Research & Technological Development for a TransAtlantic Grid

DataTAG

Technical Report DataTAG-2003-1
FP5/IST DataTAG Project

Web Services for Integrated Management: a Case Study

J.P. Martin-Flatin and P.A. Doffoel

22 November 2003

www.datatag.org

EU grant IST 2001-32459

Web Services for Integrated Management: a Case Study

Jean-Philippe Martin-Flatin, CERN
Pierre-Alain Doffoel, ESCP-EAP

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 research prototype of a management platform—and managed a gigabit transoceanic testbed. In this paper, we present the lessons learned during this process and attempt to draw conclusions of general interest as to the applicability of Web Services for managing Internet 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.

INTRODUCTION

Recent discussions in the Internet Engineering Task Force (IETF), the Internet Research Task Force (IRTF), and the Distributed Management Task Force (DMTF) demonstrate the management community's growing interest in using Web-based technologies—especially those based on the eXtensible Markup Language (XML)—in management systems [1]. At the same time, the consortia in charge of standardizing Web services technologies have showed an increasing interest in management; two of the most active are the W3C Web Services Architecture Working Group (W3C is the World-Wide Web Consortium) and the OASIS Web Services Distributed Management Technical Committee (OASIS is the Organization for the Advancement of Structured Information Standards).

In the DataTAG project [2], we needed to manage gigabit network devices and systems in a distributed manner. We have 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 (Box 6), a freely available research prototype of a Java-based management platform, to Web Services. We could have used integrated and more sophisticated software suites such as IBM's WebSphere. But modifying the source code of JAMAP gave us complete control over distribution aspects 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. Then, we summarize the Web-based Integrated Management Architecture, which underpins JAMAP. After that, 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.

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: W3C and OASIS (Box 1). More recently, a new industrial consortium, the Web Services Interoperability (WS-I) organization, has begun standardizing interoperability aspects of Web services (Box 1). Other organizations not presented here work on Web services, with varying impact.

```
World-Wide Web Consortium
http://www.w3.org/2002/ws/

OASIS
http://www.oasis-open.org/

Web Services Interoperability Organization
http://www.ws-i.org/
```

Box 1: Organizations active in Web services standardization

W3C ACTIVITIES

Four aspects of Web services are currently being standardized by the W3C:

- Simple Object Access Protocol (SOAP)
- Web Services Definition Language (WSDL)
- Web Services Architecture (WS-Arch)
- Web Services Choreography (WS-Chor)

SOAP is a protocol that allows applications to exchange structured information in a distributed environment [3]. 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 is an XML language for describing Web services [4]. Building on the RPC 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, and the names and types of the procedure parameters. 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 (IR); Web applications may discover Web

services in a WSDL repository and invoke them dynamically, just as CORBA applications may discover an object's interface on the fly and invoke it using the Dynamic Invocation Interface (DII).

WS-Arch 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 latest draft [5] but the W3C has paid little attention to them so far. This activity was launched in early 2002 and the documents produced so far are not as mature as the SOAP and WSDL specifications.

WS-Chor [6] deals with the composition and description of the relationships between low-level Web services. This activity is still nascent (it was launched in early 2003) but many actors in industry consider it to be of great importance. The term *choreography* is sometimes replaced by the terms *orchestration*, *collaboration* or *coordination*, but the goal remains the same. In 2002, the W3C specified the Web Services Conversation Language (WSCL) and the Web Service Choreography Interface (WSCI). In industry, the Business Process Execution Language for Web Services (BPEL4WS) has attracted a lot of attention.

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) [7]. 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).

WS-I Activities

The focus of WS-I is on developing profiles, usage scenarios, use cases, sample applications and testing tools to facilitate interoperability between the Web service platforms of its members. This industrial consortium began its activities in February 2002; few specifications have been released to date. The most significant is Basic Profile 1.0, which consists of implementation guidelines as to how core Web services specifications should be used together to develop interoperable Web services infrastructure. Management aspects have not been addressed yet by WS-I, but interoperability issues are critical for management systems.

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 (Box 6). This research prototype of a management platform implements the Web-based Integrated Management Architecture (WIMA) [8]. WIMA leverages XML's self-description capability to integrate SNMP MIB data and 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 device management, systems management, application management, end-to-end network management, service management, 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 the HyperText Transfer Protocol (HTTP).

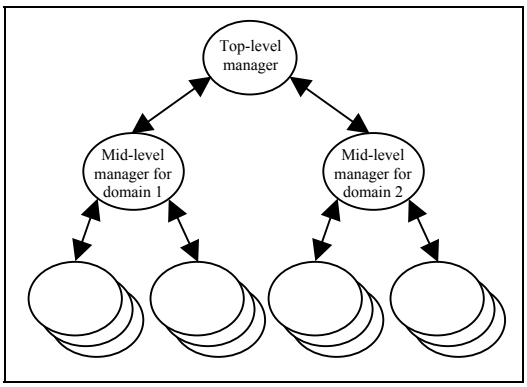

Figure 1: Distribution aspects in WIMA

WIMA supports distributed management—to be precise, a weakly distributed hierarchical paradigm if we use the taxonomy defined in [8]. 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 management vs. service management), profit center, customer (e.g., when ISPs 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 [8]:

- the *data analyzer*, which analyzes monitoring data on the fly, detects problems, and send 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 real machines can be arbitrary.

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.

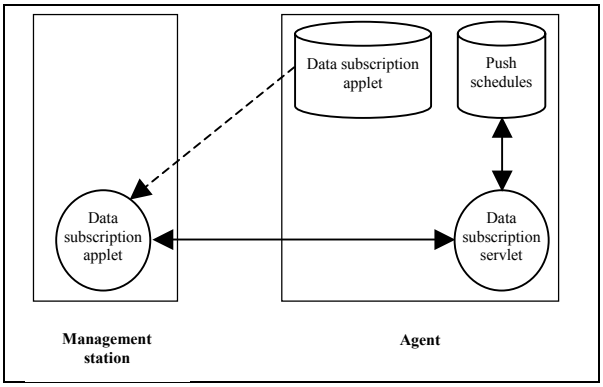

Figure 2: Publish-subscribe

Unlike most SNMP-based management platforms found to date on the market, JAMAP uses publish-subscribe (Figure 2). The SNMP Management Information Bases (MIBs) supported by the agent are published in the data subscription applet, which can be downloaded from 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 to 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., this is the case of Linux PCs and Windows PCs); otherwise, data is pushed via a proxy when we cannot run JAMAP software directly on the agents (e.g., Cisco routers in our testbed).

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.

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 [8]. First, there are good reasons for using XML in software engineering at large. Second, XML is well suited to network and systems management.

ADVANTAGES OF USING XML IN GENERAL

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, which changed several times a year for many years, XML has been through very few release cycles, and DTDs were swiftly replaced by XML Schemas. Markup languages defined in XML come and go and change regularly, but XML itself is very stable.

Second, it is portable and facilitates interoperability. The people who devised XML had learned from the mistakes of CORBA, which took years to standardize such a basic interoperability components as the Portable Object Adapter (POA). XML did have teething problems, but nothing so serious.

Third, XML is simple and easy to learn. Applications that use middleware based on XML and Web services are usually easier to develop and debug than those based on distributed object-oriented middleware. The learning curve of developers is shorter and the training costs are often lower.

Fourth, XML is already widely used in industry. As a result, most software engineering students learn it, most fresh graduates know it, and many mature developers want to study it or already learned it. (The situation is exactly the opposite with SMI, the language used to express SNMP MIBs: people do not want to waste their time learning languages and technologies that are confined to a niche market and can hardly get them a job.)

Fifth, using XML-based technologies is often a way to reduce costs. Many XML tools are available for free, which reduces the cost of entering the XML market. Interfacing legacy systems with XML tools is usually less costly than reengineering them with CORBA, EJBs or .NET. Because so many developers already know XML, sheer competition on the job market brings software development costs down. So does the fact that the same technology is used in a large spectrum of application domains: people can leverage the same libraries and tools in different application domains.

Sixth, XML Schema Definition (XSD) files allow XML parsers to validate XML files. 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 business.

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, as XML documents complying with XML schemas are less readable than XML documents complying with DTDs.

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 eventually 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 makes the "wrong" decision and selects a middleware that is abandoned by his company a couple of years later, the career of this person in this company may be drastically shortened…

The combined use of HTTP, SOAP and XML, and more recently Web services, allows for a "truce in the middleware war" [8]. XML and Web services keep a low profile compared to full-blown object-oriented distributed environments. They can cope with object-oriented models at the edges but do not require them. 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 [8].

Third, as far as the communication model is concerned, XML is useful to represent 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 (NETCONF) 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 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.

DISADVANTAGES OF USING XML

XML is not the panacea, however, and it also presents some disadvantages.

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.

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 and DTDs produced by slow-paced organizations such as the IETF and the DMTF, when they could be produced quicker and made publicly available by vendors? Since it emerged in 1990, the SNMP management platform market has demonstrated that customers are not eager to use SNMP MIBs and information models produced by standards bodies: they want functionality.

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 outweighed by the advantages of using XML [8].

HOW TO USE WEB SERVICES IN A MANAGEMENT PLATFORM

In the Internet world, most management platforms focus on four aspects [8]:

- *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 from time to time, 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, usually 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 leverage and make sense of the data gathered by regular management (data collection). Examples include security analysis and daily usage reports generation.

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

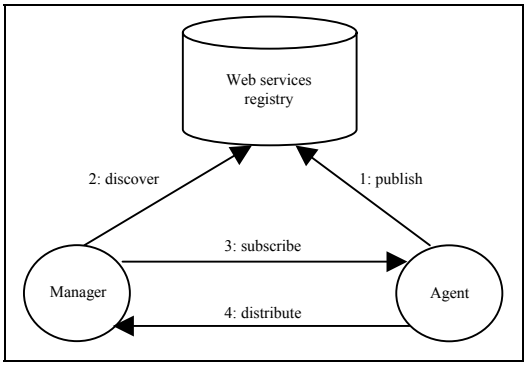

Figure 3: Four phases of regular management

The four phases of monitoring (publication, discovery, subscription and delivery) are explained in detail in [8] and summarized in Figure 3. A priori, Web services could be used at all levels in JAMAP:

- *Publication phase*: To allow agents to publish what management data (e.g., SNMP OIDs or CIM objects) managers can subscribe to (respectively, in hierarchical management, to allow mid-level managers to publish what data can be subscribed to by top-level managers).
- *Discovery phase*: To allow managers to discover dynamically what management data agents can push (respectively, to allow the top-level manager to discover what data mid-level managers can push).
- *Subscription phase*: To allow managers to subscribe to data on 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 management data to managers (respectively, to allow mid-level managers to push data to the top-level manager).

PUBLISH-SUBSCRIBE AND DISCOVERY WITH UDDI

The Universal Description, Discovery and Integration (UDDI) technology is often advertised, or thought of, as a general-purpose technology implementing publish-subscribe for Web Services [9]. As we needed such a mechanism in JAMAP, we studied, deployed, tested and evaluated UDDI.

To assess the relevance of this technology to management applications, 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.

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 [9].

After investigating a number of platforms listed in the Soapware directory [10] or mentioned in the <xml-dist-app@w3.org> mailing list, we selected Systinet's WASP UDDI (Box 6), which offers a good balance between features, conformance to 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 [7]).

The XML schema that underlies UDDI registries is rather simple. It consists of four elements [11]:

- *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. In Box 2, we see that CERN, when viewed as a business entity, offers two types of services to entities outside CERN: fundamental research in physics and applied research in information technology (IT).

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 [9]. 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. An example of this scenario is depicted in Box 3. Within CERN, IT Division 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".

```
<businessEntity>
<name>CERN</name>
<description>
Research Laboratory in Particle Physics
</description>
<contacts>
<contact>
<personName>Director of Public Relations
</personName>
<phone>12345</phone>
</contact>
</contacts>
<businessServices>
<businessService>
<name>Physics</name>
<description>
Fundamental research in particle physics and
high-energy physics.
</description>
</businessService>
<businessService>
<name>IT Research</name>
<description>
Applied research in IT and networking.
</description>
<bindingTemplate>
<!-- See Box 3 -->
</bindingTemplate>
</businessService>
<businessService>
...
</businessServices>
</businessEntity>
```

Box 2: UDDI: business entity and external services

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. In Box 3, we show an example of binding templates in the case of internal services. So, binding templates are the only entities that we can define to make UDDI useful to manage network devices and systems.

```
<businessEntity>
<name>CERN</name>
...
<businessServices>
<businessService>
<name>IT Division</name>
<description>
Provide IT infrastructure to physicists and
perform applied research and engineering in
Grids.
</description>
<bindingTemplate>
<description>
DataTAG Project
</description>
<accesspoint>
mailto:project.office@datatag.org
</accesspoint>
</bindingTemplate>
...
</businessService>
<businessService>
<name>Administration</name>
...
</businessService>
...
</businessServices>
</businessEntity>
```

Box 3: UDDI: business entity and internal services

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 Research". 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 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 (say, 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 possibly modeled using UDDI, even UDDIv3. 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.

XML-BASED CONFIGURATION MANAGEMENT IN JAMAP

As UDDI registries are not appropriate to publish and discover agents in integrated management, we devised a new XML schema customized for network and systems monitoring. We use XML files to publish and discover management information.

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 for each 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*).

```
  <xs:simpleType name="IPv4Address">
   <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"
fixed="true"/>
    <xs:pattern value="((1?[0-9]?[0-9]|2[0-4][0-
9]|25[0-5]).){3}(1?[0-9]?[0-9]|2[0-4][0-
9]|25[0-5])"/>
   </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="IPv6Address">
   <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"
fixed="true"/>
    <xs:pattern value="((1?[0-9]?[0-9]|2[0-4][0-
9]|25[0-5]).){7}(1?[0-9]?[0-9]|2[0-4][0-
9]|25[0-5])"/>
   </xs:restriction>
  </xs:simpleType>

  <xs:element name="agent">
   <xs:complexType>
    <xs:sequence>
     <xs:choice>
      <xs:element name="ipv4Address"
type="IPv4Address"/>
      <xs:element name="ipv6Address"
type="IPv6Address"/>
     </xs:choice>
     <xs:choice>
      <xs:element name="ipv4CollectorAddress"
type="IPv4Address"/>
      <xs:element name="ipv6CollectorAddress"
type="IPv6Address"/>
     </xs:choice>
     <xs:element ref="proxy" minOccurs="0"
maxOccurs="1" />
     <xs:element name="agentManagement"
type="xs:anyURI" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
```

Box 4: XML schema definitions for IP addresses

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 (Box 4). This allows JAMAP to validate effectively the XML documents exchanged between managers and agents. In JAMAP 1.3, publication is still manual.

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 an XML schema

(*agentManagement.xsd*) that is tailor-made for network monitoring.

```
<agentManagement>

<accessPoint>http://137.138.35.18:8080/services
/AgentConfigurationService</accessPoint>
<WsdlFileLocation></WsdlFileLocation>
<dataSubscriptionApplet>http://137.138.35.18:80
80/DataSubscriptionApplet.jsp</dataSubscription
Applet>
<subscriptionSheetServlet>http://137.138.35.18:
8080/servlet/jamap.servlet.SubscriptionSheetSer
vlet?</subscriptionSheetServlet>
<getServlet>http://137.138.35.18:8080/servlet/j
amap.servlet.Get?</getServlet>
<pushDispatcherServlet>http://137.138.35.18:808
0/servlet/jamap.servlet.PushDispatcherServlet?<
/pushDispatcherServlet>
<agentConfigurationServlet>http://137.138.35.18
:8080/servlet/jamap.servlet.AgentConfigurationS
ervlet?</agentConfigurationServlet>

<informationModels>
<snmpInformationModel>
<rfc>/mibs/rfc1213-mib.txt</rfc>
<snmpv1CommunityString>public</snmpv1CommunityS
tring>
<encodingType>plainText</encodingType>
</snmpInformationModel>
</informationModels>

</agentManagement>
```

Box 5: Example of *agentManagement.xml* file

The *agentManagement.xml* file contains agent-specific information that a manager needs to know in JAMAP (Box 5). First, we find information related to the 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.

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 precedent 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 121 (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.

LESSONS LEARNED

A number of lessons of general interest can be learned from this case study, and may hopefully prove useful to other integrated management 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 (Box 6), the Apache open-source package that implements Java servlets, is simple to use and well documented.

Web services: Axis (Box 6), 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. 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. SOAP toolkits usually come as a big 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, OASIS will come up with a solution in the future.

```
JAMAP
http://www.datatag.org/jamap/

Tomcat
http://jakarta.apache.org/tomcat/

Axis
http://ws.apache.org/axis/

WASP UDDI
http://www.systinet.com/products/wasp_uddi/overview
```

Box 6: Software packages mentioned in this article

CONCLUSION

In this paper, we have described the way we implemented Web Services in JAMAP and summarized the lessons learned in this process. Our conclusions are fourfold: (i) Web services are suitable for managing network elements and systems; (ii) XML portability facilitates integration in a heterogeneous environment; (iii) UDDI is a white-page service for e-commerce and is too coarse-grained for managing network elements and systems; and

(iv) 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 work was carried out while P.A. Doffoel was an M.Sc. student at ENSIMAG. Part of this research was funded by the FP5/IST Program of the European Union (DataTAG project, grant IST-2001-32459). The authors thank L. Bovet and C. Ledrich, who developed earlier versions of JAMAP; leveraging their work was essential to the success of this project.

REFERENCES

[1] J. Schoenwaelder, 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.
[2] DataTAG Project, home page available at <http://www.datatag.org/>.
[3] W3C, *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation, 24 June 2003.
[4] W3C, *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Working Draft, 10 Nov 2003.
[5] W3C, *Web Services Architecture Requirements*, W3C Working Draft, 14 Nov 2002.
[6] W3C, *Web Services Choreography Requirements 1.0*, W3C Working Draft, 12 Aug 2003.
[7] OASIS, *UDDI version 3.0 – UDDI Spec Technical Committee Specification*, July 2002.
[8] J.P. Martin-Flatin. *Web-Based Management of IP Networks and Systems*. Wiley, 2002.
[9] The Stencil Group, *The Evolution of UDDI – UDDI.org White Paper*, July 2002.
[10] Soapware.org, Home page available at <http://www.soapware.org/>.
[11] UDDI.org *UDDI Technical White Paper*, Sept 2000.

BIOGRAPHIES

J.P. Martin-Flatin [SM] (jp.martin-flatin@ieee.org) is technical manager of the European DataTAG project (www.datatag.org) at CERN. He coordinates research activities in networking and middleware for data-intensive transoceanic Grids. Prior to that, he was a principal MTS with AT&T Labs Research. He holds a Ph.D. degree in computer science from EPFL, Switzerland. He has worked 10+ years in industry and government agencies in the areas of network and systems management, software development, network security, and Web engineering. His research interests include integrated management, distributed systems, and software engineering. J.P. serves on the editorial board of the *Journal of Network and Systems Management*. He is a co-chair of the GGF Data Transport Research Group and a member of the IRTF Network Management Research Group. He is on the Technical Program Committees of several conferences in integrated management and Web Services.

P.A. Doffoel (pa@doffoel.com) is preparing a Master's degree in e-business management at ESCP-EAP, Paris, France. In September 2003, he received an M.Sc. degree in computer science from ENSIMAG, Grenoble, France. During his M.Sc. thesis at CERN, he improved JAMAP, an open-source research prototype of a management platform, to monitor gigabit networks and systems. Prior to that, he spent six months at Technical University of Berlin, Germany thanks to the Erasmus program (a student exchange program between European Universities). As a member of a joint project with FOKUS, he participated in the development of an e-commerce application based on Web services. His professional interests include e-business, distributed systems, Web services and distributed object-oriented programming.