On Management Technologies and the Potential of Web Services

George Pavlou, Paris Flegkas, Stelios Gouveris, and Antonio Liotta, University of Surrey

ABSTRACT

From the early days of management research and standardization in the late 1980s, there has always been a quest for a management technology that would be general enough to be used for network, system, service, and distributed application management; efficient in terms of information retrieval time, computational resources required, and management traffic incurred; and easy to use in order to reduce development time and operational costs. From protocol-based approaches such as OSI management and SNMP, attention shifted to distributed object and Web-based approaches. Recently, XML-based approaches and, in particular, Web services have been emerging as a promising Internet-based technology that might also be used for management. In this article we survey first the key aspects of protocol and distributed object approaches to management. We subsequently examine Web services as a distributed object approach to management, and consider its suitability, usability, and performance in comparison to SNMP and CORBA.

INTRODUCTION

Management research and standardization started in the mid-1980s, but after almost 20 years there is no widely adopted framework and technology that satisfies the needs of network, system, service, and application management. Initial technologies such as open systems interconnection (OSI) systems management and Simple Network Management Protocol (SNMP) were based on the manager-agent model. In the early to mid-1990s, general distributed object technologies, most notably the Common Object Request Broker Architecture (CORBA), emerged, spurring a lot of research and standardization activity regarding their use for management. These technologies are based on the principles of the open distributed processing model.

All these management technologies found their niche markets: OSI systems management for network management in telecommunication environments; CORBA for service management in telecommunication as well as some IP environments; and SNMP for network and system management in IP environments. SNMP, though, has fundamental limitations [1], which has led to initiatives for either SNMP-like technologies (e.g. COPS-PR) as well as Extensible Markup Language (XML)-based approaches, such as Juniper's JUNO-Script. Technologies based on XML are considered particularly attractive since XML-based management data can easily be integrated with other applications.

Recently, Web services has emerged as a promising XML-based technology for standardizing e-service interfaces. However, careful examination shows Web services to be very similar to distributed object technologies, so potentially it could be used for management. In this article we survey first the key characteristics of OSI systems management, SNMP, and CORBA. We then examine Web services, showing its similarities to CORBA, and subsequently explain how distributed object technologies can be used for management. We then evaluate the performance of SNMP, CORBA, and Web services using a representative example from network management that involves management information about TCP and its connections. We finally present a summary and draw our conclusions.

PROTOCOL-BASED MANAGEMENT FRAMEWORKS

THE MANAGER-AGENT MODEL

The manager-agent model defines the principles of operation for protocol-based management frameworks [2]. Managed resources are modeled through *managed objects* (MOs), which encapsulate the underlying resource and offer an abstract access interface. Any managed network/service element or distributed application should expose a "cluster" of managed objects modeling its resources across a management interface provided by an *agent*. The interface is formally defined through specification of the available managed object types or classes and the management access service/protocol. *Manager* applications access managed objects across interfaces for implementing management policies. A management application may act in both agent and manager roles; this is the case for peer-to-peer

0163-6804/04/$20.00 © 2004 IEEE

management interactions or hierarchical management environments.

An agent is a software entity that administers managed objects, responds to management requests, and disseminates spontaneous events through the management protocol. It uses an implementation-specific mechanism to access the managed objects and, given its collective view of the relevant cluster, can provide sophisticated object access facilities for bulk data transfer or selective information retrieval. In addition, it can evaluate event notifications at the source and only send them to interested managers according to predefined, potentially sophisticated, criteria. The management protocol supports access of multiple attributes from multiple objects through one request. The manager-agent model projects a communication framework in which standardization affects only the way in which management information is modeled and carried across systems, deliberately leaving aspects of their internal structure unspecified. This may result in highly optimized implementations given that there are no internal application programming interfaces (APIs) to be adhered to. The model is depicted in Fig. 1.

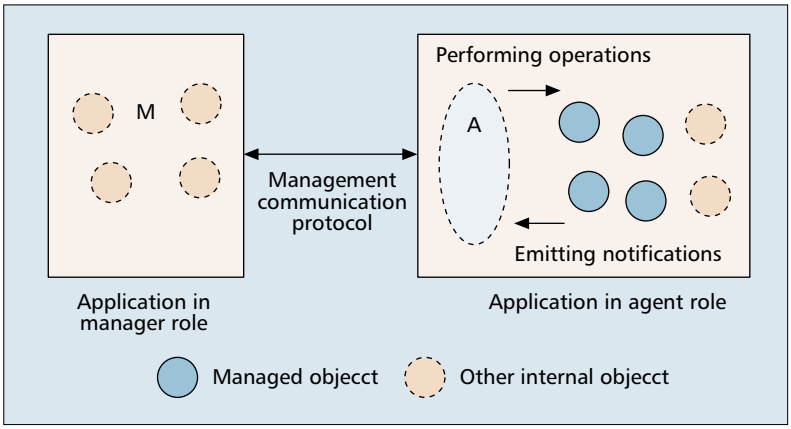


Figure 1. *The manager-agent model.*

OSI Systems Management

OSI systems management (OSI-SM) was the first management technology to be standardized; in fact, the manager-agent model was devised in its context [2]. It is a sophisticated and powerful technology, but also complicated and expensive to deploy, so it has only found use in telecommunication environments. It was the technology behind the telecommunications management network (TMN) Q/X interfaces, but even in those environments it is gradually being phased out in favor of CORBA. Still, it is accepted as the most powerful technology thus far to support features that should be essential in any management framework, so it is worth examining.

OSI-SM uses a fully object-oriented information model supporting inheritance. Managed object classes are specified in the Guidelines for the Definition of Managed Objects (GDMO) abstract language. Key deviations from object-oriented concepts are the runtime specialization of an MO instance through conditional packages and the fact that imperative commands are modeled through a generic method called an *action* with a single parameter and result, which may result in awkward parameter modeling. Telecommunication information models were initially specified in GDMO, although there is a push today toward technology-neutral information specification in the Unified Modeling Language (UML), with reverse engineering of existing models.

The Common Management Information Service/Protocol (CMIS/P) supports Get, Set, Action, Create, Delete, and Event primitives. Information access is powerful, supporting both bulk retrieval through *scoping* and selected retrieval through scoping and *filtering*. Since MOs are organized in a management information tree based on containment relationships, scoping works by selecting a number of subtree objects while filtering further eliminates selection by applying a predicate on attribute values. Connection-oriented reliable transport is used, which means that arbitrarily large amounts of bulk data may be retrieved in one go. Concerted configuration changes through a series of Set operations are supported through the OSI transaction processing facility, which "brackets" these requests and guarantees atomicity (i.e., all to succeed or none performed). Finally, the event framework is sophisticated, allowing managers to create their *event discriminators* with filtering on the event type, time, and the actual notification information.

Throughout the rest of this article, we will be using as our example management information that models a transport protocol, in particular, TCP. The latter is typically modeled as a single-instance class while its connections are modeled through another class, with connection instances "contained" in the protocol instance. Attributes of these classes have read-only properties. Protocol information could be accessed directly through Get given that its name is known (static MO), while connection information can be accessed through Get to the protocol object with "first-level" scoping; filtering could be also used to retrieve connections with particular properties. TCP and connections modeled in GDMO can be found in RFC 1214.

Simple Network Management Protocol

By SNMP [3] we effectively refer to the Internet management framework. The latter was conceived as a simplification of OSI-SM to be used with simpler IP devices. Its simplicity contributed substantially to its success, and it is widely supported by IP network devices. While OSI-SM requires managed devices to be relatively complex due to information model complexity, reliable transport, and event-based operation, SNMP designers have opted for a very simple information model, unreliable transport, and mostly polling-based operation. These decisions leave managed devices simple and have contributed substantially to its success and wide deployment. On the other hand, they shift complexity to network management centers (NMCs) and result in heavy management traffic, limiting the extent to which sophisticated management functionality can be deployed.

The SNMP information model is very simple, with scalar variables (of integer or string type) used to model managed entities. SNMP objects

While there exist distributed object technologies such as Microsoft's DCOM and Sun's J2EE, OMG's CORBA is certainly the representative one, being collectively specified so that it constitutes a real-world approach toward an ODP-inspired solution.

are formally specified in a language known as Structure of Management Information (SMI). Although we talk of objects in SNMP, they are effectively equivalent to object attributes in other frameworks. There is no inheritance and the only operations allowed on them are read and write. These objects can be either single- or multiple-instanced, the latter for tables consisting of a series of rows. Table rows are the only composite objects that can be created and deleted. This very simple variable-based rather than object-oriented approach may result in significant awkwardness and complexity when trying to model complex real entities. There is no information reuse through inheritance, while complex data types and imperative methods can only be modeled indirectly.

SNMP provides Get, Set, and Event operations. Single-instanced objects are accessed through Get; table objects are accessed through two variations of Get, GetNext, and GetBulk. SNMP objects in an agent are ordered in a linear fashion given the naming architecture, with table objects having as next the objects of the next row. GetNext and GetBulk exploit this structure, with GetNext requesting the first next of a series of objects (typically a table row) and GetBulk requesting N next objects. With GetNext, a table of R rows can be accessed through $R + 1$ serial requests. These can be reduced with GetBulk, but it is difficult to estimate N properly so as not to "overshoot" past the end of the table and at the same time minimize the required transfers. Given that SNMP uses by default UDP transport (TCP mapping as in RFC 3430 is not widely used), a response to Get, GetNext, or GetBulk should typically fit in a single packet to avoid IP-level fragmentation that increases the probability of SNMP packet loss; this reduces effectiveness for bulk transfers. There is also no selective retrieval similar to CMIS filtering. The event framework allows managers to declare interest in particular event types but without filtering. In addition, for historical reasons management stations do not rely on events, which in SNMPv1 were sent through unreliable *traps*, despite the fact that subsequent versions have included reliable *inform-requests*; as a result, management stations deploy a mostly polling-based regime.

But a fundamental limitation of SNMP is that it does not support coordinated Sets to move from configuration A to B in a consistent manner through a transaction processing facility. In fact, even simple Set commands can be problematic with SNMP; as a result, SNMP has been used mostly for monitoring rather than intrusive management. COPS Provisioning (COPS-PR) was conceived as an approach similar to SNMP that could support concerted configuration changes, but it did not maintain compatibility; nor did it fix other important shortcomings (e.g., the rudimentary information model), so it did not succeed.

A transport protocol is modeled in SNMP as a group of scalar value objects, with each such "object" having an individual name, while connections are modeled through a table row of such objects. Protocol information can be accessed through Get, while connection information can be accessed through GetNext or GetBulk. The TCP protocol and connection modeling can be found in RFC 2012.

Distributed Object Technologies

Open Distributed Processing

International Organization for Standardization/International Telecommunication Union — Telecommunication Standardization Sector (ISO/ITU-T) open distributed processing (ODP) [4] is a general framework for specifying and building distributed systems, while the Object Management Group's (OMG's) CORBA can be seen as its pragmatic counterpart. ODP came about in response to the recognition that although ITU-T and Internet Engineering Task Force (IETF) protocol-based solutions addressed the problem of heterogeneous system interconnection, the proliferation of application-layer standards and distributed applications meant that application intercommunication needed to be addressed as well. This was further fueled by the convergence of the information technology and communication sectors, and the resulting demand for standardized APIs between distributed application components and underlying platforms. Hence, the target for ODP is to facilitate distribution and interoperability, but also to achieve software portability.

ODP projects a client-server model, with distributed applications composed of objects interacting solely by accessing each other's *interfaces*. The underlying ODP platform (or ORB in CORBA) provides a number of transparencies, such as access, location, replication, failure, and resource. Clients access server objects through *interface references*, obtained through access to well-known special servers such as name servers and traders. A name server keeps a name space with interface references "advertised" by server objects, while clients could resolve names to interface references. Traders, on the other hand, keep interface references together with object "properties" (i.e., attributes with static values), and may also search the administered object space to evaluate assertions on dynamic attribute values. Clients may search for an interface reference through a predicate on properties and attribute values of the sought object. Finally, there is an underlying protocol for interoperability hidden inside the platform, with objects "sitting" on the platform through a standardized API. The ODP model is depicted on Fig. 2.

The Common Object Request Broker Architecture

While there are distributed object technologies such as Microsoft's DCOM and Sun's J2EE, OMG's CORBA [5] is certainly the representative one, being collectively specified so that it constitutes a real-world approach to an ODP-inspired solution. While manager-agent approaches such as SNMP and OSI-SM are *communications* frameworks, CORBA targets a *programmatic* interface between objects and the underlying ORB. Choices made by ITU-T/IETF on one side and OMG on the other reflect their different preoccupations: management commu-

nications for the former and distributed software systems for the latter.

CORBA uses a fully object-oriented information model supporting inheritance. Objects are effectively defined through their interfaces, which are specified in the Interface Definition Language (IDL). Methods with any argument and result parameters are possible, providing full flexibility as befits a distributed systems technology. Attributes with read or read-write properties may be also defined, which effectively results in *<attr>_get* and *<attr>_set* methods being generated. This in fact implies that, in the default case, attributes are accessed individually through a remote method call, which can be very expensive.

CORBA specifies a general remote call protocol, the General Inter-Operability Protocol (GIOP), which simply defines request, response and error packets carrying the parameters and results of operations. Its most common mapping is over TCP/IP, known as the Internet IOP (IIOP), providing reliable transport but with connection management hidden by the ORB. Objects are typically accessed through their interface methods, one in each remote call, with no support for bulk transfer or selected retrieval. There is, however, support for atomic transactions through the transaction service. Clients access server objects through *stub* images in their local address space, which hides the complexities of distributed operation and allows non-network programmers to develop distributed applications. Access can be static, through precompiled IDL stubs, or dynamic, through the Dynamic Invocation Interface (DII). Interoperable references (IORs) of interfaces may be obtained through the naming or trading service. Finally, events are disseminated through the event server that allows clients to specify the type of events they want to receive.

A transport protocol is typically modeled as a single-instance interface with connections modeled as a multiple-instance interface in CORBA IDL. Both interfaces contain attributes with read-only properties, which result in individual get methods for accessing them. The transport protocol advertises its name to the name server through which clients may obtain its interface reference. The transport protocol interface includes also a method that returns the interface references of current connections, which is used by clients to obtain these interfaces and then access connection information — this is common modeling practice in CORBA for dynamic multiple-instance objects/interfaces. The specification of the TCP IDL interface with additional methods to support bulk data retrieval, as discussed later, is shown on Fig. 3.

Web Services

Web services [6] is an emerging Internet-oriented technology that has strong analogies to CORBA. It is developed and standardized by the World Wide Web Consortium (W3C) so that Web-based e-services expose standard interfaces and are accessed in an open interoperable manner. Web services aims to put structure in Web content and associated services so that the latter are accessible by distributed applications.

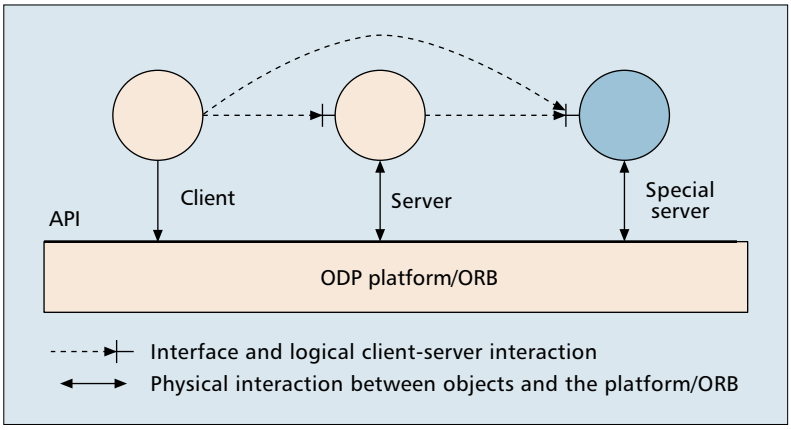


Figure 2. *The ODP/ORB model.*

Service interfaces are specified in the Web Services Description Language (WSDL), which constitutes a general XML-based framework for the description of services as communication endpoints capable of exchanging messages. It describes the service location through a Uniform Resource Identifier (URI), supported operations, and messages to be exchanged. Service inheritance is also supported. WSDL does not mandate a specific communication protocol but can support different protocol bindings; despite this, the default binding is usually Simple Object Access Protocol (SOAP, see below). WSDL can be considered as broadly equivalent to CORBA IDL. In this context, URIs are broadly equivalent to CORBA IORs. SOAP is a stateless protocol with XML-based encoding. It can support request/response/error remote call interactions and is broadly equivalent to CORBA GIOP when used that way. The default SOAP mapping on HTTP/TCP/IP can be considered equivalent to CORBA IIOP.

Service specification and interface discovery are supported through Universal Description, Discovery, and Integration (UDDI). This provides a mechanism for service providers to advertise (i.e., register) services with it in a standard form so that service consumers query services of interest and discover their location. UDDI is itself implemented as a well-known Web service in terms of interface and location. When used for service specification discovery, it is broadly equivalent to the CORBA Interface Repository; when used for interface location discovery, it is broadly equivalent to the CORBA Naming and Trading services.

It should be obvious from the above comparison that Web services can be used as a distributed object technology. Some key differences from CORBA are the following. In CORBA, the default client-server coupling is tight, with the client having precompiled knowledge of the server's interface, which supports compile time type-checking. Web service technology was initially conceived as message-oriented, with loose coupling between clients and servers through XML parsing and runtime checking only, similar to the CORBA DII. On the other hand, most Web services platforms support static coupling through stubs, albeit through proprietary mappings. CORBA supports standard language mappings

The popularity of CORBA led the TMF and Open Group to setup the JIDM taskforce that produced a static mapping, i.e., specification translation between SNMP SMI/OSI-SM GDMO and CORBA IDL, and a dynamic mapping, i.e., interaction translation for supporting generic gateways.

```
typedef struct TcpConnCounters_t {
    long activeOpens;
    long passiveOpens;
    long attemptFails;
    long estabResets;
    long currEstab;
    Time_t currTime;
} TcpConnCounters;

//…

interface i_tcp {
    TcpStaticAttrs    getStaticAttrs();    //tcpRtoAlgorithm, tcpRtoMin, tcpRtoMax
    TcpConnCounters   getConnCounters();   //tcpActiveOpens, tcpPassiveOpens, …
    TcpPacketCounters getPacketCounters(); //tcpInSegs, tcpOutSegs, tcpRetransSegs

    long              getCurrEstab();
    // …

    IntRefList        getConnRefs();
    TcpConnList       getCurrConnInfo();
};
```

Figure 3. *The TCP managed object interface in IDL.*

for languages (C++, C, Java, etc.), while Web services, being an Internet technology, addresses only the interoperability through WSDL and the protocol (e.g., SOAP/HTTP). Finally, CORBA offers sophisticated services such as naming/trading, event/notification, and transaction, security.

Given the analogy of Web services and CORBA, a transport protocol and its connections can be modeled in a similar fashion. The transport protocol is modeled as a service interface, with request/response messages for reading each of its attributes. Its interface endpoint, the URI, is advertised to the UDDI from which it can be retrieved by clients. Transport connections are modeled through another service interface, with similar request/response messages for reading relevant information. The interface endpoints can be obtained through request/response messages associated with the transport protocol. The specification of the TCP WSDL interface with additional methods to support bulk data retrieval, as discussed later, is shown on Fig. 4.

Distributed Object Technologies Used for Management

The use of distributed object technologies for network, service, and application management was a subject of intense research in the mid to late 1990s. It is now widely accepted that distributed objects are naturally suited to service and application management: service management involves mostly business process reengineering and automation, for which technologies like CORBA or J2EE are well suited; in addition, distributed applications are typically realized using distributed object technologies, so it makes sense to use the same technology to manage them. On the other hand, network and system management have relatively different requirements: large amounts of information need to be accessed, some of it real-time in nature, while concerted configuration changes must be supported across devices. So fundamental requirements of network management are support for flexible information retrieval, both in bulk but also selectively; support for fine-grained event notifications through selective criteria; and support for transactions that involve many operations to one or more devices.

The popularity of CORBA led the Tele-Management Forum (TMF) and OMG to set up the Joint Inter Domain Management (JIDM) task force that produced a static mapping (i.e., specification translation) between SNMP SMI/OSI-SM GDMO and CORBA IDL, and a dynamic mapping (i.e., interaction translation) to support generic gateways. The modeling of the transport protocol and its connections presented above in CORBA IDL and WSDL follows the spirit of the JIDM suggestions [7].

Given these developments, CORBA has started to be used for network management in telecommunication environments, with ITU-T GDMO specifications translated to IDL. But the use of CORBA, and distributed object technologies in general, presents the following problems. First, there is no support for bulk or selective data transfer; in fact, a remote method invocation per attribute is required in the default case, which can be prohibitively expensive. Second, a scalability problem may arise in making vast amounts of dynamic entities such as connections separate objects with their interfaces.

Because of these limitations, common practice for the use of CORBA for network management is to follow a semantic rather than JIDM-like syntactic translation of GDMO to IDL. Commonly accessed attributes of an object class are grouped together through an additional method that returns them, which alleviates the requirement of using separate per-attribute methods. In addition, dynamic entities such as connections are *not* modeled through separate objects/interfaces. Instead, a method that returns all of them as a "list of records" is added to the associated protocol interface. If such entities are static (e.g., semi-permanent crossconnections created and deleted through management), additional methods to create and delete them, or better to add and remove them from the current

```
<wsdl:types>
  <xsd:complexType name="TcpConnCounters">
    <xsd:sequence>
      <xsd:element name="activeOpens" maxOccurs="1" minOccurs="1" type="xsd:int"/>
      …       <!– passiveOpens, attemptFails, estabResets –>
      <xsd:element name="currEstab" maxOccurs="1" minOccurs="1" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
  …       <!– TcpStaticAttrs, TcpPacketCounters, TcpConnAttrs, … –>
</wsdl:types>
<wsdl:message name='i_tcp_getConnCounters_Request'/>
<wsdl:message name='i_tcp_getConnCounters_Response'>
  <wsdl:part name='response' element='ns0:TcpConnCounters'/>
</wsdl:message>
…       <!– i_tcp_getStaticAttrs_Request, i_tcp_getStaticAttrs_Response, … –>
<wsdl:operation name='getConnCounters'>
  <wsdl:input message='tns:i_tcp_getConnCounters_Request'/>
  <wsdl:output message='tns:i_tcp_getConnCounters_Response'/>
</wsdl:operation>
…       <!– getStaticAttrs, getPacketCounters, … –>
<wsdl:binding name='i_tcp' type='tns:i_tcp'>
  <wsdl:operation name='getConnCounters'>
    …       <!– SOAP bindings –>
  </wsdl:operation>
  …       <!– getStaticAttrs, getPacketCounters, … –>
</wsdl:binding>
<wsdl:service name='i_tcp'>
  <wsdl:port name='i_tcp' binding='tns:i_tcp'>
    <soap:address location='urn:service-uri'/>
  </wsdl:port>
</wsdl:service>
```

Figure 4. *The TCP managed object service port in WSDL.*

The use of Web Services for network management presents exactly the same problems and issues as with the use of CORBA, with the additional problem that there is not yet an event notification framework.

list, are added. This modeling approach supports bulk data transfer for multiple-instanced objects in a predefined manner, relying on the fact that connection-oriented reliable transfer is used by relevant technologies. In addition, it does not require an excessive amount of objects/interfaces to be present in network devices.

The use of Web services for network management presents exactly the same problems and issues as the use of CORBA, with the additional problem that there is not yet an event notification framework. The latter may change in the future, but even if it does not, it is fairly easy to specify and support event discriminator-like service interfaces per network device, allowing managers to select the types of events they want to receive, in a similar fashion to SNMP and OSI-SM (without full filtering). The problems of bulk data transfer and scalability can be potentially addressed as described above, but it remains to consider aspects pertinent to Web services (XML encoding overhead, etc.).

For our evaluation we will also use the TCP protocol and connections, which we have used as an example throughout this article. Based on the principles of semantic SNMP SMI to CORBA IDL and Web services WSDL mapping, the TCP protocol interface will have the following additional methods to per-attribute get methods:

- A method to retrieve all the static attributes (*tcpRtoAlgorithm*, *tcpRtoMin*, *tcpRtoMax* and *tcpMaxConn*) that are typically retrieved once
- A method to retrieve all the dynamic attributes related to connection activity (*tcpActiveOpens*, *tcpPassiveOpens*, *tcpAttemptFails*, *tcpEstabResets* and *tcpCurrEstab*)
- A method to retrieve packet counters which are typically retrieved periodically (*tcpInSegs*, *tcpOutSegs* and *tcpRetransSegs*)
- A method to retrieve the information for all the TCP connections as a list of records

The specification of those TCP methods in IDL and WSDL was shown in Figs. 3 and 4, respectively.

Evaluation

We have already implicitly evaluated the *suitability* of Web services for management, having compared them to distributed object technologies and presented the current trends in using the latter in ways that help circumvent their inherent drawbacks. We are now going to evaluate Web services in terms of usability and performance in comparison to SNMP and CORBA. We have had extensive experience with both SNMP and CORBA in the past, and recently started experimenting with Web Services, seeing them as an alternative to CORBA. A key advantage of Web services is the use of XML, which can also be used to structure and pass data between applications. Consequently, an XML-based management approach may result in economies of scale.

We experimented with three different Web services implementations, two general-purpose ones (Systinet's WASP, Apache Axis) and an optimized one that may also be used for small devices (gSOAP). We implemented small distributed applications. Given that Web services are effectively a standardized communication framework, all these platforms use different

The excess time for getting a number of attributes is a linear function of the time required for encoding/decoding and it appears that the Web Services XML encoding/decoding is roughly four times that of SNMP and CORBA for scalar types.

APIs. Apache is not user-friendly, supporting only a message-based SOAP API. On the other hand, both WASP and gSOAP are user-friendly, supporting a CORBA-like stub-based framework that hides SOAP completely and is quite easy to use. With the stub-based approach, *usability* is very good, similar to CORBA and certainly better than SNMP. There is no code portability, but the software abstractions in WASP and gSOAP are not dissimilar. On the other hand, since there are no dictated APIs, compact implementations are potentially possible.

In order to assess performance, a potential worry in Web services given the XML-based encodings and associated parsing, we implemented the TCP protocol managed object in CORBA and also used an existing SNMP agent implementation to do performance comparisons. We only wanted to assess the management infrastructure overhead and to be able to repeat experiments, so the TCP managed object had hard-wired nonchanging values for its attributes, including the values of TCP connection attributes; the total number of TCP connections was set to 40. We had to modify an existing SNMP agent in order to do this, while we implemented from scratch the CORBA and Web services versions.

The SNMP TCP managed object specification was as in the TCP managed information base (MIB) of RFC 2012. The CORBA and Web services specifications were as shown on Figs. 3 and 4, with an additional method returning all the dynamic attributes, connection- and packet-related (eight in total). For the TCP connections, we used two modeling approaches: a TCP method returning all the connections as in Figs. 3 and 4; and separate IDL and WSDL objects for each TCP connection, with a TCP object method returning the CORBA IORs and Web services URIs. The latter is a 1-to-1 mapping from SNMP SMI to IDL/WSDL; we did that in order to also evaluate the overheads of the approach.

The precise experimental environment was as follows. We used the Orbacus CORBA platform [8], WASP Web services platform [9], and NET-SNMP [10] and AdventNet SNMP platforms. We chose WASP and not gSOAP because we wanted to compare it against Orbacus, which is a general-purpose rather than small-device-targeted CORBA implementation. We also implemented everything in both C/C++ and Java in order to assess the performance difference the programming language makes, apart from the SNMP agent that was implemented only in C. Both Orbacus and Systinet support both C++ and Java. For the SNMP agent part we modified the NET-SNMP v. 5.0.1 agent implemented in C; for the manager part we used the NET-SNMP API for C and the AdventNet API for Java. Security features for SNMP and CORBA remained switched off. We used GNU C/C++ 2.95 and the Java 2 Special Edition JDK 1.3.1 versions on Linux RedHat 7.3. The managing-managed systems were placed in two different Pentium III Celeron 1 GHz Linux PCs with 256 Mbbytes of RAM, connected in "back-to-back" mode through a dedicated 100 Mb/s Ethernet.

We performed the following experiments. We measured the response times and traffic for a method returning a single attribute and one returning all eight dynamic attributes of the TCP protocol object, which are typical operations on single-instance objects in management environments; we mark them *1attr/method* and *Nattrs/method* in the figures. We also measured the response times and traffic for retrieving all the TCP connections (40 in total), a typical operation on multiple-instance objects in management environments. This was done in two ways: both through a single IDL/WSDL method of the TCP protocol object returning all the connection information and GetBulk in SNMP; and through separate TCP connection interfaces/endpoints in IDL/WSDL, with a method returning the attributes of a connection, and GetNext in SNMP. We mark these two approaches as *NMOs/method* and *1MO/method* in the figures. Note that the naming server/UDDI must be used to obtain the IOR/URI of the TCP protocol object in CORBA/Web services, but we did not consider this time in the measurements since this is typically done only once. The response time measurements were done for both C++ and Java.

The response time measurements are shown in Fig. 5, the bar chart for C++ and the table for both C++ and Java. In all the experiments, the Java response times were roughly *twice* those of C++. This makes the case for the use of C/C++ at least for the managed system part, which is common practice in management systems today. The managing system could be implemented in Java if its operation is not time-critical, but C++ results in half the Java execution time and probably even less for computationally intensive management logic. In the discussion below, we consider the C++ case when we refer to particular figures.

The response time for getting $N = 8$ attributes through one IIOP/SOAP method or one SNMP Get is only 12 percent bigger than the time to get one attribute for SNMP/IIOP but 48 percent bigger for SOAP. The excess time for getting a number of attributes is a linear function of the time required for encoding/decoding, and it appears that the Web services XML encoding/decoding is roughly four times that of SNMP and CORBA for scalar types. This was expected and is not prohibitively expensive for situations like this where eight attributes are retrieved in 3.7 ms as opposed to 2.5 ms for one attribute.

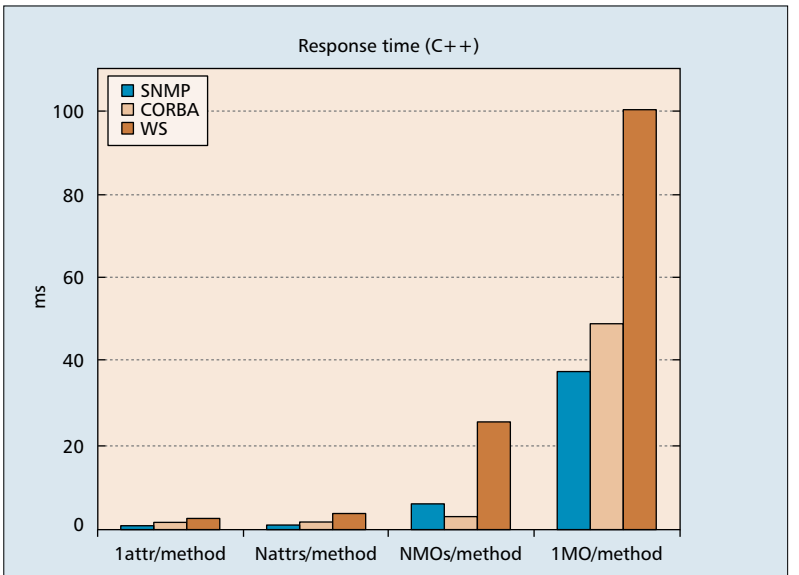
We look now at the time to retrieve the TCP connections, first through one method in IIOP/SOAP and SNMP GetBulk, and then through multiple methods on separate objects in IIOP/SOAP and through SNMP GetNext; these are shown on the right part of the charts in Fig. 5. For each row we retrieved all the objects (five in total), although given the naming principle for TCP connection table objects retrieving only the *tcpConnState* object would suffice: the naming suffix is formed from the values of the source and destination addresses and port values, so these can be obtained implicitly. Although this is the case for the TCP connection table, it is not the case for other tables, and we wanted our measurements to be representative of general table retrieval with a number of objects. In addition, we tailored carefully the SNMP GetBulk

maximum repetitions so that the response did fit in an IP packet encapsulated in a single Ethernet frame (in order to avoid fragmentation). As a result, four successive GetBulk operations were required to retrieve a table with 40 connections, retrieving again all five objects of each row.

The first thing to remark is that IIOP appears very efficient when a single method is used, resulting in relatively small additional time for 40 connections with five attributes each, in comparison to a single attribute (2.5 compared to 1.5 ms)! We were surprised by that result and checked carefully its statistical validity, but it seems to be true, testifying to the efficiency of CORBA IIOP encodings. The SOAP response time is an order of magnitude higher in this case (25 ms), which implies that XML encodings can be quite expensive for complex types. Since SNMP needs four successive GetBulk operations to retrieve the 40-row table, its overall response time is more than double that of IIOP (6 ms).

When multiple IIOP/SOAP methods and GetNext requests are used, the overall retrieval time increases substantially. SNMP is quite efficient in this case, retrieving the connection table through 41 GetNext operations in 37 ms; the 41st operation does not yield useful results but is necessary to "discover" the end of the table. IIOP/SOAP also need 41 operations, the first to retrieve the IORs/URIs of the connection objects. The IIOP time is 4 ms for the IORs and 45 ms for the connections, 49 ms in total, while the SOAP time is 6 ms for the URIs and 94 for the connections, 100 ms in total: *twice* the IIOP time and *two-and-a-half* times the SNMP time. Note that this mode of operation with one interface/service port per connection object is not really appropriate in management environments; we simply used it to quantify its overheads, which are quite high in terms of retrieval time.

Figure 6 shows the management traffic overhead. In terms of traffic, SNMP appears efficient for retrieving a single attribute, but in more plausible scenarios when multiple attributes or a whole table are retrieved, IIOP performs much better. For retrieving the 40 TCP connections with one method, IIOP needs 2252 bytes, while SNMP with four GetBulk requests/responses results in 8160 bytes in total. It should be noted, though, that if only the *tcpConnState* object had been retrieved as explained, this value would be much smaller; it is actually 1306 bytes since a single GetBulk operation is enough. Web services are far more expensive because of the XML encodings. Retrieving the dynamic TCP attributes through a single method incurs roughly five times as much traffic as SNMP and 6.5 times as much as IIOP. Retrieving the 40 TCP connections through one method incurs roughly 2.2 times as much traffic as SNMP GetBulk and eight times as much as IIOP. Finally, retrieving the TCP connections through separate IDL/WSDL objects results in large amounts of traffic for IIOP and prohibitively large amounts of traffic for SOAP. IIOP needs 9936 bytes for the 40 requests, but another 14,221 bytes for retrieving the IORs of the connection objects (i.e., approx. 24 kbytes in total). SOAP needs 76,765 bytes for the 40 requests and another 4585 bytes for retrieving the URIs, approximately 81 kbytes in total! As was also the case with retrieval times, traffic overhead is too high for modeling connections as separate IDL/WSDL objects.



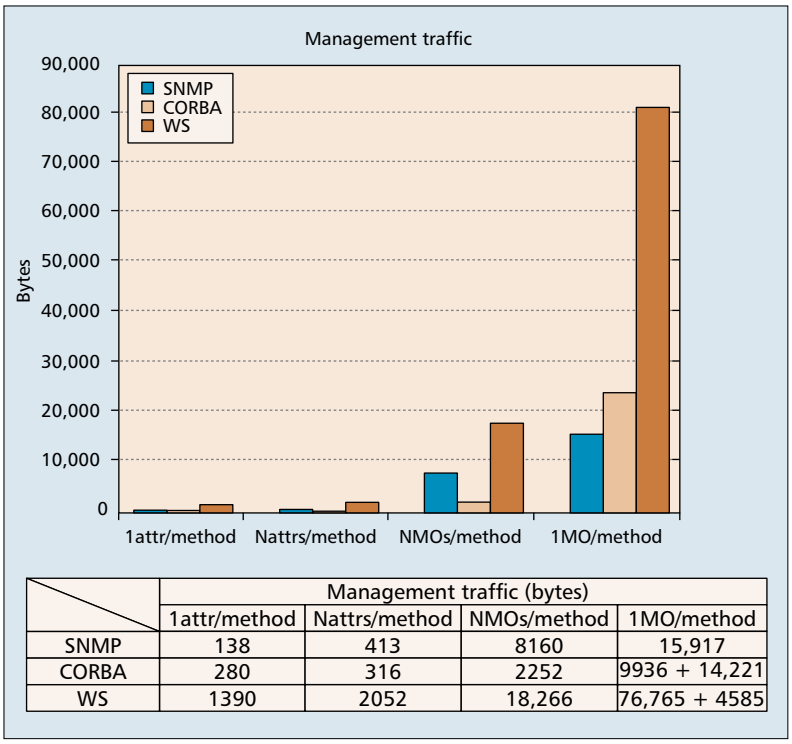
		Response time (ms)			
		1attr/method	Nattrs/method	NMOs/method	1MO/method
SNMP	C++	0.8	1	6	37
	Java	1	2	8	45
CORBA	C++	1.5	1.7	2.5	45 + 4
	Java	2	4	6	90 + 7
WS	C++	2.5	3.7	25	94 + 6
	Java	5	9	40	240 + 10

Figure 5. *Response time measurements.*

Summary and Conclusions

A number of management technologies have emerged over the last 15 years, but none of them satisfies all the requirements of management environments. At one end of the spectrum, OSI SM has been the most powerful technology, but is complicated and expensive, and relies on OSI protocols that have gone out of fashion. It was used in telecommunication environments but is gradually being phased out in favor of CORBA. On the other end of the spectrum, SNMP has been a very simple framework that became widely deployed in IP environments but fell victim to its own simplicity: its information modeling capabilities are rudimentary, it does not support bulk data retrieval and event-based management well, and, most important, it does not support configuration management well due to its lack of transaction capabilities.

Distributed object technologies, and CORBA in particular, have a number of advantages but were designed with distributed systems in mind, lacking support for bulk data retrieval; they also suffer from potential scalability problems for large managed object populations. They can be used for management, but this requires special modeling to support predefined bulk transfer through special methods and avoid modeling large populations of dynamic objects through separate objects/interfaces. These problems of



	Management traffic (bytes)			
	1attr/method	Nattrs/method	NMOs/method	1MO/method
SNMP	138	413	8160	15,917
CORBA	280	316	2252	9936 + 14,221
WS	1390	2052	18,266	76,765 + 4585

Figure 6. *Management traffic measurements.*

distributed object technologies and the ways around them through special modeling became evident from the performance evaluation in the previous section.

Web services can be seen as a distributed object technology; in fact, platform providers have been taking a CORBA-like approach with stub objects, which reinforces this view. Its use for network and systems management presents the same problems as CORBA, so exactly the same solutions can be adopted. Its usability is similar to CORBA due to the stub-based APIs and arguably better than SNMP. On the other hand, there is no security and notification support at present, which means this technology is not yet ready to be used for network management. The initial performance evaluation is encouraging but also highlights some expected problems. Information retrieval times are approximately twice those of CORBA, but the key problem is the amount of management traffic incurred due to the XML-based encodings, which can be up to eight times that of CORBA. This can be reduced through compression at the expense of slower retrieval times. The footprint of managed systems is also relatively large, but smaller than CORBA.

In summary, Web services is a promising technology but, being XML-based, has more overheads than SNMP and CORBA. On the other hand, being XML-based is also its biggest attraction, due to potential easy integration with other applications. Approaches such as OASIS Web Services Distributed Management (WSDM) and the Web Services Management Framework (WSMF) are already looking at the use of Web services as a management framework. In the next few years, research and commercial developments will determine if Web services are widely adopted and used in network and service management environments.

Acknowledgments

This work was partially supported by the EPSRC grant GR/S79992 Policy Analysis for Quality of Service Management (PAQMAN).

References

[1] J. Schoenwaelder, A. Pras, and J.-P. Martin-Flatin, "On the Future of Internet Management Technologies," *IEEE Commun. Mag.*, vol. 41, no. 10, Oct. 2003.
[2] ITU-T Rec. X.701, "Information Technology — Open Systems Interconnection, Systems Management Overview," 1992.
[3] J. Case *et al.*, "A Simple Network Management Protocol (SNMP)," IETF RFC 1157, May 1990.
[4] ITU-T Rec. X.900, "Information Technology — Open Distributed Processing, Basic Reference Model of Open Distributed Processing," 1995.
[5] Object Management Group, "The Common Object Request Broker: Architecture and Specification (CORBA)," v. 2.0, 1995.
[6] W3C, Web Services Activity Docs., http://www.w3c.org/2002/ws
[7] T. Rutt, Ed., "Comparison of the OSI Systems Management, OMG and Internet Management Object Models," Rep. of the NMF–X/Open JIDM task force, 1994.
[8] Orbacus, CORBA platform for C++ and Java, http://www.orbacus.com
[9] Systinet WASP, http://www.systinet.com/
[10] NET-SNMP libraries and tools, http://net-snmp.sourceforge.net

Biographies

George Pavlou (G.Pavlou@surrey.ac.uk) is a professor of communication and information systems at the Center for Communication Systems Research, Department of Electronic Engineering, University of Surrey, United Kingdom, where he leads the activities of the Networks Research Group. He hold a Diploma in engineering from the National Technical University of Athens, Greece, and M.Sc. and Ph.D. degrees in computer science from University College London, United Kingdom. His research interests focus on network management, networking, and service engineering, covering aspects such protocol performance evaluation, traffic engineering, quality of service management, policy-based systems, multimedia service control, programmable networks, and communications middleware.

Paris Flegkas (P.Flegkas@surrey.ac.uk) received a Diploma in electrical and computer engineering from Aristotle University, Thessaloniki, Greece, and an M.Sc. in telematics (communications and software) from the University of Surrey. He is currently working toward his Ph.D., while at the same time being a research fellow in the Networks Research Group at the Center for Communication Systems Research (CCSR), University of Surrey, working on EU and U.K. funded research projects. His research interests are in the areas of policy-based management, network and service management technologies, and IP QoS provisioning.

Stelios Gouveris (S.Gouveris@surrey.ac.uk) received a B.Sc. with honors in electrical and electronic engineering from the University of Glamorgan and an M.Sc. in telecommunications and software (telematics) from the University of Surrey. He is currently a research fellow at CCSR, where over the last four years he has been involved in a number of European and U.K. funded research projects. His research interests include network management, distributed object platforms, network programmability, and quality of service.

Antonio Liotta (a.liotta@surrey.ac.uk) obtained his first degree in computer and electronic engineering from the University of Pavia, Italy, in 1994, an M.Sc. in information technology from the Polytechnic of Milan, Italy, in 1995, and a Ph.D. in computer science from University College London in 2001. Since 2000 he has worked at the University of Surrey as a researcher and academic, and has been involved in several projects in the area of network and service management. Recent research interests have included middleware for mobile distributed systems, multimedia service control, adaptive services and applications in the context of 3G networks, Grid computing, and code mobility.