PerfSONAR: A Service Oriented Architecture for Multi-domain Network Monitoring

Andreas Hanemann[1], Jeff W. Boote[2], Eric L. Boyd[2], Jérôme Durand[3], Loukik Kudarimoti[4], Roman Łapacz[5], D. Martin Swany[6], Szymon Trocha[5], and Jason Zurawski[6]

[1] German Research Network (DFN), c/o Leibniz Supercomputing Center, Barer Str. 21, D-80333 Munich, Germany
hanemann@lrz.de
[2] Internet2, 1000 Oakbrook Drive, Suite 300, Ann Arbor, MI 48104, USA
{boote,eboyd}@internet2.edu
[3] GIP Renater, 151 Boulevard de l' Hôpital, 75013 Paris, France
Jerome.durand@renater.fr
[4] DANTE, 126-130 Hills Road, Cambridge CB2 1PG, United Kingdom
loukik.kudarimoti@dante.org.uk
[5] Poznan Supercomputing and Networking Center, Noskowskiego 12/14, 61-704 Poznan, Poland
{romradz,szymon.trocha}@man.poznan.pl
[6] Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA
swany@cis.udel.edu,zurawski@eecis.udel.edu

Abstract. In the area of network monitoring a lot of tools are already available to measure a variety of metrics. However, these tools are often limited to a single administrative domain so that no established methodology for the monitoring of network connections spanning over multiple domains currently exists. In addition, these tools only monitor the network from a technical point of view without providing meaningful network performance indicators for different user groups. These indicators should be derived from the measured basic metrics.

In this paper a Service Oriented Architecture is presented which is able to perform multi-domain measurements without being limited to specific kinds of metrics. A Service Oriented Architecture has been chosen as it allows for increased flexibility and scalability in comparison to traditional software engineering techniques. The resulting measurement framework will be applied for measurements in the European Research Network (GÉANT) and connected National Research and Education Networks in Europe as well as in the United States.

1 Introduction

The administrators of a network domain can currently make use of a lot of available tools to monitor a variety of metrics. However, the situation gets much more complicated if information about the performance of a connection involving different administrative domains is requested. Besides of examples like transnational videoconferences

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 241–254, 2005.

the monitoring of multi-domain connections is especially interesting for Grid projects which are currently being deployed across Europe and elsewhere.

The subproject *Joint Research Activity 1* of the GN2 project [4] aims at providing a framework for performing multi-domain measurements in the European Research Network (GÉANT) and the connected National Research and Education Networks (NRENs). It is carried out in close cooperation with Internet2's End-to-End piPEs [6] initiative and will result in a common system called *PerfSONAR* (Performance focused Service Oriented Network monitoring ARchitecture). The name reflects the choice of a Service Oriented Architecture for the system implementation.

A survey performed in the requirement phase of the project showed the diversity of measurements that are currently applied in different networks together with the demand for a common framework. The survey has also been useful to derive a list of metrics being of common interest. These metrics can be retrieved by active and passive measurement methodologies as well as by requesting SNMP variables. Metrics of primary interest for the users include round-trip time and one-way delay as well as its variation (aka jitter), round-trip and one-way packet loss ratio, bandwidth utilization, IP available bandwidth, and interface errors/drops. For each metric a differentiation can be made between IPv4 and IPv6, unicast and multicast, different classes of service as well as between different time-scales ranging from short-term for performance debugging to long-term for trend observation. The metric definition in the project is done in accordance with current recommendations (such as those from IETF IPPM (IP Performance Metrics) and IPFIX (IP Flow Information Export)). Currently, most public networks provide information about the core network topology, link bandwidth, and current utilization data (sometimes with some limitations), while information about other metrics is hardly provided.

The rest of the paper is organized as follows. In Section 2 requirements from different user groups concerning the networking monitoring functionality are motivated. Related Work is examined in Section 3 which includes the examination of previous projects and different software architectures interesting for the framework implementation. Our Service Oriented Architecture is presented in Section 4, while the current status of its prototypical implementation is subject to Section 5. Conclusion and future work can be found in the last section.

2 Requirements

The multi-domain monitoring framework that is addressed in the project should be able to fulfill the requirements of different user groups.

NOC/PERT: For the Network Operation Center (NOC) or people from the Performance Emergency Response Team (PERT) the framework shall provide a multi-domain perspective on the networks and juxtapose a variety of metrics. This is needed to link several information sources in order to allow for a better understanding of performance degradations in the network.

Network Managers: The framework should also allow for different kinds of policies with regard to user groups and metrics. An example policy could be that the access to

interface packet drops is only allowed for NOC and PERT staff. Therefore, it should be easy for the network managers to apply their network policies for information filtering to the framework.

Projects: A dedicated view is required for projects spanning over multiple administrative domains to show to each partner the performance of the underlying backbone network. This information complements the end-to-end view of the projects gathered by their own systems (e.g. DataGrid WP7 software [5]) to gain a better understanding of the network behavior and its impact for application tuning. Furthermore, special kinds of metrics have to be offered for the project links, e.g. an aggregated metric showing the performance of the connections to important partners.

End Users: For end users the framework shall provide a view of the backbone networks which allows to easily track whether a problem is located in the backbone or is supposed to be present in the user's local area network. This is a major improvement in comparison to the current situation (see Figure 1). Today, a user who experiencies a network performance problem is often able to get information about the local area network, but it is hardly possible to get timely information about the performance of the national research backbone and of Geant. The use of simple test tools like ping

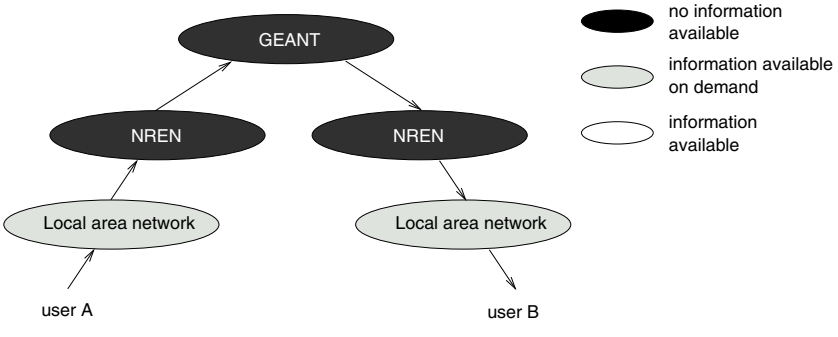

Fig. 1. Network transparency today

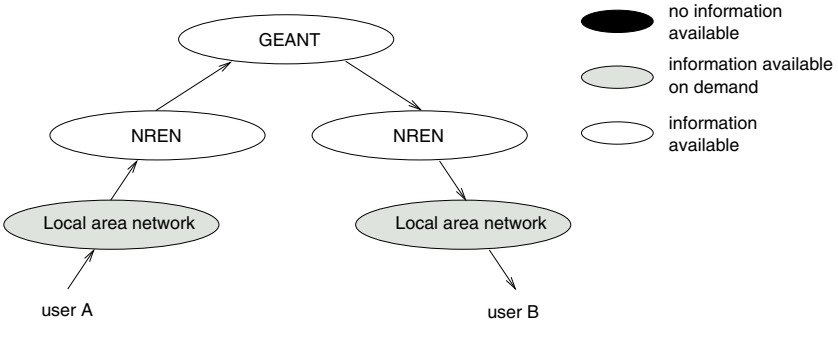

Fig. 2. Network transparency after PerfSONAR deployment

and traceroute cannot be regarded as satisfactory. The aim in the project is to provide an edge-to-edge view of the research backbones facilitating the easy identification of problems in the networks for the user (see Figure 2).

3 Related Work

This section deals with monitoring projects related to the purpose of JRA1. In addition, the different possibilities for the realization of the PerfSONAR framework using the Service Oriented Architecture paradigm are evaluated.

3.1 Monitoring Projects

One of the issues of the EU-funded INTERMON [10] project was inter-domain QoS monitoring. The modeling in the project is based on abstractions for traffic, topology, and QoS parameter patterns. This allows to run simulations for network configuration planning. To fulfill these goals, the project has based its entire design on a huge centralized and complex database for topology, flow, and test information, collecting all network data in a single location. However, such model is not acceptable in a multi-domain environment. It is not conceivable that an entity supersedes the others and has a complete control of other networks. Also, while INTERMON centralizes the collection of pre-defined measurements, JRA1 provides an architecture where any entity can, based on authentication and authorization rules, schedule new types of measurements and run tests over the multi-domain network. JRA1 has a more focused goal and has real production constraints from NOCs, requesting data for day-to-day operation of the networks. There are also many constraints for allowing distributed policies among the different networks, for exchange of monitoring data and access to on-demand test tools.

The MonALISA project [13] has also provided a framework for distributed monitoring. It consists of distributed servers which handle the metric monitoring for each configured host at their site and for WAN links to other MonALISA sites. JRA1 shares the idea of servers acting as a dynamic service system and providing the functionality to be discovered and used by any other services or clients that require such information. Even though MonALISA has similar concepts to our approach, JRA1 details its application to multi-domain environments with mechanisms for measurements spanning independently managed domains, especially with respect to metrics concatenation and aggregation. The MonALISA system relies on JINI (see Section 3.2) for its discovery service.

The PlanetLab [15] initiative is also related to our work. It is a huge distributed platform over currently 568 nodes, located in 271 different sites. It enables people being members of the consortium to access the platform (or a part of it) to run networking experiments. Most of the projects which run over the PlanetLab infrastructure deal with network monitoring and management in general. Those tests aim at properly designing services at a large scale. The architecture is similar to PerfSONAR - resources are made available through designed architecture services. In PlanetLab a node manager (i.e. access interface for each node) has been proposed which just allocates local resources based on a policy enforced by the infrastructure service. Even though analogies

between the components defined in the two projects can be identified, the PlanetLab infrastructure service is centralized and relies on a single database. Similar to INTERMON it can therefore not be applied as is to a multi-domain environment.

The Enabling Grids for E-SciencE (EGEE) project, which has been launched in April 2004 and is funded by the European Commission, continues the work of the DataGrid project [5]. The project aims at network service deployment for the Grid community. This includes the development of network interfaces and the use of network services like performance measurements and advance reservations. The task of its subactivity JRA4 [7] is to retrieve network measurements from a set of domains by using Web Services. The measurement services being accessed can be divided into end-to-end measurements (host-to-host) and backbone measurements using dedicated monitoring equipment.

EGEE JRA4 has in total designed and developed three prototypes which communicate with both kinds of measurement services. The third protoype communicates with PerfSONAR instances located in different domains to retrieve backbone measurements for capacity, bandwidth utilization, and available bandwidth. It decides on the correct service end point to contact by using a discovery functionality which is currently statically configured. The transportation layer of EGEE being implemented as a Web Service translates between different versions of the GGF NMWG schema [14].

3.2 Implementation Options

In JRA1 it has been decided to realize the network monitoring framework by adopting the Service Oriented Architecture (SOA) paradigm. This new paradigm in software engineering proposes to use independent pieces of software (called "services") which can be orchestrated to collaborate in order to reach a common goal.

A SOA has several advantages in comparison to traditional software architectures. A large task can be split into independent services which helps to avoid monolithic software blocks being difficult to maintain. At runtime, the services can be dynamically added/dropped which results in an increased flexibility and robustness. Furthermore, different implementations of a SOA design do not have to realize all services if only a part of the functionality is needed.

The most common technology for the realization of a SOA are Web Services. From the evolving Web Service standards WS-Notification is of particular interest for the project and is going to be adapted in multiple scenarios. It deals with the communication between services using publish/subscribe mechanisms and therein supports two data flow models: Direct notifications (WS-BaseNotification) from service to client, where the service itself maintains a list of interested clients, and brokered notifications (WS-BrokeredNotification), where a client can act as a broker and can have several clients of his own. The specification contains standardization of message exchanges and XML schema specifications.

In JRA1 several options have been considered for the implementation of the framework using a SOA. Besides of Web Services are also other possibilities as described in the following.

JINI. One possibility to implement a distributed Service Oriented Architecture is to apply JINI [11] which is provided by Sun Microsystems under an open source licence.

It is a set of Java APIs and runtime conventions that facilitate the building and deploying of distributed systems. JINI itself is fully implemented in Java and uses its integral mechanisms such as remote method invocation (RMI).

Applications using JINI are treated as a set of cooperating distributed services. There is one specialized service called Lookup Service where other services can register and client applications can fetch information about required services.

The default communication between services is done in RMI offering remote procedure calls, but also any other protocol offering message passing is allowed (one can choose UDP or TCP on transport level and any protocol on application level). The discovery of the Lookup Service depends on the existence of multicast in the network. Without multicast the address(es) of Lookup Service(s) need to be well-known.

Even though some mechanisms like the lookup procedure are interesting for the project, it has been decided not to use JINI because of its tight Java coupling and limitations like the necessity to use multicast.

JXTA. Sun Microsystems also promotes the open source peer-to-peer software architecture JXTA [12] which can be regarded as complementary to JINI. The basic entities defined by JXTA are peers which can interact by using defined protocols. These protocols deal with peer discovery, peer information exchange, peer-to-peer routing, and the establishing of communication channels (called pipes).

A variety of different types of low-level message transport protocols, such as HTTP, TCP/IP, and TLS (Transport Layer Security) can be used in the current reference implementation of JXTA. One of the JXTA protocols enables the communication between peers connected to different low-level network types.

There is a reference implementation of the architecture in Java, but other implementations in C, Ruby, Phyton, or Perl are also available. Like in other P2P architectures peers can be added/dropped at any time. To allow for a service provisioning change using other resources, no physical network addresses are used, but a JXTA addressing scheme is applied.

JXTA is supposed to be suitable for peer-to-peer applications, even though only few information about successful JXTA-related projects can be found best to our knowledge. The measurement framework in JRA1 cannot be classified as a true peer-to-peer scenario as we have different classes of service in our framework (see Section 4). Therefore, JXTA would have to be combined with JINI or Web Services.

Apache Axis. For the implementation of Web Services Apache Axis [1] which is a popular Simple Object Access Protocol (SOAP, [18]) implementation has been examined. Axis is mainly a Java platform for creating and deploying Web Services. A C/C++ version of Axis is available as well. The Axis package also provides an application called SOAPMonitor to view and debug SOAP communications received and replied to by deployed Web Services. A web-based interface for viewing and managing services is also part of the package. Application servers such as Tomcat [19] are required to use the Java version of Apache Axis. The C/C++ version needs a Web Server (such as Apache) and uses Xerces C++ [20] to parse SOAP. The examination of Axis in the project has focused on the Java version.

Deploying services can be done in many ways with all of them requiring the use of Web Service Deployment Descriptors (WSDD). It can work with or without server side stubs and can also create service descriptions in Web Service Description Language (WSDL) at run time. It is also possible to have clients without stubs thus making it easier to have dynamic invocations.

In the Axis case, a client connects to the web (application) server and feeds a SOAP message, which conforms to the service WSDL definition. Axis converts the SOAP message to proper method calls of the classes that implement the service (business logic). Axis is therefore called a *SOAP proxy*. A Document Type/Literal Style of Web Service, which uses an XML document as input in the request, can be deployed by using Axis. Information required by the service to handle and satisfy the request (including the operation name) are "derived" from this XML Document.

The use of SOAP, WSDLs and the possibility to use Document Type/Literal Style of Web Services makes it easy to deploy services without worrying about how the clients are built (using Java/C++/Perl/Python/etc.). The clients only need to build acceptable XML documents (conformant to a defined schema) and use the standard SOAP protocol in order to make use of the service. The operating system or the platform used on the client side bears no importance or effect on the entire process. Consequently, the use of Web Services (and hence Axis, which provides an "easy-to-use" Web Service (SOAP) implementation) has an edge over its competitors.

Globus Toolkit. The Globus Toolkit 4.0 [9] has evolved from a Web Services-based reimplementation of a software suite from the Grid community. It is based on Axis, but has build several modules on top of it which can be used independently. Its modules include implementations of the Grid Resource Allocation Manager (WS-GRAM), Replica Location Service, Monitoring and Discovery Service (MDS), etc.

It intends to implement a stable reference implementation of the Web Services Resource Framework (WSRF, refactoring of the Open Grid Services Infrastructure specification, OGSI) in Java, C, and Python. The management of stateful resources provided by WSRF is interesting, as stateful resources like links are encountered in the JRA1 context.

The Globus Toolkit is much more difficult to deploy than Axis. Even though some modules are application agnostic so that they are supposed to be suitable for network monitoring, it needs to be examined whether it makes sense to incorporate some Globus Toolkit modules in the framework or to implement some more lightweight functionalities for the JRA1 purpose on top of Axis directly. A sophisticated module of Globus is the implementation of the Grid Security Infrastructure which is hard to deploy due to the needed certificates which have to be provided and managed.

Summary. In the project it has been chosen to implement the monitoring framework by using Web Services. Web Services provide a lot of flexibility in the client programming as these only have to be conformant to an XML description, i.e. there are no dependencies from operating systems or programming languages. The communication can happen via the easy-to-use SOAP. Our examination of Apache Axis showed that it is a mature open source tool for implementation. Some modules of the Globus Toolkit could be adopted in later stages of the project.

4 Multi-domain Monitoring Framework and Service Oriented Architecture

The monitoring framework which is designed by JRA1 as well as JRA1's PerfSONAR system being applied for the middle layer of the framework are outlined in this section.

4.1 Monitoring Framework

The general monitoring framework which is explained in detail in the following is depicted in Fig. 3.

The Measurement Points are the lowest layer in the system and are responsible for measuring and storing network characteristics as well as for providing basic network information. The measurements can be carried out by active or passive monitoring techniques. The Measurement Point Layer of a domain consists of different monitoring components or agents deployed within the domain. A monitoring agent provides information on a specific metric (e.g., one-way delay, jitter, loss, available bandwidth) by accessing the corresponding Measurement Points. Each network domain can, in principle, deploy Measurement Points of its choice.

The Service Layer is the middle layer of the system and consists of administrative domains. It allows for the exchange of measurement data and management information between domains. In each domain, a set of entities (services) is responsible for the domain control. Each of them is in charge of a specific functionality, like authentication and authorization, discovery of the other entities providing specific functionalities, resource management, or measurement of network traffic parameters. The interaction of the entities inside a domain as well as the access to the Measurement Point Layer or other domains may not be visible to the end user. Some of the entities contain an interface which can be accessed by the User Interface Layer.

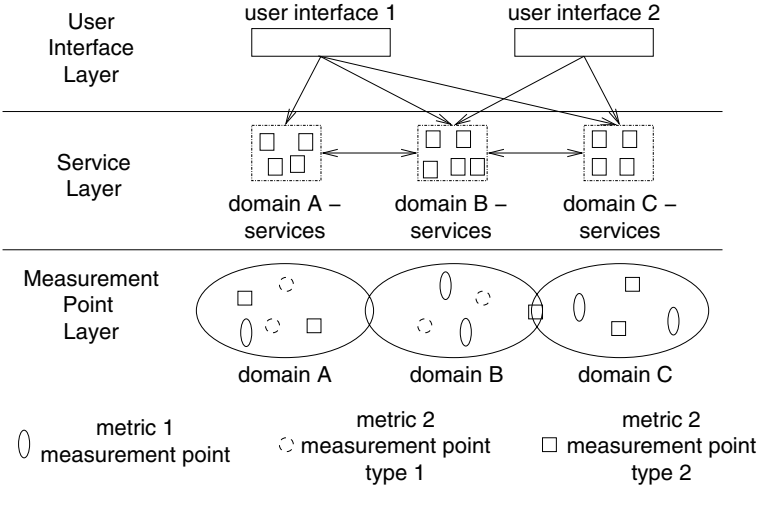

Fig. 3. JRA1 architecture proposal

The User Interface Layer consists of visualization tools (user interfaces) which adapt the presentation of performance data to be appropriate for the needs of specific user groups. In addition, they may allow users to perform tests using the lower layers of the framework. From the user interface perspective, the Service Layer provides an additional level of abstraction to hide the differences between Measurement Points deployed in the different domains.

The design aim is to provide the main functionalities in the Service Layer as independent entities to allow for an increased flexibility of the system: existing elements may be easily replaced or new ones inserted. Even if the number of entities is large each one can be identified and invoked using discovery functionalities.

4.2 Service Oriented Architecture

There are three general categories of performance measurement data, i.e., active and passive measurement results as well as network state variables (SNMP variables) that can be thought of as data producers and are provided by the Measurement Point Layer. From the user or network administrator point of view, analysis tools, threshold alarms, and visualization graphs can be thought of as data consumers which are contained in the User Interface Layer. Between data producers and data consumers is a pipeline of aggregators, correlators, filters, and buffers, which can be regarded as data transformers and data archives. Data producers, consumers, transformers, and archives are all resources that need to be discovered and (possibly) protected from over-consumption using authentication and authorization.

A services-based measurement framework implements each of these roles as an independent service: Lookup (LS), Authentication (AS), Measurement Archive (MAS), Transformation (TS), and Resource Protector (RPS). These services form the Service Layer. The Measurement Point Layer is also regarded as it contains Measurement Point Services (MPS).

Users of any service, whether they are end user applications or other services, are specified as clients. Providers of any service are denoted as servers. Therefore, many services can be both client and server, depending upon the context. To achieve this, all data providers implement a publisher interface and all data consumers implement a subscriber interface. When a data flow is requested, the consumer provides a handle to a subscription interface if it wants a push interaction. If it does not provide a subscription handle, the data producer creates a publisher interface that the consumer can poll.

The service interactions are depicted in Fig. 4 and are referenced at the end of the description of each service. In [2] use case examples for the application of the framework can be found.

Measurement Point Service (MPS). The Measurement Point service creates and publishes measurement data by initiating active measurements or querying passive measurement devices [3]. A common interface to these capabilities is required for ease of integration into the monitoring system as a whole. MPSs use a measurement setup protocol to allow the user to request measurements to be made for a specified set of parameters and then publish the results of these measurements to one or more sub-

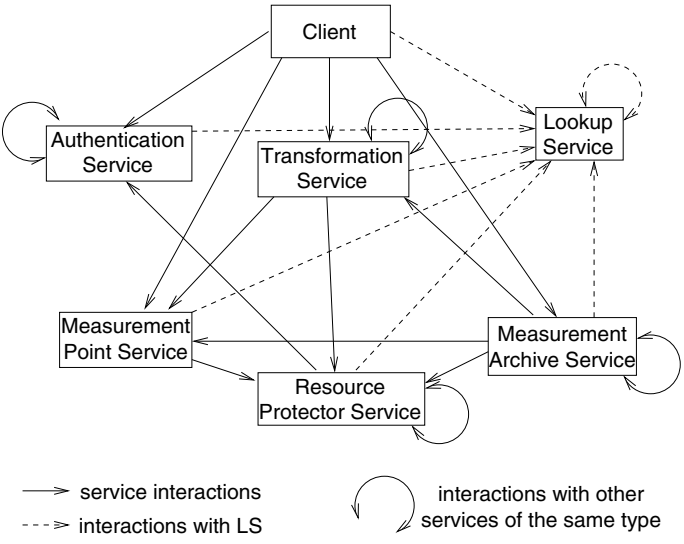

Fig. 4. Interactions of Services

scriber interfaces. Legacy capabilities (e.g., existing active measurement tools, Netflow, SNMP) can be wrapped within an MPS.

When acting as a server, the MPS accepts measurement requests and uses a push method for data publishing. In this case the client has to provide in advance one or more subscriber handles to send the results directly to it. It is also possible to send data indirectly via a TS (see below). When acting as a client, the MPS registers its own presence with an LS and publishes measurement data to subscribers. The MPS may send resource availability and authorization requests to the Resource protector service.

Lookup Service (LS). Services register their existence and capabilities, subject to locally-determined policies and limits, with a Lookup Service. The registration is performed using a join protocol. A service may register for a limited period of time or leave without disrupting the interaction of other services. Clients discover needed services by querying an LS using the lookup protocol. The first LS is found by one of several approaches, including multicast, well-known servers, or internal configuration. Once an LS is found, additional LSs are identified by querying the first one. LSs register themselves with other LSs and are organized using peer-to-peer distribution techniques.

The lookup protocol of the service network defines the kinds of queries a client can make when looking for a resource. The LS is not a simple name-based directory service. Queries about the services are based upon attributes such as service type, required authentication attributes, and service capabilities, as well as more complex constructs, such as network location or community affiliations. It is expected that the information in the LS will be much more expansive than a typical UDDI directory, although this could be one way to implement the LS information store. It is likely a more generic store will be put in place to allow for more targeted service specific information to be registered. This could be implemented using a direct XML database with XQuery capabilities.

When the service acts as a server the LS accepts requests for service-related information, registration and deregistration requests (including advertisements from other LSs announcing their existence), and keep-alive requests. When acting as a client, the LS registers its own presence to other LSs. The service can also work in peer-to-peer networks where an LS shares directory information with other LSs.

Measurement Archive Service (MAS). The Measurement Archive service stores measurement data in database(s) optimized for the corresponding data type and publishes measurement data produced by MPSs and/or TSs. In addition to providing a historical record for analysis, the MAS serves to reduce queries to the MPS by effectively offloading the publication to multiple clients. MAS makes use of a set of protocols: storage setup protocol which is used to setup the MAS to accept and store measurement data from a publisher, e.g. MPS and measurement data retrieval protocol to get measurement data from MAS by the client.

In case the MAS is perceived as a server, it accepts and stores setup requests as well as publication requests. The publication request includes a subscription handle and the results are sent directly to the client (or indirectly via a TS). As a client, the MAS registers its own presence with an LS, subscribes to an MPS, other MAS, or TS, and publishes measurement data to subscribers. The MAS may send resource availability and authorization requests to the RPS.

Authentication Service (AS). The Authentication service provides the authentication functionality for the framework as well as an attribute authority. The AS supports clients with multiple identities, including individual identities that represent different roles at different times. Role-based authentication using attribute assertion style authorization protects the privacy of the user [17]. This typically means a handle is created to provide additional information about the attributes of that user and that resources can use that handle to make queries about the user subject to the privacy policy. Communities of multiple administrative domains that accept each other's authentication can be formed by federating ASs. Federation details are held solely in the AS and are hidden from other services within a given administrative domain. In other words, the *trust* relationship within a domain is between the domain's services and the local AS domain, while the *trust* relationship between any two federated domains is managed by the ASs.

When acting as a server, the AS accepts authentication requests and attribute requests via its interfaces. As a client, it registers its own presence with an LS and may query other ASs for attributes of a federated identity.

Transformation Service (TS). The Transformation service performs a function (e.g., aggregation, correlation, filtering, or translation) upon measurement data. The TS subscribes to one or more servers and publishes to one or more clients, making it a key component of a data pipeline within the measurement framework. For example, a TS might compress datasets from more recent, high-resolution data to less recent, low-resolution data and publish that data to an MAS. A TS also might read from multiple data publishers to create a specific correlation. A very simplistic data analysis example would be a threshold detection operation that then pushed data out for the purposes of

triggering a Network Operations Center alarm. For the Alarm Notification Service (a type of TS) the WS-Notification standard is going to be applied for distributing alarms.

When considering the TS as a server, it accepts publication requests. If the request includes a subscription handle, the results are sent directly. If no subscription handle is included, the TS returns a publisher handle to the client which is then responsible for initiating dataflow. When TS acts as a client, it registers its own presence with an LS, subscribes to one or more MPs, MAs, or TSs, and publishes measurement data to subscribers. The TS may send resource availability and authorization requests to the RPS.

Topology Service (ToS). The Topology service is a specific example of a TS used to make topological information about the network available to the framework. It collects topological information from a variety of sources (i.e. multiple MPSs) and uses algorithms to derive the network topology. The ToS also reflects multiple network layers. That is, the topology can be described on the domain level through network elements, but also by wavelengths representing the physical level. Understanding the network topology is necessary for the measurement system to optimize its operation. For example, the LS relies on the ToS to determine MPS that are "closest to" interesting network locations (e.g. routers). Thus, in the same way that a host may ask for an MPS instance that has a particular set of properties, a service component can also request information about node proximity. Additionally, the Topology service may be used for overviews/maps that illustrate the network with relevant measurement data.

Resource Protector Service (RPS). The Resource Protector service is used to arbitrate the consumption of limited resources, such as network bandwidth. This service is distinct from the individual MPSs to allow the consumption of resources that are common across multiple types of MPSs to be tracked in a single place. (For example, a one-way latency test would be adversely effected by a throughput test going over the same network interface.) The RPS also has a scheduling component to deal with the consumption of time-dependent resources. When measurement activities are involved, resources may be related to the measurement infrastructure or real network resources. The RPS can allocate portions of a resource based upon configuration rules and can schedule the time-dependent resources. Services that consume resources contact the associated RPSs to allocate them. Because RPSs reduce scheduling flexibility, RPSs should only be deployed to protect limited resources. In other words, some MPSs do not have to contact an RPS at all.

Authenticated requests provide a way of making attribute assertion queries back to the authenticating entity. A handle is included within the Authentication Token that is sent with the request. This makes it possible for the RPS to determine whether a particular resource requestor has the right to access a given resource without being completely aware of the identity of the requestor.

If the RPS acts as a server, it accepts authorization and resource availability requests. If it acts as a client, the RPS registers its presence with an LS. The RPS may request authorization and resource availability for other resources from other RPSs. The RPS may request additional attribute information about an authenticated identity from an AS.

5 Prototypical Implementation

This work is based upon lessons learned from many European and international initiatives and deployed measurement frameworks, including DANTE's perfmonit project [4] and Internet2's piPEs project [6]. The work is also carried out with respect to efforts of the GGF Network Measurement Working Group [14] to develop schemas for interoperable measurement frameworks.

A prototypical implementation is currently carried out to realize the model as Web Services aiming at the retrieval of link utilization data from several networks. The focus of this implementation is to validate the framework design and the service interactions.

Simplified versions of the services are applied to reduce the complexity of the architecture at the first stage. The number of services and their complexity will increase over time by adding additional modules, features, and measurement types. The initial service is the Lookup Service which is needed to locate other services (in this case MAS and MPS). The crucial portion of the prototype system is the MAS, which is initially a wrapper around Round Robin Databases (RRD) [16] and provides link utilization statistics. in the beginning, several MASs have been deployed in multiple domains, making use of different RRD collections already performed in these networks and providing a picture of a few research networks' utilization both from Europe (e.g. GÉANT) and North America (e.g. ESnet [8]). As mentioned before, the prototypical implementation is already used by EGEE [7] to retrieve link utilization data for enabling its own Grid network monitoring.

Two other phases are targeted in the prototype. The first extension will be to add auto-registration capabilities to the LS, so that any service coming into live could register its capabilities and will automatically be known by the LS. It is also intended to add new measurement capabilities like packet loss and interface errors to the MAS. It is considered to replace user scripts with intuitive graphical interface for test setup, data retrieval and presentation.

6 Conclusion and Future Work

In this paper a motivation has been given for the necessity of a multi-domain network monitoring framework. Due to the deficits of existing frameworks with respect to flexibility and the disregard of organizational boundaries such a framework is subject to the JRA1 project. The examination of different implementation options has resulted in choosing a Web Services approach using Apache Axis and maybe some modules of the Globus Toolkit.

While the project primarily aims to provide a monitoring framework for the involved research networks, the open source tool development will also make it feasible to apply the framework to other multi-domain network monitoring scenarios.

Acknowledgments

The authors wish to thank Nicolas Simar (DANTE) and Thanassis Liakopoulos (GRNET) for the collaboration in the project and their valuable comments on previous versions of the paper.

References

1. Apache Axis. http://ws.apache.org/axis.
2. J. Boote, E. Boyd, J. Durand, A. Hanemann, L. Kudarimoti, R. Lapacz, N. Simar, and S. Trocha. Towards multi-domain monitoring for the european research networks. In *Proceedings of the Tenera Networking Conference 2005 (TNC 2005)*, Poznan, Poland, June 2005. TERENA.
3. T. Chen and L. Hu. Internet performance monitoring. *Proceedings of the IEEE*, 90(9):1592–1603, September 2002.
4. DANTE homepage including information about GÉANT, performit and GN2 projects. http://www.dante.net/.
5. Wp7 - network services, DataGrid project. http://ccwp7.in2p3.fr.
6. E2Epi performance evaluation system (piPEs), Internet2, End-to-End Performance Initiative. http://e2epi.internet2.edu/.
7. Joint Research Activity 4, Enabling Grids for E-SciencE (EGEE) project. http://egee-jra4.web.cern.ch/EGEE-JRA4/.
8. Energy Sciences Network. http://www.es.net.
9. Globus toolkit, version 4.0. http://www.globus.org/.
10. INTERMON project. http://www.intermon.org/.
11. JINI network technology, Sun Microsystems. http://www.sun.com/software/jini and http://www.jini.org.
12. JXTA, Sun Microsystems. http://jxta.org/.
13. MONitoring Agents using a Large Integrated Services Architecture (MonALISA), California Institute of Technology. http://monalisa.caltech.edu/.
14. Network measurements working group (NMWG), Global Grid Forum. http://www-didc.lbl.gov/NMWG.
15. PlanetLab project. http://www.planet-lab.org/.
16. Round robin database tool homepage. http://people.ee.ethz.ch/ oetiker/webtools/rrdtool/.
17. Specification of the general architecture, protocolas, and message formats of the shibboleth mechanism. http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-arch-protocols-06.pdf.
18. Simple Object Access Protocol, World Wide Web consortium. http://www.w3.org/-2000/xp/Group.
19. Apache Tomcat, Apache Jakarta project. http://jakarta.apache.org/tomcat/.
20. Xerces xml parser, c++ version, Apache project. http://xml.apache.org/xerces-c/.