In L.J. Zhang and M. Jeckle (Eds.), *Web Services – Proc. 2nd European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany, September 2004, Springer, LNCS 3250, pp. 239–253

Web Services for Integrated Management: a Case Study

Jean-Philippe Martin-Flatin[1], Pierre-Alain Doffoel[2] and Mario Jeckle[3]

[1] CERN, Geneva, Switzerland
jp.martin-flatin@ieee.org

[2] ESCP-EAP, Paris, France
pa@doffoel.com

[3] University of Applied Sciences Furtwangen, Furtwangen im Schwarzwald, Germany
mario@jeckle.de

Abstract. As evidenced by discussions in standards organizations, vendors and the user community have recently showed a growing interest in using XML technologies for management purposes. To investigate the relevance of this approach, we have added support for Web Services to JAMAP (a Java-based research prototype of a management platform) and managed a gigabit transoceanic testbed. In this paper, we present the main lessons learned during this process and attempt to draw conclusions of general interest as to the applicability of Web Services for managing IP networks and systems. Our main conclusions are that XML, WSDL and SOAP are useful, especially for configuration management, whereas UDDI is not adequate. To date, we still lack a standard way of publishing, discovering and subscribing to Web Services for the purpose of managing network devices and systems.

1 Introduction

Recent discussions in the Internet Engineering Task Force (IETF), Internet Research Task Force (IRTF) and Distributed Management Task Force (DMTF) demonstrate the management community's growing interest in using Web technologies—especially those based on the Extensible Markup Language (XML)—in management systems [16]. At the same time, the consortia in charge of standardizing Web Services technologies have showed an increasing interest in management.

In the DataTAG Project [7], we wanted to manage gigabit network devices and systems in a distributed manner. We had a cluster of PC servers and network devices (routers and switches from different vendors) at both ends of our transoceanic gigabit network (at CERN in Geneva, Switzerland and at StarLight in Chicago, IL, USA). To assess the suitability of Web Services for the purpose of management, we ported JAMAP [11], an open-source research prototype of a management platform, to Web Services. We could have used integrated and more sophisticated software suites such as IBM's WebSphere and BEA's WebLogic, or a management product such as HP's Web Services Management Framework. But using JAMAP gave us complete control over the code and allowed us to change each component independently of the others.

The remainder of this article is organized as follows. First, we review the main building blocks of Web Services. Next, we summarize the management architecture that underpins JAMAP. Then we analyze the advantages of using XML-based technologies in integrated management. Next, we describe how we leveraged Web

Services in JAMAP and draw some lessons of general interest. Finally, we conclude and present directions for future work.

2 Overview of Web Services

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 World Wide Web Consortium (W3C) [25] and the Organization for the Advancement of Structured Information Standards (OASIS) [15]. More recently, a new industrial consortium, the Web Services Interoperability Organization (WS-I) [24], has begun standardizing interoperability aspects of Web Services.

2.1 W3C Activities

The W3C's activities on Web Services are split into four areas:

- Simple Object Access Protocol (SOAP)[1]
- Web Services Definition Language (WSDL)
- Web Services Architecture (WSA)
- Web Services Choreography (WS-Choreography)

SOAP [8][9] is a protocol that allows applications to exchange structured information in a distributed environment. The SOAP binding used by most applications 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 consists of a header and a body. The header is optional and carries metadata; the body is mandatory and includes the actual application payload. 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. The SOAP specification also specifies an XML representation for Remote Procedure Call (RPC) invocations and responses carried in SOAP messages.

WSDL [6][10] is an XML language for describing Web Services. Building on the base communication mechanism defined by SOAP, each Web Service is characterized by its signature, which consists of a procedure name, the type of the result returned by this procedure, the names and types of the procedure parameters, and the actual message pattern chosen for communication. Multiple Web Services can be published in a single WSDL file, often called a WSDL repository. If we compare Web Services with CORBA, the WSDL language is similar to the Interface Definition Language (IDL); a WSDL repository is similar to CORBA's Interface Repository; Web applications may discover Web Services in a WSDL repository and invoke them

1. At the time of its definition, the acronym stood for Simple Object Access Protocol. In the standardized version, SOAP is no longer an acronym.

dynamically, just as CORBA applications may discover an object's interface on the fly and invoke it using the Dynamic Invocation Interface. But the underlying development paradigm fundamentally differs. CORBA requires a developer to create an IDL description before implementing clients and servers that match the defined interface, whereas WSDL descriptions may be deployed for generating implementations but their use is not mandated. WSDL-compliant interface descriptions may be provided after the initial creation of the service. Additionally, the central storage of WSDL descriptions within a designated repository is not required by the Web Service paradigm. The service provider may also choose to serve WSDL descriptions at the physical location where the service is offered. When doing so, no standard repository has to be offered, although standard lightweight discovery mechanisms may be deployed [3].

WSA [5] defines core architectural concepts (e.g., discovery and life cycle) and relationships that are central to the interoperability of Web Services. It also defines a set of constraints and examines how the architecture meets the Web Services requirements expressed by stakeholders. Management issues are mentioned in the final document but the W3C has paid little attention to them. To date, this activity has not produced a formal standard prescribing a fixed architecture; it merely collected current concepts and terms and documented their relationships.

Choreography deals with the composition of Web Services and the description of relationships between Web Services. It is also known as orchestration, collaboration or coordination. The nascent W3C activity on choreography [12] is essentially based on Sun's Web Service Choreography Interface (WSCI) [1].

2.2 OASIS Activities

OASIS is an industrial consortium responsible for standardizing many domain-specific aspects of Web Services, especially ebXML, a markup language for e-business. The main general-purpose technology standardized by OASIS and relevant to integrated management is Universal Description, Discovery and Integration (UDDI). We will come back to UDDI later.

The OASIS Web Services Distributed Management Technical Committee recently began working on Web Services management. This activity covers two aspects: using Web Services to manage distributed resources and managing Web Services (the latter includes modeling a Web Service as a manageable resource).

In the area of choreography, the OASIS Web Services Business Process Execution Language Technical Committee took over the BPEL4WS [19] proposal by Microsoft, IBM *et al.* and is now standardizing it under the name Business Process Execution Language (BPEL). The main objectives of this work are to describe process interfaces for business protocols and define executable process models.

2.3 WS-I Activities

WS-I focuses on developing profiles, usage scenarios, use cases, sample applications and testing tools to facilitate interoperability between the Web Services platforms of its members. This industrial consortium began its activities in February 2002; few specifi-

cations have been released to date. The most significant is Basic Profile 1.0 [4], which consists of implementation guidelines as to how core Web Services specifications should be used together to develop an interoperable Web Services infrastructure. Management aspects have not been addressed yet by WS-I, but interoperability is critical for management systems.

3 WIMA and Design of JAMAP

Now that we have summarized the state of the art in Web Services, let us study the design of the open-source software used in this project: JAMAP. This research prototype of a management platform implements the Web-based Integrated Management Architecture (WIMA) [14] in Java. WIMA leverages XML's self-description capability to integrate SNMP[1] Management Information Base (MIB) data and Common Information Model (CIM) objects in a seamless manner, which is particularly useful in heterogeneous environments. WIMA is well suited to integrated management, that is, the integration of management data pertaining to network devices, systems, applications, end-to-end networks, services, etc.

In WIMA, agents publish the monitoring data and notifications they can send, and management applications (*managers*) subscribe to them in a semi- or fully automated way. The same publish-subscribe mechanism is used for manager-to-manager communication, when managers are organized hierarchically to manage a large domain or different domains. Data transfers are based on HTTP.

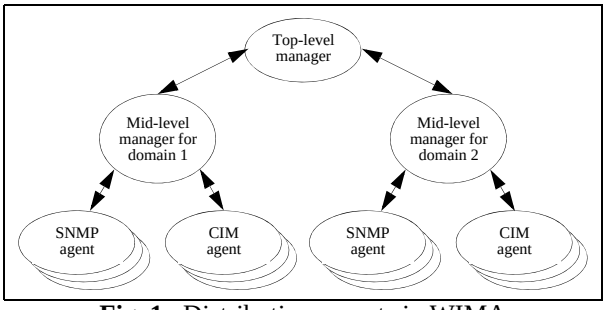

Fig. 1. Distribution aspects in WIMA

WIMA supports distributed management—to be precise, a weakly distributed hierarchical paradigm if we use the taxonomy defined in [14]. This allows administrators to split an organization into multiple management domains if the amount of management data to process grows large. Management domains may be defined according to many criteria: geographical location, management area (e.g., network

1. SNMP is the Simple Network Management Protocol [18], currently the main standard for managing networks.

management vs. service management), profit center, customer (e.g., when Internet Service Providers do virtual hosting), etc.

In the example depicted in Figure 1, the organization has adopted a three-tier management hierarchy. We have one top-level manager for the entire organization, one mid-level manager per management domain, and a number of agents (up to a few hundred) per domain.

In WIMA, the top-level manager runs the event correlator, which is the smart part of the management application. JAMAP 1.3 features a simple rule-based engine implemented in Java.

In WIMA, each mid-level manager is broken up into five components [14]:

- the *data analyzer*, which analyzes monitoring data on the fly, detects problems, and sends events to the event correlator when problems are detected;
- the *data collector*, in charge of collecting and filtering data on a regular basis for the purpose of monitoring; incoming data can be processed immediately by the data analyzer, archived in the data repository, or both;
- the *notification collector*, in charge of receiving incoming SNMP traps and CIM events, filtering them and forwarding them to the event correlator; it may also archive some of these incoming events in the data repository;
- the *configuration manager*, in charge of automatically configuring agents;
- the *background analyzer*, which performs data mining and non-realtime analysis on the data archived in the data repository; it may also send events to the event correlator when problems are detected.

WIMA allows each mid-level manager to be distributed across several physical machines. The mapping between the five previous components and physical machines is not constrained by WIMA.

In JAMAP 1.3, the first three components are implemented and the fourth is under development. Data collection, notification collection and data analysis are fully distributed, whereas event correlation is not. Each mid-level manager can comprise one or several data collectors (e.g., one for network management and another for service management), one or several notification collectors (e.g., one for SNMP traps and another for CIM events), and exactly one data analyzer.

Unlike most SNMP-based management platforms found to date on the market, JAMAP uses publish-subscribe (see Figure 2). SNMP MIBs supported by the agent are published in the data subscription applet, which can be downloaded by any management station. This applet communicates with the data subscription servlet inside the agent. This allows the manager to subscribe to specific MIB Object Identifiers (OIDs) and specify a push frequency for each OID. The data subscription servlet updates the push schedules in a persistent repository. From then on, management data is pushed regularly by the agent to the manager. The data is sent directly to the manager when we can run JAMAP software directly on the agent (e.g., when the agent is a Linux PC or a Windows PC); otherwise, data is pushed via a proxy (e.g., in our testbed, when the agent is a Cisco router).

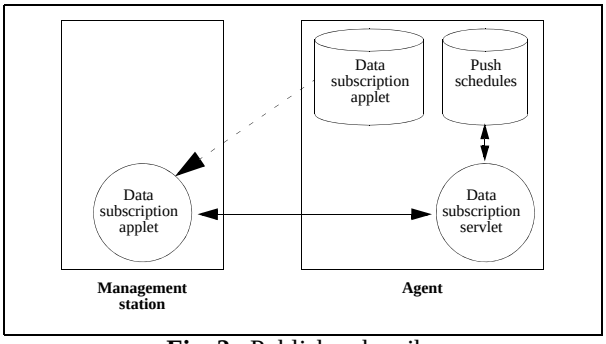

Fig. 2. Publish-subscribe

When JAMAP was implemented, the focus was on the communication and organizational aspects. Release 1.3 includes no fancy GUIs and no dynamic discovery of the network topology. The event correlator supports a limited set of rule templates, and no state is currently retained by the rule engine between successive push cycles.

4 Why Use XML in Integrated Management?

The main reasons for using Web technologies in general, and XML in particular, in integrated management are analyzed in detail in [14] and only summarized here. Some of them are generic, others are specific to network and systems management.

4.1 Advantages of Using XML in General

Probably the main advantage of using XML in software engineering is that it is both standard and stable. It is backed by the W3C, which has a good track record of independence vis-à-vis any particular vendor's interests. Unlike Java, whose specifications changed several times a year for many years, XML has been through very few release cycles, and the ill-designed DTDs were swiftly replaced by XML schemas. Markup languages defined in XML come and go and may change often, but XML itself is very stable.

Second, it is platform-neutral and facilitates interoperability. The people who devised XML had learned from the mistakes of the Common Object Request Broker Architecture (CORBA); e.g., the Object Management Group (OMG), which is in charge of CORBA, took many years to standardize the Portable Object Adapter, even though important operations had not been standardized in the Basic Object Adapter and required platform-specific extensions.

Third, XML is simple and easy to learn. The learning curve of developers is usually short and training costs low (many XML tools are freely available).

Fourth, XML is widely used in industry. As a result, most software engineering students want to learn it, people who recently graduated usually know it, and many mature developers want to study it.

Fifth, using XML-based technologies is often a way to reduce costs. With XML and Web Services, organizations can often implement simple solutions based on loosely coupled middleware, instead of reengineering legacy applications and replacing legacy systems.

Sixth, W3C's XML Schema Definition (XSD) allows XML parsers to validate XML documents. This increases the robustness of XML-based distributed applications. Robustness has become a major concern in industry, as a growing number of enterprises entirely depend on the availability of their software systems to run their businesses.

Seventh, and more arguably, XML-based applications are easier to debug because XML is human readable. This argument is less compelling than it used to be, because a large number of XML documents now use metamodel mappings rather than model mappings [14].

4.2 Advantages of Using XML in Network and Systems Management

Middleware technologies have blossomed in the past decade. Until recently, when administrators purchased a management platform, they had to choose between CORBA, EJBs, .NET, proprietary middleware, etc. This variety of poorly compatible solutions increases development costs for vendors, who need to support multiple technologies; it augments purchase costs for customers, who pay for this variety even if they do not need it; but most of all, it decreases the safety and long-term visibility of customers' investments. If an administrator selects a middleware that is abandoned by his company a couple of years later, he will bear the responsibility for this "wrong" decision.

XML and Web Services allow for a "truce in the middleware war" [14]. Compared to full-blown object-oriented distributed environments, where everything must be an object, they keep a low profile. They can cope with object-oriented models at the edges but do not require them; they can also deal with data models; they can even extract data from old proprietary repositories designed several decades ago. By trying to achieve less and by successfully tackling interoperability from day one, XML-based technologies constitute a rather safe investment.

Another advantage of using XML in management is that it allows for a clean separation between information and communication models. The limitations of SNMP, which bundles the two, are explained in [14].

Third, as far as the communication model is concerned, XML is easy to use for representing management data in transit between agents and managers, or between managers in hierarchical management.

Fourth, regarding the information model, the success encountered by XML schemas is compelling: they have been adopted exceptionally quickly throughout the industry, and there is nothing specific to integrated management that should prevent this industry from leveraging XML. This message has been advocated by the DMTF for years; the IETF may soon be convinced, as evidenced by the recent work of the Network Configuration Working Group.

Fifth, as we experimented during this project, XML is appropriate for expressing persistent management data, especially configuration files, in a heterogeneous environment.

Sixth, XML facilitates the integration of multiple management areas (in our case, network management and systems management) by offering a general-purpose means of representing self-describing data, whatever the data. By doing so, it makes it easy to deal with the heterogeneity of information models found in real life.

4.3 Disadvantages of Using XML

XML is not the panacea, however. Its main disadvantages are threefold.

First, it is verbose. This increases network overhead, but also processing time and resource consumption at the edges. Although this is generally not an issue, as most layered software architectures used today are quite verbose, it can be problematic for resource-constrained equipment such as embedded systems, cell phones and inexpensive commodity devices. To cope with bandwidth problems arising from XML's verbose textual notations, W3C recently started the XML Binary Characterization Working Group, which deals with binary compression of XML content.

Another problem is that XML schemas and DTDs are so simple to create that vendors lack incentives to comply with standards. Why should self-describing data comply with XML schemas produced by slow-paced multi-vendor consortia, when they could be produced quicker and made publicly available by each vendor? Since it emerged in 1990, the SNMP management market has demonstrated that customers are not eager to use SNMP MIBs and information models produced by standards bodies: they want functionality. This problem can be alleviated by using Extensible Stylesheet Language Transformations [13], a generic and standard mechanism for transforming automatically instances of one XML vocabulary into another.

Last, validating an XML document takes time and consumes CPU and memory resources. Some agents cannot afford to do that. Therefore, management applications cannot always rely on validation. This is unfortunate because validating incoming XML documents is a proven way to make management applications more robust.

These disadvantages exist, but they are in our view outweighed by the advantages of using XML.

5 How to Use Web Services in a Management Platform

In the Internet world, most management platforms focus on four aspects [14]:

- *Regular management* consists of management tasks that run continuously, in pseudo-real time, over long periods of time. It encompasses monitoring, data collection (for background analysis), and notification handling.
- *Ad hoc management* consists of management tasks that run occasionally, if need be, for a short time. It comprises troubleshooting and short-term monitoring, and operates in pseudo-real time.

– *Configuration management* consists in changing the setup of an agent to make it operate differently.
– *Background analysis* includes all the management tasks that run in the background (as opposed to pseudo-real time) and strive to make sense of the data gathered by regular management (data collection). Examples include security analysis and the generation of daily usage reports.

For the sake of conciseness, let us focus on monitoring, a form of regular management that is implemented in JAMAP 1.3.

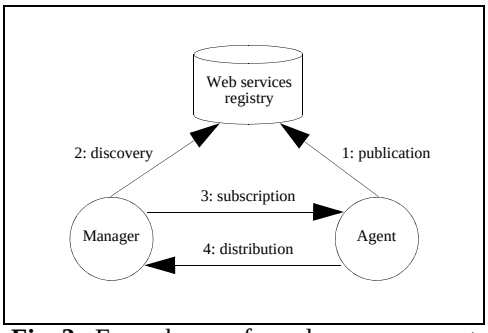

Fig. 3. Four phases of regular management

The four phases of monitoring (publication, discovery, subscription and delivery) are explained in detail in [14] and summarized in Figure 3. A priori, Web Services could be used at all levels in JAMAP:

– *Publication phase*: To allow agents to announce what data (e.g., SNMP OIDs or CIM objects) can be pushed to managers (respectively, in hierarchical management, to allow mid-level managers to announce what data can be pushed to top-level managers).
– *Discovery phase*: To allow managers to discover dynamically what management data can be pushed by agents (respectively, to allow the top-level manager to discover what data can be pushed by mid-level managers).
– *Subscription phase*: To allow managers to subscribe to data provided by agents by invoking Web Services directly on these agents (respectively, to allow the top-level manager to subscribe to data provided by mid-level managers).
– *Distribution phase*: To allow agents to push data to managers (respectively, to allow mid-level managers to push data to the top-level manager).

6 Publish-Subscribe and Discovery with UDDI

The UDDI technology is often advertised, or thought of, as a general-purpose technology implementing publish-subscribe for Web Services [20][22]. As we needed such a mechanism in JAMAP, we studied, deployed, tested and evaluated UDDI.

To analyze the relevance of this technology for managing our testbed, we first adopt a top-down approach and study where UDDI registries allow us to store data useful for integrated management. Next, in a bottom-up approach, we model the management information that we want to store in a publish-subscribe registry and investigate how it fits with UDDI.

6.1 Top-Down Approach

Three versions of UDDI have been specified to date. UDDIv1 is now considered historic. When we conducted this work, the market was dominated by UDDIv2 and had not yet begun migrating to UDDIv3. The main new features in UDDIv3 are registry interaction and versioning [20].

After investigating a number of platforms listed in the Soapware directory [17] or mentioned in the `xml-dist-app@w3.org` mailing list, we selected Systinet's WASP UDDI [23], which offers a good balance between features, compliance with the latest specifications and free availability to researchers. The version that we tested (release 4.5.2) implements UDDIv2 and supports a few UDDIv3 features (e.g., subscriptions and notifications, which allow clients to automatically receive notification of changes made to registered Web Services).

The XML schema that underlies UDDI registries is rather simple. It consists of four elements:

- *businessEntity*: describes a business or an organization; the information provided here is equivalent to the yellow pages of a telephone directory: name, description, contact people, etc.
- *businessService*: provides a high-level description of a service provided by a company or an organization in business terms; this information is similar to the taxonomic entries found in the white pages of a telephone directory.
- *bindingTemplate*: provides a technical description of a given business service; it includes either the access point (e.g., a URL or an e-mail address) of this Web Service or an indirection mechanism that leads to the access point.
- *tModel*: technical models contain (i) pointers to technical documents used by developers of Web Services and (ii) metadata about these documents; they represent unique concepts or constructs, facilitate reuse and enable interoperability; they are primarily used as sources for determining compatibility between providers and consumers of Web Services and as keyed namespace references.

The first three elements are hierarchically structured: a business entity logically contains one or several business services, and a business service logically contains one or several binding templates. The fourth element lies outside this hierarchy: a binding template includes references to technical models (*tModels*) but does not contain the

tModels themselves; as a result, a single *tModel* may be referenced by several binding templates.

The UDDI business entities and services are very coarse-grained. In practice, they can be used in two ways. First, a UDDI registry may be used to advertise the business services offered by a given business entity to other companies. A business publishes its core business activities and a potential customer may want to discover this information.

Alternatively, or sometimes concurrently, a UDDI registry may be used to announce within a company the services offered to its own staff. This use was not initially envisioned in UDDIv1 and UDDIv2, but the authors of UDDIv3 now advocate it as an important use of UDDI [20]. Corporations are sometimes organized into profit centers that run as independent businesses and charge each other. Using UDDI registries to discover what services are available within a corporation makes sense in such environments. For example, within CERN, IT and Administration (among others) offer services to all CERN staff. Both of them can therefore be modeled as business services within the business entity named "CERN".

Next in the UDDI hierarchy, the concept of binding template is more flexible than business entities and services: we can put more or less anything we want into it. So, binding templates are the only entities that we can define to make UDDI useful to manage network devices and systems.

6.2 Bottom-Up Approach

In JAMAP, publish-subscribe operates at a much finer-grained level of abstraction than UDDI business entities and services. For instance, a managed element may want to publish that it supports SNMPv2c and the version of MIB-II specified in RFC 1213; another may publish that it supports CIM Specification 2.2, CIM Core schema 2.7 and CIM System schema 2.7. How can we model that in a hierarchical way à la UDDI?

An intuitive information model would be as follows. Within the business entity "CERN", we find the business service "IT". Within the latter, we find the finer-grained service "DataTAG networking research". Within the latter, we find the service "gigabit network monitoring". Within the latter, we want to list all the agents that can be managed by JAMAP. Each agent then wants to announce the information models (SNMPv2c, CIM, etc.) that it supports. For a given agent (e.g., `w02gva.datatag.org`) and a given information model (e.g., "SNMP"), we want to advertise the list of data models (e.g., SNMP MIBs) supported by this agent. Since many agents only support portions of MIBs, we want agents to be able to publish what SNMP OIDs they support. And finally, we want to allow managers (programs) to subscribe to each OID at a given frequency (e.g., retrieve `ifInOctets` every 15 minutes).

This model has two shortcomings: the DataTAG project involves several research institutes, not just CERN, and we monitor a testbed network that does not belong to CERN.

A more suitable information model would be the following. Within business entity "European Union", we have another finer-grained business entity called "FP5/IST

Projects". Within this entity, we find an element "project" of which "DataTAG Project" is an instance. Within a project, we find partners and activities. We model project partners by a sequence of XML elements each called partner; one of them is "CERN". So, our XML schema would already require four layers where UDDI only provides one: the business entity.

Next, project activities can similarly be modeled as a sequence of XML elements each called activity; one of them is "network monitoring". This activity can legitimately be modeled as a Web Service, since it is indeed a service offered to all project partners. Within this activity, we define two management domains: one called "CERN", which covers all the testbed systems and network devices located in Geneva; and another called "StarLight", which covers equipment in Chicago. In each management domain, we have agents, one per managed element. Each agent then advertises the information models that it supports. The rest of the information model is similar to the previous.

Unfortunately, this cannot be modeled using UDDI. We investigated whether finer-grained information could be stored in UDDIv2 and UDDIv3. We tried to leverage the *instanceParms* element of an *instanceDetails* structure in a *bindingTemplate*, to no avail.

With the simple information model currently supported by UDDI, we can only model that a company supports a management application and provides an access point for it at a given URL. This model is much higher level than what is required by JAMAP for the purpose of publish-subscribe and discovery between agents and managers. This conclusion is valid for all three versions of UDDI.

7 XML-Based Configuration Management in JAMAP

As UDDI registries are not appropriate to publish and discover agents in integrated management, we devised an XML schema for monitoring networks and systems with JAMAP, and used XML files to publish and discover management information.

7.1 Publication and Coarse-Grained Discovery

In JAMAP 1.3, a manager can discover all the agents within its domain by parsing an XML configuration file (`networkMap.xml`) that describes the organization's network. This file contains the addresses of the agents, management domain by management domain, and, for each agent, dynamic information such as the URL of its data collector servlet, an optional proxy address, and the URL of the agent's configuration file (`agentManagement.xml`).

To increase robustness, we strived to be precise in the XML schema definition file (`networkMap.xsd`). For instance, an IP address is not simply a string: we give precise definitions of IPv4 and IPv6 addresses. This allows JAMAP to validate effectively the XML documents exchanged between managers and agents. In JAMAP 1.3, publication is still manual.

7.2 Fine-Grained Discovery

Each agent has an `agentManagement.xml` file associated with it. This file may be published by the agent itself or another machine. It can be validated against the tailor-made XML schema mentioned above.

```
<agentManagement>
  <accessPoint>
    http://137.138.35.18:8080/services/AgentConfigurationService
  </accessPoint>
  <WsdlFileLocation></WsdlFileLocation>
  <dataSubscriptionApplet>
    http://137.138.35.18:8080/DataSubscriptionApplet.jsp
  </dataSubscriptionApplet>
  <subscriptionSheetServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.SubscriptionSheetServlet?
  </subscriptionSheetServlet>
  <getServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.Get?
  </getServlet>
  <pushDispatcherServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.PushDispatcherServlet?
  </pushDispatcherServlet>
  <agentConfigurationServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.AgentConfigurationServlet?
  </agentConfigurationServlet>
  <informationModels>
    <snmpInformationModel>
      <rfc>/mibs/rfc1213-mib.txt</rfc>
      <snmpv1CommunityString>public</snmpv1CommunityString>
      <encodingType>plainText</encodingType>
    </snmpInformationModel>
  </informationModels>
</agentManagement>
```

Fig. 4. Example of `agentManagement.xml` file

The `agentManagement.xml` file contains agent-specific information that a manager needs to know in JAMAP (see Figure 4). First, we find information pertaining to the configuration Web Service of the agent (access point and WSDL file location). Next, we find the URL of the Java applet used for manual configuration. Then, we define the URLs of a number of Java servlets run by the agent: one for configuring the agent, another for pushing data to the manager, etc. Last, we specify the information models and data models that are supported by the agent. For the SNMP information model, each entry indicates the RFC defining a specific version of an SNMP MIB, the community string, and the encoding: Basic Encoding Rules, XML, serialized Java, plain text, etc. For the CIM information model, each entry includes the name of the schema, its version number, and the encoding.

One advantage of our XML schema is that it makes it easy to add new information models or agent-specific information. Another is that it allows for strong validation of XML documents.

7.3 Subscription

Subscriptions can be either manual or automated. If a subscription is done manually, we can call directly the subscription Web Service on the agent by using the information retrieved from the previous files. Alternatively, we can use an applet if a URL is specified. When subscriptions are automated, we just need to edit an XML

subscription file that is located at a certain URL and contains, for each agent, the subscribed data and its push frequency.

The automation of subscriptions relies on simple criteria, as illustrated by the following examples:

- in management domain “CERN”, for all devices of type “Linux PC” supporting the SNMP information model, retrieve the `ifInOctets` and `ifOutOctets` columnar objects every 15 minutes if the PC supports RFC 1213 (MIB-II), and retrieve the `hrSWRunPerfCPU` and `hrSWRunPerfMem` columnar objects every 5 minutes if the PC supports RFC 1514 (Host Resources MIB);
- in all management domains, for all devices of type “Cisco 76xx”, retrieve the `ifInOctets` and `ifOutOctets` columnar objects every 5 minutes.

8 Lessons Learned

A number of lessons of general interest can be learned from this case study and may hopefully prove useful to other projects.

Easy to use: As claimed by many Web enthusiasts, XML is indeed easy to use and debug. The Simple API for XML (SAX) makes it very easy to parse an XML document in Java and validate it against an XML schema. Tomcat [21], the Apache open-source package that implements Java servlets, is simple to use and well documented.

Web Services: Axis [2], the Apache incarnation of Web Services, works in simple cases but still suffers from teething problems. Web Services discovered in a WSDL repository cannot be invoked dynamically if they use complex types (other than string and integer); instead, one has to use stubs à la CORBA. This explains why `WsdlFile-Location` is empty in Figure 4. Also, invoking a Web Service from within an applet requires the Java applet security scheme to be turned off, unless one uses commercial security certificates. Hopefully, future versions of Axis will address these issues.

Portability: XML is highly portable and is very appropriate for defining configuration files used by management applications. Similarly, both Tomcat and Axis are portable: they work fine on Linux 2.4.20, Windows XP and Windows 2000 platforms.

SOAP: SOAP can be used to push data between managers and agents but it offers little flexibility. A SOAP toolkit usually comes as a black box: there is often no easy way to control the HTTP connection underneath (e.g., for setting socket or TCP options, or for using long-lived HTTP connections). This is a problem for JAMAP, since we want to avoid the overhead of frequently setting up and tearing down HTTP connections.

Discovery: There are currently no standard ways of publishing, discovering and subscribing to fine-grained Web Services for integrated management. UDDI is too coarse-grained for our purposes. Hopefully, a new solution will come up in the future.

9 Conclusion

In this paper, we have described how we implemented Web Services in JAMAP and summarized the lessons learned in this process. Our conclusions are fourfold: Web Services are suitable for managing network devices and systems; XML portability facilitates integration in a heterogeneous environment; UDDI is a white-page service for e-commerce and is too coarse-grained for managing network devices and systems; and we still lack a standard way of publishing, discovering and subscribing to monitoring services between managers and agents.

In the future, it would be useful to devise a generic mechanism to publish, discover and subscribe to management Web Services. This mechanism should make it possible for management application designers to use any XML schema, as opposed to a fixed schema as in UDDI. Another interesting challenge would be to allow generic components of a management application to discover and bind to one another by using Web Services. Would the flexibility offered by component software be outweighed by the decrease in performance?

Acknowledgments

This research was carried out while P.A. Doffoel was an M.Sc. student at ENSIMAG doing an internship at CERN. Part of this work was funded by the FP5/IST Program of the European Union (DataTAG project, grant IST-2001-32459).

References

1. A. Arkin, S. Askary, S. Fordin *et al.*, (Eds.), *Web Service Choreography Interface (WSCI) 1.0*, W3C Note, World Wide Web Consortium, 2002. Available at `http://www.w3.org/TR/2002/NOTE-wsci-20020808/`.
2. Axis, `http://ws.apache.org/axis/`.
3. K. Ballinger, P. Brittenham, A. Malhotra *et al.*, (Eds.), *Web Services Inspection Language (WS-Inspection) 1.0*, IBM, 2001. Available at `http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html`.
4. K. Ballinger, D. Ehnebuske, M. Gudgin *et al.*, (Eds.), *Basic Profile Version 1.0*, Final Material, Web Services Interoperability Organization, 2004. Available at `http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html`.
5. 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/`.
6. R. Chinnici, M. Gudgin, J.J. Moreau *et al.* (Eds.), *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Working Draft, World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/WD-wsdl20-20040326/`
7. DataTAG Project, `http://www.datatag.org/`.
8. 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/`.

9. 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/`.
10. M. Gudgin, A. Lewis and J. Schlimmer (Eds.), *Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns*, W3C Working Draft, World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326/`.
11. JAMAP 1.3, `http://www.datatag.org/jamap/`.
12. N. Kavantzas, D. Burdett and G. Ritzinger (Eds.), *Web Services Choreography Description Language Version 1.0*, W3C Working Draft, World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/`.
13. M. Kay (Ed.), *XSL Transformations (XSLT) Version 2.0*, W3C Working Draft, 2003. Available at `http://www.w3.org/TR/2003/WD-xslt20-20031112/`.
14. J.P. Martin-Flatin, *Web-Based Management of IP Networks and Systems*, Wiley, 2002.
15. Organization for the Advancement of Structured Information Standards (OASIS), `http://www.oasis-open.org/`.
16. 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, Oct. 2003.
17. Soapware.org, `http://www.soapware.org/`.
18. W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd Edition, Addison-Wesley, 1999.
19. S. Thatte (Ed.), *Business Process Execution Language for Web Services (BPEL4WS) Version 1.1*, 2003. Available at `http://www.ibm.com/developerworks/library/ws-bpel/`.
20. The Stencil Group, *The Evolution of UDDI—UDDI.org White Paper*, July 2002.
21. Tomcat, `http://jakarta.apache.org/tomcat/`.
22. UDDI.org, *UDDI Technical White Paper*, Sept. 2000.
23. WASP UDDI, `http://www.systinet.com/products/wasp_uddi/overview/`.
24. Web Services Interoperability Organization (WS-I), `http://www.ws-i.org/`.
25. World Wide Web Consortium (W3C), `http://www.w3.org/`.