

On the standardisation of Web service management operations

Jeroen van Sloten, Aiko Pras, Marten van Sinderen

Abstract— Given the current interest in TCP/IP network management research towards Web services, it is important to recognise how standardisation can be achieved. This paper mainly focuses on the standardisation of operations and not management information. We state that standardisation should be done by standardising the abstract parts of a WSDL document, i.e. the interfaces and the messages. Operations can vary in granularity and parameter transparency, creating four extreme operation signatures, all of which have advantages and disadvantages.

Keywords— Network management, Web services, operation granularity, WSDL, SOAP.

I. INTRODUCTION

OVER the years the Simple Network Management Protocol (SNMP), in its different versions [1], has grown to be the most commonly used network management platform in IP networks. However, while originally developed in an environment where networks were small, bandwidth was scarce and processing power on networked devices was low, the design choices made then are nowadays becoming apparent limitations in the network management area.

A. SNMP limitations

One of the design choices for the first version of SNMP was simplicity. Therefore, the data model and the protocol were kept as simple as possible in order to achieve a high acceptance from software developers. Matters such as security and transactions had been postponed for later research, to keep implementation of managers and agents simple and inexpensive. This has resulted in the wide acceptance of SNMP and the availability of many proprietary and open source implementations of both managers and agents. Equipping networked devices had become very simple as a result. However, even though this has contributed for a large part to the success of SNMP, nowadays there are more demands towards network management which makes discussion on improvements and alternatives valid.

The growth of networks and the development of more and more networked devices have led to a situation where scalability problems of SNMP become a serious issue. Networks have grown tremendously in size since the late eighties and early nineties. More systems and devices are connected to form larger networks and moreover, agents on these devices commonly have more capacity to support more management information.

Centre for Telematics and Information Technology, University of Twente. PO Box 217, 7500 AE Enschede, The Netherlands. {sloten, pras, sinderen}@cs.utwente.nl.

A considerable amount of management information, such as IP, TCP, interfaces and accounting information, is stored in tables that continue to grow for each device that is added to a network. Tables are a cause of inefficiency in SNMP-based network management and therefore growing tables are likely to cause even more inefficiency. The result of this is that, while networks continue to grow, the total amount of management information also grows considerably and thus makes SNMP less feasible to use.

Even though developing SNMP applications is made relatively easy due to the simplicity and availability of open implementations, SNMP remains a domain specific technology which is not always easily understood by many developers. Some of the reasons are that the SNMP data model is very data-oriented by nature and there are only a few basic SNMP primitives. In principle, each version of SNMP has *get*, *set* and *trap* primitives and a few variations on them. Having very few primitives could help keeping agents very simple, but it is also a reason that agents and managers have been developed which only offer basic functionality. Most of the current modelling practices are object-oriented by nature, whereas SNMP's data model is not. This is regarded as one of the main shortcomings of SNMP [2, p.49].

Given the fact that SNMP offers a notification primitive, it is awkward to notice that most of the monitoring is done by a polling mechanism: after certain periods of time a manager sends SNMP requests for information to a number of agents. This means that for each request, two messages are transferred (one message to request information and one message in response) whereas a notification would only need to transmit one message from the agent to the manager. Also the same type of information is being gathered, marshalled and unmarshalled which unnecessarily loads both the agents and the managers. It is not difficult to conclude that this is a very inefficient way of monitoring.

In the recent past there have been several meetings by interested parties, such as the Network Management Research Group [3] of the Internet Research Task Force [4] and the Internet Architecture Board [5], about the future of network management (RFC3535 [6]). They all acknowledge the problems that SNMP currently poses. Schönwälder et al. [7] give a short summary of several of these meetings and their overall conclusion was that so-called evolutionary approaches are expected to fail (because of the mentioned limitations of SNMP) and that more focus should be put on revolutionary approaches, most notably XML-based

approaches. This outcome, combined with the fact that there has not been substantial output yet, has made for ongoing IETF workgroups such as Evolution of SNMP (EoS) [8] and SMIng [9] to be discontinued.

B. Towards Web services based management

One of the emerging standards based upon XML is Web services [10]. Web services is a generic technology, because it is XML-based and platform/programming language independent. One can clearly notice an industry-wide interest in Web services, supported by the growing availability of various related application servers and development tools. Web services are expected to become a standard part of future operating systems, which should have a growing familiarity among many users and developers in effect. This makes it to be a very promising technology, even though standards, such as security, transactions, choreography, etc. are still under development.

The availability, combined with being a generic technology and an open standard, makes it easier for people to develop applications using Web services. Apart from dedicated management applications, one can also think of presenting management information in a spreadsheet or storing management information in databases simply by calling a Web service that is already present in the operation system. SNMP also makes clear that availability of applications is a key factor for the market acceptance of a technology [7]. But these advantages of Web services are very general and not only relevant to network management. What is very important for network management is that there is a standardised form in which management information is defined and how this information is accessed. It remains hard to develop a management application when management information and its accessors are not standardised. With regard to the accessors, hereafter called operations, there is a need for a variety of operations and most notably, more efficient operations than SNMP offers.

C. Goal

The goals of this paper are: firstly, to explain how standardisation of management operations can be achieved for Web services and secondly, what form these management operations can take. Hereby we will discuss the merits of both very specific and very generic management operations and compare them to existing Web services-based management approaches. Our intention is to abstract from the definition of management information, or at least leave choices open in how management information should be defined in order to use it for Web services.

D. Approach

We will firstly present a short state of the art of Web services in section II, where the focus is placed on the definition of Web services. This is followed section III about how standardisation of Web services operations can be achieved in the definition of Web

services. An elucidation on the form of management operations is given in section IV, where we will distinguish two degrees of freedom. We will conclude with section V which gives an overview of already existing Web services-based management approaches.

II. WEB SERVICES BACKGROUND

MANY people will nowadays regard the Web as a large collection of web sites, web portals and all other kinds of information displays. Most certainly this is and will remain a very important aspect of the Internet. However, the machine-aware part of the Internet is becoming increasingly important, for it is currently under heavy development and the technologies look promising. The machine-aware part referred to is called Web services for which the World Wide Web Consortium gives the following definition [11]: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards". Simply said, Web services make it possible for machines to communicate with each other by means of standardised messages and regardless of specific hardware or software that a machine uses. Of course, the only requirement is that a machine is able to process Web service requests or send responses.

Web services commonly communicate through the exchange of Simple Object Access Protocol (SOAP) messages [12] which is a standardised form of XML messages. SOAP is language and platform independent and it allows programs to communicate through standard communication protocols, such as HTTP or SMTP. However, Web service communication is not limited to SOAP only. One can for instance also use HTTP-GET or HTTP-POST messages instead of SOAP messages to access the same service, depending on the implementation of the Web service of course. This is depicted in figure 1.

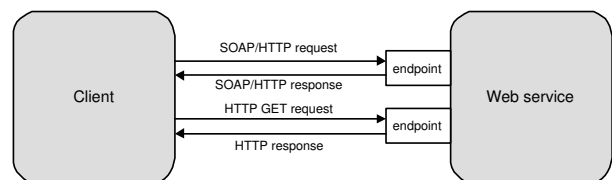


Fig. 1: Web services communication

A Web service is described in a Web Service Description Language (WSDL) document [13]. This document exposes the operations, which parameters to pass to an operation, via which protocols an operation can be accessed and on which location (i.e. the IP number or domainname) the Web service resides. Web services are by design highly extensible and therefore its descriptions too. WSDL documents can either be

kept very simple, using the basic elements and data-types (from XML Schema [14] for instance), but they can also be defined in a modular manner, distributed to any extent and using self-defined data-types of any complexity. SOAP also defines a basic message structure, which can be extended with (extra) headers, attachments and fault messages.

For the explanation of the most important WSDL elements definitions from WSDL version 2.0 will be used. The main differences with the previous version (1.1) is that the element `<porttype>` is now called `<interface>` and `<port>` is now called `<endpoint>`. An example of a WSDL document is shown in listing 1. This lists the main elements and shows the relation between them.

Listing 1: WSDL example

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">

  <types />

  <message name="getNumberOfTcpConnsRequest">
    <part name="index" type="xs:int"/>
  </message>

  <message name="getNumberOfTcpConnsResponse">
    <part name="tcpconns" type="xs:int"/>
  </message>

  <interface name="cmsStatistics">
    <operation name="getNumberOfTcpConns">
      <input message="getNumberOfTcpConnsRequest"/>
      <output message="getNumberOfTcpConnsResponse"/>
    </operation>
  </interface>

  <binding name="cmsSoapBinding" type="cmsStatistics">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getNumberOfTcpConns">
      <soap:operation
        soapAction="http://example.com/cms/getNumberOfTcpConns"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="encoded"
          namespace="http://example.com/cms/message/"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </output>
    </operation>
  </binding>

  <service name="connectionManagementService">
    <endpoint name="cmsSOAP" binding="cmsSoapBinding">
      <soap:address location="http://example.com/cms"/>
    </endpoint>
  </service>
</definitions>
```

A WSDL document has `<definitions>` as root element. The namespaces can be defined as attributes of this element. Each Web service is defined by means of a `<service>` element and can be accessed through endpoints. An endpoint specifies at which address this particular service can be accessed and which protocol should be used for that. Suppose the Web service can be accessed using both SOAP messages over

HTTP and HTTP-GET messages, the physical locations of both endpoints need not necessarily be the same. Listing 1 shows that the *connectionManagementService* can be accessed only with SOAP at location "http://example.com/cms". If this service can also be accessed with HTTP GET messages, it should have a second endpoint such as in listing 2 which also shows how the location of each endpoint can be different.

Listing 2: Endpoints

```
<service name="connectionManagementService">
  <endpoint name="cmsSOAP" binding="cmsSoapBinding">
    <soap:address location="http://example.com/cms"/>
  </endpoint>
  <endpoint name="cmsHTTP" binding="cmsHttpBinding">
    <soap:address location="http://example2.com/cms"/>
  </endpoint>
</service>
```

An interface exposes the operations of the Web service. This can be compared to a function library or a class in a common programming language. Within an operation one can define what the input and output messages are with the `<input>` and `<output>` elements. Each of these elements corresponds to a (SOAP) message exchanged between the client and the service. The structure of such a message is defined in a `<message>` element to which the input or output refers. A message describes exactly of which parts it consists with `<part>` elements. WSDL distinguishes several types of operations that are distinguished from one another by the order of the input and output blocks. This is called a message pattern [15].

An interface is merely an abstract description of the Web service operations while a `<service>` element more concretely describes where this interface is located. The mapping of the abstract description to a location is done with a binding. A binding specifies what kind of messages are exchanged and in which style. In the example SOAP messages are used over HTTP, which is defined in the `<soap:binding/>` element. For each message that is defined in the interface, the binding specifies how the contents should be interpreted: the encoding.

III. STANDARDISATION FOR NETWORK MANAGEMENT

IN order to easily use Web services for network management there should be agreement on the management information and operations. In other words, there is a need for standardisation of information and operations. Standardisation of information is a very broad subject and it is not directly related to Web services. Therefore we will abstract from it and just assume we have certain management information defined in a MIB, such as in SNMP. Note however, that there have been efforts to represent MIBs in XML [16], [17] and reverse engineer MIBs to deduce a UML [18] class diagram [19] that may be interesting when stan-

dardising information. The focus here will be on standardisation of operations.

WSDL provides a mechanism to describe a Web service in a modular manner. The elements `<import>` and `<include>` are used for this. Both elements provide the same functionality of separating different components of a WSDL description, but `<include>` does this for components from the same target namespace, whereas `<import>` is used for different target namespaces. This allows for several WSDL components to be stored at different locations.

Without going into detail on the Universal Description, Discovery and Integration (UDDI) standard, we will mention that the UDDI Technical Committee recommends a division of WSDL documents into a "service interface definition" part and a "service implementation definition" part. The service interface definition part contains the `<message>`, `<interface>` and `<binding>` elements and the service implementation definition part the `<service>` element.

However, we argue that bindings could better be defined separate from the service interface definition. Suppose a service is deployed at many different locations, all of which can be accessed by the same binding. In that case, it would not be necessary to define the binding at each location where one location would be sufficient. Bindings could possibly also be standardised, thereby making operations available through a default protocol and defining a standard encoding of the message parts. Figure 2 shows how a WSDL document is built up, when a modular approach is used. This shows how a WSDL document can be separated into an abstract part containing the messages and interfaces (the **what** part), and two concrete parts: a binding (the **how** part) and a service (the **where** part).

For standardisation, the abstract part of a WSDL document is the most important, since the operations and messages are defined here. Based on WSDL documents a management application can be built, which again stresses the need for standardisation.

When standardising the abstract part of a WSDL definition, the need for a modular approach is implied. This implication is strengthened by the fact that standards should be defined independent from each other and for different purposes, whilst running on the same management agent and offering a similar means of access. Much like SNMP, where for example the IF-MIB [20] and HOST-RESOURCES-MIB [21] are also defined by different persons, each standardising a disjoint set of management information, while all information from both MIBs can be accessed in a similar manner.

With regard to the standardisation of interfaces in the abstract part, the extensibility mechanism provided by WSDL could also be used. This mechanism is the possibility to extend an interface with more operations. WSDL provides an attribute *extends* for the `<interface>` element for this, which points to one or more other interfaces. As shown in listing 3 interface

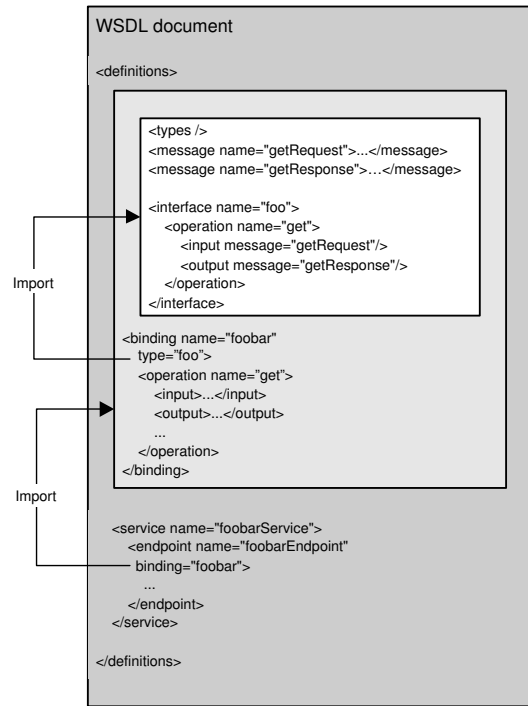


Fig. 2: WSDL import mechanism

intf1 exposes certain operations, such as *op1*. Interface *intf2* exposes the same operations as *intf1*, but can also add extra operations as shown in table I.

Listing 3: Interface extension

```

<interface name="intf1">
  <operation name="op1" ... />
  ...
</interface>

<interface name="intf2" extends="intf1">
  <operation name="op2" ... />
  ...
</interface>

```

TABLE I: Interface extension.

interface	exposed operations
intf1	op1, ...
intf2	op1, op2, ...

IV. WEB SERVICES-BASED MANAGEMENT OPERATIONS

THE previous section described how a Web service should be composed when only parts of it are standardised. But when discussing standardisation of management operations, it first has to be made clear what exactly should be standardised and what the implications of standardisation are on certain types of operations. In order to reason about the implications, the different types of operations need to be defined.

We distinguish four extreme forms that operations can take: coarse and fine operations both with and without transparent parameters as depicted in figure

3. It shows an example that on one side has a very coarse operation called *Get* and on the other side a very fine operation called *GetIfInOctets*. Both can have varying levels of parameter transparency. An example of an operation on an intermediate level is *GetIfInfo*, which is more specific than *Get*, but not as detailed as *GetIfInOctets*. More on this shall be explained in the following sections.

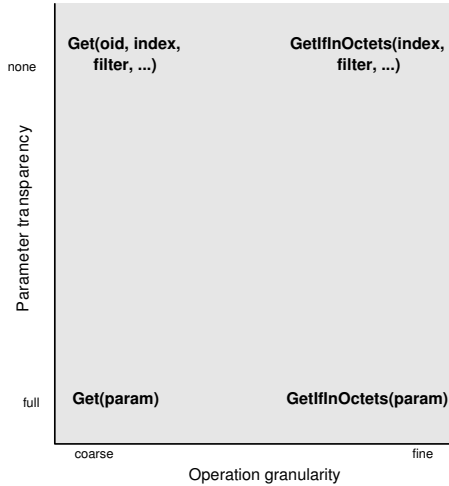


Fig. 3: Operation extremes

A. Parameter transparency

For WSDL operations, management information is exchanged through (SOAP) messages. These messages can consist of several parts. Note that there is a difference between the traditional view of parameters passed to an operation and the parts of the messages that are involved in a WSDL operation. Of course, each parameter can be explicitly defined in a `<part>` element. Input parameters would then be parts of an input message and vice versa for the output parameters. An example is given in figure 4, which explains how messages are defined in case each parameter is mapped to a single part.

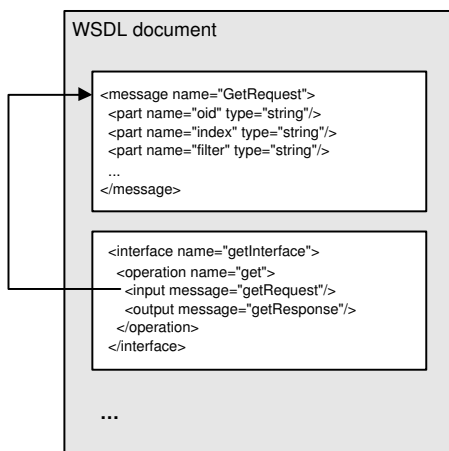


Fig. 4: Operation with non-transparent parameters

On the other hand, parameters can also be combined and/or serialised in such a way, that the param-

eters are not described at WSDL level anymore. Merely their serialisation, is described in a message part. This is called parameter transparency. In that case, serialising should be done upon composing a message and deserialisation upon receiving it. This means that besides awareness of the operation parameters, a management application (and agent) should also offer (de-)serialisation capabilities. Figure 5 shows what a WSDL message looks like when operation parameters are transparent at WSDL level.

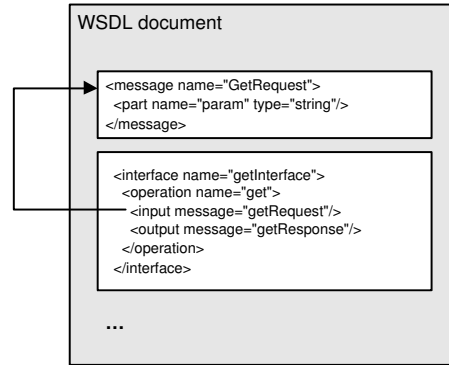


Fig. 5: Operation with transparent parameters

An advantage of transparency is that management information is abstracted from protocol level, so the structure of information can change without having to modify the operation messages. Suppose a Web service supports an operation (irrespective of granularity) with full transparent parameters, such as in the bottom half of the graph of figure 3. All parameters are serialised into one message part, called *param*. In case the parameters are serialised in an XML structure, it can be contained in a message where the part has (XML-)string type. On both the manager and agent side, a generic XML parser can then be used to extract the parameters from the message.

Suppose management information is also defined in XML structures [16], [17], when the agent receives an information request, selecting the information from these structures can be easily done using XPath [22] expressions.

On the manager side however, it means that there should be functionality to firstly create such an XML data structure before the Web service can be called. This entails that it can only be used in either more specific network management applications or by more experienced users/developers. In other words, parameter transparency offers the use of operations very expressively, but this is only useful for professional users who need this kind of flexibility. For a PC user in his home environment who wants to include some management information in his spreadsheet, this will become too complicated. In that case, a simple operation is required where a user does not need to create complicated parameter structures in order to compose the messages in a correct form.

Therefore we can conclude that only if expressiveness is wanted or the use of generic XML parsers is ac-

cepted, it is interesting to have operations with transparent parameters. Non-transparent parameters make operations easy to understand for users and easy to develop for developers.

B. Operation granularity

The other degree of freedom is operation granularity: the level of variation between very coarse and very fine operations. In order to illustrate this, we will assume to have a managed system of which system information and network interface information can be requested. The variables for system information are rather straightforward, namely its location and uptime: *SysLocation* and *SysUptime*. The network interface information is a bit more complex, since a system can have more than one interface. Therefore the same variables can be retrieved for each single network interface: *IfInOctets*, *IfOutOctets*, *IfInErrors* and *IfOutErrors*. Figure 6 shows a containment diagram, which depicts these several types of information we can retrieve from a system.

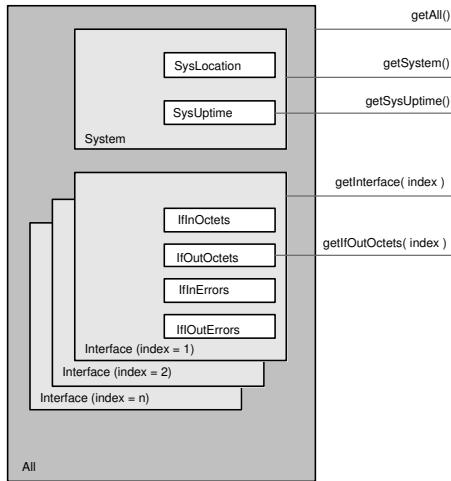


Fig. 6: Containment diagram

In order to retrieve this information from the system, we can define operations that request the managed system for this information. If we consider using very fine operations we would get operations such as *getSysUptime* or *getIfOutOctets* as is also shown in figure 6. So a fine granularity of operations means that for each variable a single operation is defined. Operations for network interface information need to have a parameter supplied to identify the interface.

Suppose one wants to request all information from one network interface. This would result in calling an operation for each single variable. Therefore, a bit coarser operations can also be defined, such as *getSystem*, *getInterface(index)* or perhaps even all information contained in the managed system. In that case, there is not only a single operation for each variable, but also for each container where *All* is the container that contains 'everything'.

An advantage of this approach is that if the naming of the operations precisely defines the functionality, it

is very clear to a user which operation to call to retrieve the information he wants. Generally speaking, the parameters passed to an operation, be it transparent or not, can be simpler since selecting an object is already done by choosing the corresponding operation. In the case of more than one instance of an object (such as the network interface example) one has to provide an instance identifier.

Suppose someone wants to select different types of variables, such as all *System* variables and the *IfInErrors* of an interface with *index = 2*, then one is forced to call two separate operations. If we consider the containment hierarchy as a tree of fine operations (figure 7), then we can state that one fine operation does not allow selection in separate branches of the tree, e.g. one operation can not retrieve both *System* and *IfInErrors*. Only the operation corresponding to the node where these branches meet (*getAll*) would make it possible to retrieve this information. However, *getAll* does not only retrieve *System* and *IfInErrors*, but much more information that in this case would be redundant.

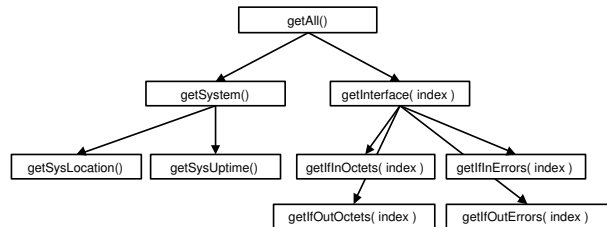


Fig. 7: Containment tree

Suppose filtering the result would be possible on the agent side. This makes it possible to get all information that satisfies some criteria, or get all information with some exceptions or up until a certain depth of the tree. We will use the term filtering for any of these kinds of criteria. Theoretically filtering would make it possible to retrieve any single variable or container only using *getAll* together with a filter. This leads to the concept of a very generic operation, for instance *get*. When provided with a container-name or variable-name, a possible index and perhaps a (simpler) filter, *get* can be used to retrieve any kind of information on any level in the containment tree. In that case, *get* is an example of an extreme coarse operation, having a very generic name and used for more than a single task, in fact for all tasks.

This behaviour makes *get* a very expressive operation, because with one single operation one can get any type of information from the managed system. It does make the parameters that should be passed to the operation more complex, whether they are transparent or not.

It also poses a more complex naming problem. With fine operations, the network manager only needs to know which index to provide when it requests information from an object that can have more than one instance. On the other side, with coarse operations a network manager needs to know how to address any

objects or instances on the agent side.

V. WEB SERVICES-BASED MANAGEMENT APPROACHES

THUSFAR there have been several approaches for network management with Web services.

Hewlett Packard [23] has developed a logical architecture for managing computing resources through Web services. This is called The Web Services Management Framework (WSMF) and it has now been adopted by OASIS [24]). It is developed to address the growing need of businesses to integrate their systems, and more specifically the management of those systems. The framework provides a collection of interfaces that expose a certain type of management information for so-called managed objects. Each interface has operations that are related to a specific task, such as monitoring, discovery or configuration. The WSMF allows for interfaces to be extended or new interfaces to be added for a managed object. The aim is to provide a generic, platform independent interface to management information. The operations provided by the interfaces that are standardised, are generally operations with a fine granularity which serve a specific task. When needed, the extensibility of Web services can be used to specify new non-standardised operations. Since common interfaces provide common operations, one single interface (and thus its operations) can also be used for a collection of managed objects.

Ricardo Neisse et al. [25] introduce the idea of defining methods on different levels of granularity instead of merely copying the SNMP primitives. In this SNMP to Web services gateway, methods are defined on a so-called protocol-level or on an object-level. Methods on protocol-level are translations of SNMP primitives: *Get*, *GetNext* and *Set*, whereas on the object-level there is a specific *Get* method for each scalar and table object, such as *GetSysLocation* or *GetIfTable*. A *Set* method is created for each writable object, i.e. *SetSysLocation* or *SetIfAdminStatus*. So the protocol-level gateway has a few operations with very coarse granularity, whereas the object-level gateway supports only operations with very fine granularity. The incentive of this research project was to conduct a bandwidth comparison between the gateway fine coarse operations and the gateway with fine operations. The result of this comparison was that protocol-level gateways are only interesting when just a few SNMP objects are concerned. This type of gateway uses SNMP object identifiers and the SNMP style of communication: a response message for each single value. The object-level gateway reduces network traffic, because it can send collected management information back to the manager in one SOAP message. This turns out to be more efficient with a high number of instances (this number varies for compressed or uncompressed messages and for SOAP over HTTP or over HTTPS). Therefore an object-level gateway is of particular interest for configuration management, where typically large amounts of information is trans-

ferred.

NETCONF [26] is a Working Group of the IETF and chartered to produce a protocol suitable for network configuration. Configuration typically entails relatively simple tasks such as up- or downloading whole configurations. It therefore needs only a few basic operations to transfer large amounts of data. The NETCONF protocol offers a small set of coarse operations to manage device configurations and retrieve device state information. However, these set of coarse operations is meant to be extensible with finer operations when specific functionality is required. But considering the expected usage of coarse operations, there is no need for standardising finer operations. Communication is performed through the exchange of XML messages, which makes NETCONF acknowledge that using SOAP for message exchange is definitely interesting [27]. It offers the functionality that is wanted, but more important it is widely supported on many platforms and understood by more and more people. It could therefore be used as a Web service, especially since Web services also provide standard support for importing definitions in case of extensible operations.

VI. CONCLUSION

In summary, standardising Web services-based network management operations should be done by standardising the messages, interfaces and possibly the types of a WSDL document, i.e. the abstract part. Types could also be imported from XML Schema or other schema definitions, for example. This allows for the reusability of operation definitions for several bindings or services. The concrete WSDL (with a location and possibly a protocol binding) of a network management Web service can then import the abstract WSDL document and thereby using standardised operations to monitor or configure management information. Interfaces have the extra feature that they can be extended, therefore allowing simple, generic interfaces be defined and where necessary, more complex, specific interfaces based on these.

Operations with fine granularity are generally very simple to use. They are easily understood, provide limited functionality per operation and are therefore very suitable for easily creating management applications. Should an application require coarser operations, there is still the possibility to extend interfaces with these operations. Of course, extending an interface with new fine operations providing new functionality is also an option.

Non-transparent parameters are also a means of providing easy understanding of operation functionality. It also makes it possible to simply use these operations in office applications, such as a spreadsheet, or in a simple database environment, for there is no need for extra tools like XML parsers or other tools to (de-)serialise parameters in case of higher transparency.

What becomes clear from the existing Web services approaches, is that two of them offer a small basic set

of coarse operations and at the same time acknowledge the need for extensibility with finer operations. Only the WSMF provides some basic fine operations, but they can be applied to similar managed objects that reside on different locations. The WSMF also allows for extension of its operations.

The work presented here is part of ongoing research. We strongly encourage interested parties to react and comment on the discussed issues.

REFERENCES

- [1] R. Frye, D. Levi, S. Routhier, and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework, RFC3584," RFC 3584, Internet Engineering Task Force, August 2003.
- [2] Jean-Philippe Martin-Flatin, *Web-Based Management of IP Networks and Systems*, John Wiley & Sons, Ltd., Chichester, West Sussex, PO19 8SQ, England, 2003.
- [3] "Network Management Research Group," <url:http://www.ibr.cs.tu-bs.de/projects/nmrg/>.
- [4] "Internet Research Task Force," <url:http://www.irtf.org>.
- [5] "Internet Architecture Board," <url:http://www.iab.org>.
- [6] J. Schönwälder, "Overview of the 2002 IAB Network Management Workshop, RFC3535," RFC 3535, Internet Engineering Task Force, May 2003.
- [7] J. Schönwälder, A. Pras, and J.P. Martin-Flatin, "On the future of internet management technologies," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 90–97, Oct. 2003.
- [8] "IETF: Evolution of SNMP Working Group," <url:http://www.ietf.org/ietf/eos/>.
- [9] "IETF: Next Generation Structure of Management Information Working Group," <url:http://www.ietf.org/ietf/smimg/>.
- [10] "W3C: Web Services Activity," <url:http://www.w3.org/2002/ws/>.
- [11] "W3C: Web Services Architecture (8 August 2003)," <url:http://www.w3.org/TR/2003/WD-ws-arch-20030808/#whatis>.
- [12] "W3C: XML Protocol Working Group," <url:http://www.w3.org/2000/xp/Group/>.
- [13] "W3C: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," <url:http://www.w3.org/TR/wsdl120/>.
- [14] "W3C: XML Schema," <url:http://www.w3.org/XML/Schema>.
- [15] "W3C: Web Services Description Language (WSDL) Version 2.0 Part 2: Message Patterns," <url:http://www.w3.org/TR/2003/WD-wsdl20-patterns-20031110/>.
- [16] "Avaya Labs Research - XML based Mgmt Interface," <url:http://www.research.avayalabs.com/user/mazum/Projects/XML/>.
- [17] F. Strauß and T. Klie, "Towards XML oriented internet management," in *Proc. 8th IFIP/IEEE International Symposium on Integrated Network Management*, Colorado Springs, Mar. 2003, pp. 505–518.
- [18] "Unified Modeling Language," <url:http://www.omg.org/uml/>.
- [19] J. Schönwälder and A. Müller, "Reverse engineering internet MIBs," <url:http://www.ibr.cs.tu-bs.de/vs/papers/im-2001.pdf>.
- [20] K. McCloghrie and F. Kastenholz, "The interfaces group MIB using SMIv2," RFC 2233, Internet Engineering Task Force, Nov. 1997.
- [21] P. Grillo and S. Waldbusser, "Host resources MIB," RFC 2790, Internet Engineering Task Force, Mar. 2000.
- [22] "W3C: XML Path Language (XPath) Version 1.0," <url:http://www.w3.org/TR/xpath/>.
- [23] "HP - Web Services Management Framework," <url:http://devresource.hp.com/drc/specifications/wsmf/>.
- [24] "OASIS Web Services Distributed Management TC," <url:http://www.oasis-open.org/committees/wsdm/>.
- [25] R. Neisse, R. L. Vianna, L. Z. Granville, M. J. B. Almeida, and L. M. R. Tarouco, "Implementation and bandwidth consumption evaluation of snmp to web services gateways," *IEEE/IFIP Network Operations & Management Symposium*, Apr. 2004.
- [26] "IETF: NETCONF Working Group," <url:http://www.ops.ietf.org/netconf/>.
- [27] T. Goddard, "NETCONF over SOAP," Internet-Draft, feb 2004, <url:http://www.ietf.org/internet-drafts/draft-ietf-netconf-soap-01.txt>.