ACTIVE MANAGEMENT OF MULTI-SERVICE NETWORKS

I. Marshall*, J. Bates^, M. D. Spiteri^, C. Mallia*, L.Velasco*

Abstract

The increasing need for rapid introduction of new services and highly customised service offerings poses additional management challenges for today's networks. In this paper we propose a flexible distributed management system based on role driven policies and active layer networking that addresses these demands. Management information is distributed via a hierarchy of Information Servers which accommodate different storage and propagation requirements. The design and implementation of these Information Servers is described in detail as is their integration with an existing commercial management solution.

1. Introduction

Usage of multi-service networks is increasing rapidly as global take-up of the heterogeneous Internet increases. At the same time the number of services offered is growing exponentially, along with the associated management costs. To minimise the cost and complexity of this large distributed system, novel approaches to system and service management are required. Previous proposals have typically involved some attempt to reduce the interdependency of system components and enable management actions to be undertaken purely on the basis of the state of individual components. One approach is 'Management by Delegation', as proposed in [1]. Here, the management of applications is delegated to the servers where they execute, and thus the processing load is shifted onto the application server. However, such a system retains a centralised model for information handling and delegation, and appears unlikely to scale to the degree envisaged.

The use of mobile agents has been proposed as an alternative solution [2,3]. The agents are migratory programs that move through the system collecting data. Rather than communicating data to a central source the agents may act autonomously, either alone, or in collaboration with nearby agents to resolve system problems. However the agents tend to be large and complex, and their migration through the system imposes considerable overhead, reducing performance.

Active networks [4] aim to enable network clients to easily add new services. Users of an active network can supply the programmes and policies required for their custom services, in transport packets alongside their data. This increases the available network functions dramatically, and at the same time reduces network operating costs. The latter benefit is achieved by effectively delegating expensive management functions associated with the client-defined services, to the clients. However, to realise the benefits, a more flexible management system is required. In particular, there should be no restriction on who can delegate to what, nor on the location of the 'delegator'. One must also provide a common information model for server and network resources ensuring management information can be passed around the system in a scalable manner.

We propose an active management solution for multi-service networks based on role-driven policies and Application Layer Active Networking (ALAN) [5]. Our approach avoids many information handling problems by using a lightweight scalable mechanism for information transfer. The proposed Information Management System consists of a hierarchy of '*store and forward*' information stores, with events being classified by their propagation characteristics and storage duration. Our solution is flexible and able to accommodate the requirements posed by the active networks far more easily than current solutions such as Tivoli that assume unitary administration authorities.

*BT Labs, Adastral Park, Martlesham Heath Ipswich IP5 3RE

^Laboratory for Communications Engineering, Department of Engineering, University of Cambridge, UK.

2. Distributed Management System

The major system entities involved in the provision and operation of services on today's networks are the owners, providers, administrators and consumers [6]. The aim of our management architecture is to enable management to be performed by all these system *stakeholders*. Each stakeholder can have a number of roles, each role having associated with it the authority to act in selected parts of the system. Managers can adopt any role they are authorised to adopt, from any location, and should be regarded as mobile.

The management system controls a set of networked entities (including routers, caches and servers) capable of autonomous actions. The actions to be undertaken at each entity are determined using knowledge that entity possesses about itself (local knowledge) and policies (these being supplied by remote managers) that specify responses to system events. The local knowledge will consist of status data on elements controlled by the autonomous entity and will be obtained, for instance, from SNMP MIBs, local logs and tables, local test routines (ICMP based for example), and any other appropriate source. The number of policies applicable to an event instance will typically be constrained by the local knowledge of the entity. Furthermore, each entity has a policy defined by its administrator, that expresses the local precedence order of the management roles authorised to provide policies. Where conflicting responses are possible the system should use the policy defined by the manager with the strongest role. All of the policies are based on the work of Sloman et. al. [7,8] and allow manager rights (*authorisations*) and responsibilities (*obligations*) to be linked to the manager's role. The capability to dynamically modify the current set of policies on a node effectively allows for the run-time modification of its behaviour and forms the basis of our active management proposal.

The possible actions include download and execution of management utilities and other executable programs that may be required but are not already installed. Programs added to the system using the active capability discussed in [4], have policies distributed with them that specify their usage and behaviour. These program policies are associated with the strongest role of the program provider at the entity where the program is used.

2.1 Management information

The four principal vehicles for management information transfer in the system outlined above are *requests*, *policies*, *events* and *programs* (Fig. 1).

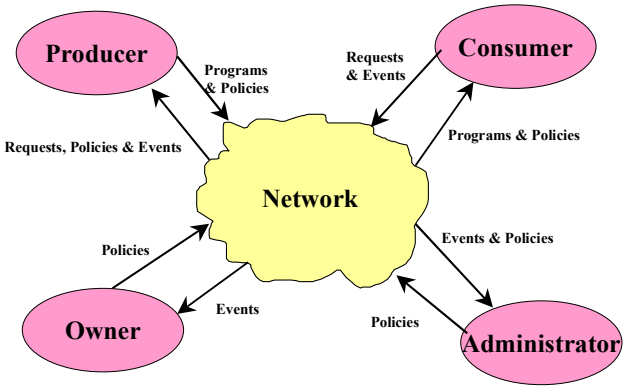


Figure 1 - Information transfer

Programs are identified using URNs or URLs, and are obtained via HTTP 'GET' requests from an HTTP cache hierarchy. In other words they are distributed via a demand led *pull* mechanism. By contrast, policies and events require a *push* mechanism to enable entities to update managers with event messages, and managers to update entities with new policies. Furthermore, the distribution mechanism for events and policies must meet a number of important requirements. For many events guarantees of delivery must be provided, even if the target is currently off line. Latency guarantees will often be needed as will event filtering and consolidation. For the cases when there is a policy implosion at a particular entity, or an event implosion at the site currently occupied by a particular management role, buffering will also be needed.

2.2 Distribution of information

To meet the information management needs of our distributed management system, we propose the use of multicast on a network of '*store and forward*' information servers for the distribution of events, policies and requests. So the policies associated with a management role determine the multicast addresses on which managers adopting that role should listen for events. Entities listen for updates on addresses mandated in existing policies, and entities send each event message to all the multicast groups specified in the policies triggered by that event. To reduce address proliferation, many policies will mandate use of well known addresses, however, due to the limited number of multicast addresses available some address translation will be needed from local scopes to global scopes.

Fig 2 illustrates the network of '*store and forward*' information servers we propose to meet the distribution requirements of the management information. This network is being described in detail next.

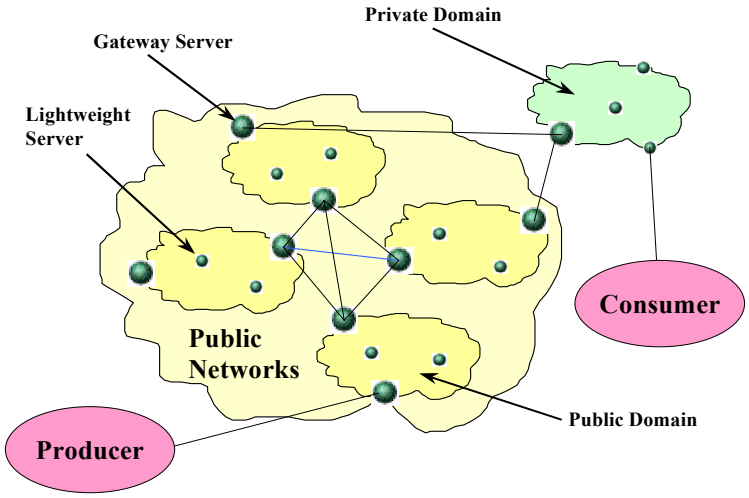


Figure 2 - Network of Information Servers

We envisage two classes of information server; lightweight partially-functional servers which can be co-located with any entity, and fully functional servers which will typically be located at gateway nodes. The lightweight servers are intended to enable load sharing within domains, and will support only those processes that are heavily loaded in the host domain. They will only have knowledge of their local (default) gateway servers. On the other hand, the gateway servers maintain a table of neighbouring gateway servers to which information may have to be routed. This table is analogous to a BGP/MBGP routing table in that it enables selection of the most appropriate 'next' hops, but since not all BGP/MBGP routers will be information servers a separate protocol is required. The table will contain a number of multicast addresses, each corresponding to a different combination of neighbouring servers.

So the distribution of events takes place as follows: entities in a domain multicast events on local addresses. The gateway server then transmits any events required by managers outside the domain using a multicast address used by the appropriate neighbour servers (or a series of unicasts if multicast is not available). Therefore the events are tunnelled across the WAN. A tunnelled IGMP will thus also be required. External requirements are communicated by attempting to join the local group. The gateway server will catch and log all such attempts by monitoring tunnelled IGMP requests at the appropriate external port and checking the authorisation of the remote manager. Event messages will also be reliably stored by the gateway server (for as long as is specified in the message through a timeout parameter), to enable recovery from faults and outages. A manager or information server in recovery mode will poll neighbouring servers for any queued messages, and pull them down in order of arrival. Managers can also elect to retrieve some or all of their messages in this way at any time, by simply not joining the multicast groups. The gateway server will forward join requests to its neighbours using the tunnelled IGMP.

Distribution of policies and requests follows a similar pattern (not surprising since requests and policy changes are themselves represented as events). However, since this is a proactive activity, an extra initialisation step is necessary. The manager who is the source of the policy must first identify the local multicast address for his policy to use. In the event that his role has a well-known address this is simply obtained from a data server (e.g. using the *whois* service or through HTTP). However, it is necessary to provide for new roles that could arise. This has been left for further research, as protocols for dynamic allocation of multicast addresses are currently not fully developed.

3. Design and Implementation of IMS

We have designed and implemented the central component of the proposed management system, the Information Management Server. As shown in Fig 3, an IMS server instance is composed of several functional components.

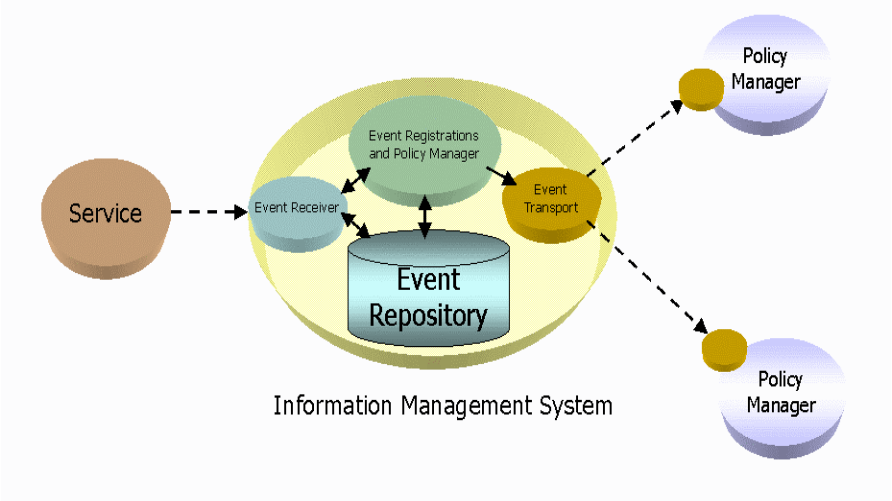


Figure 3 - Information Server

The IMS server is implemented in Java and it offers an API to define different event types, as required by specific domains. An event type comprises an event schema together with the names and data types of the event fields. Initially, we have modelled event types and templates as java classes within java class hierarchies, so as to take advantage of the strong typing system of this language. Java serialisation was then used to translate event instances into byte sequences to be transported across the network to registered clients. However we have found that this implementation posed unnecessary restrictions. The data transmitted could only be understood by java components on either end, and furthermore, the same version of java libraries was required. This assumed a common administration control throughout and would present difficulties in evolving the system. More importantly, it didn't allow for clients written in other languages to register for and receive events. We have thus decided not to use the java type system. Instead we have defined a language-independent message structure, a set of messages, and provided support for version information. This allows the format to evolve over time and future components to be made downwards compatible. For the event meta-types and parameters we have decided to support the types defined in the ODMG 2.0 specification Java Binding (with the exception of the collection types, which we have decided not to support).

The current implementation of the IMS abstracts the actual network transport mechanism. The event transport system provides a protocol independent API for event communication The defined set of messages and data types, together with the support for version information and for the dynamic discovery of event schemas, allow for the transport functionality to be provided by different protocols. Currently we support the TCP protocol and use java at both ends, but other protocol implementations can be easily provided.

Clients can request event meta-data from an IMS server or they can acquire them via third parties. They can then register their interest for specific instances of events by supplying templates for events to be matched

against. They can also specify the desired registration policies. In the current implementation we support he following policies:

- Immediate delivery: as soon as an event is received by the IMS server, is checked against the client template and if it satisfies the criteria it is transmitted to the client
- Delayed delivery: clients can specify that events should be kept on the IMS server for x number of seconds and then transmitted. Or that they are stored until the client makes an explicit request for them.
- Priority delivery: clients can specify the priority they assign in event instances and the IMS server serves high priority registrations first.
- Limited registration: clients register for a specified number of events, or for a specific period of time.
- Bounded delivery: clients require the delivery of no more than a maximum number of events every a specified time interval.

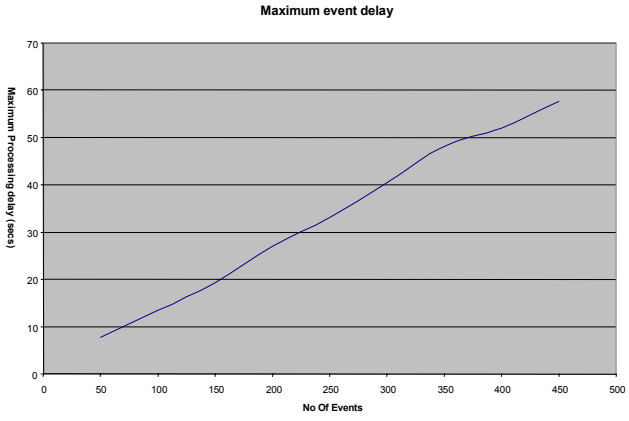
An instance of an IMS server can support all or a subset of the above policies. Event clients can request the list of supported policies and thus discover new policies, as they become available. More importantly they can modify, or even delete, their existing registrations with an IMS server dynamically.

Finally, the event storage functionality is provided by the event repository which is implemented in C/C++ and deployed on Linux. It offers the ability to store and retrieve event instances in temporal order and in real-time and it can handle multiple client read and write connections. An advanced query language [9,10] is provided to enable temporal data mining. This allows for manipulation of stored event classes and instances, as well as for event sessions. It can also be used by clients to access event repositories directly. The event repository, if required, can support only a subset of the complete functionality to optimise performance. So for instance, it can serve as a simple '*store and forward*' buffering service while clients will still be able to use the query interface, but only for simple store and retrieval operations.

4. Integration of IMS

For our experiments, we have used a number of NT machines - effectively our nodes - and equipped each of these with a set of rules. These represented the current set of policies on the node and for their specification we have used a commercial tool, namely RoboMon form Heroix™. RoboMon rules can refer to both system and application specific statistics and variables; they specify conditions which when met, result to predefined actions among which the generation of events. The propagation of events to remote nodes is accomplished via RPC calls to remote RoboMon processes.

We were interested in the scalability of the event propagation mechanism as the number of events generated on a node increases. To study this we have used a set up with two nodes – termed A and B – with A being configured to forward events to B. A was a dual processor 350 MHz Pentium and B a single 266Mhz Pentium. This enabled A to simulate a large number of machines sending events to a single management station. We wrote a custom rule that generated a configurable number of events at fixed time intervals. In the next diagram we can see that the maximum processing delay at node B increases with the number of events sent by A.



The rule was configured to fire every minute and generate 50*N events with N being initially 1 and increasing by 1 on demand. When N was 9 (e.g. 450 events/min) the delay in processing was almost a minute. Clearly, any increase in the number of events per firing of the rule would result in delays for all events.

To address this identified shortcoming in the event propagation mechanism, we deployed on the sender node A an instance of the IMS server. RoboMon uses a Microsoft Access database to store the generated

events and so we wrote a java program that monitored this database. As soon as new events were generated, these were pulled and pushed on the IMS server. We defined an event type for RoboMon events and run a custom written client on node B. This registered with the IMS server for the receipt of RoboMon events. For a special category of RoboMon events, namely service alarms, the client specified a policy of immediate delivery. For all the other events, the specified policy was for delivery of a maximum of 400 events per minute.

We have found that despite the overheads associated with pulling the events off the Access database using JDBC and the extra steps involved in sending them via the IMS server, our simulated management station was able to receive the important service alarm events with very good latency (~secs). This contrasts very favourably with the delays incurred previously for all events once the maximum handling capacity was reached. In this case the IMS server acted effectively as a buffer for events that the client could not handle. When the simulated "event storm" had passed, that is when the custom rule was disabled and the event queue on the server was cleared, the client could modify its "store and forward" policy with the IMS server to request immediate delivery for all events. We could regard RoboMon as an example of an existing management system already in place and deployed. The proposed solution could be integrated in the proposed way and address scalability problems as well as accommodating different delivery requirements and clients with various processing capabilities.

5. Conclusions

The proposed IMS meets the requirements of our flexible delegated management architecture. It supports flexible subscription policies which satisfy real management needs for differentiation and subsequently different treatment of events. Furthermore, it allows the development of more flexible transport policies and it can accommodate different services requiring different transport characteristics, e.g. events over TCP, events over CORBA IIOP or events over HTTP, capable of tunnelling through firewalls. The IMS offers plug-in functionality to implement both subscriptions - to allow clients to specify events that match their interests, and propagation - to deliver matching events to all subscribers. The two supported delivery models, multicast and server side filtering, meet the distribution needs of our information management system. Future work will include a detailed performance characterisation of a store and forward hierarchy of information stores deployed on a testbed of management nodes.

6. References

[1] Yemini Y., Goldszmidt G., Yemini S., "Network Management by Delegation", Proceedings of the 2nd International Symposium on Integrated Network Management, April 1991.

[2] Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., " An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology", ICC 1999.

[3] Ku H., Luderer G., Subbiah B., "An Inteligent Mobile Agent Framework for Distributed Network Management", Proceedings of the IEEE Global Telecommunications Conference (Globecom '97). Pp160-164, 1997.

[4] Marshall I., Cowan J., Crowcroft J., Fry M., Ghosh A., Hutchinson D., Sloman M., Waddington D., "Alpine - Application Level Programmable Inter-Network Environment", BT Technology Journal, April 1997.

[5] Fry M., Ghosh A., " Application Layer Active Networking", Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH '98), June 98.

[6] Chapman M., Bagley M., "ROSA Architecture Progrmaming Infrastructure Support Requirements"., British Telecom Technology Journal, November 1992.

[7] Lupu E., Sloman M., " A Policy-based Role Object Model", Proceedings of the 1st IEEE Enterprise Distributed Object Computing Workshop (EDOC '97).

[8] Sloman M., "Policy Driven Management for Distributed Systems", Plenum press Journal of Network and Systems Management, Plenum Press

[9] Spiteri M., Bates J., "An Architecture to support Storage and Retrieval of Events", *Proceedings of MIDDLEWARE 1998*, Lancaster, United Kingdom, September 1998.".

[10] Bates J., Bacon J., Moody K., and. Spiteri M., "Using Events for the Scalable Federation of Heterogeneous Components", *Proceedings of 8th ACM SIGOPS European Workshop*, Sintra, Portugal. September 1998.

[11] Nwana H., Ndumu D., "Introduction to Agent Technology", BT Technology Journal, Vol. 14, No 4, 1996.

[12] Baldi M., Gai S., Picco G., "Exploiting Code Mobility in Decentralised and Flexible Network Management", Proceedings of the 1st International Workshop on Mobile Agents (MA'97), pp, 13-26, 1997.