-cycle. A few moments later, the Web service checks whether the PC has resumed its activities; it sends back a response to the management application, with status `succeeded` or `failed`.

Using a Web service for this task allows the management application to abstract away this complexity, which is equipment specific. Binding a given piece of equipment to the right reboot Web service then becomes a matter of configuration or publication in a registry.

Example 2: Simple script

An example of simple script is the retrieval of routing tables from a group of routers. If multiple routing protocols are supported by a router, all of its routing tables need to be downloaded. This information can be subsequently exploited by the management application for debugging transient routing instabilities in the network. Alternatively, this Web service can also be used on a regular basis for performing routing-based event masking, for instance in networks where routes change frequently ("route flapping") and network topology-based event masking does not work satisfactorily.

In this example, the Web service providers can be the routers themselves; another possibility is to have a single Web service provider that runs on a third-party machine (e.g., an SNMP gateway or proxy) and interacts via SNMP with the different routers concerned by this task [72].

TYPE 2: ELABORATE MANAGEMENT TASKS AS WEB SERVICES

We now go one level up in abstraction and present four examples of elaborate management tasks: core problem investigation, ad hoc management, policy enforcement check and long-lasting problem investigation.

Example 1: Core problem investigation

In a management platform, when the event correlator has completed the root-cause analysis phase, we are left with a small set of core problems that need to be solved as soon as possible. When the solution to the problem is not obvious, we then enter a phase known as problem investigation. Different technologies can be used during this phase: an Event-Condition-Action (ECA) rule-based engine, a case-based reasoning engine, etc. On this occasion, the management application can launch programs[1] to retrieve more management information to further investigate, and hopefully solve, each outstanding problem.

Assuming that we do not have performance constraints that impose to keep all processing local[2], these programs can be delegated to other machines in the form of Web services, thereby contributing to balance the load of the management application across multiple machines and to improve scalability. They run autonomously, gather data independently, access the data repository on their own, etc.

For instance, when the response time of a Web server is unacceptably long for an undetermined reason, it can be useful to monitor in great detail the traffic between this server and all the hosts interacting with it. This can be achieved by a Web service monitoring, say, for 30 minutes all inbound and outbound traffic for this server and analyzing this traffic in detail afterward, using fast data mining techniques. This task can involve many fine-grained operations; for instance, it can retrieve Remote MONitoring (RMON) MIB [85] OIDs in bulk at short time intervals, or it can put a network interface in promiscuous mode and sniff all traffic on the network. The duration of this debugging activity can be a parameter of the Web service operation. Upon successful completion, the Web service returns the cause of the problem; otherwise, it may be able to indicate whether it found something suspicious.

Example 2: Ad hoc management

Ad hoc management tasks [53] are not run regularly by the monitoring application. Instead, they are executed on an *ad hoc* basis to investigate problems as they show up. These tasks can be triggered interactively (e.g., by staff in a NOC), or automatically by the management application (see example 1 with the slow Web server). These tasks are good candidates for Web services: it makes sense to delegate these self-contained and independent programs to third-party machines that do not have the same constraints, in terms of resource availability, as the main management platform that runs the bulk of the management application[3].

For instance, if the administrator suspects that the enterprise is currently undergoing a serious attack, he/she may launch a security analysis program that performs pattern matching on log files, analyzes network traffic, etc. This program will run until it is interrupted, and report the attacks that it detects (or strongly suspects) to the administrator using SMS messaging, paging, e-mail, etc.

Example 3: Policy enforcement check

Another interesting case of elaborate management task is to check that policies used in policy-based management [81] are enforced durably. For instance, let us consider a classic source of problems for network operators: `telnet`, the main enemy of policies. The bottom line of this scenario is invariably the same, with variations in the details.

1. These programs can be executable binaries, byte-code for virtual machines, scripts, etc.
2. For instance, that would be the case if we had to interact with thousands of finite state machines only accessible from the address space of the event correlator.
3. We do not assume a centralized management platform here. When management is distributed hierarchically over multiple managers, the problem is the same as centralized management: each manager can be a bottleneck for its own sub-domain.

A technician in a NOC is in charge of debugging a serious problem. Customers are complaining, the helpdesk is overwhelmed, his boss called him several times and is now standing next to him, watching over his shoulder, grumbling, doing nothing but stressing our technician. The latter feels he is alone on earth. All the management tools at his disposal indicate that there is an avalanche of problems, but they all fail to find the cause of these problems. After checking hundreds of configuration settings on many devices, our technician decides to stop services and reconfigure network cards one by one, hoping that this will eventually point to the problem. Using `telnet`, he modifies the configuration of many operational routers, trying all sorts of things... Doing so, he gradually starts to build a picture of what is going on: he had a similar problem several years ago, a misconfigured router was reinjecting routes when it should not. Today's problem is in fact very different, but our technician is now convinced he knows what is going on. Against all odds, after an hour of frantic typing, the situation starts to improve. Everything is still slow, but there seems to be a way out of this serious problem. And when he modifies the 27th router, bingo, the problem is gone and everything comes back to normal! His boss applauds, thanks the Lord for sending him such a great man, goes back to his office and phones his own boss, to make sure he does not get the blame for this major outage...

Is the incident closed? Yes, because things work again. Is the problem solved? No, because our technician still does not know what the problem was. Is the situation back to normal? Not at all: first, the same problem could resurface at any time; second, by changing the configuration of dozens of devices, our technician has placed several timed bombs in the network, without his knowing it. These timed bombs are big potential problems that are just waiting for an occasion to go off. And why so? Because he cannot roll back all the unnecessary changes that he made. Under stress, he did many things that he cannot remember anymore; and because he is dealing with operational networks, he is now very cautious about modifying anything. Yet he did several changes that he may soon regret. He changed configuration settings that are supposed to be entirely controlled by PDPs, and that will disappear in a few hours because they are overwritten everyday, at the same time, during slack hours. Will the problem resume immediately after the next policy update? He also changed many configuration settings that have nothing to do with the actual problem. Will they cause new problems in the future?

All network operators, all corporations, and presumably most large enterprises have been through this one day. This problem is as old as `telnet`. Under stress, NOC staff can modify (and break) just about everything, irrespective of their skills. One solution is to regularly copy the configuration settings (notably those that enforce policies and are generated automatically) of all network devices in a data store, at least once a day, and analyze every day the changes that were made in the past 24 hours. This process can be partially automated; it is a self-contained and coarse-grained activity, independent of the rest of the management application; it does not have to run on the main management platform. In other words, it should lend itself very well to being ported to Web services. In fact, not only some, but all configuration settings that are under the control of a PDP should systematically be checked like this, by Web services or other means.

In this case, the Web service providers should probably be third-party machines, not operational routers, to avoid temporarily saturating them with management tasks.

Example 4: Long-lasting problem investigation

If we go back to example 1, some of the outstanding core problems may not be solved within a predefined time (e.g., 10 minutes). In such cases, it is appropriate to systematically outsource the resolution of these problems to third-party machines, which take over these investigations and run them as background tasks, possibly taking hours or days to come to a conclusion. Transferring these problem investigations to external entities can be achieved in a loosely coupled and portable manner using Web services. The objective here is to conserve precious few resources on the machine running the bulk of the management application, while not giving up on important but hard problems.

TYPE 3: MACRO-COMPONENTS AS WEB SERVICES

This time, we break up the management application into very large chunks that can be physically distributed over multiple machines. The idea is to implement the network transparency recommended by SOA to improve the scalability of the management application.

Example 1: Distributed monitoring application

In today's management platforms, the main building block of the management application is the monitoring application. We propose to break up this large, monolithic, single-vendor monitoring application into large chunks (macro-components) wrapped up with coarse-grained Web services. These macro-components can be provided by several vendors and integrated on a per-organization basis, depending on site-specific needs and customer-specific price conditions.

This integration seldom requires dynamic binding at run-time, because presumably the macro-components that make up management software will not change often. This restructuring of the monitoring application brings more competition among vendors and can drive costs down (feature-rich management applications capable of integrating the management of networks, systems and services are still incredibly expensive).

For instance, one vendor can provide elaborate graphical views of the network with 3D navigation, while another provides an event correlator with a smart rule description language, and a third provides the rest of the monitoring application. Standard Web service-based communication would enable these loosely-coupled services to work together.

Example 2: Integration of management

Web services can also be useful to integrate the different management platforms in charge of network management, systems management, service management, the trouble-ticket system, technology-specific platforms, etc. Doing so, they have the potential to nicely complement today's management solutions, for instance when the management platforms used in an enterprise cannot use the same event correlator due to integration problems (a serious problem that prevents management from being integrated).

TYPE 4: WEB SERVICES BETWEEN MANAGING/MANAGED ENTITIES

The fourth type of coarse-grained Web services is different from the previous three. Instead of focusing on the delegation granularity, it addresses the issue of service discovery. We present three examples: dynamic discovery of the manager, transparent redundancy of the managing entity, and dynamic discovery of the top-level manager.

Example 1: Dynamic discovery of the manager

In this example, instead of configuring the address of the managing entity in each agent of its domain, administrators rely on Web services for managed entities to discover their managing entity dynamically at run-time. This discovery typically uses a standard registry, either configured in the managed entity or discovered at boot time (e.g., with a multicast operation).

Example 2: Transparent redundancy during discovery

When managed entities think they bind directly to their managing entity, they can in fact connect to an intermediary Web service in charge of improving the robustness of management. This intermediary can transparently support two redundant managers with a hot stand-by (i.e., two machines that receive and analyze the same management data in parallel). The primary is active, the secondary is passive. If the primary fails, the secondary can take over transparently at very short notice. Alternatively, this intermediary can balance the load between multiple managers (which are all active). This can be useful in environments where managers experience important bursts of activity.

Example 3: Dynamic discovery of the top-level manager

In distributed hierarchical management (a technique commonly adopted in large enterprises and corporations [53]), the management domain is under the responsibility of a managing entity known as the *top-level manager*. The domain is logically split into sub-domains, usually for scalability reasons, and each sub-domain is under the responsibility of a managing entity called the *sub-level manager*.

In this scenario, instead of configuring the address of the top-level manager in each sub-level manager, administrators rely on Web services for the sub-level managers to discover automatically the top-level manager of their domain. This operation typically uses a registry.

6.5 MIGRATING TO SOA: IS IT WORTH THE EFFORT?

After studying the changes that the migration to SOA requires in management applications, let us ponder over these changes. Are they reasonable? Do the advantages of SOA outweigh the drawbacks of reengineering existing management applications?

WE NEED SOMETHING NEW

In the last decade, we have seen a number of proposals that challenged the organizational models of existing management architectures: Management by Delegation [92], mobile code [3], intelligent agents [42], multi-agent systems [36], and more recently P2P [86] and Web services. So far, none of them have succeeded in durably influencing the architecture and design of the management applications used in real life. Why should Web services succeed where other technologies failed?

As we saw in Section 1.2, the main problem with today's management platforms is that we are prisoners of habits that date back to an era when networks were small and simple to manage, resources were scarce, dependencies were not highly dynamic, event correlation was simple because we did not bother to integrate the management of networks, systems and services, etc.

Today, architecting an integrated management application for a corporation is very difficult, and the tools available on the market often fall short of customers' expectations. Designing these complex applications boils down to designing large and highly dynamic distributed applications. This requires the state of the art in software engineering, and the reengineering of integration management applications. Migrating to SOA can be the occasion for this clean slate approach.

Another important reason for adopting coarse-grained Web services is that they offer a solution for integrating multi-vendor dedicated management platforms (e.g., a platform dedicated to network management with another platform specialized in service management). This integration is a weak point of many of today's management platforms, and a number of corporations or large enterprises currently struggle to achieve this integration.

No Guarantee of Success

Businesswise, migrating integrated management platforms to SOA is not a guaranteed win. There is much uncertainty. The market of management applications needs to change its habits, and whether this will happen is difficult to predict. Decision makers in the IT industry are told every year that a new "revolution" has just occurred, so they tend to be skeptical. Convincing them will not be easy.

Oddly enough, the main non-technical driver for change may be the comfort offered by homogeneity: if everyone in the software industry adopts SOA, then there is a strong incentive for management applications to also migrate to SOA: few people enjoy being the last dinosaur in town...

Paradox

Migrating management platforms to Web services leads to an apparent paradox. On the one hand, SOA is about preserving investments in legacy systems by wrapping up existing applications with Web services. The goal here is to avoid the high cost of reengineering applications, especially complex ones. Based on this principle, using Web services in management applications should not require profound changes in these applications: we should not have to reengineer them. On the other hand, using Web services as getters and setters of managed objects defeats the point of coarse-grained and autonomous services in SOA. What is the catch?

This paradox is only apparent. It is due to a confusion between applications and services in SOA parlance. Services are meant to be reusable by several applications or other services; applications themselves are not. Management applications are not used as services by other applications: they are at the top of the food chain, so to say. Once SOA-enabled, applications can make use of different Web services without being entirely reengineered. But there is no free lunch in IT: they need to be made SOA-enabled in the first place.

Performance, Latency

The performance of Web service-based management solutions is still very much an open issue. Web services use SOAP messages underneath. This messaging technology, based on the exchange of XML documents, is notoriously inefficient [44]. In corporations and large organizations, integrated management platforms have drastic requirements in terms of performance. Breaking the management application into Web services-enabled macro-components, elaborate management tasks and simple management tasks may turn out to be totally impractical for performance reasons.

The question is particularly acute for communication between macro-components. For instance, wrapping up the event filter, the event aggregator and the event correlator with different Web services could lead to a sharp decrease in performance, because these macro-components need to exchange many events per time unit in large environments, latency must be kept low, and SOAP message exchanges are slow compared to the current tightly coupled practice. A performance analysis of management platforms using coarse-grained Web services is outside the scope of this chapter, but this matter deserves attention in follow-up work.

7 Conclusion

In this chapter, we summarized the main standards pertaining to Web services and presented different ways of using this middleware technology in integrated management. Fine-grained Web services are a direct mapping of current habits in management: each Web service operates at the managed object level. This is the approach currently adopted by standards bodies (e.g., in WSDM and WS-Management). We showed that it is efficient when many compressed managed objects are transferred in bulk. Unfortunately, it does not comply with SOA, the distributed computing architectural style that underpins Web services. Coarse-grained Web services, conversely, are SOA compliant and work at a higher level of abstraction. Each Web service corresponds to a high-level management task (e.g., balancing the load of a virtual resource across multiple physical resources) or a macro-component of a management application (e.g., an event correlator). Coarse-grained Web services are autonomous (each of them fulfills some functionality on its own), loosely coupled with the rest of the management application, and coarse-grained. We distinguished several categories of Web service for integrated management and illustrated them by examples.

In the future, will integrated management adopt the vision of Web services promoted by SOA? Answering this question is difficult because the market is only partially driven by technical considerations. From a business standpoint, adopting Web services makes sense because the software industry is massively adopting this technology, so there are many tools available and many experts in the field. From an engineering standpoint, Web services make it possible to organize management applications differently. This is appealing for architecting integrated management applications aimed at large organizations. It also opens new horizons regarding the organizational model of standard management architectures: we now have the technology to do without the "manager does everything, agent does nothing" principle. Web services make it possible to break up management platforms into independent macro-components, packaged as services, which can bring more competition among vendors and drive costs down. For instance, one vendor may provide elaborate views with Graphical User Interfaces (GUIs) while another may provide the event correlator. They also facilitate the integration of network, systems and service management tasks. But the performance of coarse-grained Web services for building management solutions is still an open issue that deserves thorough investigations. The same problem was true of CORBA and J2EE a few years ago, and satisfactory solutions appeared a few years later. Can we count on this for Web services too?

ACKNOWLEDGMENTS

The work presented in this paper was supported in part by the EMANICS Network of Excellence (#26854) of the IST Program funded by the European Commission. Early versions of this material were published in [24], [54] and [77].

REFERENCES

[1] G. Alonso, F. Casati, H. Kuno and V. Machiraju, *Web Services: Concepts, Architectures and Applications*, Springer, 2004.

[2] A. Alves, A. Arkin, S. Askary *et al. (Eds.)*, *Web Services Business Process Execution Language Version 2.0*, Public Review Draft, OASIS, August 2006. Available at: `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html`

[3] M. Baldi and G.P. Picco, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications", in R. Kemmerer and K. Futatsugi (Eds.), *Proc. 20th International Conference on Software Engineering (ICSE 1998)*, Kyoto, Japan, April 1998, pp. 146–155.

[4] K. Ballinger, D. Ehnebuske, C. Ferris *et al.* (Eds.), *Basic Profile Version 1.1*, Final Material, Web Services Interoperability Organization, April 2006. Available at `http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html`

[5] T. Banks, Web Services Resource Framework (WSRF) – Primer v1.2, Committee Draft 02, May 2006. Available at: `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`

[6] T. Bellwood, *UDDI Version 2.04 API Specification*, UDDI Committee Specification, OASIS, July 2002. Available at: `http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf`

[7] T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web", *Scientific American*, May 2001.

[8] K.P. Birman, "Like It or Not, Web Services Are Distributed Objects", *Communications of the ACM*, Vol. 47, No. 12, pp. 60–62, December 2004.

[9] D. Booth, H. Haas, F. McCabe *et al.*, (Eds.), *Web Services Architecture*, W3C Working Group Note, World Wide Web Consortium, 2004. Available at: `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`

[10] D. Box, E. Christensen, F. Curbera *et al.*, *Web Services Addressing (WS-Addressing)*, W3C Member Submission, August 2004. Available at: `http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/`

[11] V. Bullard and W. Vambenepe (Eds.), Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1, OASIS Standard, August 2006. Available at: `http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.htm`

[12] V. Bullard and W. Vambenepe (Eds.), Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 2, OASIS Standard, August 2006. Available at: `http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.htm`

[13] B. Burke and R. Monson-Haefel, *Enterprise JavaBeans 3.0*, 5th edition, O'Reilly, 2006.

[14] N. Catania, P. Kumar, B. Murray *et al.*, Overview—Web Services Management Framework, Version 2.0, July 2003. Available at: `http://devresource.hp.com/drc/specifications/wsmf/WSMF-Overview.jsp`

[15] K. Channabasavaiah, K. Holley and E. Tuggle, "Migrating to a Service-Oriented Architecture, Part 1", IBM, December 2003. Available at: `http://www-128.ibm.com/developerworks/webservices/library/ws-migratesoa/`

[16] R. Chinnici, J.J. Moreau, A. Ryman *et al.* (Eds.), *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Candidate Recommendation, March 2006. Available at: `http://www.w3.org/TR/2006/CR-wsdl20-20060327/`

[17] R. Chinnici, H. Haas, A.A. Lewis (Eds.), *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*, W3C Candidate Recommendation, March 2006. Available at: `http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327/`

[18] M. Choi, J.W. Hong and H. Ju, "XML-Based Network Management for IP Networks", *ETRI Journal*, Vol. 25, No. 6, pp. 445-463, December 2003.

[19] M. Choi and J.W. Hong, "Performance Evaluation of XML-based Network Management", Presentation at the *16th IRTF-NMRG Meeting*, Seoul, Korea, 2004, `http://www.ibr.cs.tu-bs.de/projects/nmrg/meetings/2004/seoul/choi.pdf`

[20] J. Clark and S. DeRose, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, November 1999. Available at: `http://www.w3.org/TR/1999/REC-xpath-19991116/`

[21] L. Clement, A. Hately, C. von Riegen *et al.* (Eds.), *UDDI Version 3.0.2*, UDDI Spec Technical Committee Draft, OASIS, October 2004. Available at: `http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm`

[22] F. Curbera, Y. Goland, J. Klein *et al.*, "Business Process Execution Language for Web Services (BPEL4WS) Version 1.1", May 2003. Available at: `http://www-106.ibm.com/developerworks/library/ws-bpel/`

[23] DMTF, *Web Services for Management (WS-Management)*, Version 1.0.0a, DSP0226, April 2006. Available at: `http://www.dmtf.org/standards/published_documents/DSP0226.pdf`

[24] T. Drevers, R. v.d. Meent and A. Pras, "Prototyping Web Services based Network Monitoring", in J. Harjo, D. Moltchanov and B. Silverajan (Eds.), *Proc. of the 10th Open European Summer School and IFIP WG6.3 Workshop (EUNICE 2004)*, Tampere, Finland, June 2004, pp. 135–142.

[25] R. v. Engelen, "gSOAP Web services toolkit", 2003, `http://www.cs.fsu.edu/~engelen/soap.html`

[26] R. v. Engelen and K.A. Gallivany, "The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks", in *Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, May 2002, pp. 128–135.

[27] T. Erl, *Service-Oriented Architecture: Concepts and Technology*, Prentice Hall PTR, 2005.

[28] R.T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures*, Ph.D. Thesis, University of California, Irvine, CA, USA, 2000. Available at: `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`

[29] R. Fielding, J. Gettys, J. Mogul *et al.* (Eds.), *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*, IETF, June 1999.

[30] S. Graham, A. Karmarkar, J. Mischkinsky *et al.* (Eds.), *Web Services Resource 1.2 (WS-Resource)*, OASIS Standard, 1 April 2006.

[31] M. Grand, *Patterns in Java, Volume 1: A Catalog of Reusable Design Patterns Illustrated with UML*, Wiley, 1998.

[32] M. Gudgin, M. Hadley, J.J. Moreau *et al.* (Eds.), *SOAP 1.2 Part 1: Messaging Framework*, W3C Candidate Recommendation, World Wide Web Consortium, 2002. Available at: `http://www.w3.org/TR/soap12-part1/`

[33] M. Gudgin, M. Hadley, J.J. Moreau *et al.* (Eds.), *SOAP 1.2 Part 2: Adjuncts*, W3C Candidate Recommendation, World Wide Web Consortium, 2002. Available at: `http://www.w3.org/TR/soap12-part2/`

[34] H. Haas and C. Barreto, "Foundations and Future of Web Services", Tutorial presented at the *3rd IEEE European Conference on Web Services (ECOWS 2005)*, Växjö, Sweden, November 2005.

[35] H.G. Hegering, S. Abeck and B. Neumair, *Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application*, Morgan Kaufmann, 1999.

[36] M.N. Huhns and M.P. Singh (Eds.), *Readings in Agents*, Morgan Kaufmann, 1998.

[37] IBM developerWorks, *New to SOA and Web Services*. Available at: `http://www-128.ibm.com/developerworks/webservices/newto/`

[38] IRTF-NMRG, *Minutes of the 11th IRTF-NMRG Meeting*, Osnabrück, Germany, September 2002. Available at: `http://www.ibr.cs.tu-bs.de/projects/nmrg/minutes/minutes-011.txt`

[39] ISO/IEC 10746-1, *Information Technology—Open Distributed Processing—Reference Model: Overview*, International Standard, December 1998.

[40] ITU-T, *Recommendation X.753, Information Technology—Open Systems Interconnection—Systems Management: Command Sequencer for Systems Management*, ITU, October 1997.

[41] ITU-T, *Recommendation M.3400, Maintenance: Telecommunications Management Network—TMN Management Functions*, ITU, February 2000.

[42] N.R. Jennings and M.J. Wooldridge (Eds.), *Agent Technology: Foundations, Applications, and Markets*, Springer, 1998.

[43] P. Kalyanasundaram, A.S. Sethi, C.M. Sherwin *et al.*, "A Spreadsheet-Based Scripting Environment for SNMP", in A. Lazar, R. Saracco and R. Stadler (Eds.), *Integrated Network Management V—Proc. of the 5th IFIP/IEEE International Symposium on Integrated Network Management (IM 1997)*, San Diego, CA, USA, May 1997, pp. 752–765.

[44] J. Kangasharju, S. Tarkoma and K. Raatikainen, "Comparing SOAP Performance for Various Encodings, Protocols, and Connections", in *Proc. of the 8th International Conference on Personal Wireless Communications (PWC 2003)*, Venice, Italy, September 2003, Springer, LNCS, Vol. 2775, pp. 397–406.

[45] N. Kavantzas, D. Burdett and G. Ritzinger (Eds.), *Web Services Choreography Description Language Version 1.0*, W3C Candidate Recommendation, World Wide Web Consortium, November 2005. Available at: `http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/`

[46] D. Levi and J. Schoenwaelder (Eds.), *RFC 3165: Definitions of Managed Objects for the Delegation of Management Scripts*, IETF, August 2001.

[47] D. Libes, *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*, O'Reilly, 1994.

[48] M. Little, “Transactions and Web Services”, *Communications of the ACM*, Vol. 46, No. 10, pp. 49–54, October 2003.

[49] K. Ma, “Web Services: What’s Real and What’s Not?”, *IEEE Professional*, Vol. 7, No. 2, pp. 14–21, March-April 2005.

[50] C.M. MacKenzie, K. Laskey, F. McCabe *et al.* (Eds.), *Reference Model for Service Oriented Architecture 1.0*, Committee Specification 1, OASIS, August 2006.

[51] F. Manola and E. Miller, *RDF Primer*, W3C Recommendation, February 2004. Available at: `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`

[52] D. Martin, M. Burstein, D. McDermott *et al.*, “Bringing Semantics to Web Services with OWL-S”, *World Wide Web: Internet and Web Information Systems*, to appear in 2007.

[53] J.P. Martin-Flatin, *Web-Based Management of IP Networks and Systems*, Wiley, 2003.

[54] J.P. Martin-Flatin, “Distributed Event Correlation and Self-Managed Systems”, in O. Babaoglu *et al.* (Eds.), *Proc. of the International Workshop on Self-* Properties in Complex Information Systems (Self-Star 2004)*, Bertinoro, Italy, May 2004, pp. 61–64.

[55] J.P Martin-Flatin, P.A. Doffoel and M. Jeckle, “Web Services for Integrated Management: a Case Study”, in L.J. Zhang and M. Jeckle (Eds.), *Web Services—Proc. of the 2nd European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany, September 2004, Springer, LNCS, Vol. 3250, pp. 239–253.

[56] K. McCloghrie, D. Perkins, and J. Schoenwaelder (Eds.), *RFC 2578: Structure of Management Information Version 2 (SMIv2)*, IETF, April 1999.

[57] K. McCloghrie and F. Kastenholz (Eds.), *RFC 2863: The Interfaces Group MIB*, IETF, June 2000.

[58] D.L. McGuinness and F. van Harmelen, *OWL Web Ontology Language Overview*, W3C Recommendation, February 2004. Available at: `http://www.w3.org/TR/2004/REC-owl-features-20040210/`

[59] S. McIlraith., T.C. Son and H. Zeng, “Semantic Web Services”, *IEEE Intelligent Systems*, Vol. 16, No. 2, pp. 46–53, March-April 2001.

[60] A. Nadalin, C. Kaler, R. Monzillo *et al.*, *Web Services Security: SOAP Message Security 1.1 (WS-Security)*, OASIS Standard Specification, February 2006.

[61] R. Neisse, R. Lemos Vianna, L. Zambenedetti Granville *et al.*, “Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways”, *Proc. of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, Seoul, Korea, April 2004, pp. 715–728.

[62] Net-SNMP home page, `http://net-snmp.sourceforge.net/`

[63] OASIS Web Services Distributed Management (WSDM) Technical Committee, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm`

[64] OASIS Web Services Resource Framework (WSRF) Technical Committee, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`

[65] OASIS Web Services Transaction (WS-TX) Technical Committee, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx`

[66] L. O’Brien, L. Bass and P. Merson, *Quality Attributes and Service-Oriented Architectures*, Technical Report CMU/SEI-2005-TN-014, Software Engineering Institute, Carnegie Mellon University, PA, USA, September 2005. Available at: `http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn014.pdf`

[67] R. Orfali, D. Harkey and J. Edwards, *Client/Server Survival Guide*, 3rd edition, Wiley, 1999.

[68] E. O’Tuathail and M. Rose, *RFC 4227: Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)*, IETF, January 2006.

[69] C. Pahl, “A Conceptual Framework for Semantic Web Services Development and Deployment”, in L.J. Zhang and M. Jeckle (Eds.), *Web Services—Proc. of the 2nd European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany, September 2004, Springer, LNCS, Vol. 3250, pp. 270–284.

[70] M.P. Papazoglou, “Web Services and Business Transactions”, *World Wide Web: Internet and Web Information Systems*, Vol. 6, No. 1, pp. 49–91, March 2003.

[71] M.P. Papazoglou and D. Georgakopoulos (Eds.), Special issue on “Service-Oriented Computing”, *Communications of the ACM*, Vol. 46, No. 10, October 2003.

[72] M.P. Papazoglou and W.J. van den Heuvel, “Web Services Management: A Survey”, *IEEE Internet Computing*, Vol. 9, No. 6, pp. 58–64, November-December 2005.

[73] S. Parastatidis, S. Woodman, J. Webber *et al.*, “Asynchronous Messaging between Web Services Using SSDL”, *IEEE Internet Computing*, Vol. 10, No. 1, pp. 26–39, January-February 2006.

[74] G. Pavlou, “Chapter 2. OSI Systems Management, Internet SNMP, and ODP/OMG CORBA as Technologies for Telecommunications Network Management”, in S. Aidarous and T. Plevyak (Eds.), *Telecommunications Network Management: Technologies and Implementations*, pp. 63–110, IEEE Press, 1998.

[75] G. Pavlou, P. Flegkas and S. Gouveris, “Performance Evaluation of Web Services as Management Technology”, Presentation at the *15th IRTF-NMRG Meeting*, Bremen, Germany, January 2004.

[76] G. Pavlou, P. Flegkas, S. Gouveris *et al.*, “On Management Technologies and the Potential of Web Services”, *IEEE Communications*, Vol. 42, No. 7, July 2004

[77] A. Pras, T. Drevers, R. v.d. Meent and D. Quartel, “Comparing the Performance of SNMP and Web Services-Based Management”, *IEEE e-Transactions on Network and Service Management*, Vol. 1, No. 2, December 2004.

[78] J. Rao and X. Su, “A Survey of Automated Web Service Composition Methods”, in J. Cardoso and A. Sneth (Eds.), *Proc. of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, July 2004, Springer, Vol. 3387, pp. 43–54.

[79] J. Schönwälder, A. Pras and J.P. Martin-Flatin, “On the Future of Internet Management Technologies”, *IEEE Communications Magazine*, Vol. 41, No. 10, pp. 90-97, October 2003.

[80] M. Singh and M.N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, Wiley, 2005.

[81] M. Sloman, “Policy Driven Management for Distributed Systems”, *Journal of Network and Systems Management*, Vol. 2, No. 4, pp. 333–360, December 1994.

[82] M. Sloman and K. Twidle. “Chapter 16. Domains: A Framework for Structuring Management Policy”. In M. Sloman (Ed.), *Network and Distributed Systems Management*, Addison-Wesley, 1994, pp. 433–453.

[83] J. v. Sloten, A. Pras and M.J. v. Sinderen, “On the Standardisation of Web Service Management Operations”, in J. Harjo, D. Moltchanov and B. Silverajan (Eds.), *Proc. of the 10th Open European Summer School and IFIP WG6.3 Workshop (EUNICE 2004)*, Tampere, Finland, June 2004, pp. 143–150.

[84] Soapware.org, `http://www.soapware.org/`

[85] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd Edition, Addison-Wesley, 1999.

[86] R. Steinmetz and K. Wehrle (Eds.), *Peer-to-Peer Systems and Applications*, Springer, LNCS, Vol. 3485, 2005.

[87] The Stencil Group, *The Evolution of UDDI—UDDI.org White Paper*, July 2002. Available at: `http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf`

[88] UDDI.org, *UDDI Technical White Paper*, September 2000. Available at: `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`

[89] W. Vogels, “Web Services Are Not Distributed Objects”, *Internet Computing*, Vol. 7, No. 6, pp. 59–66, November-December 2003.

[90] WASP UDDI, `http://www.systinet.com/products/wasp_uddi/overview/`

[91] K. Wilson and I. Sedukhin (Eds.), Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1, OASIS Standard, 01 August 2006. Available at: `http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.htm`

[92] Y. Yemini, G. Goldszmidt and S. Yemini, “Network Management by Delegation”, in I. Krishnan and W. Zimmer (Eds.), *Proc. of the 2nd IFIP International Symposium on Integrated Management (ISINM 2001)*, Washington, DC, USA, April 1991, pp. 95–107.

[93] C. Zhou, L.T. Chia and B.S. Lee, “Semantics in Service Discovery and QoS Measurement”, *IEEE Professional*, Vol. 7, No. 2, pp. 29–34, March-April 2005.

[94] Zlib home page, `http://www.zlib.org/`