Policy-based Resource Management for Application Level Active Networks

Ioannis Liabotis, Ognjen Prnjat, Lionel Sacks
University College London, Torrington Place, London WC1E 7JE, England, UK
email: {iliaboti | lsacks | oprnjat}@ee.ucl.ac.uk

Abstract

The emerging active network concepts, particularly those focusing on the application level, will provide the users with the flexibility of deployment of customised services on the operator's infrastructure. This will be achieved through deployment of user customised control code in virtual execution environments residing on the network nodes such as routers and servers. In order to achieve efficient network operation, the resource consumption on both element and network levels must be managed. In an active networking environment resource requirements come in three categories: bandwidth, memory and processing. Based on the active networking architecture proposed in the IST project ANDROID, we develop resource models and metrics that take into account these three resource categories. The resource models are further used to define the necessary classes of policies in order to achieve efficient resource management. Policies are expressed in XML, and an example policy is given. Finally, we present an experiment comparing the use of alternative resource allocation policies. We also briefly discuss some security issues in the application level active networks.

Keywords: Application Level Active Network, Resource Model, Resource Management, Policies, XML.

1. INTRODUCTION

Application level active services are based on programs supplied by the users of the network. Those programs will run on equipment owned by the operators, enabling users to have access to custom services that will be managed by them without the operators' intervention. This will give flexibility to the user making the network even more attractive and the services more versatile. Effectively, these active network concepts are an example of grid computing [GridComp] philosophy for telecommunications. Various active network architectures have been proposed but none of them has more than a very basic management system. The IST project ANDROID (Active Network Distributed Open Infrastructure Development) is based on application level active networking (ALAN) [Fry99]. In the ANDROID scenario, users supply service components as Java applications that run on specially designed execution environments on active nodes. Management responsibility is delegated to system components that make autonomous decisions in response to changes in their local operating environment.

In an application level active network, resources come into three categories: network, computation and storage. Quality of service in conventional networks is provided by protocols and algorithms that manage network resources. Bandwidth management, priorities of queues in routers and traffic classification are some of the mechanisms used to offer the desired quality of service [Blak98] [Brad94] [Brun98]. In an active services network, computation intensive processes offer the final services to the user community. Thus, the available network bandwidth is not the only parameter that influences the overall performance of a service. The total delay that data packets suffer in an end to end service is the sum of the network delay and the time needed to process data in active nodes. Moreover, during processing in active node data and code need to be stored in memory or disk making necessary to manage those resources as well.

In this paper, we first present the active services architecture. Next, we introduce the candidate resource model for the active servers. Then, we discuss resource management policies and resource mechanisms. Moreover, we present the experimental results comparing the alternative resource allocation policies. In this context, an example XML policy used in given. In the conclusive sections, we also tackle some security issues in this environment.

2. THE ACTIVE SERVICES ARCHITECTURE

The ANDROID architecture is based on the application level active network (ALAN) [Fry99]. ALAN provides an environment in which developers can engineer applications through the network by utilising platforms on which 3rd party software can be dynamically loaded and run [Mars99a]. Combined with a high performance IP network, this would provide a context in which a wide variety of network products and services could be engineered; thus providing a candidate environment for advanced data services, grid computing and ultimately the network computer. The ALAN system consists of regular client and server applications that are located in the existing Internet. Clients and servers communicate through Dynamic Proxy Servers (DPS) that are located in strategic points between them. DPS provides an Environment for the Execution of Proxylets - EEP. Proxylets are downloaded to the DPS and executed on behalf of the users. Those applications provide functionalities that enhance the level of service or introduce new services to the final user. End-to-end active services are provided by one or more DPSs (EEPs) executing one or more proxylets. WWW servers can be used for proxylet storage. Messaging is done in XML and is carried over HTTP [Mars99b].

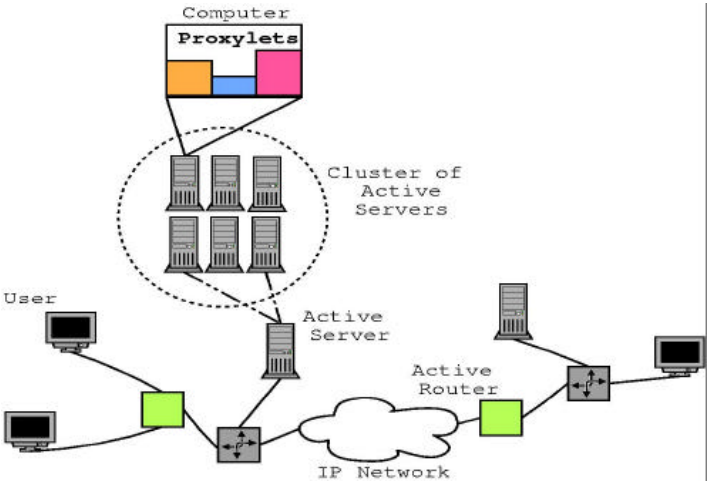

Figure 1 - Application layer active network architecture

These concepts were developed further in the IST project ANDROID, which focuses on the development of a scalable, lightweight management infrastructure for the ALAN-based active networks. ANDROID system is an event driven, policy enabled [Slom99] management system [Mars00]. The focus of ANDROID is on developing the management for primary issues of starting and maintaining the services and on resource and security management. In the ANDROID scenario active nodes are distinguished in two categories: the active routers and the active servers (Figure 1). The active router provides an execution environment that runs dynamically loaded routing software components. Those components offer to users customised routing tables according to user defined policies. Flexibility is restricted by allowing users to provide only configuration policies for components that are carefully selected by the router operators. The active server is the second type of active node that offers a lot more flexibility to users. It can be considered as an end system with a full protocol stack. It also provides an execution environment capable of running user-provided processes that are unrestricted above the transport layer [Mars99a]. Safe execution can be provided if each individual user has its own dedicated server. In case where more than one user shares the

same server special mechanisms have to be built in order to provide a safe and reliable execution environment for users' code.

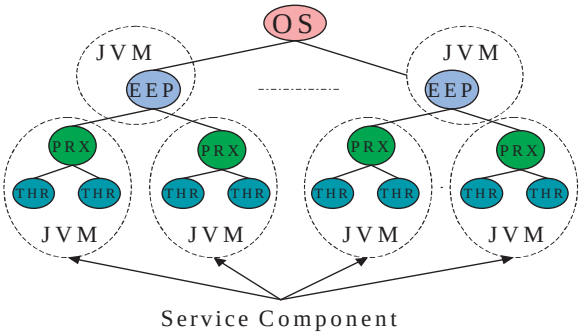

Figure 2 - Active server architecture

Figure 2 gives the general architecture of an active server, the execution environment and the proxylets that run on it. As discussed above, an active server is an end system with a specific general purpose OS. Multiple execution environments for proxylets (EEPs) are allowed to run on each active server. Each execution environment is allowed to run one or more proxylets. If we consider the ALAN architecture, the execution environment for proxylets is a Java Virtual Machine (JVM). Each proxylet runs on its own JVM and can consist of more than one Java threads. The management system targets to manage locally the resources consumed by the proxylets and execution environments. Thread resource consumption needs to be managed by application providers or the users themselves and is out of the scope of the management system.

Note that an active service might be composed of multiple processes that are executed in different active nodes. End user services can thus be separated into three categories. Services that are offered by persistent processes and can be used by multiple users, services that are offered by processes that are started on user demand and can be used by a single user and finally single object services that consist of only one process.

3. RESOURCE MODEL

Based on the above discussion we design a resource model that considers all the resources that are available to the active services users. Figure 3 gives an overview of the resource model in UML [UML]. The basic classes of resources are computation, storage and network. Each user is consuming an amount of those resources, that might change over time according to availability and user's demand.

An active server can be considered as a single processor computer, a multiprocessor computer or a cluster of computers that are connected with high-speed links. Thus management of computational resources can be either allocation of processes to different processors, or allocation of processes to different machines in the same cluster or even allocation of CPU time in the same single processor computer. CPU resources can be described in utilisation percentages, but that can cause ambiguity in case of different processing platforms with different capabilities. For example 60% utilisation of a 400Mhz Pentium II is less computing power than 60% utilisation of a 750Mhz Pentium III. Thus benchmarking of processing power becomes necessary. Although MIPS are not an absolute metric for the performance of a CPU and the behaviour of all applications running on this CPU, it can be considered as a safe general measure of the processing power offered. For application specific needs MFLOPS or other CPU specific capabilities (*e.g.* graphics processing) can be measured and

considered. The basic mechanism to control the CPU resources in a single or multiprocessor computer is the process scheduler which is responsible for assigning processor time to processes.

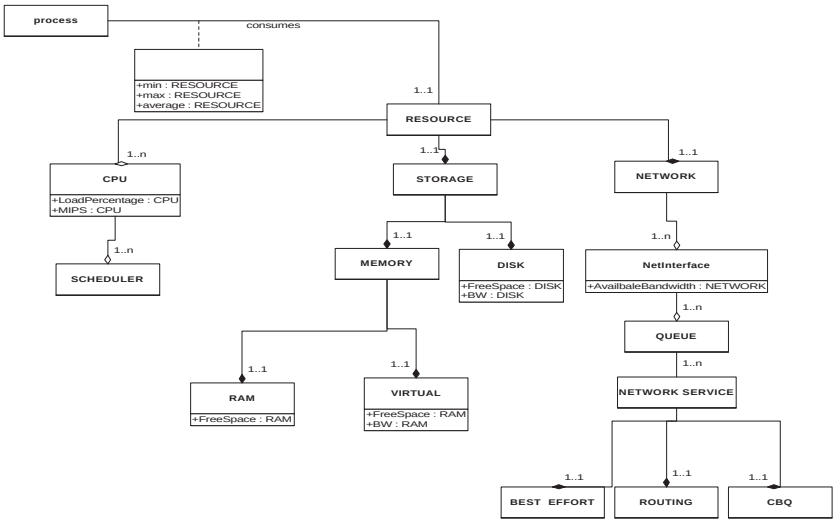

Figure 3 - A simple resource model of an active node

Storage resources are grouped in two categories: memory and disk. The basic attributes that characterise storage resources are available space and bandwidth. Memory and disk space are simply allocated by allowing each process to use a maximum number of bytes. Monitoring of the usage has to take place and admission control should take actions in case the process exceeds its limits. Disk bandwidth can be a critical parameter when a disk is shared by many applications that perform read and write operations - such as fast changing caches.

Network resources are: available network bandwidth, network services such as diff-serv mechanisms [Blak98][Nich99], queues and other protocols. Here we do not describe management of network resources, except for the basic manipulation of packet priorities in server queues, as this is out of scope of this work.

It is important to point out that user requirements are difficult to capture, as resource utilisation is different from service to service and between the users of a specific service. For new services a benchmark of possible resource utilisation can be supplied as an estimate of the resources needed by the service. Then, according to the actual user requirements, new estimates can replace the old ones.

4. POLICIES, RESOURCE ALLOCATION, AND THE QOS MODEL ENFORCEMENT

Management policies describe the rules that must or may be applied to active nodes in order to force a desirable service quality. Policies in the ANDROID system are expressed in XML [XML]. XML is used since it can operate in totally heterogeneous platforms - being independent of operating systems and programming languages. Furthermore, XML can be extensible according to users needs, having an unlimited vocabulary using newly defined tags. In this scenario, users and system operators send policies to active nodes. User policies describe their resource requirements, *e.g.* the minimum, maximum and the average amount of CPU power they require for a specific time period. Administrative policies describe actions

that have to be applied when a new service request is arriving to the system and check operation conditions of existing services. The basic classes of policies are policies for persistent multi-user processes, policies for the introduction of new single user services and policies for single object services. Application providers or larger user communities set up persistent multi-user processes. An initial resource allocation is done for those proxylets. As more requests for service are arriving to the active node the management system is responsible for supplying or denying new resources in order to service the new users and meet the Quality of Service (QoS) guaranties that the service wishes to provide. For single user services or single object services, the user must send policies specifying the amount of resources he/she requires. The management system will then decide on the action that should be taken.

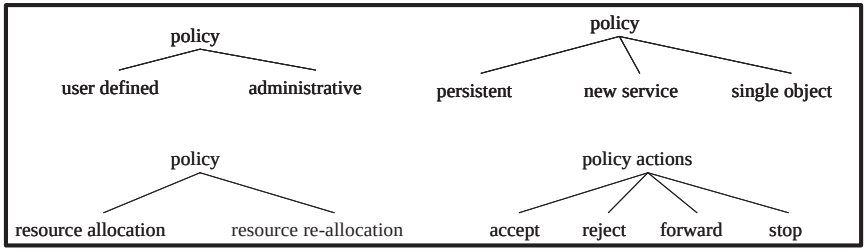

Figure 4 - Policy types and actions

The management system needs to take actions when a request for service arrives; or a request for resource allocation or reallocation is placed by user policies; and whenever a process is not complying with the policies that govern its execution. The basic actions of the management system are to: (i) *accept a request for service, either multi-user or single user/single object*, (ii) *reject a request for service if service cannot be processed*, or (iii) *forward request to another active server*. The management system is also able to (iv) *stop a process in case of errors*. Figure 4 summarises the above discussion and presents a view of policy classification and the available policy actions.

Several types of applications such as networked multimedia require a level of predictable performance. General-purpose operating systems are not designed with this in mind. Applications running on those operating systems cannot have any level of predictable performance. A way of offering guaranteed levels of QoS for applications is to run each service component in a dedicated server or to have more than one service in one server but trying to keep the utilisation at very low levels. This mechanism provides to applications guaranteed quality of service, having as a drawback the increased cost of service, since a lot of resources remain unused for long time periods.

In order to have a cost-effective network, resources have to be shared among different processes. We can distinguish between different classes of resource sharing. *Fair sharing* will not provide any guaranties for QoS in applications. *Sharing based on priorities* can provide some precedence to some applications over the others but in case of high resource utilisation does not provide any guaranty on execution time. *Hard resource allocation* [Brun98] can give QoS guaranties that a process that is competing for resources will execute at a predictable rate, which is determined by the fraction of the resources that is reserved for that process, regardless of the intensity of the competition for the resource. In this model processes might or might not be allowed to share any other free resources available to the system. Different classes of services can be defined so that some processes get guaranteed minimum resources and other fair-share the remaining ones. That way the resource consumption is maximised reducing the cost of service but making the required mechanisms more complex and less

interoperable as modifications to existing operating systems are required. A basic requirement from a system that gives guaranteed resources is that it must achieve this transparently and without requiring from applications to have specific features. Figure 5 summarises the possible methods for resource allocation in an active server.

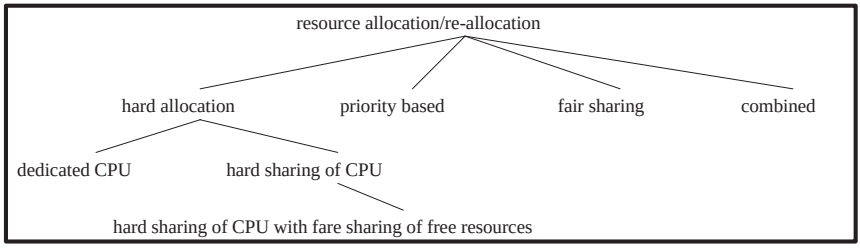

Figure 5 - A classification of resource allocation mechanisms

The QoS model in the ANDROID system is currently under development and its implementation is envisaged to be finalised in the final project demos in late 2002. The QoS model will be based on the above discussion. The XML resource management policies will be used to control the resource allocation, and thus QoS, on the ANDROID active servers. The user-specified resource management requirements, in form of XML policies, will be combined with the resource-related administrative policies defined by the active server operators, so as to yield the final resource management actions that will be enforced by the resource allocation controller on the active server. The resource allocation controller will utilise the resource allocation mechanisms discussed above so as to enforce the agreed QoS. The XML DTD depicting the general policy structure has been specified in the ANDROID project. An example policy used in our experiment is given in the following section.

5. AN EXAMPLE OF CPU RESOURCE MANAGEMENT

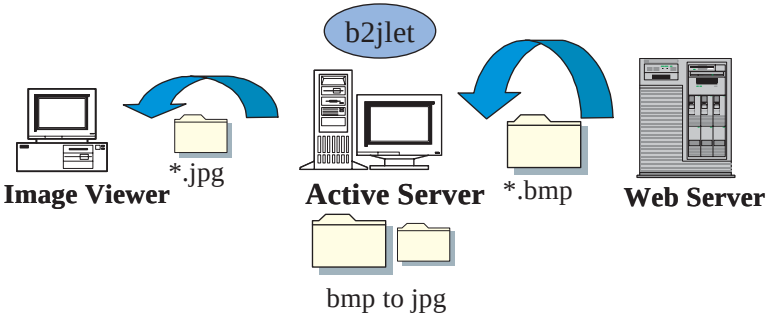

Figure 6 - The experimental set-up

In order to validate our resource management approach we ran a test proxylet that performs Bitmap to Jpeg image conversion. The user application is a Java image viewer that displays images sequentially. The final effect that the user gets is a sequence of images that were captured from a web camera. The images are stored in a web server in Bitmap format. We assume that there is not enough network bandwidth between the Image Viewer and the Web Server in order to transfer the bitmap images with the desirable frame rate. This could happen

if the application user is connected to the network using a modem. We select an active server that is located close to the Web Server, and we run “b2jlet” proxylet which converts bitmap images to Jpeg. That way the image viewer receives Jpeg images, which require less bandwidth to be transferred. Figure 6 illustrates the application and the proxylet that runs on behalf of the user on the active server.

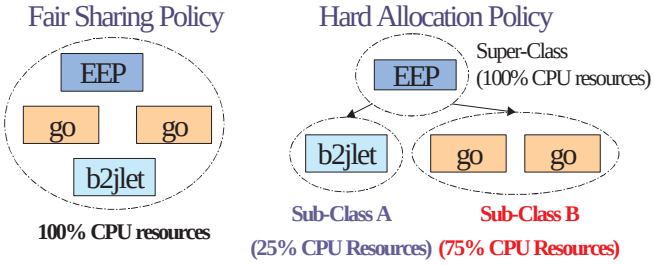

Figure 7 - Fair sharing and hard allocation policies

Now that the available bandwidth is not a restriction the frame rate on the image viewer depends only on the time needed for the images to be transformed by the proxylet. We assume that the end user requires a frame rate of no less than 1.5 frames per second.

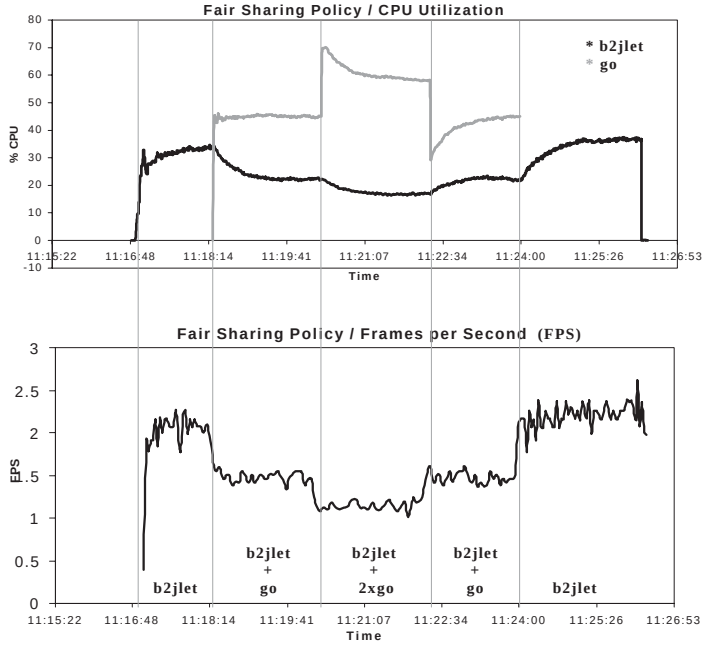

Figure 8 - CPU utilisation and frame rate for the fair sharing policy

Initially the policy on the active server is a fair sharing policy. All proxylets running on this active server consume 100% of the resources using the priorities that the operating system (FreeBSD 3.4) scheduler assigns (Figure 7). When the “b2jet” is the only proxylet running on the machine the image viewer receives images with a frame rate of 1.8 to 2.2 frames per second (Figure 8, bottom). The CPU utilisation of “b2jlet” is around 30% (Figure 8, top). A different user the requests another proxylet to start running. The new proxylet, “go”, implements a CPU intensive process that, if it was running alone in the active server, would consume 100% of the CPU. The two proxylets are sharing the CPU according to priorities assigned by the operating system’s scheduler. This has as an effect that the frame rate on the image viewer drops to around 1.5 frames per second (Figure 8, bottom), which is acceptable, considering the initial user requirements. Then, another user requests to run the proxylet “go”. Now all three proxylets are running, sharing 100% of the CPU. The frame rate on the image viewer now drops to 1.2 frames per second witch is not acceptable any more (Figure 8, bottom). As shown in Figure 8, top, the CPU utilisation of “b2jlet “ was less than 20%.

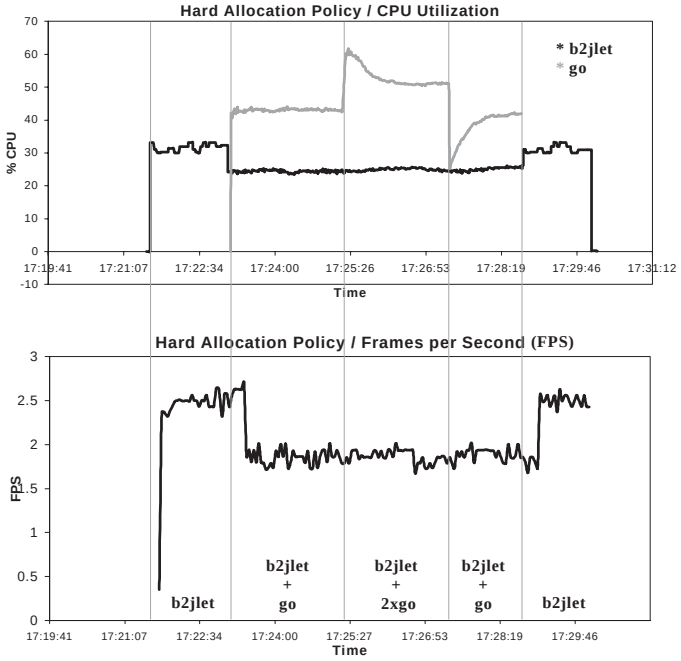

Figure 9 - CPU utilisation and frame rate for the hard allocation policy

Next, we run the same experiment with a hard allocation policy. In order to implement hard allocation in the active server we used the EclipseBSD [EclipseBSD], which is a patch to the FreeBSD kernel that adds various schedulers [Brun98] in order to support hard allocation of the resources such as CPU, network and disk bandwidth. The hard allocation policy allowed “b2jlet” to use at least 25% of the CPU at all times, regardless of the overall system load (Figure 7). The other 75% of the CPU could be shared among all processes. Before the introduction of “go”, “b2jlet” performed the same as with the fair sharing policy (Figure 9). When the two instances of “go” started running, the CPU usage of “b2jlet” dropped to 25%

giving a frame rate of around 1.7 (Figure 9, bottom), which was acceptable according to the initial user requirements.

```
<policy>
    <creator>
        <authority>
            <domain>USER</domain>
            <role>ApplicationProvider</role>
        </authority>
        <identity>/VideoApplication/ADMIN</identity>
        <reply_address>128.40.40.18</reply_address>
    </creator>
    <info>
        <name>Resource Allocation for Image Transcoder</name>
        <modality>Obligation</modality>
        <servicelevel>
            <delivery>Guaranteed</delivery>
            <execution>Optional</execution>
        </servicelevel>
        <priority>1</priority>
    </info>
    <subject var = "RAC">
        <subjectlist>ResourceManager</subjectlist>
    </subject>
    <trigger>
        <event>
            <source>USER</source>
            <category>ResourceManagement</category>
            <eventtype>ProxyletLoad</eventtype>
        </event>
    </trigger>
    <actions>
        <action>
            <target>ResourceAllocationController</target>
            <method>AllocateCPU(b2jlet,25%)</method>
        </action>
    </actions>
</policy>
```

Figure 10 - Example XML policy

The XML policy enforcing this approach to resource allocation is shown on Figure 10. The policy conforms to the ANDROID policy DTD. After the creator tags describing the creator details, the policy information is given, including policy name, modality, and service level. The policy subject is the resource manager, which, on the receipt of the "load proxylet" event, will invoke the allocateCPU method on the resource allocation controller. The parameters of this call are the identity of the target proxylet, and the percentage of the CPU to be allocated to this proxylet.

Experiments demonstrate that the required QoS can be achieved on a single platform, using the hard allocation resource management policy. Thus, QoS can be guaranteed, while the resource utilisation is maximised by allocating more than one proxylet to an active server.

6. CONCLUSIONS AND FURTHER WORK

The active network principles aim to add flexibility to the existing network by allowing the users to deploy their customised services on the operator's infrastructure. In an active services environment, apart from the network resources, there is a need to manage the computation and storage resources as well. Services running on active servers, such as distributed multimedia services, have strict resource requirements. Thus, predictable performance and QoS guarantees are required. This is easy to achieve using multiple platforms, but more difficult in a single platform. There is thus a need to build a management system that will be able to provide QoS guaranties in a shared system. Possible modifications to current OSs might be necessary but the extent of modifications has to be limited so that an interoperable and low cost management system is achieved. Moreover, control decisions have to be local, reducing the administrative overhead for the management of services.

In this paper, we have considered the resource management issues relevant in active network scenarios. The work in this paper was conducted in the context of the IST Project ANDROID, which is developing a scaleable, lightweight, policy-based management system for application-level active network scenarios. We have captured the managed resources available in an active network through a resource model encompassing CPU, storage, and network resources. Moreover, we have discussed in detail the candidate XML-based resource management policies. Policies expressed in XML give a flexible way of describing management responsibilities and resource requirements without the need of implementation specific details. However, in order to conduct efficient implementation of these policies and to be able to guarantee a desired QoS, resource allocation mechanisms have to be in place. The execution environment needs to have a simple interface that will allow efficient resource management reducing the overall complexity of the system, while at the same time maximising the resource utilisation. Thus, different classes of resource sharing mechanisms on a single active server were discussed in detail. Finally, we have given an example of our resource management principles, by comparing the efficiency of two distinct resource management policies. An example policy was given in XML. The experiments demonstrated that the required QoS can be achieved on a single platform, using the hard allocation resource management policy. Thus, QoS can be guaranteed, while the resource utilisation is maximised by allocating more than one proxylet to an active server.

Further implementation of the QoS model in the ANDROID environment is under development, and the full set of resource management policies is envisaged to be available for the final project demo in 2002.

The second key management focus of the ANDROID project is the security in the application level active networking environment. Of particular interest is the security of the active servers. In this context, a number of distinct security policies are under development. These include authentication of the proxylets (based on JAR - Java Archive - signing) and of the proxylet deployers (based on the role matching), proxylet resource access control mechanisms (to be enforced through the modification of the particular EEP files in use), and the in-service policing mechanisms.

7. ACKNOWLEDGEMENTS

This work was partially conducted in the context of the IST project ANDROID. Authors would particularly like to thank Ian Marshall, Mike Fisher and Paul Mckee for discussions on some of ideas presented in this paper.

8. REFERENCES

[Blak98] S. Blake, et al, "An Architecture for Differentiated Services", RFC 2475, December 1998.

[Brad94] B. Braden, et al., "Integrated Services in the Internet Architecture: An Overview", RFC 1633, July 1994.

[Brun98] J. Bruno, E. Gabber, B. Özden, and A. Silberschatz, "The Eclipse Operating System: Providing Quality of Service via Reservation Domains", Proceedings of the USENIX 1998 Annual Technical Conference, New Orleans, Louisiana", June 1998.

[EclipseBSD] http://www.bell-labs.com/project/eclipse/

[Fry99] M. Fry, A. Ghosh, "Application Level Active Networking", Computer Networks, 31 (7) (1999) pp. 655-667.

[GridComp] http://www.gridcomputing.com/

[Mars99a] I. W. Marshall, et al, "Application-level Programmable Network Environment", BT Technology Journal, Vol. 17, No. 2, April 1999.

[Mars99b] I. W. Marshall, M. Fry, L. Velasco, A. Ghosh, "Active Information Networks and XML", in "Active Networks" ed. S. Covaci, LNCS 1653 pp. 60-72, Springer Verlag, 1999.

[Mars00] I. W. Marshall, J. Hardwicke, H. Gharib, M. Fisher, P. Mckee "Active Management of Multiservice Networks" Proceedings of the Network Operations and Management Symposium (NOMS), 2000.

[Nich99] K. Nichols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", RFC 2638, July 1999.

[Slom99] M. Sloman, E. Lupu, "Policy Specification for Programmable Networks", Proceedings of IWAN '99 Conference, Springer-Verlag, 1999.

[UML] Rational Software Corporation, Unified Modelling Language, http://www.rational.com/

[XML] Extensible Markup Language (XML) W3C Recommendation 10. 2. 1998; http://www.w3.org/XML/