Security and Resource Policy-based Management Architecture for ALAN Servers

Temitope Olukemi, Ioannis Liabotis, Ognjen Prnjat, Lionel Sacks

University College London, Torrington Place, London WC1E 7JE, England;email: {tolukemi / iliaboti / oprnjat / lsacks}@ee.ucl.ac.uk

Abstract: *Application Layer Active Networks (ALAN) allow quick and efficient deployment, on the active servers, of user-customised services (proxylets). Programmability above the transport layer makes this approach distinct form other active network initiatives. This scenario raises the issues of efficient resource management on the active server. Moreover, the deployment of user-specified processes has to be highly secure so as not to harm the active server operator platform. The IST project ANDROID is using a flexible generic specification for policies, in XML, allowing a wide range of policies to be expressed and processed in a common framework. This paper presents the security and resource management architecture developed to support the application of the ANDROID policy-based principles to manage the ALAN servers. We present the architecture, as well as the sample policy sets. The prototype security and resource management implementation were demonstrated during two real-life trials and the results are presented here.*

Key words: ALAN, Policy-based management, XML, Resource and security management.

1. INTRODUCTION AND PROBLEM FIELD

Application Level Active Networks (ALAN) [1], provides an environment in which developers can engineer applications through the network by utilising platforms on which 3rd party software (proxylets) can be dynamically loaded and run [2][3]. The ALAN system consists of client and server applications that are located in the existing Internet. Communication is enhanced through customised services provided by user-deployed proxylets which run on the Execution Environment for Proxylets - EEP, which is a Java Virtual Machine (JVM) running on an active server (the host platform). Proxylets provide functionalities that enhance the level of service or introduce new services to the user. End-to-end active services are

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35620-4_43

D. Gaïti et al. (eds.), *Network Control and Engineering for QoS, Security and Mobility*

provided by one or more active servers executing one or more proxylets. The IST project Active Network DistRibuted Open Infrastructure Development (ANDROID) is focusing on management of ALAN-enabled networks. The objective is to develop a flexible, policy-based [4] system that enables monitoring and control of active nodes and services. The key issues are the ability to effectively manage the active server resources, and the security of those resources. Here we present the specific developments of the larger ANDROID management architecture [5] - those concerning the security and resource management of the active servers.

2. ANDROID MANAGEMENT PRINCIPLES

ANDROID focuses on the development of a scalable, lightweight management infrastructure for the ALAN-based active networks. ANDROID system is an event driven, policy enabled [4] management system [6][7]. ANDROID active nodes are either active routers or active servers. Active router provides an execution environment that runs dynamically loaded customised routing software components. Flexibility is restricted by allowing users to provide only configuration policies for components selected by router operators. Active server (an end system with a full protocol stack) offers more flexibility to users, by allowing deployment of proxylets unrestricted above the transport layer. If more users share the same server management mechanisms have to be built in order to provide a safe and reliable execution environment. An active server is an end system with a specific general purpose Operating System (OS). Multiple EEPs are allowed to run on each active server. Each EEP is allowed to run one or more proxylets. The ANDROID EEP is a Java Virtual Machine (JVM) - FunnelWeb [8]. Each proxylet runs on its own JVM and can consist of more than one Java threads. The management system secures and manages locally the resources consumed by the proxylets and EEPs.

The ANDROID approach to management is event-driven and policy-based. Policies specify actions that should be applied when particular events occur [9]. When a policy is triggered, conditions involving locally available information are evaluated and the actions initiated. Policies and events are communicated between the components through the management information distribution (MID) system [10][5]. ANDROID approach strives to facilitate the exchange of management information between heterogeneous systems, by defining a lightweight, flexible, and extensible policy and event schemas., in XML [11][12]. This allows platform independence and facilitates the management information exchange between heterogeneous systems. The ANDROID XML schemas specify features common to all policies or events in a standard way and also allow flexibility in the definition of specific information sets for managing particular components of a system. The policy schema [5] consists of 6 elements: *creator* identifies the origin of the policy; *info* describes information not relevant for policy rules; *sender* identifies the forwarding path of the policy; *subject* identifies entities that respond to the policy; *trigger* relates the policies to events that trigger them; *action* represents the behaviour that the policy triggers, dependent on some conditions. A policy is to be interpreted by a subject which performs actions on targets, dependent on some conditions. The event schema consists of 7 elements: *event-id* (unique); *time*

(when the event occurred); *timetolive*; *source* (event originator); *sequence* (number of events produced from a source); *info*; and the *data* element which allows any well-formed XML to be included, describing the event-specific information.

3. ACTIVE SERVER ARCHITECTURE

Here we present the resource and security management functionality needed for policy enforcement on the active servers (ASs). We first capture the required functionality in terms of use-cases, and then elaborate, through UML, on the required infrastructure of the ASs.

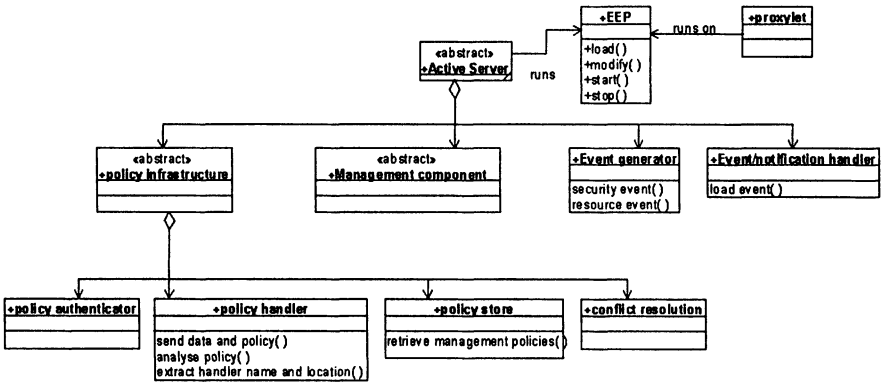

Figure 1 - Active server overview

Use-cases include the service set-up and in-service use cases, initiated by User or AS Operator. *Service negotiation and set-up* involves the interactions between the business players: negotiation of the resource usage, SLA definition, and set-up of policies. *Service initialisation* involves the user requesting the start of the service by sending events to the operators. After performing resource and security checks based on policies, operators load the proxylets. *Runtime service management* involves the operators monitoring the behaviour of the proxylets, and, in case of unexpected behaviour, applying the relevant policies so as to preserve the resource integrity and security of the platforms. *Service modification use-case* involves the modification of the service parameters, either by the user or the operator. Specific sub-cases are: the *reallocation* of the proxylet making up a service to another processing platform; requesting the *increase or decrease of the resources* dedicated to the proxylet. *Service termination* involves stopping the service by the operator or the user.

UML class diagram models are used for detailed design. The AS (Figure 1) hosts one or more EEPs which run one or more proxylets. On each AS, there is a policy infrastructure, providing for *policy authentication*, *generic policy handling*, *policy storing*, and basic *conflict resolution*. The AS also supports the *event/notification handling* functionality. Finally, the specific management functionality is located on the AS: *security and resource managers*. These two components are XML-enabled: they can receive events/policies, interpret them, and apply management actions on targets. These components can also generate XML events - via *event generator*.

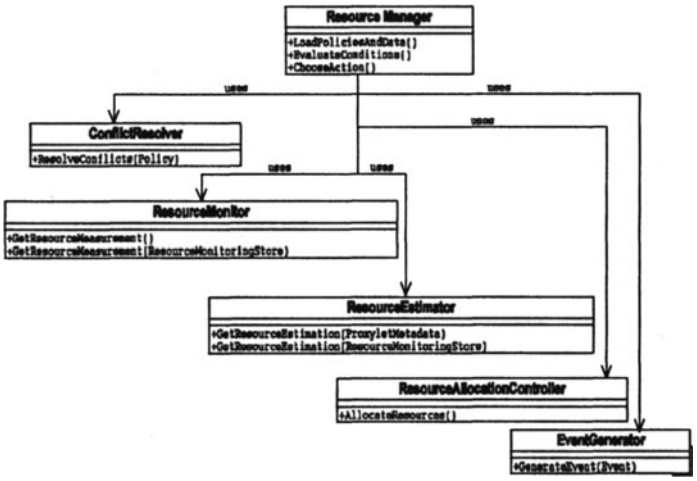

Figure 2 -Resource manager

Proxylets and EEPs running on an AS consume the resources provided by the OS. Resource management is important from the operator perspective, (dictating resource consumption of user processes); and user perspective (proxylets adapting resource usage levels dynamically). 3 main categories of the local resources are CPU (Kernel and User modes), storage (memory and disk) and network [13]. Resource management involves the tasks of resource monitoring - observing the consumption of resources by processes; and resource (re)allocation - closing the local control loop, *e.g.* (re)scheduling a process. The *resource manager* (RM) (Figure 2) uses *resource monitoring* and *estimation* components. It also performs basic *conflict resolution.* Management actions are either communicated in the form of events, created via *event generator*, to the EEP controller (which then enforces the actions on EEP), or are directly enforced on the resources via the *resource allocation controller.*

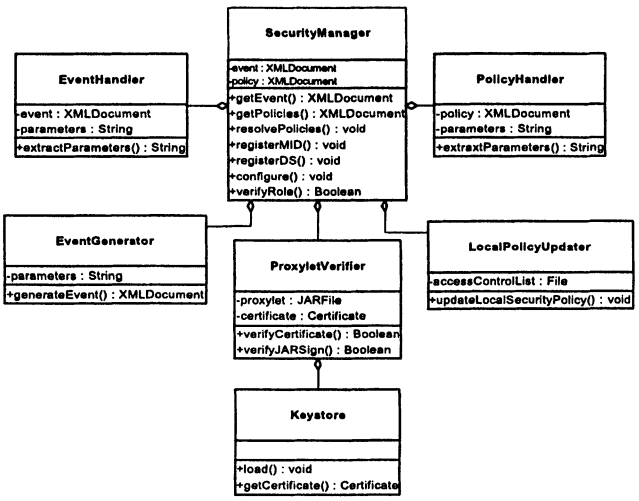

Figure 3 - Security manager

The *security manager* (SM) (Figure 3) performs the policy-controlled deployer and proxylet authentication, and set-up of java.policy file which restricts the runtime proxylet access to the resources [14] in FunnelWeb. SM receives events from event handlers and policies from the policy store. It performs basic conflict resolution. The *event handler* extracts relevant parameters from the XML event. The *policy handler* extracts parameters from XML policies. The *proxylet verifier* performs verification of the signature of the proxylet JAR file. It also verifies that the certificate used to sign the JAR file is a valid one contained in the *keystore. Event generator* receives parameters from the security manager and generates XML events which are forwarded to the MID. The *local policy updater* updates the local java.policy file on the AS using information from an access control list which is derived from the relevant policies.

The UML sequence diagram (*service initialisation* use-case) is shown in Figure 4. When the user-defined "start" event arrives at the active server, it is handled as shown on the first portion of Figure 4. Policy handler analyses the event/policy, retrieves the relevant management policies and forwards the event data and policies to the SM. SM performs the authentication of the proxylet and the deployer, and the update of the java.policy file. Then, the SM sends the security event (in this case the "load proxylet" event) via the event generation component to the event handler. This event is the authorisation of SM to the resource manager notifying it to load the proxylet.

The specifics of the resource management are as follows. The RM receives the resource management policies from the policy handler. The actions might also depend on the measured/estimated resource information. Considering this info, conditions are evaluated, relevant action is set, and the event generator is notified to create the relevant event, in this case "load proxylet". This event is the authorisation of the RM targeted at the EEP and notifying it to load the proxylet.

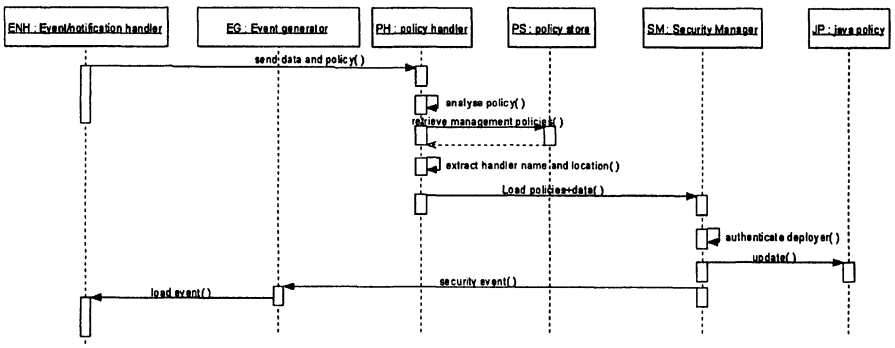

Figure 4 - Service initialisation - security related sequence diagram

4. POLICIES

4.1 Security management

Based on use-cases we categorise the security policies in: policies for service initialisation, reallocation and termination; and for runtime management. This categorisation is based on the type of events that occur in use-cases, and on the event source. Most of the policies in group 1 originate from users, while the group 2 policies are mainly established by the AS operator. Since not all the policies can be described here, the following shows the example security events, conditions and actions associated with the *service initialisation use-case*. Different combinations of these (Table 1) define the full set of policies.

EVENT	CONDITION	ACTION
eLdPrx	pAuthDeployer pAuthProxylet	aLocSPU aLdPrx aDnLdPrx aSecAl
eRnPrx		aRnPrx aDnRnPrx aSecAl

Table 1 - Security management policies (service initialisation)

Events:
Load proxylet event (eLdPrx): received by SM when a user or AS operator wants to load a specified proxylet on the AS.
Run proxylet event (eRnPrx): received by the SM when a user or AS operator wants to run a proxylet that has already been loaded to the AS.
Conditions:
pAuthDeployer: returns True if deployer of the proxylet is authenticated, *i.e.*, has provided a certificate and False otherwise.
pAuthProxylet: returns True if the proxylet has been signed with a valid certificate (of the creator) and False otherwise.
Actions:
aLdPrx: SM allows the proxylet to load - generates "load proxylet" event.
aRnPrx: SM allows the proxylet to run - generates "run proxylet" event
aLocSPU: invokes a direct method call on SM which creates/updates a local policy file with the proxylet/user authorisation to the local AS resources.
aSecAl: involves the generation of an event informing the AS operator that a security violation has occurred.
aDnLdPrx, aDnRnPrx: involves sending an event back to the user notifying that the request could not be fulfilled by the AS.

The example security policy shown in Figure 5 is triggered by a load proxylet event. There are two conditions (**pAuthDeployer** and **pAuthProxylet**) that have to be satisfied before the actions **aLdPrx** and **aLocSPU** are invoked.
Policy-ID (Policy 2)
Event (Load proxylet), **Event Originator** (User)
Condition (If "Deployer authenticated" and "Proxylet authenticated")

Policy Originator (AS Operator)
Action 1 (Invoke Load_proxylet method on resource manager)
Action 2 (Update the java.policy file related to the proxylet)

```
<?xml version = "1.0" encoding = "UTF-8"?>
<policy xmlns = "http://www.android.org/policy" xmlns:xsi =
"http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
"http://www.android.org/policy file:///C:/docs/policy.xsd">
      <creator>
            <authority>
                  <admin-domain>EE</admin-domain>
                  <role>Admin</role>
            </authority>
            <identity>/AS/ADMIN</identity>
            <reply_address>127.0.0.1</reply_address>
      </creator>
      <info>
            <policy-id>270920011237</policy-id>
            <modality>Obligation</modality>
      </info>
      <subject>
            <domain>
                  <role>Security</role>
            </domain>
      </subject>
      <trigger>
            <event-id>eLdPrx</event-id>
      </trigger>
      <actions>
            <condition>
                  <operand>pAuthDeployer</operand>
                  <operator>Equals</operator>
                  <operand>True</operand>
                  <and/>
                  <operand>pAuthProxylet</operand>
                  <operator>Equals</operator>
                  <operand>True</operand>
            </condition>
            <action>
                  <target>
                        <domain>
                              <role>Resource-Manager</role>
                        </domain>
                  </target>
                  <data>
                        <method>aLdPrx</method>
                  </data>
                  <target>
                        <domain>
                              <role>Security Manager</role>
                        </domain>
                  </target>
                  <data>
                        <method>aLocSPU</method>
                  </data>
            </action>
      </actions>
</policy>
```

Figure 5 - Example security management policy (service initialisation use-case)

Another example policy for the *runtime service management use-case* is shown on Figure 6. This use case involves the AS operator monitoring the behaviour of the proxylets, and, in case of unexpected behaviour, applying the relevant policies so as to preserve the resource integrity and security of the platform. The policy shown in Figure 6 is triggered by the eResProfVio event which indicates to the security manager that the resource profile violation has been carried out by the proxylet. If the deployer of the event is correctly authenticated, the security manager invokes the aStPrx and aSecAl actions: it stops the proxylet and raises a security alarm.

```
<?xml version="1.0" encoding="UTF-8" ?>
<policy xmlns="http://www.android.org/policy"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.android.org/policy
http://www.ee.ucl.ac.uk/~tolukemi/policy.xsd">
<creator>
      <authority>
            <admin-domain>EE</admin-domain>
            <role>Admin</role>
      </authority>
      <identity>/AS/ADMIN</identity>
      <reply_address>127.0.0.1</reply_address>
</creator>
<info>
      <policy-id>2260420021050</policy-id>
      <modality>Obligation</modality>
</info>
<subject>
       <domain>
            <role>Security</role>
      </domain>
</subject>
<trigger>
      <event-id>eResProfVio</event-id>
</trigger>
<actions>
      <condition>
            <operand>pAuthDeployer</operand>
            <operator>Equals</operator>
            <operand>True</operand>
      </condition>
      <action>
            <target>
                  <domain>
                        <role>Security Manager</role>
                  </domain>
            </target>
            <data>
                  <method>aStPrx</method>
                  <method>aSecAl</method>
            </data>
      </action>
</actions>
</policy>
```

Figure 6 - Example security management policy (runtime management use-case)

4.2 Resource management

EVENT	CONDITION	ACTION	
eLdPrx	pCPUthr pMEMthr pDISKthr	aLdPrx aAlRs aMRMP	aDnLdPrx aFLd aNtfAR
eRnPrx	pNETthr pPredMeta	aRnPrx alRs aMRMP	aDnRnPrx aFRn aNtfAR

Table 2 - Resource management policies (service initialisation)

Based on the use-cases, we categorise the resource management policies into policies for service initialisation, run-time management, service reallocation, resource increase/decrease, and service termination. Based on the originator we also categorise policies into policies originated by users and by AS operators. Main difference is the delivery and/or execution guaranties they offer. In case of conflicts AS operator policies have precedence over user policies. Since not all the policies can be described here, the Table 2 gives the events, policies and actions associated with the *service initialisation use-case*. Policies are identified by the conditions that the resource manager has to evaluate. Actions can be applied directly to targets or can be sent as events through the notification system to the appropriate management component. Combinations of events, conditions, and actions define the full set of policies (Table 2).

Events:

eLdPrx; eRnPrx; explained in section 4.1 on security management polices.

Conditions:
pCPUthr: this condition type describes a range of policies that are associated with CPU usage. The operands are measured values (in percentages) such as: total time, kernel time, user time; and the statistics.
pMEMthr, pDISKthr: these types of conditions have as operands the amount (in absolute values or percentages) of free/used memory/disk space.
pNETthr: operands are: incoming/outgoing bytes per second and statistical information on those values (average, standard deviation, percentiles).
The above conditions are checked by the resource monitor.
pPredMeta: involves the predicted future resource usage using information found in the proxylet metadata and is checked by the resource estimator.
Actions:
aLdPrx: RM allows the proxylet to load - generates "load proxylet" event.
aRnPrx: SM allows the proxylet to run- generates "load proxylet" event.
aAlRs: a direct method call to the resource allocation controller that will enforce hard resource allocation or change process priorities.
aMRMP: is related to conflict resolution and will be used in the future to set valid policy in case of conflicting requirements.
aDnLdPrx, aDnRnPrx: involve sending an event back to the user notifying that the request could not be fulfilled.
aFLd, aFRn: forwarding actions are applied when the AS cannot load/run the requested proxylet. The original load/run event is sent to another AS.
aNtfAR: The AS notifies the active router (AR) that the request for service has been forwarded to another AS.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<policy xmlns = "http://www.android.org/policy" xmlns:xsi =
"http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
"http://www.android.org/policy file:///home/iliaboti/policy.xsd">
     <creator>
          <authority>
               <admin-domain>AS</admin-domain>
               <role>AS Operator</role>
          </authority>
          <identity>ASOperator1</identity>
          <reply_address>ASOP@as1.org</reply_address>
     </creator>
     <info>
          <policy-id>Policy1</policy-id>
          <modality>obligation</modality>
     </info>
     <subject>
          <domain>
               <role>Resource-Manager</role>
          </domain>
     </subject>
     <trigger>
          <event-id>eLdPrx</event-id>
     </trigger>
     <actions>
          <condition>
               <operand>Total-CPU-Usage</operand>
               <operator>LessThan</operator>
               <operand>60%</operand>
          </condition>
          <action>
               <target>
                    <domain>
                         <role>EEP-Controller</role>
                    </domain>
               </target>
               <data>
                    <method>aLdPrx</method>
               </data>
          </action>
     </actions>
</policy>
```

Figure 7 - Example resource management policy (service initialisation)

The example policy of Figure 7 demonstrates one event-condition-action combination that appears during the *service initialisation use-case.* This AS Operator specified policy allows loading of user requested proxylets if the Total CPU time load of the AS is less that 60%.
Policy-ID (Policy 1)
Event (Load Proxylet), **Event Originator** (User)
Condition (If "Total CPU time load" less than 60%)
Policy Originator (AS Operator)
Action (Send Load proxylet event to EEP Controller)

5. SOFTWARE DEMONSTRATION

The resource manager (RM) and the security manager (SM) were demonstrated in two distinct real-life trials of the ANDROID project. The first trial [5], in Essen, involved the demonstration of the prototype resource monitoring component, consisting of the monitoring GUI and the measuring proxylet (R-let). The resource GUI was used to launch the R-let at the remote active server (AS). AS was secured by the SM, which performed proxylet deployer authentication on the basis of the authentication policy. Monitoring of resources (CPU, memory) on the AS is then done by the R-let. The sequence of events in the demo is as follows. The SM receives a XML event from resource monitoring GUI instructing it to load the R-let. The SM parses the event, and checks the local policy store for a policy associated to this event. After authenticating the user that sent the event the SM loads the R-let via FunnelWeb. From then on the monitoring application can contact FunnelWeb for running proxylets or R-let for monitoring information.

The second ANDROID demonstration in Paris involved three scenarios demonstrating the security and resource manager functionality on the AS. The scenarios involved the enforcement of the security and resource management policies given in Figure 5 and Figure 7. The security and resource managers were implemented to interoperate sequentially, where the request to load the proxylet was first authorised by the security manager, and then the load request would be forwarded to the resource manager, which would invoke the chosen action on the EEP.

The first scenario involved the successful loading of the proxylet, where first the security manager performed the proxylet and deployer authentication as specified by the security policy in 5. Next, the resource manager verified that there are enough resources, using the resource monitor component. According to the resource management policy given in Figure 7, which specifies that the proxylet can be loaded if there is no more of 60% of CPU used, the resource manager loaded the proxylet via the EEP, since the 60% condition was met.

The second scenario involved the attempt to load an incorrectly signed proxylet. According to the security policy specified in Figure 5, the security manager denied the incorrectly signed proxylet to load on the AS.

The third scenario involved the authorisation of the security manager to run the proxylet, which was correctly signed. However, the proxylet was denied to run by the resource manager, since the 60% CPU condition was not met. The condition was not met because a dummy resource-consuming proxylet was running on the AS, which was consuming 100% of the CPU resources.

The security and resource manager screens are captured on Figure 8. The top part of the screen shows the security manager, with the Keystore and policy file locations. The bottom part shows the resource manager. On the left side, the AS CPU utilisation (in this case 100%) and the relevant policy location are depicted. The right hand side of the screen shows the resource manager log, where it can be seen that the proxylet was denied to load since the CPU utilisation condition was not met.

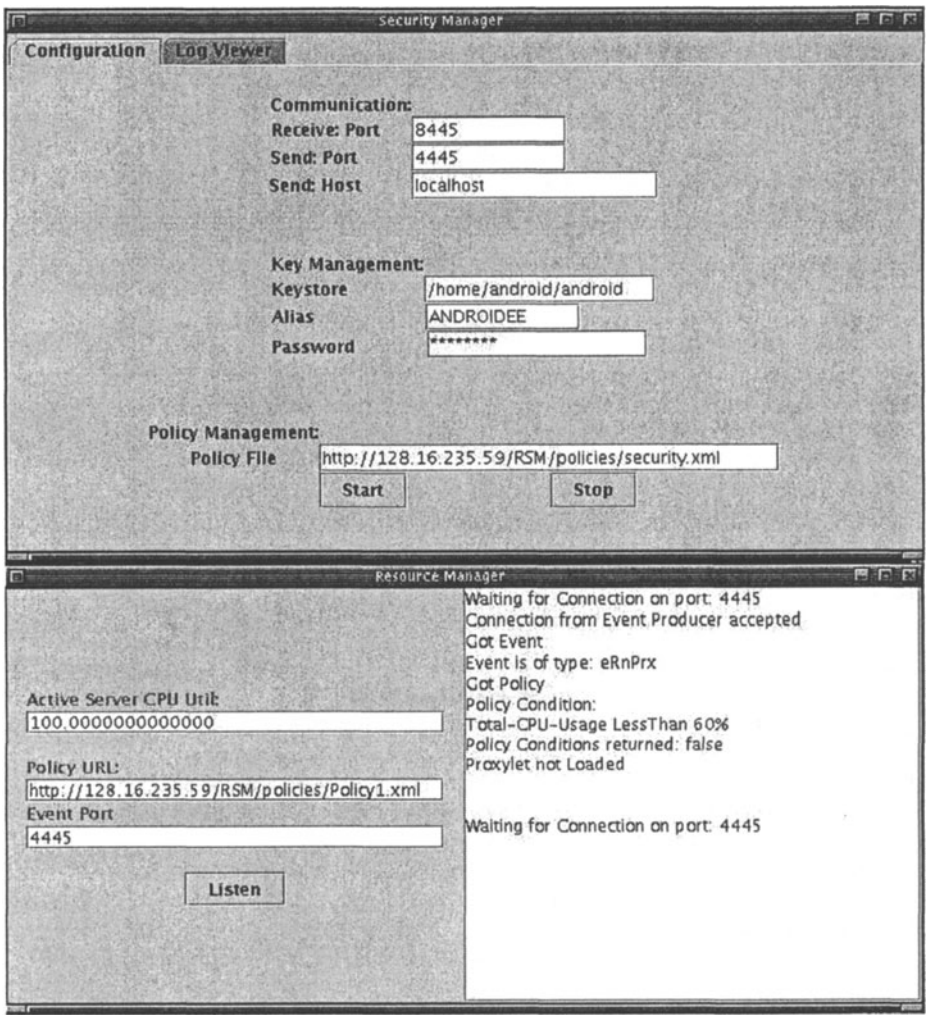

Figure 8 - Security and resource managers - demo screens

The use of the policy-controlled security and resource managers in the ANDROID project demo demonstrated their applicability in the real-life trial. Both components effectively demonstrated the policy-enforcement on the ALAN active servers, including proxylet and deployer authentication, and proxylet admission control based on the locally interpreted resource information.

6. CONCLUSION

This paper discussed the policy-based management architecture focused on managing the security and resource aspects of ALAN servers. The approach developed by the ANDROID project adopts a lightweight extensible policy and event schemas that effectively capture the information needed for management of large distributed configurable systems such as ALAN-enabled networks. The resource and security management architecture developed is fully policy-enabled. A thorough UML-based analysis and design allowed the development of the components necessary to support the policy enforcement on the ALAN active servers. This approach also allowed the specification of a wide range of policies relevant for the target active network scenarios. The security and resource management architecture and policies presented here were successfully deployed in a real-life trial of the project, being a distinct part of the larger ANDROID development.

Authors would like to thank Mike Fisher and Paul Mckee of BT for their work on the generic policy and event specifications; Ian Marshall of BT/UCL for discussions on some of the ideas, Ken Carlberg and Piers O'Hanlon of UCL for help during the ANDROID project demo.

7. REFERENCES

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

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

[3] 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.

[4] Sloman M., "Policy Driven Management for Distributed Systems", Journal of Network and Systems Management, 1994.

[5] O. Prnjat et. al., "Policy-based Management for ALAN-Enabled Networks"; IEEE 3rd International Workshop on Policies - Policy 2002, Monterey, CA, USA, June 2002.

[6] Marshall I. W., Hardwicke J., Gharib H., Fisher M., Mckee P., "Active Management of Multiservice Networks", Proceedings of NOMS 2000.

[7] Marshall I. W., Gharib H., Hardwicke J., Roadknight C., "A Novel Architecture for Active Service Management", IEEE/IFIP IM Symposium 2001.

[8] FunnelWeb http://dmir.socs.uts.edu.au/projects/alan/

[9] Damianou N., et. al., "Ponder: A Language for Specifying Security and Management Policies", Imperial College Research Report DoC 2001, January 2000.

[10] Natarajan R., McKee P, Mathur A.P., "A XML Based Policy-Driven Information Service", IEEE/IFIP Integrated Management Symposium (IM'2001), Seattle, May 2001.

[11] W3C, "XML Schema Part 0: Primer – W3C Recommendation, 2 May 2001", [www] http://www.w3.org/TR/xmlschema-0

[13] Ioannis Liabotis, et. al., "Policy-based Resource Management for ALAN", Proceedings of the 2nd IEEE LANOMS 2001.

[14] Prnjat O., et. al., "Integrity and Security of the Application Level Active Networks"; IFIP WATM'2001 and EUNICE'2001; Sept. 2001.

[12] W3C, "XML Schema Part 2: Datatypes – W3C Recommendation, 2 May 2001", [www] http://www.w3.org/TR/xmlschema-2