Type:
Experiences

Title:
A XML based Policy-Driven Information Service

Abstract:
The use of XML as a vendor, implementation and operating system neutral means of describing system events and system management policies is described. The need for handling heterogeneity in emerging distributed systems by management solutions and the utility of XML in this regard is discussed. The design and architecture of a prototype implementation of a XML based policy-driven information server is described.

Key-Words:
Emerging Management Technologies and Frameworks
XML
Policy-Driven Management
Experiences
Information Service

Authors:
Ramkumar Natarajan
Software Engineering Research Center
1398, Department of Computer Sciences
Purdue University, West Lafayette,
IN 47907-1398, USA
Ph: +1-(765)-494-7812
nrk@cs.purdue.edu

Paul McKee
BT Adastral Park, Martlesham Heath
Ipswich, IP5 3RE, UK
paul.mckee@bt.com

Aditya P. Mathur
Software Engineering Research Center
1398, Department of Computer Sciences
Purdue University, West Lafayette,
IN 47907-1398, USA
apm@cs.purdue.edu

A XML based Policy-Driven Information Service

Ramkumar Natarajan
Software Engineering Research Center
1398, Department of Computer Sciences
Purdue University, West Lafayette,
IN 47907-1398, USA
nrk@cs.purdue.edu

Paul McKee
BT Adastral Park, Martlesham Heath
Ipswich, IP5 3RE, UK
paul.mckee@bt.com

Aditya P. Mathur
Software Engineering Research Center
1398, Department of Computer Sciences
Purdue University, West Lafayette,
IN 47907-1398, USA
apm@cs.purdue.edu

Abstract

The use of XML as a vendor, implementation and operating system neutral means of describing system events and system management policies is described. The need for handling heterogeneity in emerging distributed systems by management solutions and the utility of XML in this regard is discussed. The design and architecture of a prototype implementation of a XML based policy-driven information server is described.

1. Introduction and Motivation

There is a rapid growth in the number and flexibility of new services being offered on current networks. Also, the technology keeps evolving and is in a state of perpetual flux. These factors pose significant challenges for developers and maintainers of large scale distributed systems.

Developers have responded to the challenge by increasing the use of software components thus increasing the reuse of common functionality and speeding the introduction of new services. As component technology matures we expect to see the dynamic instantiation of services on users end systems or on general purpose computing nodes within the network. Active networks [1] currently under development allow network users to easily add new services. Users may supply their own application code for execution on routers within the active network increasing dramatically the range of available network functions dramatically.

Added to this is the fact that in a growing Internet economy, leveraging existing technology infrastructure and integrating it with external infrastructure is becoming critical to businesses. This leads to highly heterogeneous distributed systems. The explosion in the number and flexibility of applications, coupled with the heterogeneity of emerging distributed systems will demand flexible and novel management solutions to both system and service management.

We believe that a successful management solution in such an environment must possess the following attributes:

- Ability to devolve authority to the lowest possible level, in many cases to the clients themselves. This demands a very flexible approach to system management, where there should be no restriction on the degree of delegation allowed. This kind of delegation is required due to the sheer scale of Internet based distributed systems.
- A vendor, implementation and operating system neutral information model for resource management to facilitate interchange of management information. The model must be capable of adequately representing any entity of the system. These entities can be a network, a server, a service or any other computational or storage device. The model should also be extensible to support emerging entities in the system. Such an information model is a must given the heterogeneity and dynamic nature of emerging distributed systems.
- Ability to be tailored at run-time to meet new requirements, either in terms of providing additional functionality or adapting to environmental changes. Given the dynamic nature of evolving network systems, any management solution must be extensible and adaptable to the changing environment.
- Ability to interact and co-operate with other management solutions. Increasingly, services are provided by businesses, which in turn make use of other services provided by other businesses. A successful management solution for such an environment must be one that is capable of not only managing its own domain, but also capable of co-operating and interacting with external management solutions.

With these objectives in view, a prototype implementation of an information server was built. Details of the design and implementation of the prototype are presented in the rest of the paper, organized as follows. Section 2 discusses the generic architecture of a system where such an information service can be used. The use and advantages of XML in this scenario is discussed in Section 3. The architecture and design of the information server is described in Section 4. Section 5 analyzes related work. Future directions are elaborated in Section 6.

2. Architecture of an Information Service

An Information Server can be viewed as a generic extension of the basic unit of a publish and subscribe service. In this context, we define an Information Server as one that can receive directives from management and act on the information that it receives from system components based on these directives. The system component [2], can be a service, an object, an interface, a method or a physical entity with a corresponding representation in the system. The type of actions that an Information Server can perform on the information it receives can range from simple subscription based forwarding to sophisticated aggregation, filtering and logging of Information. Apart from this, custom extensions can be provided to take specific actions based on recognised information. From a management perspective, we have positioned the Information Server as a means of tracking and handling system events. Thus the information server in this context receives event information from the system components and acts on them as per the policies specified by management.

In a previous publication [3], the initial structure of a network of store and forward event servers that we propose to use in our heterogeneous management system has been described. Implicit in this structure is the assumption that the entire domain that is to be managed is partitioned and organized in some manner. One way of achieving this partitioning and organization is based on logical partitioning of the deployment space into zones [2].

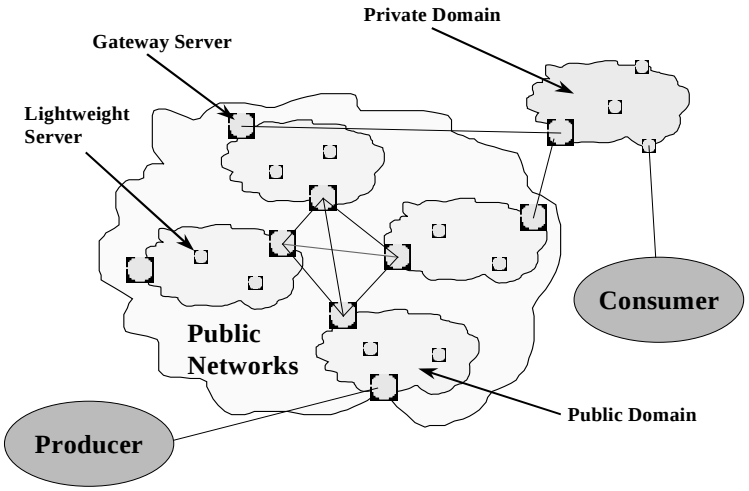

Figure 1. Architecture of an Information Service

As shown in Figure 1, our proposed architecture contains two classes of information servers, lightweight partially functional servers which can exist with any entity in the system, and fully functional servers that will typically be located at gateways between networks. The prototype implementation described in this paper is a lightweight server. The lightweight servers are intended to support load sharing of events within a single domain. Any event or policy it receives that is outside this domain will be passed to the associated gateway server. At this point in the design we place no constraint on the nature of events or policies except that they be well-formed XML documents whose structure will be described in the next section. We propose a generic event collection service that may be used in a variety of scenarios. Some sample scenarios where such a service can be used are in handling:

- Events reporting failures that require corrective action
- Events that record usage of resources for billing
- Events relating to system security
- Events that collect and report routine usage data specific to the application domain.
- Events that report configuration changes
- Events used in performance testing and debugging the application.

Similarly, no prescriptive policy syntax is required. As explained later, the system uses a wrapper mechanism to identify a policy that conforms to a generic template. However, the user has the freedom to alter the exact form of their policies based on individual requirements. The system stores all policies received, and then matches detected events against available policies. The policy generation can be further augmented and automated, if the generator of events is capable of providing policy templates for its events.

We propose to use multicast on this network for the distribution of policies, events and requests. Roles assigned to the store and forward nodes control the multicast groups they join and the sets of policies and events they may receive. Event based systems are of considerable interest in supporting the evolution of current distributed systems into those that will support the emerging class of dynamic applications that require a response to asynchronous distributed happenings. Event based systems allow any type of event or combination of events to be used as triggers for further actions within the system. Some examples of event based management systems are GEM from Imperial College [4] and work at the University of Cambridge

[5] both of which use the concept of event filtering, which we use as part of our proposed architecture.

3. The Use of XML

One of the design goals of our system was flexibility. As stated earlier, the system was aimed at being capable of handling heterogeneity. It was also aimed at being extensible. Another important concern was the capability to utilize existing communication protocols and security mechanisms.

As an example of the heterogeneity that is expected to be handled by the system, an application may be created using components developed in different languages, on different operating systems and we would like to use a single management structure. As mentioned previously, we have proposed the use of XML as an information exchange medium in component based systems and believe that it is of equal applicability in this management context.

XML was proposed by the World Wide Web consortium (W3C) [6] as a simplified form of SGML and a replacement for HTML. It allows the separation of a document's content and any visual presentation. An advantageous spin off of this separation is the use of XML in applications where visual display is not important, and the document's contents in this case being treated purely as data. XML is being used to describe components and applications in a vendor and language neutral way and therefore already has a role in distributed systems. XML is also being used as a data interchange format between components and applications in loosely coupled large-scale applications.

XML provides a simple standard way to delimit and interpret text data. It provides a set of rules for forming semantic tags that break a document into parts and identifies those parts. Unlike the fixed set of tags in HTML, XML is extensible; it is a meta-markup language and the user can define tags as needed. XML tags name the element being described and named attributes modify the tagged structure. This flexibility allows the user to formally describe a syntax they have developed and share it in a programming language and platform agnostic way with others. XML is often referred to as self-describing in that each item carries a name that can be referred to in an external model. In addition, XML allows the structural definition of the data to be specified with the data itself. This provides a mechanism for initial validation of the data, so that it conforms to a generic pattern and then further interpretation of the individual elements can be delegated to sub-systems that are capable of doing so. Also, due to its status as a proposed replacement for HTML, XML transport receives the same support that HTML currently has. In addition to all these factors, XML has the added advantage of being a text-oriented format that makes it human-readable.

We propose that XML be used to describe events, policies and even more. A component's metadata can be described in XML and might contain a list of the events it could produce. This would allow automatic examination of components and component based applications to produce a list of events with a suggested list of required policies. Such automation should contribute to the generation of significantly more robust applications. Our initial policy representation used in this work is based upon the work of Sloman et al. [7] [8] that describes a policy notation for both authorisation (rights) policies and obligation (responsibility) policies. Because we want flexibility in policy definitions, we have also examined the representation of policies expressed using Ponder [9] being developed by Sloman et al at Imperial College. Ponder is a more complicated policy language but it is still possible to express policies as well formed XML documents.

Driven by our stated need for flexibility our current system is based on wrappers. Any communication entering the system conforms to a generic pattern of a message, which is composed of a fixed (but extensible) header and a flexible body. The header of the message is analyzed to identify the message type and the message is passed on to the appropriate sub-system for further processing. Thus, both events and policies arrive in the system as messages with a header. The header contains information that is used in interpreting the contents of the body. It also contains information on the intended recipient of the message, which can be a sub-system of the information server or any other entity in the system. We therefore make no assumptions about the policy definition language or event structure. Each message will be handled or matched according to its own document type definition. The overall structure of an incoming message containing an event is described in Figure 2. As mentioned above, the event template consists of a fixed header and a flexible body. This is a generic template for an event message and can be extended by embedding XML fragments into the individual fields that could be interpreted further.

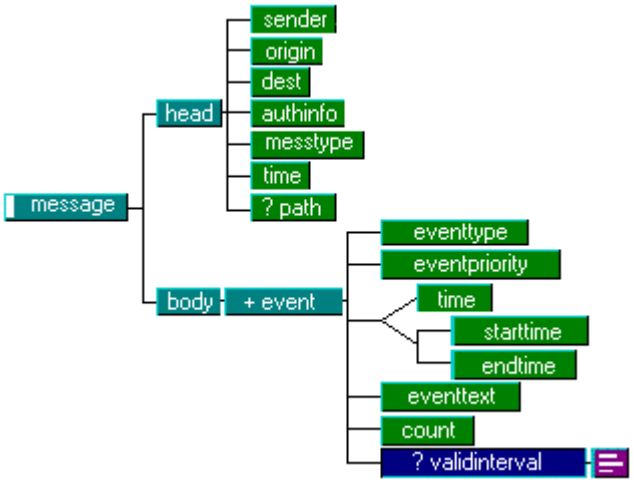

Figure 2. Template of an Event message

Should the message contain a policy then the template shown in Figure 3 is applicable.

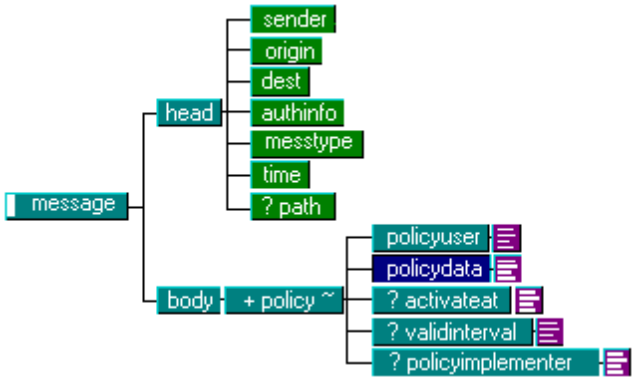

Figure 3. Template of a Policy Message

Figure 3 is the generic template for any policy entering the system. The actual data pertaining to the policy is stored in the policydata field. As an example, the

policydata field structure for one of the sub-systems of the information server is shown in Figure 4.

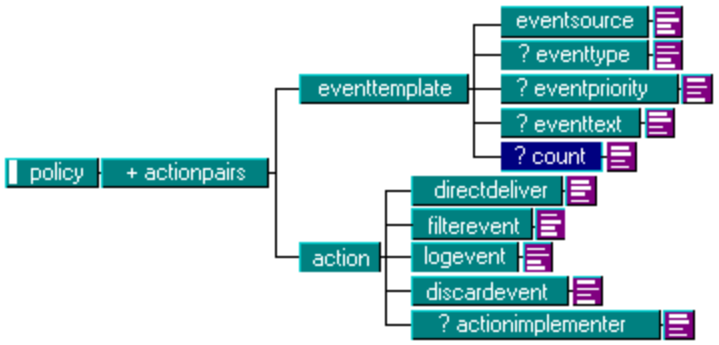

Figure 4. Policy data for the EventHandler sub-system

As can be seen from the above example, using XML lends itself to easy extension of the generic templates to match specific requirements or any particular representation of events and policies that the user might choose.

4. Architecture of an Information Server

Figure 5 shows the overall architecture of the information server prototype implementation. The prototype has been implemented completely in Java.

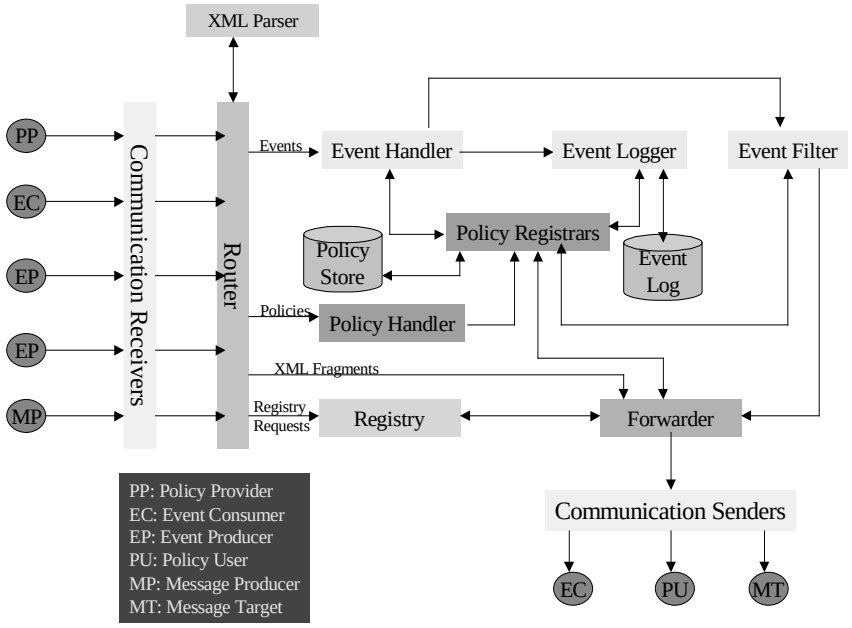

Figure 5. Architecture of an Information Server

4.1 Users of the Information Server

Before analyzing the architecture in detail, here is a brief overview of the proposed users of the information server. The users of the server can be broadly classified into two groups: Information senders and Information receivers. It is possible that the same component is both an information sender as well as a receiver. Information senders can be further classified into the following categories:

- Event Producers, which send information about local events to the server.

- Event Consumers, which can register interest in specific or generic events from individual Event Producers or classes of Event Producers.
- Policy Producers, who can generate policies that can control the usage and operation of the information server.
- Message Producers, who can produce messages that can be routed to their destinations by the Information Service, provided the service has knowledge of the destination.

Similarly, the Information receivers can be categorized as follows:

- Event Consumers, which receive information on event occurrences for which they have registered an interest with the service.
- Message Targets, who can register with the service in order to receive messages targeted at them.
- Policy Users, who are the targets of propagated policies that are relevant to them.

4.2 Communication Layer

The system was designed keeping in mind the need to separate communication mechanisms from the operation of the server. Thus, communication was implemented as a separate controllable layer. This layer acts as the single point of contact between the external world and the server and is in turn controllable through policies sent to the system. Due to this separation of communication from the operation of the server, it is possible to use different protocols for the communication. This flexibility aids interoperability between heterogeneous systems and ultimately will allow the choice of transport protocols that offer definite qualities of service, or guarantees of correctness for different classes of communication. At present, as a lowest common denominator approach, a simple TCP/IP based communication mechanism has been implemented. At the client end, a similar abstraction is provided that hides the complexity of the underlying communication from the client. To achieve this, a simple multicast-based discovery mechanism has been implemented that lets clients connect transparently with an Information Server. The discovery mechanism itself has been implemented as a swappable component, so if needed directory services such as JNDI or similar alternatives can be used.

4.3 Router

The flexibility of the server in handling various message types correctly and its extensibility to handle new message types is implemented through the router component. The router parses the incoming XML document and proceeds to analyze the header in detail. At present, we are using the Xerces XML parser from Apache [10] and parsing the incoming document into a DOM tree [11]. The rationale behind parsing the incoming documents into a DOM tree is that the documents are expected to be small (typically around 1 KB). Apart from this, the DOM tree representation lends itself nicely to the handling of documents whose actual structure is determined by the initial contents of the document itself. The router examines the header portion of the incoming message to determine the destination and type of the message. Once this is done, it passes the parsed message on to the relevant sub-system if it is intended for the server. Otherwise, it passes it on to the forwarder sub-system that takes care of forwarding services.

4.4 Policy Handler

Any message that is identified as a policy by the router is passed on to the policy handler. The policy handler is aware of the general template of a policy and analyzes the policy message further to determine the intended subject (user) of the policy. The policy handler then redirects the policy to the relevant policy registrar. In our current implementation we have policy registrars associated with each sub-system, that controls policies relevant only to that sub-system. It is possible to have a central policy registrar that replaces these individual policy registrars, though it might lead to a loss of flexibility in terms of controlling the policy registration mechanism of the sub-systems.

4.5 Policy Registrar

The policy registrar is the component that is responsible for storing and controlling the policies that pertain to a particular sub-system. The policy registrar keeps track of policies, activating or de-activating them as mandated by the policy producer or the policy itself. In addition, it also does garbage collection of the policy store by removing expired policies from the store. The policy store itself is separate from the registrar. At present, due to the continuous need for retrieving policies, they are stored as parsed objects in custom data structures. As XML storage and retrieval technology matures, we expect the policy store to be a standard XML database that can be queried through conventional techniques.

4.6 Event Handler

Any message that is identified as an event by the router is passed on to the event handler. The event handler is aware of the generic template of an event and queries its policy registrar to find matching policies for each incoming event. Based on the matching policies, the event handler takes appropriate action on the event. At present, this could be either passing the event on to the event filter for sophisticated filtering of the information, logging it into a local event log, discarding the event, delivering it to interested recipients or taking a custom action specified by the policy.

4.7 Event Logger

The event logger acts as an intelligent interface to a local event log. If a policy is in place to log events that match a particular template, then the event handler redirects these events to the event logger. The event logger stores these events in the local log. Apart from this, the event logger can be controlled by policies that state the retention period for events in the local log, periodic purges of events matching templates or transmission/back-up of the local store (selectively if needed). At present, the event log itself is implemented to use XSet [12], an in-memory implementation of a XML database with limited ACID semantics.

4.8 Event Filter

The event filter acts as the mechanism for devolving intelligence and authority to the local information servers. It is envisioned that the event filters will act based on policies to perform sophisticated prioritisation, filtering, aggregation, averaging, threshold detection or other operations on incoming raw events. Through this implementation of policy-driven filtering, the decision-making process of the management system will be spread through the system rather than being concentrated on individual nodes.

4.9 Registry

The registry acts the mechanism that keeps track of recipients interested in receiving notifications on occurrences of events. It also keeps track of known message targets, which is used in forwarding messages to the intended target. In this aspect, the registry acts like a conventional address book. It must be noted though that this functionality is not a necessary part of the service and its removal does not affect the rest of the service. Again, the registry can be controlled by policies to allow or deny registrations, determine lease duration for registrations etc.

4.10 Forwarder

The forwarder is the primary point through which all outgoing communication from the information server is directed. It interacts with the registry to determine the transmission mechanism for each message and takes care of the delivery. The forwarder can also be controlled through policies to determine transmission characteristics to meet protocol and reliability requirements

4.11 Implementation details and sample usage

The server implements prioritization of events and policies by using different class-based queues for the messages after they cross the router sub-system. Thus, based on their priority determined from controlling policies on the router, any incoming message is placed into an appropriate class queue for the corresponding sub-system.

At present, the server does not employ a pool of parsers to concurrently parse incoming message fragments. The rest of the server is capable of handling messages concurrently. Setting up a parser pool would lead to much better throughput of the system in terms of number of messages processed per unit time.

Due to the flexible nature of the server, typical message sizes would depend on the user's information model for the messages. For the event and policy representation that we have used in our work, messages are typically 1 KB long.

To conclude our discussion on the prototype we give a sample usage scenario for such a server. There are many large-scale commercial products such as webservers that are supplied with comprehensive management tools by their vendors. These management tools are capable of managing single instances or large clusters but are usually single vendor solutions. Our ultimate goal is the unified management of an enterprise's complete set of applications and not just those of a single vendor. If our proposed solution is to be viable it is essential that we be able to generate and collect events from such commercial products. To explore the feasibility of using our information store with a commercial product, we used Weblogic Server by BEA systems [17] as an example of a large scale widely used commercial product. The Weblogic server provides an Event API [18] in which events are published internally and external entities may subscribe to receive events of interest. The clients of the server can specify conditions in an evaluator method. When these conditions are satisfied the event occurs and a client specified action method is called. The action method can generate an XML fragment and externalise it for collection by the Information server. The Information Server can be used as an extension to such an application that is capable of externalizing events. In such a situation, the Information Server can be plugged into the application using a bridge like the one mentioned above, that converts the native application events into XML fragments corresponding to the Information Server's input and vice-versa. This would add dynamic management ability to the application by letting the manager specify policies that take

different actions, either on the application's components or otherwise, based on different events. As an example, a policy to divert traffic or alert administrators in the presence of high loads can be enforced on a Weblogic server that is capable of providing traffic events per unit time.

5. Related work

The information service we have proposed is quite different from conventional event services or similar systems with publish-subscribe semantics. Unlike traditional publish-subscribe systems, the proposed information service also has the capability of implementing management policy at all levels. Thus apart from the simple filtering capabilities of publish-subscribe systems, the information service can be made capable of making management decisions at the lowest possible levels in the system.

Generally, existing event-based services can be categorized as channel-based, subject-based or content-based. Due to the flexibility of XML in expressing self-defining data, the proposed information service can be made to fit into any of these three categories based on the requirements.

Among the channel-based event services is the CORBA Event Service [13]. The CORBA Event Service provides only rudimentary filtering of events and does not lend itself to easy extensions for implementing additional functionality into the service itself. The Jini distributed event model [14] provides another framework for event services. The Jini event model puts the onus of event propagation on the event producers and consumers themselves. This does not scale well and leads to complex interactions in the system, in the absence of third party agents such as the one described in this work.

SIENA [15] is an event-based service that offers a scalable and content-based publish-subscribe mechanism. Gryphon [16] is another system that also provides content-based messaging. These systems primarily focus on efficient matching and distribution of messages based on their contents. In contrast, the information service has been focussed on being flexible and capable of being extended at run-time to perform actions based on matching events. Given the flexibility and expressive power of XML, the notations of these systems can be expressed in XML. Thus it is conceivable that the techniques developed in these two systems can be applied to the information service to improve its efficiency.

6. Summary and future work

We have presented the motivation for a management solution that is capable of handling heterogeneity and is highly flexible and extensible. The utility of XML in the implementation of such a solution has been discussed. The design and implementation of a prototype information server along with the architecture of a generic information service has been presented. The information service's relation with similar work has also been discussed.

There are a lot of issues that need to be addressed with the current implementation of the information server. The actual topology and implementation of the proposed information service architecture is one of them. As mentioned earlier, the service is envisioned as one that uses IP multicast in local domains and heavyweight gateway servers to get across domains. Gryphon [16] and SIENA [15] are systems that provide efficient techniques for propagation of events and policies through such a service and this is an area of interest for future work.

Empirical studies need to be made to determine the scalability and efficiency of such a service. Transactional semantics for policy implementation is another

improvement that can be made on the existing implementation. The current implementation does not provide any transactional semantics or implementation guarantees for policies. This can be modified to provide different classes of guarantees for policy implementation ranging from best effort to guaranteed implementation.

Storage and retrieval technology for XML based policies and events is another area of active development. The use of emerging schema standards and XML databases in the system can lead to improvements in reliability and error-recovery. Conflict resolution in the presence of multiple governing policies is another important area of work.

Integrating the information service with current management solutions should be attempted and the impact of the service in a truly heterogeneous environment with multiple management solutions needs to be demonstrated. The overhead associated with such a service in this scenario has to be measured to determine the feasibility and requirements of the service.

References

[1] I. W. Marshall et al "Application-level programmable internetwork environment". BT Technology Journal, Vol. 17, No. 2, pages 82-94, April 1999

[2] Baskar Sridharan, Sambhrama Mundkur and Aditya P. Mathur. "Non-intrusive Testing, Monitoring and Control of Distributed CORBA Objects". In *Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages*, pages 195-206, Mont-Saint-Michel, France, June 2000.

[3] I. W. Marshall et al "Active management of multi-service networks". In *Proceedings of the IEE colloquium 99/147 - control of next generation networks*

[4] M. Mansouri-Samani, M.Sloman, "A Generalised Event Monitoring Lanaguage for Distributed Systems", *IEE/IOP/BCS Distributed Systems Engineering Journal*, vol 4, no 2, pages 96-108, June 1997.

[5] M. Spiteri and J. Bates, "An architecture to support storage and retrieval of events", *Proceedings of MIDDLEWARE 1998, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Lancaster, UK, September 1998.

[6] http://www.w3.org/TR/1998/REC-xml-19980210

[7] E. Lupu, M. Sloman, "A Policy Based Role Object Model", In *Proceedings of the 1st International Workshop on Enterprise Distributed Object Computing (EDOC'97)*, pages 36-47, Gold Coast, Australia, October 1997.

[8] D. A. Marriott, M. Sloman, Nicholas Yialelis, "Management Policy Service for Distributed Systems", Imperial College Research Report DoC 95/10, September 1995.

[9] http://www-dse.doc.ic.ac.uk/policies/ponder.html

[10] http://xml.apache.org/xerces-j/index.html

[11] http://www.w3.org/TR/REC-DOM-Level-1

[12] http://www.cs.berkeley.edu/~ravenben/xset/

[13] Object Management Group, Event Service Specification v 1.0, December 1997.

[14] Sun Microsystems Inc., The Jini Distributed Event Specification (EV) v 1.0.1, November 1999.

[15] A. Carzaniga, D. S. Rosenblum, A. L. Wolf, "Interfaces and Algorithms for a Wide-Area Event Notification Service", *Technical Report CU-CS-888-99*, Department of Computer Science, University of Colorado, October 1999.

[16] G. Banavar et al, "An efficient multicast protocol for content-based publish-subscribe systems", In *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, Austin, TX USA, May 1999

[17] http://www.bea.com/products/weblogic/server/index.html

[18] http://www.weblogic.com/docs51/classdocs/API_em.html