Experiences in Using MUWS for Scalable Distributed Monitoring

Aimilios Chourmouziadis[1], Oscar F. Gonzalez Duque[1], George Pavlou[2],
[1]Centre for Communications Systems Research (CCSR), University of Surrey, Guildford, UK,
[2]Networks and Services Research Lab, University College London, UK.

***Abstract*- Efficient Web Services (WS) based network monitoring of managed devices is a difficult task due to the relatively big overhead WS impose. In the past we proposed mechanisms to perform distributed monitoring efficiently, minimizing the relevant overhead. Standardization of WS operations is also important in order to achieve interoperability. The WS Resource Framework (WSRF) tries to standardize the messages exchanged with resources representing the state of a device. Adopting WSRF's concepts, the Management Using Web Services (MUWS) standard aims to support device management in an interoperable manner. In this paper we propose methods to use the mechanisms introduced in our previous work combined with MUWS in order to establish the means to retrieve management information efficiently and at the same time achieve interoperability. We also present our experiences in using custom as well as standardized solutions for monitoring devices that range from small to large resource-capable systems. We describe the motivations for this research and present ideas on techniques that need to be adopted for WS based monitoring based on what we have learned in the process.**

***Index Terms*—WS, MUWS, WSRF, interoperability**

I. INTRODUCTION

OVER the past few years, various research groups defined many WS-based specifications for Network and Service Management (NSM). Some of the most prominent work comes from the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Distributed Management (WSDM) group.

From this group of most notable importance are two specification documents; (a) the MUWS specification [1] (b) the Management of Web Services [2] specification (MOWS). In the MUWS specification authors describe how to manage the resources of devices with the use of WS. In MUWS the authors also recognize the importance of managing state when interacting with distributed resources. This is necessary because although WS are typically stateless [3], "the interface maintained by a WS is clearly stateful, since its behavior is defined with respect to the underlying state" [4].

MUWS concepts of managing state come from the Web Services Resource Framework (WSRF [5]) which partitions the functionality required for managing state into several specifications and introduces the concept of the WS-Resource [8]. Expanding the work in [5], MUWS introduces the concept of the manageable resource, a refinement of a WSRF resource. Through the use of the concept of the manageable resource and standardization, MUWS promotes interoperability in managing a resource's state. Despite the interoperability benefits, the use of standards such as MUWS, raise efficiency and scalability issues as for example is explained in [6].

In the past we have suggested mechanisms to improve the efficiency of WS management applications [7], [15], [16]. We have built and deployed a query tool that tries to increase the performance of management applications for monitoring and event reporting. This tool supports (a) distributed monitoring as part of an architecture that gives a complete view of management services through ONE agent by supporting federation of management requests. (b) it supports efficient bulk and selective retrieval through information processing in cases where the entire state of a device is not required and (c) it exploits the conceptual relationships between state data, structuring the WS encompassing them in hierarchies for effective monitoring (the importance of exploiting the relationships between state data is shown in [25] and [17]).

Reading through the specifications of MUWS three features of the latter seem very promising for integration of our query tool and architecture with MUWS for efficient and effective monitoring and interoperability. The first is the support of MUWS for resource specific query languages such as the one supported by our query tool. The second feature is the support that MUWS provides in defining relationships between WS, based on the Resource Properties (RPs) the latter share. Our query tool exploits these relationships for monitoring and MUWS promotes an XML schema for defining them as common MUWS RPs. In our previous work [7] we introduced how to use relationships between management state data for efficient monitoring. Based on the relationships between state data we introduced the idea of structuring a hierarchy of WS interfaces which can be searched more efficiently using our query tool. The third feature is the support of MUWS to WSRF's WS-ServiceGroup (WS-SG) specification [11]. The concepts in [11] can be used for composition of resources based on RP they share. This way collective access to RP can be achieved. Even more, resources can be grouped forming hierarchies of resources enabling better access to RP for monitoring. Our query tool exploits-requires such hierarchies for effective retrieval of state information. Enabling combination of custom solutions for management with standards can promote efficient management solutions especially now that the use of

This work was supported partly supported by the EU EMANICS Network of Excellence on the Management of Next Generation Networks IST-026854, and the IST ENTHRONE phase 2 projects.

978-1-4244-3487-9/09/$25.00 ©

custom solutions within a domain and the use of standards at the edges are promoted so as to increase WS scalability [20].

Based on the above this paper will try to perform the following: (a) show how to potentially integrate our query tool and distributed monitoring architecture with MUWS (b) evolve the concept of navigation of relationships between WS for monitoring even further compared to our work in [7], this time for relationships between MUWS' WS-Resources (c) present a full monitoring example where navigation of relationships between WS-Resources takes place (d) share our experiences in integrating our query tool and architecture with Apache MUSE (a MUWS framework implementation), (e) test the performance of MUWS based on a number of monitoring scenarios that require use of a query tool against a custom framework we have built for monitoring and SNMP.

To perform the above the rest of the paper is organized as follows. Section 2 provides a background on WSRF, MUWS and our query tool and architecture. Section 3 analyzes the potential for integrating our query tool and architecture with MUWS and evolves the concept of navigating relationships between WS-Resources for monitoring. Section 4 analyses an example of navigating relationships between WS-Resources for monitoring. Section 5 analyzes our experiences in integrating MUWS and our query tool and architecture, and analyzes measurements on specific scenarios for polling based monitoring. In section 6 we draw our conclusions.

II. Background And Analysis

A. Managing state with stateful resources using the WSRF framework and MUWS

Using WS for distributed management requires standardization of the interactions of WS with stateful resources in order to facilitate interoperability. As such the WSRF defines mechanisms and constructs to enable WS to access state in a consistent and interoperable manner. To retain the WS characteristic of being stateless, WSRF proposes to distinguish WS from state and RPs by introducing the WS-Resource [8] construct. WS-Resources represent an association between a WS and a number of stateful resources in terms of the *implied resource pattern.*

WSRF partitions the functionality required for managing state into five interrelated specifications, each of which tackles a different aspect of managing state. The WS-Resource standard [8] analyzes how to associate a WS with a resource. The WS-ResourceProperties (WS-RP) standard [9] defines how RP are modeled in RP documents exposing a number of the available properties of a resource to users (consumers) that want to access, change or delete them. The WS-Resource Lifetime (WS-RL) [10] specification standardizes the message exchanges taking place in order to manage the lifetime of a WS-Resource. The WS-ServiceGroup (WS-SG) [11] specification defines how to collectively access RPs of various WS-Resources based on several constraints. The WS-BaseFaults (WS-BF) [12] standard describes fault types for describing errors produced when exchanging messages with resources.

Based on the concepts of the WSRF, MUWS introduces a new concept; the manageable resource. A manageable resource is a refinement of a WSRF resource. A resource is manageable when it exposes a set of manageability capabilities. The latter is a set of RPs, operations, events, metadata describing the specific behavior of a resource and defining its ability to be managed. Some of the operations MUWS defines for accessing and managing resources allow the managing entity to access RPs of a resource, subscribe to events, or, control the resource. Most of these operations are inherited by WSRF and the WS-BaseNotification [13] standard, the latter being used for manipulating event information. Some of these operations are given in Table I.

TABLE I
SOME OF THE MUWS OPERATIONS TO MANAGE WS-RESOURCES

QueryResourceProperties (QRP)	Retrieves specific RPs using a query language (e.g. using XPath).
QueryRelationshipsByType (QRBT)	Retrieves information for particular relationships resources share.
Subscribe	Requests that specific notifications are sent to an event consumer
Notify	Notifies a consumer about an event

B. A custom Query Tool for information processing and distributed monitoring

The query tool we have developed in [7] supports bulk and selective retrieval through information processing, exploiting the relationships between state data for effective distributed monitoring. This is performed through the use of four types of queries each one with a special functionality. All queries are carried as parameters in operations exposed by WS interfaces. Each WS interface exposes management data from SNMP MIB's which represent state data from a managed device. The types of queries supported are: (a) *Service Selection (SS)*, (b) *Single Instance Data (SID)*, (c) *Multiple Instance Data (MID)*, and, (d) *Filtering Data* (FD) queries. SS queries are a combination of WS endpoint addresses, level restrictions and relationship restrictions in order to select local or remote WS exposing the state data of a managed device from which state information will be retrieved. Level restrictions are applied in order to specify from which levels of a hierarchy of WS data can be retrieved. Relationship restrictions are applied to enforce selecting WS whose state data share specific relationships with the state data of other WS. Relationship restrictions are applied by an agent according to the hierarchical view of WS the latter has (such as that in Fig. 1). How the agent acquires this view is explained fully in [7]. *SID, MID and FD queries* are called data queries because they are used to retrieve the state data of a managed device. *SID and MID* queries allow the retrieval of single and multiple instance data respectively (i.e. a table's rows are represented by multiple instance objects) from a WS. *FD* queries can be applied to *MID* queries to filter the collected data. Usage of the four types of queries supports distributed monitoring as shown in Fig. 1.

In Fig.1 a process where the retrieval of management data from WS implementing SNMP MIB's occurs is depicted. In this figure a manager sends a request to an agent that has a WS interface using an operation it exposes. As part of the operation's operands the manager sends SS, SID, MID, FD

queries and a callback address (Fig.1 step 1). Each SS query has an association with data queries and their combination is used to retrieve management data. On receiving a number of queries the agent extracts each SS query to determine whether local or remote data from WS exposing these data need to be accessed (Fig.1 step 2 and 3). In case remote data from managed devices need to be accessed, the current agent tries to route the SS and data selection queries to the remote agent of these devices (step 4 alternative). This is achieved by extracting the remote agent's address from each SS query and sending a request to the next hop agent through which this address can be reached. This continues from agent to agent until the remote agent is reached. Each request made to a next hop agent contains the SS query and their associated data queries. The request also contains the callback address of the manager that sent these queries in the first place. At the remote agent, the manager's callback address is used to send back the required data to it using a process similar to the one for retrieving local data explained below (step 4 alternative). This process distributes the monitoring load to several agents. If local data need to be accessed (step 4), the agent uses the relationship and level restrictions in a SS query to find the WS whose state data share specific relationships with other WS. This process is called service selection. During this process the agent picks the services to retrieve data from. After service selection the agent dispatches the data queries to each WS through an operation that each one exposes. Each WS uses its query tool instance to analyze the queries it received. After determining which data to retrieve, each WS sends a request for the data to the associated managed device (step 5 and 6). Each WS responds to the agent with the required data in XML format although management data are held in programming language objects (step 8). The agent concatenates the data and sends back the response to the manager (step 9). An implementation of the query tool was presented in [16], demonstrating the scalability of our query tool to XPath.

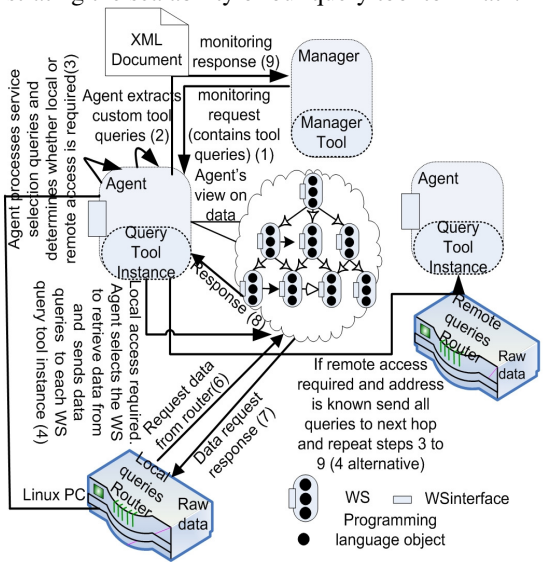

Fig. 1. Distributed monitoring using the custom query tool to support bulk and selective retrieval monitoring

In order to perform efficient distributed monitoring as described previously using the standard operations and concepts of the MUWS framework, it is required to (a) have a number of agents to manage the monitoring process (b) expose management state data using the WS-Resource concept (c) build WS interfaces supporting the standard operations of MUWS (d) support the custom tool queries within the standard operations of MUWS (e) build a hierarchy of WS-Resource interfaces as shown in Fig. 2 using the WS-SG specification. All the previous characteristics can be supported using the concepts of the MUWS architecture with some adjustments to the architecture described in Fig.1. The details of this are presented in section 3.

III. Integrating MUWS With Our Query Tool And Architecture

A. *The MUWS potential to support our distributed monitoring scheme*

The QRP operation in Table 1 enables retrieval of RPs in a selective or bulk manner. This is achieved using a query language. MUWS supports two well known query languages; XPath v1.0 and v2.0. The use of one or the other is supported by a dialect attribute pointing to the specification of their syntax. In theory any resource specific query language can be used as stated in [14]. This clearly shows that we can use our own query tool with MUWS for data retrieval, especially since it has been proven to be more scalable than XPath ([15], [16]).

The *QueryRelationshipsByType* (QRBT) operation also presents great application potential for use with our query tool. QRBT is an operation that can be used for retrieving information about relationships that exist between WS-Resources. According to MUWS, relationships are RPs inside a RP document that resources share. In the past, research has been conducted by the authors where relationships between management data representing the state of a managed device were used in order to search the management data hierarchy more effectively [17]. This way bulk access to network management state data was provided. At the time though programming concepts such as classes and containment were used to model the state of managed devices and the relationships between state data (OSI-SM). Relationships in OSI-SM though are shared between state data and the objects that encompass them. Since WS or WS-Resources also encompass state data or RP, they also share relationships. To define these relationships MUWS provides elements to model them as RPs and provides the QRBT operation to access them. This way MUWS standardizes the means to define and access relationships among WS-Resources. This is vital to support our agents functionality in Fig.1 for effective monitoring.

A difference though between the work in [17], and using relationships to search a hierarchy of WS or WS-Resources, is that object oriented principles such as containment facilitated the structuring of state data in hierarchies with different levels of abstraction. This allowed searching for state data more effectively. WS or WS-Resources offering access to RP don't do this by default. This is where the WS-SG Specification [11] supported by MUWS comes into play. With the use of WS-SG

collective access to WS-Resources can be provided using membership constraints based on PRs that resources share.

A. Building WS-Resource hierarchies

To structure a hierarchy of WS as that in Fig. 1, this time though with WS-Resources we need to evolve the ideas presented in [7] and introduce three new rules for structuring WS-Resources in hierarchies, as well as advance the idea about relationships between state data-RPs that can be used for monitoring. An example of how to organize WS-Resources representing management state data in hierarchies is given in Fig. 2. In this figure examples of 5 types of relationships between management data are given. Containment relationships are the most common relationships and are the basis for building hierarchies of objects or WS-Resources. In programming language terms, objects at higher level of a hierarchy (i.e. level 0) contain a portion of management state data from objects at lower levels of a hierarchy (i.e. level 1) as well as their own data. The same can happen with WS-Resources. Using WS-SG concepts, a WS-Resource at level 1 can be created from WS-Resources at level 2 and thus contain RPs from the latter layer as well as its own. Thus access to WS-Resources at level 2 can be provided from level 1. A very good example of containment relationships can be borrowed from SNMP Management Information Bases (MIBs). An SNMP table *contains* management state data representing its columns and rows. Another type of relationship common to SNMP MIBs is *augmentation*. A table *augments* another table when both have common row identifiers. When *augments* relationships or any other type of relationship apart from containment exist between two WS-Resources, those resources lie at the same level. If a WS-Resource shares containment and other relationships with other resources, containment is the dominant relationship when classifying WS-Resources in hierarchies. Based on containment and the above two rules, the hierarchy in Fig.2 is built between data of the Traffic Engineering (TE) MIBs (RFCs 3812, 3813, 3814).

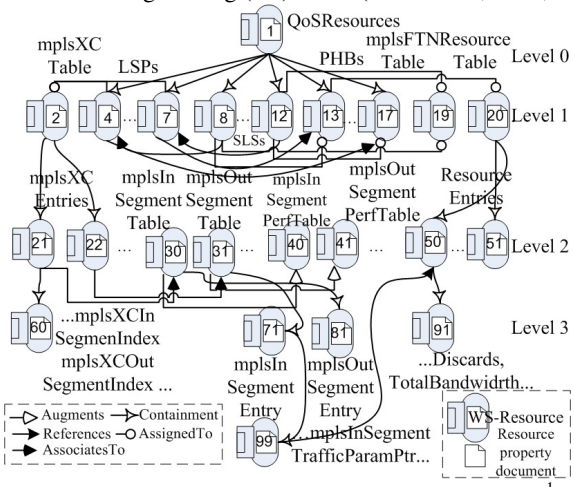

Fig. 2. Organizing WS-Resources state data in hierarchies[1]

In Fig. 2 another common relationship between SNMP MIBs is a *References* relationship. *References* relationships occur when for example an attribute references another attribute. An example of this relationship is the *mplsInSegmentTraffic-ParamPtr* attribute in the *mplsInSegmentTable* of the MPLS Label Switching Router (LSR) [18] MIB referencing a row of the *mplsTunnelResourceTable* in the MPLS Traffic Engineering MIB [19] containing the characteristics of a QoS Traffic Class. *AssociatesTo* and *AssignedTo* relationships are also very common between the traffic engineering MIBs. A Label Switched Path (LSP) is associated to a Per Hop Behaviour (PHB-traffic class). A Service Level Specification (SLS-traffic contract) is assigned to a PHB.

B. Distributing monitoring of WS-Resources

In order to support our distributed monitoring architecture and query tool using the concepts of the MUWS framework, it is vital to meet the requirements given at the end of section 2.

1) First Requirement

In terms of the requirement to have a series of agents to manage the monitoring process, the MUWS framework specification defines that it supports both agentless or with an agent implementations when managing WS-Resources. Thus supporting the agents of the architecture in Fig. 1 with MUWS is possible.

2) Second and Third Requirement

In terms of exposing management data using the WS-Resource concept for the architecture in Fig. 1 there are 4 conditions that need to be met. First each WS exposing a manageable resource should expose the MUWS operations through a WS addressing endpoint. Second resources should be exposed in terms of RP. Third RP documents should be linked with the WS interfaces through their WSDL portType elements (to form a WS-Resource). Fourth the agent itself should be a WS-Resource having access to all the other WS-Resources. The first condition can be satisfied as long as the WS of Fig. 1 are built in order to expose the standard operations of MUWS. The second condition can also be satisfied since the WS-RP specification and thus MUWS also support implementations of the RP document that are not instances of an XML Schema. This is necessary since our query tool needs to dynamically construct the RP document and their values from data held in programming language objects by binding these elements to an XML document instance. MUWS allows resource specific implementations of the RP document and thus the second condition can be met without risking being non-conformant to WSRF's and MUWS' concepts for exposing state as RPs. For the third condition each WS exposing management data should be linked with a RP document by referring to it in its WSDL portType. This can be achieved with any WSDL implementation. For the fourth condition to be met the agent should use the concept of the WS-SG specification to group WS-Resources so that collective access to resources is provided. This can be achieved as explained in the fifth requirement below but also requires the agent to be a WS-Resource. This can be supported by the architecture of Fig. 1 by making the appropriate changes so that the WS representing the agents to be turned into WS-Resources.

[1]Fig. 2 is not an exhaustive list of relationships between resources and should not be considered as normative but only as a possible way to structure resources.

Fulfilling the second requirement mandates that both the agent and management data be exposed as WS-Resources exposing the standard operations of MUWS. This means that the third requirement is also satisfied.

1) Fourth Requirement

The fourth requirement can be achieved using the *QRP* and *QRBT* operations of MUWS. The *QRBT* operation can be used to retrieve the relationships that WS-Resources share in order for an agent to build its conceptual view of WS-Resources. This way when a SS query is dispatched from a manager to an agent, the latter will be able to select the WS-Resources to retrieve data from. The *QRP* operation can be used to perform bulk and selective retrieval of RPs from WS-Resources using our query tool. This requires the QRP operation to support our custom tool queries. This can be achieved by using the dialect attribute of the *QRP* operation to point to a specification of our query tool. Since [14] states that query languages are resource specific, this should not be a problem. Fig. 3 shows a usage example of the QRP operation to support our tool queries.

2) Fifth Requirement

Fulfilling the fifth requirement requires using the WS-SG specification to build a hierarchy of WS-Resources. Using this hierarchy, data from WS-Resources can be retrieved more efficiently. In order to build such a hierarchy, containment can be the relationship between WS-Resources that can serve as the basis of a member constraint to build the levels of the hierarchy. Fig. 4 gives an example of a containment relationship between two WS-Resources one of which resides at level 2 of the hierarchy and one at level 3. In this figure, the relationship type and level association elements can be used by the agents of our architecture to build a conceptual hierarchy of WS-Resources and are defined in a separate XML schema (belonging in the *rel* namespace). Inserting schema specific information such as those in the *rel* schema inside the *type* and *participant* elements of the MUWS schema is allowed by the latter in order to describe any schema specific information about WS relationships. WS-Resources can share other types of relationships apart from containment given in Fig. 4. As long as relationships are defined in MUWS within relationship elements and stored as RPs of a WS-Resource, our agents can look them so as to build the hierarchy of Fig. 2.

In order to actually support the tree of Fig. 2, collective access from a WS-Resource at a higher level to WS-Resources of lower levels is necessary. As such it is necessary for a higher level resource to be able to access the WSDL operations and RP documents of WS-Resources at lower levels. For WSDL 2.0 this is straightforward due to its extensible nature. Accessing operations from the WSDL document of another WS-Resource necessitates that the latter references these operations in its portType. WSDL 2.0 allows this through an extension attribute in the portType definition. In order for a WS-Resource to be able to have access to other RP of other WS-Resources, the WSDL RP document schema of the former has to reference the WSDL RP schemas of the latter resources. This in WSDL 2.0 can be achieved by first defining each RP document of a WS-Resource in a separate XML schema. Then using the import attribute of an XML schema inside the WSDL document of a WS-Resource it is possible to refer to elements of other schemas of other resources. WSDL 1.1 though is not as extensible as WSDL 2.0. As such, in order to perform the above with WSDL 1.1, the schemas and operations of lower level resources have to be manually imported in the WSDL document of the higher level WS-Resource. It is obviously more flexible to build hierarchies of WS-Resources with WSDL 2.0.

```
<wsrp:QueryResourceProperties>
  <wsrp:QueryExpression Dialect='http:131.227.88.70/custom/query>
    <qrt:SS_Query>{http://192.168.50.4:8080/WS-
          Resource/1/,2,3,AssociatesTo*Augments}
     </qrt::SS_Query>
     <qrt:MID_Query>{mplsInSegmentPerfEntry[ ]}
     </qrt:MID_Query>
     <qrt:FD_Query>{mplsinSegmentPerfDiscards<=500}
     </qrt:FD_Query>
    <qrt:SS_Query>{http://192.168.60.3:8080/WS-
         Resource/2/,2,3, AssociatesTo*Augments}
     </qrt:SS_Query>
     <qrt:MID_Query>{mplsInSegmentPerfEntry[ ]}
     </qrt:MID_Query>
     <qrt:FD_Query>{mplsinSegmentPerfDiscards>=1000}
     </qrt:FD_Query>
     <qrt:CBackAddress>…</qrt:CBackAddress>
  </wsrp:QueryExpression>
</wsrp:QueryResourceProperties>
```

Fig. 3. Custom tool queries with MUWS's QRP operation

```
<muws2:Relationship>
  <muws2:Name>…</muws2:Name>
  <muws2:Type><rel:containment></muws2:Type>
  <muws2:Participant>
     <muws1:ManageabilityEndpointReference>..EPR2…
     </muws1:ManageabilityEndpointReference>
     <wsa:EndpointRefence>...EPR2…
     </wsa:EndpointReference>
     <muws1:ResourceId>…</muws1:ResourceId>
     <muws2:Role>…</muws2:Role>
     <rel:Lvl>3</rel:Lvl>
  </muws2:Participant>
  <muws2:Participant>
     <muws2:Self/>
     <muws2:Role>…</muws2:Role>
      <rel:Lvl>2</rel:Lvl>
  </muws2:Participant>
</muws2:Relationship>
```

Fig. 4. Defining relationships of WS-resources as RPs

IV. WS-RESOURCE MONITORING EXAMPLE

Having explained how to use MUWS to achieve the requirements of the distributed monitoring architecture in Fig.1, it is now possible to provide an example. This example will show how to use MUWS operations for distributed monitoring across a network domain.

In the distributed monitoring example using MUWS operations we have to imagine that the architecture in Fig. 1 has been transformed to support the WS-Resource concept and the operations of MUWS. In addition, we need to assume that each agent has a view on a hierarchy of WS-Resources that looks like the one between WS-Resources in Fig. 2. The example begins by having the manager of Fig.1 query the WS-Resource interface of the agent associated with the local queries router by invoking its QRP operation. This operation carries the data shown in Fig.3. The agent extracts the <qrt:SS_Query> elements from the QRP operations and checks the addresses they contain. The agent thus realizes that

the first query is for the local router and the second for a remote router. As such, it dispatches the remote SS query and its associated MID and FD queries to the next hop remote agent by invoking the latter's QRP operation. In this operation the agent also inserts the callback address of the manager. Back to the local queries agent, monitoring resumes by having the latter process the local SS query. The agent then determines that the manager wants to retrieve properties from the WS-Resources in level 2 and 3 that can be reached by first following relationships of type *AssociatesTo* and then type *Augments* starting the search from WS-Resource 1. The agent then searches the conceptual tree by invoking each WS-Resource's QRBT operation by searching for relationships of type *AssociatesTo*. This eliminates all WS-Resources apart from 30,31,4 to 7, 13 to 17 because they cannot be reached by an AssociatesTo relationship. The agent then queries again the remaining WS-Resources for relationships of type *Augments*. The latter results in having the agent select WS-Resources 40 and 41 in order to retrieve state data. Then the agent applies the level restrictions which mandate selecting only WS-Resources between and including levels 2 and 3. Since WS-Resources 40 and 41 belong to level 2, they remain selected. The agent then dispatches the MID and FD queries to WS-Resources 40 and 41 using their QRP operations. In each WS-Resource the mplsInSegmentPerfEntry instances are selected which have mplsInSegmentPerfDiscards values less than 500 [2]. WS-Resource 41 does not contain any information as the ones requested by the agent and thus sends an empty response. WS-Resource 40 responds to the agent with the result contained in an XML document. The agent would normally concatenate results, but in this case it does not happen because from the WS-Resources selected, only WS-Resource 40 provides a number of RPs. The agent then sends back to the manager the result in XML format. A similar process as for the local agent also takes place in the remote agent when it receives the remote queries.

I. Experimental Work

To test the performance of MUWS and our query tool for monitoring WS-Resources we need to introduce two scenarios based on a QoS MPLS network and the traffic TE MIBs [18], [19] which require information processing as well as retrieving data in a bulk or selective manner.

The first scenario, is one that we used in [15] and [16] to test the performance of XPath 1 & 2 and our query tool. In this scenario a QoS network interface fails and the manager after being notified by an event needs to determine the contracts and the Logical Switched Paths (LSPs) that are affected for a small (30 LSPs) and a big network (900 LSPs) (for the measurements 6 LSPs and contracts are affected for both types of networks thus only the volume of data that needs to be searched changes - this is why Fig.5 contains only one traffic overhead figure). Based on this scenario we will test the performance of MUWS against a Custom Framework (CF), using XPath 1 or 2 or our query tool for MUWS, and only our query tool for the CF. For this scenario either when using XPath 1 or 2 or our custom query tool three queries (qry1,qry2, qry3 in Fig.5) must be sent from the manager to an agent. As shown in [15] the XPath 1 and 2 queries are more verbose than our query tool and require more merging and filtering operations thus more processing.

For the measurements of this scenario we used Java 1.6.0. The MUWS framework implementation is supported by Muse 2.2.0 which is based on Apache Axis 2.1.1 software to deploy and build WS. Axis uses the Streaming API for XML processing (StAX) pull parser to efficiently split an XML stream into small sized chunks. As such it can build a partial XML infoset tree in memory in an incremental manner, allowing applications to start processing the XML content even before the entire document has been parsed minimizing latency. One of the most interesting features of Axis 2.0 is its AXIOM object model (AXIs Object Model) and its built-in support for the W3C XOP (XML-binary Optimized Packaging) and MTOM (Message Transmission Optimization Mechanism) standards used in the latest version of SOAP attachments. These two standards work together providing a way for XML documents to logically include blobs of arbitrary binary data into SOAP messages. XOP and MTOM are crucial features of the new generation of Web services frameworks since they finally provide interoperable attachment support, increase performance and end the current problems in this area [21]. Version 2.1.1 of AXIS 2 though has problems with its StaX XML stream utilities so we tweaked Muse to support version 2.1.4. Muse currently only supports XPath 1.0 so in order to support XPath 2.0 and our query tool we tweaked Muse according to the guide in [22]. Additionally Muse only currently supports XML representations of RPs so we had to change its source code to support JAXB object representations (raw data) of an XML document and schema which our query tool requires. The CF which uses our query tool also uses Axis 2.1.4. For XPath 1 support we used JAXP 1.4 and for XPath 2 Saxon 8.9 by Michael Kay one of the standard authors of XPath 2.0. Both implementation of XPath are conformant to the standards, come from reliable bodies and provide full support to XPath capabilities. To calculate traffic overhead we used Linux's tcpdump utilities. For latency we used Java's currentTimeMillis() function using the average of 10 results for each sample. One thing to note is that version 9.0 of Saxon introduces considerably more latency than version 8.x so we opted for version 8.

From the measurements in Fig.5 we can see that our CF is less verbose than MUSE (6639 bytes to 10137). This explains why both for a small and a big network the MUWS version that uses our query tool introduces a bit more latency than the CF (230ms vs 238ms for a small network, 349ms vs 384ms for the big network). This would not be the case though if MUWS was using XPath 1.0 or 2.0. The fact that XPath 1.0 as explained in [16] has more verbose queries, introduces MUWS with more traffic overhead. Additionally the fact that XPath 1.0 and 2.0 require more filtering and merging operations introduces considerably more latency for MUWS than our CF especially when the amount of information to be processed increases (1.5 to 13 times more latency for XPath 2, 2.9 to 17 times more latency for XPath 1). The latter is attributed as explained in [16] in DOM's inefficiency. As such

[2]MplsInSegmentPerfEntries and MplsInSegmentPerfDiscards do not appear as properties of WS-Resources in Fig.2 because the figure would look crowed.

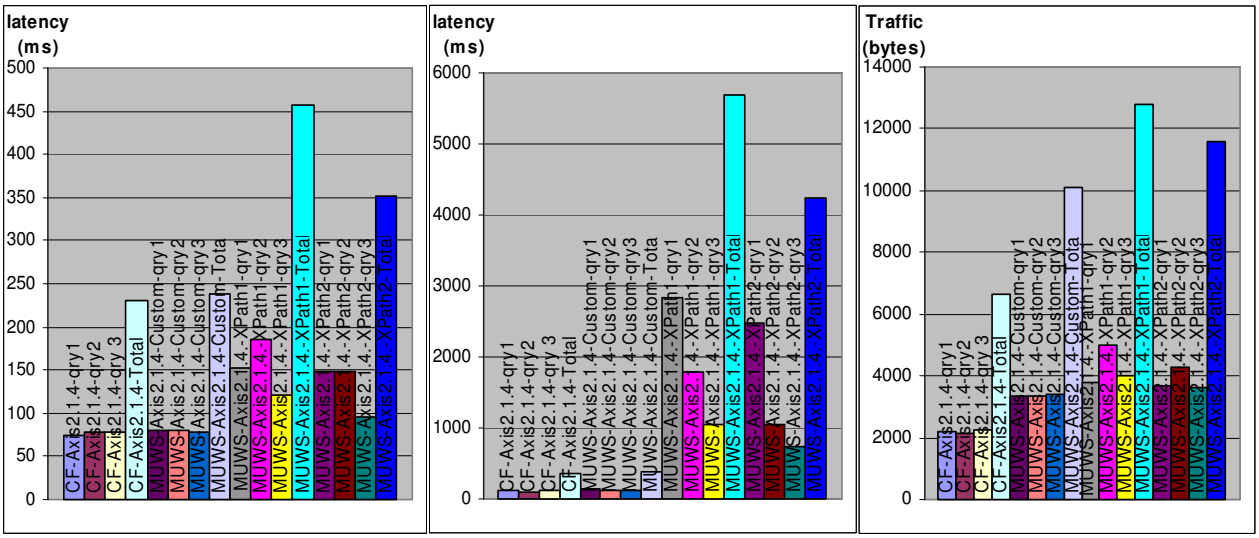

Fig. 5. Latency and traffic overhead for a small and a big QoS network for scenario 1

MUWS when using XPath 1.0 or 2.0 would not be a scalable solution for monitoring and even for event reporting.

In the second scenario the manager would like to query an agent to determine to which traffic class (Per Hop Behavior-PHB) an LSP in the ingress router of an MPLS network is assigned. For this scenario we will test the performance of MUWS against the CF built in Apache AXIS 2.1.4 & 1.1.4 and the SNMP GetNext and GetBulk operations for a varied number of LSPs (from 1 to 980 objects). Please note that SNMP cannot process data at the agent and thus the SNMP manager fetches all the required data so that it can process them. In the measurements the processing overhead at the SNMP manager is not included.

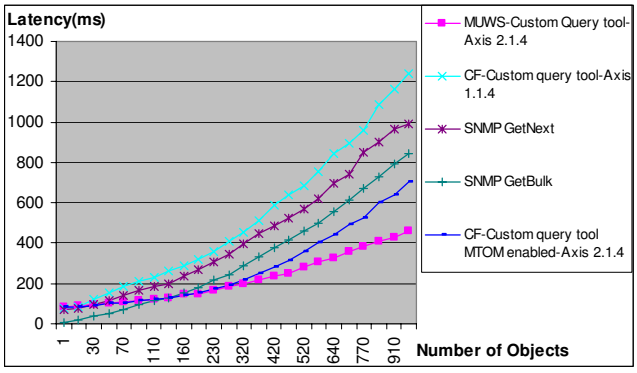

Fig. 6. Scenario two latency measurements

During these measurements we noticed that when the size of the AXIOM element returned after a query to the agent for AXIS 2.1.4 and the CF exceeds 4 KBs an invocation exception occurs. We managed to be able to return responses greater than 4KBs only when we enabled the MTOM mechanism of AXIS which "binarizes" the AXIOM response element. Though we can not compare MUWS and the CF on the same terms this is a good opportunity to check the performance of the new binarization scheme of AXIS 2, since techniques like the SOAP with Attachments (SwA) API or embedding data in xs:base64Binary or xs:hexBinary encodings suffered from interoperability and performance issues respectively (the latter schemes increased message size by a factor of 1.33x to 2x [23], [24]) (AXIOM=up to 25% smaller size for normal character encodings and faster processing without the overhead of encoding and decoding base64 data [21]). Additionally we noticed that the MTOM feature is not supported by Muse probably because the latter uses DOM node elements to form a response serialized in an XML Document (not AXIOM elements).

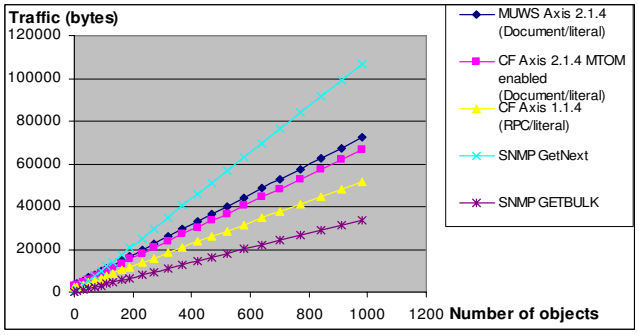

Fig. 7. Scenario two traffic overhead measurements

As it can be observed in Fig. 6, the latency performance of MUWS using Axis 2.1.4 against the custom framework using Axis 1.1.4 has been decreased by 3 times when the number of objects retrieved reaches 980. It is even worthy of attention that MUWS latency performance exceeds the performance of the SNMP GetBulk operation when more than 130 objects are retrieved not even considering the processing overhead for SNMP at the manager. Compared to GetNext, MUWS performs better in terms of latency when more than 50 objects are retrieved. From Fig.6 we can also observe that the binarization scheme of Axis 2.1.4 performs quite well. When less than 160 objects are retrieved, the CF with MTOM enabled performs a bit better than Muse but when this number is exceeded the fact that data are exchanged using Mutlipart MIME binary encodings increases latency for the CF, though its performance is still better than SNMP (130 to 980 objects).

In terms of traffic overhead MUWS and the CF either with Axis 1.1.4 and 2.1.4 performs better than SNMP GetNext operations (after more than 45 or 60 objects are retrieved for CF and MUWS respectively) and worse than SNMP's

GetBulk operations. Additionally we can observe that MUWS is more verbose than the CF.

I. CONCLUSIONS

OASIS and DMTF are two groups in the process of standardizing the WS operations on manageable resources for the purpose of increasing the interoperability of WS management applications. Collaboration between the two working groups has led them to issue a roadmap in order to converge their frameworks so as to provide the basis to build interoperable management applications in an end to end fashion. Efficiency though as shown in [6] is an issue that these groups have to solve. A great aspect of the specifications of these two groups though, is that they have left undefined all the implementation details of their frameworks standardizing only a few common aspects of the capabilities of managed devices and the operations to access these capabilities. This is a key requirement for open management which can lead to efficient implementations of management applications.

One such application was a custom query tool built by us to support distributed monitoring as part of an architecture of management agents on network devices. The tool also exploits the relationships between management data so as to provide bulk and selective retrieval capabilities through information processing over the state data of managed devices.

In this paper we have shown how to integrate our query tool and distributed monitoring architecture with MUWS by satisfying a number of requirements. We have also evolved the concept of navigation of relationships between WS. As such we have shown how to structure hierarchies of WS-Resources for effective monitoring by introducing 3 rules and we have also elaborated on the relationships that exist between state data that can be exploited for monitoring. Additionally in this paper we have also presented a full monitoring example where navigation of relationships between WS-Resources takes place and where information processing is required for bulk and selective retrieval of RPs. To show the potential of MUWS for efficient and scalable monitoring we have shown how to integrate our query tool with Apache MUSE, an implementation of MUWS. After integrating our query tool we have tested the performance of MUWS based on two monitoring scenarios that require use of our query tool against XPath 1 and 2 and against a custom framework we have built for monitoring and SNMP.

In the measurements we have shown that MUWS is more scalable and efficient for monitoring when using our query tool against XPath 1 which is currently supported by the Apache Muse software and also more scalable than XPath 2 (both in terms of traffic and latency overhead). Based on the Apache Axis 2.1.4 engine we have also shown that MUWS can be quite scalable in terms of latency compared to SNMP even when not considering the information processing operations that have to be performed by the latter for the scenarios we examined. In terms of traffic overhead MUWS starts producing less traffic than SNMP's GetNext traffic when more than 60 objects are retrieved but always more traffic than GetBulk. Nevertheless MUWS can still be a scalable solution for network monitoring in terms of traffic and latency overhead when using our custom query tool.

On the other hand we must recognize that XPath and standardized solutions also have great benefits. XPath implementations have more functionality, they enable the user to be more expressive when processing management information, and they are well known tools that many are familiar with. Learning a new tool even with a simpler syntax such as our query tool might not be desirable. As such it is probably more desirable to use XPath implementations with MUWS at the edges of a network domain especially for interoperability while using custom solutions within a network domain. The use of such an approach is recognized in [20] for event reporting as more scalable but it can also be applicable as we have shown in this paper for polling based monitoring.

REFERENCES

[1] W. Vambenepe, "WS Distributed Management: Management Using Web Services (MUWS 1.1) Part 1&2", OASIS standards, August 2006.

[2] *H. Kreger et al,* "WS Distributed Management: Management of Web Services 1.1" OASIS, 01 August 2006.

[3] W. Vogels, "Web Services are not Distributed Objects: Common Misconceptions about the Fundamentals of Web Service Technology", IEEE Internet Computing, December 2003.

[4] I. Foster et al, "Modeling stateful Resources with Web Services", IBM library white paper, May 2004.

[5] K. Czajkowski et al "The Web Services Resource Framework v1.0" OASIS standard version 1.0, May 2004.

[6] G. Moura, G. Sanchez, R. Gaspary "On the Performance of WS Management Standards-An Evaluation of MUWS and WS-Management for Network Management" IM 2007, May 2007, pp. 459-468.

[7] A. Chourmouziadis, G. Pavlou, "Efficient Information Retrieval in Network Management Using Web Services", DSOM 2006, October 2006.

[8] S. Graham, J. Treadwell, "The WS-Resource", OASIS standard ver. 1.2.

[9] S. Graham, J. Treadwell, "The WS-ResourceProperties" OASIS standard v.1.2, April 2006.

[10] L. Srinivasan, T. Banks, "The WS-ResourceLifeTime" OASIS standard v 1.2, April 2006.

[11] T. Maguire, T. Banks, et al, "The WS-ServiceGroup" OASIS standard v. 1.2, Apr. 2006.

[12] L. Liu, S. Meder, "The WS-BaseFault", OASIS standard v.1.2.

[13] S. Graham et al, "Web Services Base Notification".

[14] K. Klein, J. Cohen et al "Towards converging Web service standards for resources, events, and management" white paper March 2006.

[15] A.Chourmouziadis G.Pavlou, "Web Services Monitoring: An Initial Case Study on the Tools Perspective", poster, NOMS 2008.

[16] A.Chourmouziadis, G. Pavlou, "An Evaluation of WS Tools to Address Efficient Information Retrieval for Network Management", to appear in the International Symposium on Computer Networks (ISCN 2008).

[17] G. Pavlou, A. Liotta, et al "CMIS/P++: extensions to CMIS/P for increased expressiveness and efficiency in the manipulation of management information", INFOCOM 98, April 1998.

[18] C. Srinivasan, et al, "*MPLS LSR* MIB*"* RFC 3813, June 2004.

[19] C. Srinivasan, et al, "*MPLS TE* MIB*"* RFC 3812 June 2004.

[20] H. Saiedian, S. Mulkey "Performance Evaluation of Eventing WS in Real-Time Applications" IEEE magazine, Vol 46, p.p. 106-111.

[21] D**.** Sosnoski**, "**Java Web services, Part 2: Digging into Axis2: AXIOM**",** IBM article, Nov 2006

[22] D. Jemiolo, "Craft Custom Query Dialects with Apache Muse", IBM article, June 2007.

[23] Y. Yang, "Faster Data Transport Means Faster Web Services with MTOM/XOP", DevX article, http://www.devx.com/xml/Article/34797

[24] A. Bosworth, D. Box, et al,"XML, SOAP and Binary Data", white paper from BEA and Microsoft, February 2003.

[25] J. Schonwalder, A. Muller, "Reverse Engineering Internet MIBs", International Symposium of Integrated Network Management, May 2001