A Secure Infrastructure for Service Discovery and Access in Pervasive Computing

Andrej Cedilnik, Lalana Kagal, Filip Perich, Jeffrey Undercoffer, and Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
100 Hilltop Circle, Baltimore, MD 21250
{acedil1, fperic1, junder2, lkagal1, joshi}@cs.umbc.edu
phone: 410-455-3971
fax: 410-455-3969

Abstract

Security is paramount to the success of pervasive computing environments. The system presented in this paper provides a communications and security infrastructure that goes far in advancing the goal of anywhere - anytime computing. Our work securely enables clients to access and utilize services in heterogeneous networks. We provide a service registration and discovery mechanism implemented through a hierarchy of service management. The system is built upon a simplified Public Key Infrastructure that provides for authentication, non-repudiation, anti-playback, and access control. Smartcards are used as secure containers for digital certificates. The system is implemented in Java and we use Extensible Markup Language as the sole medium for communications and data exchange. Currently, we are a solely dependant on a base set of access rights for our distributed trust model however, we are expanding the model to include the delegation of rights based upon a predefined policy. In our proposed expansion, instead of exclusively relying on predefined access rights, we have developed a flexible representation of trust information in Prolog, that can model permissions, obligations, entitlements, and prohibitions. We have explored the use of delegations of these concepts to extend the web of trust. In this paper, we present the implementation of our system and describe the modifications to the design that are required to further enhance distributed trust. Our implementation is applicable to any distributed service infrastructure, whether the infrastructure is wired, mobile, or ad-hoc.

Keywords: Pervasive Computing, Security, Distributed Trust, SmartCards, Extensible Markup Language.
[1]

[1] Technical Report,TR-CS-01-12, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County

1 Introduction

As computing becomes pervasive people will live in Intelligent Space we will work in "SmartOffices" and live in "SmartHomes". An office presents an interesting setting to explore the intelligent space scenario. Often it is one of many rooms in a building of many floors, and the occupants may be highly mobile within this limited environment. Intelligent Spaces provide services and resources with which the user will interact through a laptop computer, PDA, cell phone or some other computational device using short range wireless communications such as Bluetooth, IEEE 802.11, or Infrared.

The ubiquitous computing paradigm calls for access to computational and computer controlled resources at anytime from anywhere. In furtherance of the goals of ubiquitous computing, "SmartOffices", "SmartHomes", and Intelligent Spaces have been the focus of research efforts on both the academic and industrial fronts. Project Centaurus [9] [8], is designed to minimize the load on portable devices and provide a media independent infrastructure and communication protocol for the provision of services. Designed for operation within pre-defined space, users can access services registered within that same space. Project Centaurus is responsible for maintaining a list of available services, and executing them on behalf of any user requesting them. Moreover, it minimizes resource consumption on a user's device by avoiding the need to have the services installed on each device that wishes to use them, which is advantageous for most resource-poor mobile Clients.

Our work, referred to as Centaurus2, extends the work completed in Project Centaurus by deploying decentralized services throughout an enterprise. Service access is made possible through a service management hierarchy that utilizes strong cryptographic controls that provide for authentication, anti-replay prevention, and non-repudiation. Additionally, Centaurus2 uses a capability management system to enforce access control to services, and smart-card technology to facilitate security within the system.

Like Project Centaurus, Centaurus2 employs *Communications Managers*, *Service Managers, Services*, and *Clients* (Services and Users). In Project Centaurus a grouping of a Service Manager, Services and Users operate as an autonomous system at any given instance, **while in Centaurus2 access to services from any user to any service is made possible even though the particular user and service may be connected to disparate Service Managers.** Centaurus2 accomplishes remote access to services through a hierarchical-based service management infrastructure. In this hierarchy, all Service Managers are connected, via a tree-like structure with the top-level Service Manager defining the root of a given domain. The primary feature of the hierarchical configuration of service management is the relegation of services on the basis of domains, buildings, floors, and particular rooms or areas. Additionally, Centaurus2 defines inter-domain relationships where a user in her "SmartOffice" is provided with the ability to access services in her "SmartHome".

Communication between Users and Services is performed via the hierarchy of Service Managers and is medium independent. Addressing is executed throughout the use of a "handle" that uniquely identifies placement within the hierarchy. Additionally, Centaurus2 enables inter-domain service access through the use of a bridge. The bridge is established between the root Service Managers of separate domains. This bridge allows a user in their SmartOffice to reach services running in their SmartHome. Security is paramount to the ubiquitous computing paradigm. Accordingly, security is fundamental to Centaurus2, which employs the *Centaurus2 Certificate Authority* and the *Centaurus2*

Capability Manager. Using a simplified and lightweight Public Key Infrastructure, trust management within Centaurus2 is distributed across all entities within the Centaurus2 system. Centaurus2 uses smart-card technology for digital certificate and key storage as well as for cryptographic functionality.

In Centaurus2 users and services are all treated equally as Clients enabling a user device that accesses services to also offer its own services. For example, a mobile device used by a delivery vehicle to access route information could also run a GPS service. Service Managers, through which Clients connect, broker requests and are responsible for authenticating Clients and enforcing access control; consequently, the Client is only concerned with a reciprocal authentication with its Service Manager. Moreover, even though a user has access to a service and the request for service has been authorized and delivered the service is the final arbiters as to who may utilize it.

The concepts we employ are applicable across the broad spectrum of ubiquitous and pervasive computing environments because Centaurus2 provides an infrastructure for securely reaching resources at anytime from any place irrespective of the underlying communications protocol.

Though this architecture solves the issue of controlling access to services in a "SmartSpace", it does not accommodate users that are foreign entities, that is entities that are not known to the system in advance. Also, access rights tend to be rather static, as clients are not able to receive permission to access a Service to which they are not pre-authorized. To overcome these issues we propose the incorporation of some of our previous work on trust management [7] into our working model. This prior work includes the development of policies for trust management, including this information into an entity's certificate. It also includes checking the entity's certificate to verify that it has certain rights based on the policy, and is permitted the delegation of rights to third parties. This is different from static access control lists, because of access control only works for known entities and deferring of access rights is not possible.

This paper is organized as follows: Section 2 discusses other research and technologies briefly comparing Project Centaurus and Centaurus2 to similar projects. Section 3 details the system design and architecture. Section 4 describes our implementation, Section 5 discusses our ongoing work and describes our proposed design for including distributed trust and we conclude the paper with Section 6, which is a brief summary of our work.

2 Related Work

Though there are several academic and commercial projects that are aimed at realizing the *SmartSpaces* scenario, none of them use *distributed trust* as a way to resolve the complex security issues. So, after discussing some of the projects related to *SmartSpaces*, we will briefly describe some work done on *distributed trust*.

2.1 SmartSpaces : Related Work

Some of the projects dealing with *SmartSpaces* are the joint Unisys Corporation/Orange [1] experimental house in Hertford, England, UC Berkeley's Project Ninja [4] [5], the University of Washington's Portolano project [3], and Stanford's Interactive Workspaces Project [2].

As demonstrated by the Unisys/Orange project, the concept of SmartHomes is transitioning from the purely

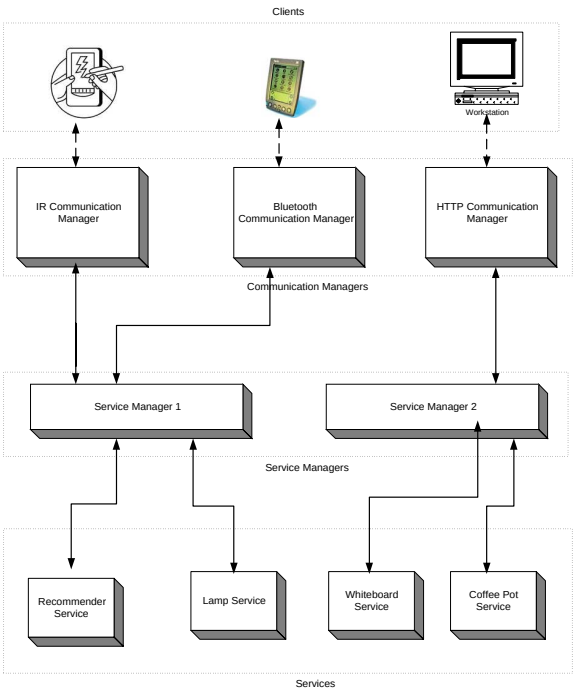

Figure 1: Centaurus System Components

academic to industry oriented research. The Unisys/Orange project is an experimental "intelligent" house that responds to voice commands to "dim the lights' or "turn up the volume on the television". In addition to voice, the home owner can interface with the house through a Wireless Application Protocol (WAP) telephone, web browser, or a Personal Digital Assistants (PDA). The Unisys/Orange project demonstrates that as ad-hoc mobile systems are developed and introduced they will be some combination of both wired and wireless communications. Accordingly, security must also be a paramount for success and public adoption of any system.

In the Centaurus project [8], the main design goal is the development of a framework for building portals to services using various types of mobile devices. Centaurus provides a uniform infrastructure for access to heterogeneous hardware and software components. It uses a language based on XML as the sole data exchange format between the service requester and service provider. This language called Centaurus Capability Markup Language (CCML), provides an extensible and simple content description that enables the creation of a user interface.

The Centaurus Service Manager is designed to be decentralized where Services can dynamically join and leave the system. A Service Requester (user of a service) can either be an end-user or another service. This hybrid architecture allows one service to be a composition of many services. Figure 1, illustrates the relationship between system components in the Centaurus model.

Another important research project is UC Berkeley's Project Ninja [4], [5] which employs Capability Managers, Certificate Authorities, and a hierarchy of Service Discovery Service Servers. However, unlike Project Centaurus,

it does not delegate state management to the Services themselves nor does it allow the Service Manager to serve exclusively as a cache. This approach is at a disadvantage because as the complexity of distributed state management increases the fault tolerance of the system decreases. For security and information assurance Ninja utilizes encryption between all entities within the system. This implies a high computational overhead on the endpoints of the communication regardless of whether the endpoint is a PDA, cell phone, or a powerful workstation. Centaurus2 does not make the assumption that the end points are computationally robust and instead relies on a simplified Public Key Infrastructure (PKI). The entities in the Centaurus2 system enjoy non-repudiation, authentication, and protection from replay attacks vis-à-vis the simplified PKI while access control to services is provided by the Centaurus2 Capability Manager(s). Moreover, while Project Centaurus and Centaurus2 are protocol and communications medium independent, Ninja is not.

The Portolano Project [3] focuses on User Interfaces, Distributed Services, and Infrastructure. Specifically, the Portolano vision of the user interface is one where the focus is away from the execution of explicit user commands, instead making use of autonomous agents that act on the user's behalf. Likewise, they propose that in order to support wider applicability distributed services need to be more openly organized into extensible horizontal layers instead of vertical integrated monolithic services. In Centaurus2 this extensibility is accomplished by the ad hoc nature in which a service-Client can register with any Service Manager within its domain making its service available to all authorized clients regardless of location.

Stanford's Interactive Workspaces Project [2] which endeavors to provide a system for interconnecting and integrating heterogeneous COTS legacy devices and software components. In addition, to provide interoperability, their endpoints communicate through a mediating infrastructure that transforms data so it will be compatible from one device type to another. The concept of data translation differs significantly from the Centaurus approach where XML is used as the sole format for data exchange.

2.2 Distributed Trust : Related Work

Matt Blaze's PolicyMaker [13] is probably one of the first forays into *distributed trust*. Though the concept has its roots in Pretty Good Privacy (PGP) [16], Simple Public Key Infrastructure (SPKI) [6], Simple Distributed Security Infrastructure (SDSI) [15] and Role Based Access Control (RBAC) [11] [12].

PGP [16] is a simple way of sending secure email using a *web of trust*, without exchanging a key and without a central authority. In PGP, a keyholder (an individual associated with a public/private key pair) learns about the public keys of others through introductions from trusted friends. The largest problem associated with PGP is key distribution and management.

The Simple Public Key Infrastructure (SPKI) was the first proposed standard for distributed trust management [6]. This solution, though simple and elegant, includes only a rudimentary notion of delegation, which is crucial to the developed of *distributed trust*.

PolicyMaker [13] is able to interpret policies and answer questions about access rights. Unfortunately, the development of policy is slightly complicated and not easy for non-programmers to use. This poses quite a problem, as it is generally non-programmers who will define the policies.

Role Based Access Control [11] [12] is probably one of the best known methods for access control, where entities are assigned roles, and there are rights associated with each role. Unfortunately, this is difficult for systems where it is not possible to assign roles to all users and foreign users are common.

The above mentioned models are very powerful, however they do not meet all the requirements of trust management. Generally security systems should not only authenticate users, but also allow users to delegate their rights and beliefs to other users securely and provide a flexible mechanism for this delegation. The above systems either support only authentication ignoring delegation altogether, or support delegation to some extent without providing the flexibility needed, or do not provide sufficient restrictions on delegation of rights.

We drew on the key points of most of the above-mentioned schemes and designed an infrastructure that uses X.509 certificates and policies to enforce security. A policy contains basic/axiomatic rights, rights associated with roles, rules for delegation, and rules for checking the validity of requests. Our system will allow an entity in the system to delegate any right that it may have. Whether these delegations are honored depends on the policy. Constraints can be added to both the actual delegation and to the delegatee, tightening control on the rights and permissions. In our model, we use a *redelegatable* flag that controls whether the permission can be further delegated. We have found that these features address the main issues of trust management, authentication and delegation, successfully.

3 Design

Our system is designed to control access to services within a *SmartSpace*. Centaurus2 is designed as a framework so that clients (users and services) can move, attach, detach, move and re-attach at any point within the framework. Additionally, we provide a bridging mechanism so that a user in one domain can reach a service in another domain provided that the user has the appropriate permissions.

There are five functional components within the Centaurus2 system. *The Centaurus2 Certificate Authority* is responsible for generating x.509 version 3 digital certificates for each entity in the Centaurus2 system and for responding to certificate validation queries from Service Managers. The *Centaurus2 Capability Managers* maintains a database of the group membership of Centaurus2 entities and answer requests for group membership. Next is the *Communications Managers*, which provides a communication gateway between a Client device and a Service Manager. Its sole purpose is to abstract and translate communications protocols. The fourth component is the *Service Manager*, which brokers requests between registered user-Clients and service-Clients. Finally, users and services are treated equally as *Clients*. Access rights of others to a particular client entity are exclusively determined by the entity itself; and set when it registers with a Service Manager. This equality of users and services is a minor variation from Project Centaurus original design; however, it allows a Client to access services while at the same time providing some its own services to others.

3.1 Centaurus2 Certificate Authority

The Centaurus2 Certificate Authority is used to produce x.509 version 3 digital certificates [14]. In Centaurus2 certificate request and issuance is ancillary to the system. When a certificate request from a Centaurus2 entity

is filled, the entity receives its requested x509 v3 certificate signed by the Certificate Authority and the Certificate Authority's self signed certificate, which is subsequently used to validate other entities' certificates. These certificates are stored and protected on a client's smartcard.

Certificate generation and signing is typically a one time occurrence for any entity within the Centaurus2 System. In a typical PKI the Certificate Authority makes its registrant's public certificates available in an on-line repository and provides an on-line Certificate Revocation List (CRL) where inclusion indicates that a given certificate is, for one of many possible reasons, invalid. As previously stated, Centaurus2 uses a simplified PKI. In Centaurus2 each entity presents its certificate to its Capability Manager when it registers. Rather than use a CRL to signal a problem with an entity, the entity's entry in the Capability Manager is blocked, consequently preventing all access by that entity to the Centaurus2 system. This precludes the necessity of maintaining a CRL, which must be signed by the Certificate Authority each time it is modified.

A Centaurus2 Service Manager verifies the authenticity of its stored copy of the Certificate Authority's certificate by sending the Certificate Authority a validation query. The Certificate Authority replies to the query with $E_{privatekey}((verifier))$. In Centaurus2 the Certificate Authority's certificate SHA-1 message digest is used as the verifier. To verify the validity of its copy of the Certificate Authority's certificate the Service Manager tests if:

$$(CertificateAuthority'scertificateSHA-1messagedigest) = D_{publickey}(E_{privatekey}((verifier))).$$

When the test passes it assumes that the copy of Certificate Authority's private key is valid and any object signed by that key is also valid.

Our Lightweight PKI, in contrast to the traditional PKI, does not maintain a CRL, does not transmit its key via the network, and does not distribute user's certificates or public keys. Rather, the Service Manager verifies that its copy of the Certificate Authority's certificate remains valid. This is done without transferring any keys or certificates via the network. In turn the Service Manager will ensure that all certificates that it receives from clients have been signed by the Certificate Authority.

3.2 Centaurus2 Capability Manager

The Centaurus2 Capability Manager is responsible for maintaining and communicating group membership(s) of all entities in the Centaurus2 system. Entities include Service Managers and Clients (users and services). Group membership may be as general as "*umbc.edu*" (meaning that only entities in the group umbc.edu are allowed access), more restrictive as "*cs.umbc.edu*", or even so granular as only the named Client "*ajoshi*", which implies that only the named Client is allowed to access a particular service.

When the Centaurus2 Capability Manager is initialized it reads its x.509 v3 digital certificate and its PKCS#11 [10] wrapped private key from a secure file and stores it into local memory. It also reads and indexes the capability file containing the group membership of all entities within the system, as well as storing the time stamp of the capability file.

When a Service Manager's group membership request is received, the Centaurus2 Capability Manager compares

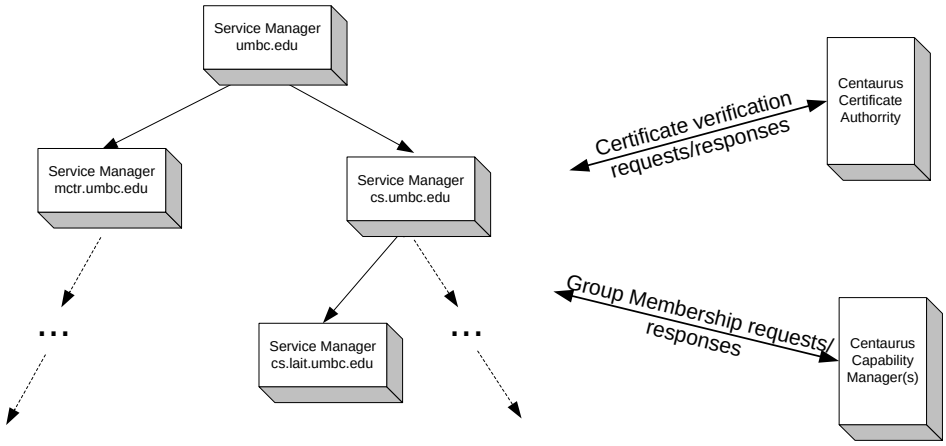

Figure 2: Service Manager, Capability Manager, and Certificate Authority Overview

the current time stamp on the capability file with the time stamp of the last file read, if they are not equal it re-reads the capability file. This feature allows for a dynamic administration, to include rights revocation, of the capability file.

In response to a group membership request, the Centaurus2 Capability Manager sends a message containing the subject's group memberships. The response is digitally signed with the Centaurus2 Capability Manager's private key.

Figure 2 shows a high level view of the Centaurus2 Certificate Authority, Centaurus2 Capability Manager(s), and Service Managers. It should be noted that there could be multiple Capability Managers where one Capability Manager serves a cluster of Service Managers. In the event of multiple Capability Managers each instance will replicate the capability database. During initialization, each Service Manager learns the location of its Capability Manager from its configuration file and communicates with that Capability Manager directly. At the first communication exchange between the Service Manager and the Capability Manager, the Service Manager requests and validates the Capability Manager's certificate, which it receives encoded in a signed CCML message.

3.3 Centaurus2 Service Manager

The Service Manager is responsible for processing Client Registration/De-Registration requests, responding to registered Client requests for a listing of available services, for brokering Subscribe/Un-Subscribe and Command requests from user-Clients to service-Clients, and for sending Service Updates to all subscribed users whenever the state of a particular service is modified.

Service Managers are arranged in tree-like structure and form the core of the Centaurus2 system. Service Managers are identified by their "locations", or handles. With the exception of Group Membership requests to the Capability Manager and certificate validation requests to the Certificate Authority all messages are sent and routed through the hierarchy of Service Managers using the handle to determine where to forward each message.

All Clients (users and services) rely upon the Service Manager to which they are registered to enforce security,

access control, and to broker requests for services. Consequently, each Client is only concerned with the trust relationship with its immediately connected Service Manager. In turn, Service Managers establish trust relationships with each other. Consequently, trust between Clients is transitive through the Service Managers. Figure 3 illustrates the relationship between a Client-User, Service Manager, and Client-Service within the context of single Service Manager. The Communication Manager is hidden in the cloud labeled "Any Medium".

When a Service Manager initializes, it reads its handle, its parent's handle, its Capability Manager's address, the Centaurus2 Certificate Authority's address, and the handle of pre-selected subset of Service Managers from a configuration file. In our implementation, handles are of the form "*umbc.edu*", "*cs.umbc.edu*", or "*lait.cs.umbc.edu*", etc. Each Service Manager starts with its own digital certificate and corresponding private key, and the digital certificate of the Centaurus2 Certificate Authority. Upon start up the following sequence of events occur:

1. Send a certificate verification request to the Centaurus2 Certificate Authority to ascertain that the local copy of the Certificate Authority's certificate is valid.

2. Receive a signed certificate verification response from the Centaurus2 Certificate Authority. Verify the signature of the response using the stored certificate of the Centaurus2 Certificate Authority; since we use the certificate SHA-1 message digest as a nonce, verify the value of the SHA-1 message digest contained in the returned message to that stored in the Certificate Authority's certificate for equality.

3. Send a certificate request to its Capability Manager.

4. Receive a certificate response from the Capability Manager, verify that the certificate contained in the message was signed by the Centaurus2 Certificate Authority and also verify that the signature of the message is valid.

5. If the Service Manager is not the root Service Manager of the domain (handle $\neq$ parent's handle) register with the parent Service Manager by sending a CCML registration message that contains a copy of the registering Service Manager's digital certificate that has been converted from its ANS.1 encoding to hexadecimal string. The entire message is signed as shown by Equation (2), which is also converted to hexadecimal string and inserted into the CCML message.

6. The receiving Service Manager extracts the certificate, verifies the signature and verifies the expression using Equation (3) to prevent replay attacks. Here Δ is the maximum round trip time of any message in the Centaurus2 system. When all points are successfully performed, the receiving Service Manager registers the sending Service Manager in its database of pending Clients and sends a group membership request to the Capability Manager.

$$\text{SHA-1(CCML message)} \tag{1}$$

$$E_{privatekey}(\text{SHA-1(Original Registration Request CCML message)}) \tag{2}$$

$$TimeStamp(\text{Original Registration Request CCML message}) + \Delta \cong TimeStamp(\text{Service Manager}) \tag{3}$$

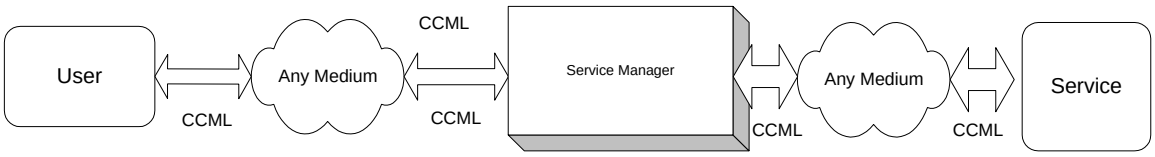

Figure 3: Clients within the context of a single Service Manager

7. Once the group membership response is received from the Capability Manager the Client registration is changed from pending to registered and a digitally signed registration acknowledgment containing the parent Service Manager's digital certificate is sent to the registrant. Note: the Capability Manager's entry for a Service Manager's group memberships only contains the Service Manager's own handle.

8. Once a registration acknowledgment has been received the registering Service Manager accepts communications from Clients.

9. Each Client registers with any Service Manager in a similar fashion. This process is covered in detail under the Client subsection.

Generally in a PKI system, certificates are made available in an on-line repository. Consequently, when a user needs an entity's digital certificate it is requested from such a repository and assumed to be valid once it is received and has verified the certificate signing chain along the entire path to the top level signature authority. In the general PKI implementation certificate repositories and CRLs have a high degree of administrative overhead. This overhead and the accompanying network traffic imposed by certificate acquisition and the signature verification is mitigated in Centaurus2 by its simplified PKI framework. As previously stated, each Centaurus2 entity possess its certificate and presents that certificate upon registration. All entity's attributes, to include the validity of its certificate, are validated through a single query to the Capability Manager. In addition, the authenticity of the presented certificate is verified by ensuring that the certificate was signed by the Certificate Authority.

The Service Manager maintains a database of Client profiles for all entities registered with it. Information contained in the profile includes the Client's certificate (or the certificate of the Client's Service Manager if the Client did not initially register with the Service Manager), group memberships, location (the Service Manager to which it is immediately connected), name, and permissible access groups.

3.4 Client

Client registration and subsequent access to services occurs on an ad hoc basis. All Clients must register with a Service Manager prior to accessing any services or making its services available. At registration, a Client will, in addition to sending its digital certificate, transmit a list of group memberships required for other Clients to access its services. The Service Manager will store the Client's certificate, list of access memberships, and the list of group memberships (acquired from the Capability Manager) for the Client in the Service Manager's user profile database. Generally, a user-Client will send an empty access list indicating that no other Client may be granted access to it,

whereas a service-Client will include a list of groups where membership in at least one of the listed groups is required for access to that Client's service.

The following process enumerates the sequence of events during Client registration to a Service Manager:

1. When a Client initially registers with a Service Manager, the *Source Location* and *Destination Location* fields of the registration message are blank. Otherwise the Source Location will contain the name of the Client's immediately connected Service Manager and the Destination Location will contain the location of the additional Service Manager to which the Client wishes to register.

2. Additionally, during initial registration, the Client registration message will include a copy of the Client's digital certificate. The initial and all subsequent registration requests contains a list groups wherein membership allows access. This message is digitally signed by the Client.

3. Upon receipt by the Service Manager, if the *Source Location* and *Destination Location* fields are blank the Service Manager will follow the same sequence of steps that a parent Service Manager follows when registering one of its child Service Managers. If the *Source Location* and *Destination Location* fields are not blank the Service Manager will verify the message, place its own digital certificate into the message, re-sign the message and forward the registration message towards the destination Service Manager.

4. When a Service Manager receives the registration message described in the previous step, it follows the same sequence of steps that a parent Service Manager follows when registering one of its child Service Managers.

5. When a registration response message from a non immediately connected Service Manager is received by the Client's immediately connected Service Manager the Service Manager will verify the distant Service Manager's certificate and create a profile for the distant Service Manager in its user database.

6. Included in the response to its initial registration message, the client receives a copy of the Service Manager's digital certificate. Recall that all entities in the Centaurus2 system receive both their requested certificate and a copy of the Certificate Authority's certificate in response to a certificate request. The client uses the copy of the Certificate Authority's certificate to authenticate the certificate received from the Service Manager.

Once a Client has successfully registered, it is given an interface to all services to which it has rights to and which are also registered to the same Service Manager. This includes an interface to all other Service Managers that the Client's Service Manager is aware of.

Figure 4 depicts the Client located at Service Manager *cs.umbc.edu* as also registered with the Service Manager *lait.cs.umbc.edu* and using the service registered there. Note that a Communications Manager logically resides within the area labeled "Any Medium".

Figure 5 illustrates a scenario where: User-1 is registered with both Service Manager-3 and Service Manager-2. Communications from User-1 to Service Manager-2 will pass through Service Manager-1, however, Service Manager-1 will not verify those messages. The Service Manager to which the Service is registered verifies all messages from a user to a Service.

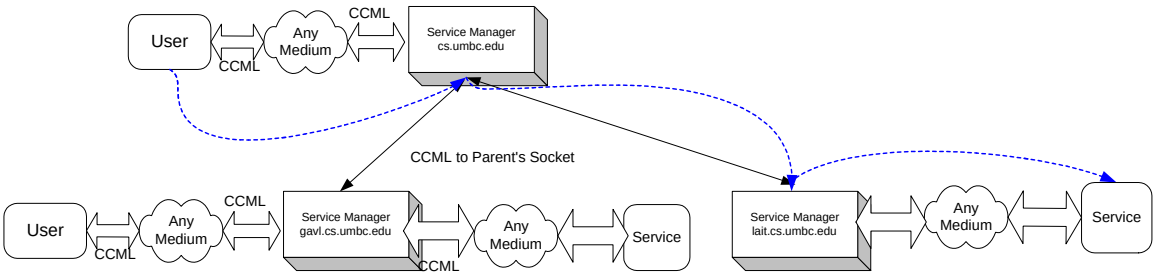

Figure 4: Multiple Service Manager registrations

3.5 Entity Communication and Service Discovery in Centaurus2

Clients (users and services) are able to communicate irrespective of the underlying communications medium. Accordingly, to ensure communication through the hierarchy of Service Managers, Centaurus2 uses handles for addressing. For example, a User registered to the Service Manager at LAIT.CS.UMBC.EDU wishing to subscribe to a Service registered at MCTR.UMBC.EDU would send the request addressed as follows:

Source name:	User
Source Location:	LAIT.CS.UMBC.EDU
Destination Name:	Service
Destination Location:	MCTR.UMBC.EDU

The CCML message (registration, subscribe, or command) will be sent to the Clients's Service Manager who in turn will re-sign it and forward the request up or down the hierarchy of Service Managers or to a immediately connected Client. Forwarding is based upon comparing the handle of the destination location to the handle of the present location. The response will be returned in the same manner.

As previously stated when a Client (service) registers in Centaurus2 it transmits a list of groups where membership in those groups implies that another Client (user) is permitted access to the service. Additionally, the Service Manager requests a list of group memberships from the Capability Manager to assure that the service is permitted to connect to the Service Manager.

Although a user-Client is only given an interface to services to which the user is authorized, all requests to access Clients are re-verified to ensure that the user does possess the necessary rights. This validation for access rights is shown on Equation (4).

$$Client.groupmembership \cap OtherClient.accessgroups \tag{4}$$

Inter-Domain Communication is the interaction between Clients each of which fall under separate root Service Managers. Inter-Domain communication is based upon an agreement between the two communicating Domains. When two domains agree to bridge, one root Service Manager (SM-A) will register to the corresponding root Service Manager (SM-B), and in addition will exchange acceptable access groups in its CCML registration message.

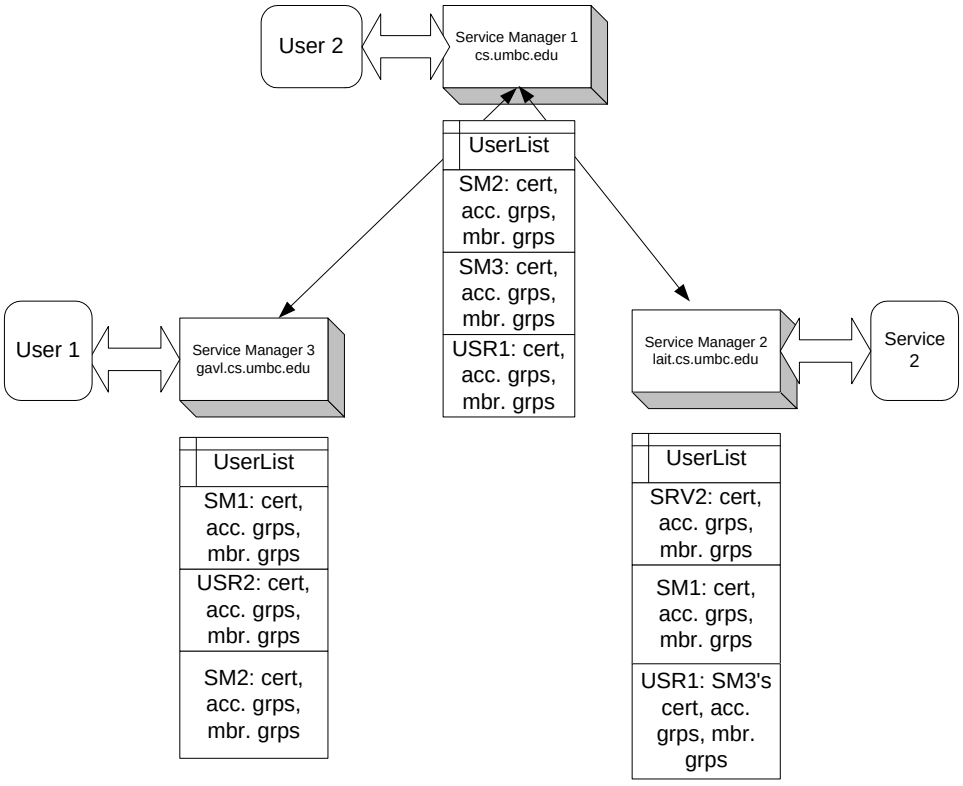

Figure 5: Example of Multiple Service Manager Registration showing Service Manager UserList entries

The Capability Manager for SM-A will include SM-B in its profile list, which includes the requisite group memberships needed for access. Consequently, SM-B will only be visible to those users operating within the domain of SM-A who have access rights to SM-B. When a Client under SM-A with rights to some resource on SM-B registers with the SM-A it will see SM-B, and upon registration to SM-B it will see all of SM-B's attendant services. SM-A will resign any messages destined for the domain rooted at SM-B, in turn SM-B will verify the signature, resign the message and forward it to the appropriate Service Manager within its domain.

4 Implementation

Our goal of instantiating security, specifically: authentication, authorization, non-repudiation, and anti-playback, as a primary component of Centaurus2 was heavily influenced by the desire to make security as unobtrusive as possible. We believe we have designed a system that is both highly secure and where security is nearly transparent to the user. We accomplished such task through the use of following tools and mechanisms:

1. A simplified Public Key Infrastructure
2. X.509 version 3 Digital Certificates
3. Smart Cards

4. PKCS #11 containers for private keys stored on computing devices

5. Capability Matrix

The sole action required of the individual entity (i.e., Service Manager, Service, Centaurus2 Capability Manager, or Client) to ensure secure operation is the one-time entry of their pass phrase during system initialization to enable the reading of their private key into memory. Accordingly, if the entity fails to enter the correct pass phrase any and all messages sent by that entity will be ignored. Moreover, the entity is only provided feedback from the system once they have been positively identified and verified.

4.1 Simplified Public Key Infrastructure and X.509 version 3 Digital Certificates

Typically a Public Key Infrastructure consists of a Certification Authority (CA) to include Registration Authorities (RA), certificate holders, users that validate digital signatures and their certification paths from a known public key of a trusted CA, and repositories that store and make available certificates and Certificate Revocation Lists (CRLs).

Accordingly, our simplified Public Key Infrastructure consists of the Centaurus2 Certificate Authority having a self-issued and self-signed certificate containing its public key, and a certificate issued to each entity (Service Manager, Client, and Centaurus2 Capability Manager) in the system. That is each entity in the system is a certificate holder.

Rather then implementing certificate repositories and a Certificate Revocation List (CLR), each entity has its own certificate and each Service Manager has copy of the Certificate Authority's certificate. Upon start up, and optionally at configurable intervals, a Service Manager verifies the Certificate Authority's certificate that it possesses. In turn this certificate is used to verify that each certificate presented to the Service Manager has been signed by the Certificate Authority. In addition, instead of maintaining a CRL, the Capability Manager(s) has an entry for each valid user on the system. The absence of an entry in the Capability Manager's capability matrix for any entity blocks that entity from any and all accesses to the system.

Smart Cards are used as a security provider for a Client's cryptographic functionality as well as storage for his digital certificate and private key. At initialization the Client unlocks the card by entering a Card Holder Verification Value (CHV) gaining access to the card. The digital certificate is exported from the card and is available for presentation whenever the Client registers to a Service Manager. The Client then makes an SHA-1 message digest of CCML messages. This digest is imported to the card where it is digitally signed using the private key, and the signature is exported for placement into the CCML message.

PKCS #11 describes syntax for private-key information. Private-key information includes a private key for some public-key algorithm and a set of attributes. The PKCS #11 standard also describes syntax for encrypted private keys. A password-based encryption algorithm, as described in PKCS #5, is used to encrypt the private-key information. The private key of Service Managers, the Centaurus2 Capability Manager, Services and those Clients that do not have smart cards, is stored in a PKCS #11 container in a regular File System.

4.2 Client Capabilities

The Centaurus2 Capability Manager is responsible for responding to requests for group membership from Service Managers. Accordingly, the Capability Manager maintains a database of all users (including Service Managers) and their group memberships. Group membership can be of the form "*gavl.cs.umbc.edu*", "*perich.net*", or may be as granular as the individual Client, i.e.: "*ajoshi*". Each Service Manager determines the entity's access rights based upon group membership and forwards requests to Clients based on those rights. The Client trusts the Service Manager to only send commands from Clients having the appropriate rights. The Client, however, has ultimate jurisdiction on responding to those commands and may, for some reason, choose to ignore the other Client's command. Moreover, during registration the Client sends the Service Manager a list of groups wherein membership is required in order to access its service.

4.3 Security Protocol

The Service Manager is responsible for ensuring the integrity of the Centaurus2 system. Each Service Manager has a copy of the Centaurus2 Certificate Authority's certificate. As will be explained below, the Service Manager to which a Service is registered is then responsible for authorizing CCML messages destined to the Service. The following describes the security protocol implemented in Centaurus2.

Upon receiving a Client certificate verify the certificate by ensuring that the certificate was digitally signed by the Certificate Authority's private key. When signing a message, compute the signature using Equation (2), convert it to hexadecimal string and insert it into the message. When verifying the message compare the message digest from Equation (1) with the decrypted signature from the Equation (2), and verify the timestamp using Equation (3).

If it is the initial registration the registrant generates a Registration Request that includes a copy of the registrant's digital certificate. The certificate is converted from ANS.1 to hexadecimal string and inserted into the registration message. The message is then signed by the sender. Upon receipt of the Registration Request one of the following five cases will hold and the Service Manager will respond accordingly.

1. If the registration request is from a child Service Manager. The parent Service Manager verifies the certificate and the message signature, and requests the group membership from the Capability Manager. Once everything is verified, it establishes a Client profile for the child Service Manager. The profile contains the registrant's digital certificate, which has been converted from hexadecimal to string to ANS.1 encoding, access groups, member groups, name, and location. It then transmits a registration response message containing the registering Service Manager's digital certificate. In turn the child Service Manager follows the same steps to verify its parent.

2. If the Service Manager is both the source and destination Service Manager, then the originator of the message is one of its immediately connected clients. The Service Manager verifies the certificate and the message's signature and requests the registrant's group membership from the Capability Manager. Once everything is verified, it establishes a Client profile storing the profile in its user database, and transmits a registration response message containing its digital certificate.

3. If the Service Manager is the source Service Manager but is not the destination Service Manager, this indicates that the registering Client has already registered with its nearest Service Manager. The source Service Manager verifies the digital signature of the message, places its certificate into the message, re-signs the message, and forwards it to the destination Service Manager.

4. If the Service Manager is neither the source nor the destination Service Manager, it forwards the message to either its parent or one of its children based upon a substring match of the destination handle and its handle.

5. If the Service Manager is the destination Service Manager and is not the source Service Manager, it verifies the certificate to ensure it was signed by the Certificate Authority, verifies the signature of the message, and requests the registrant's group membership from the Capability Manager. Once everything is verified, it adds the registrant's profile to its user database, and sends a registration response to the registrant containing its digital certificate. When the registrant's Service Manager receives the registration response it adds the sending Service Manager to its Client database, resigns the registration response, and forwards the response to the Client.

Note that if the source of a message is immediately connected to the Service Manager the Client's profile will contain a copy of the Client's digital certificate otherwise it will contain a copy of the Client's immediately connected Service Manager. All subsequent messages (de-registration, subscription request, command or update) will be verified using the certificate stored in the Client's profile in the destination Service Manager's Client database. Each Service Manager will ensure that a message forwarded to any of its immediately connected Clients is signed with the Service Manager's private key.

4.4 Operational Protocol

The following enumerates Centaurus2 message types, the initiator, the receiver and their resultant action:

1. *Registration Request* From: Client-Service Manager To: Service Manager. Contains the registrant's digital certificate and is signed by the registrant. If the registrant is already registered, it is first de-registered and access to all previously subscribed services are terminated and the registrant is re-registered.

2. *Registration Response* From: Service Manager To: Client. Signed by the sender. Transmitted once the group memberships are received from the Capability Manager.

3. *De-Registration* Request From: Client To: Service Manager. Signed by sender. The senders profile is deleted from the Service Manager, and all subscriptions are removed from Client-services.

4. *De-Registration* Response From: Service Manager To: Client. Signed by the sender, it is assumed that the destination does not receive this message.

5. *Group Membership Request* From: Service Manager To: Capability Manager. Sent to the Capability Manager from a Service Manager requesting the group memberships of some entity.

6. *Group Membership Response* From: Capability Manager To: Service Manager. Signed by the Capability Manager. Contains group memberships of the subject.

7. *Service List Request* From: Client To: Service Manager. Signed by the sending user-Client requesting a listing of available services.

8. *Service List Response* From: Service Manager To: Client (user). Signed by the sender, informs the Client of available services registered with that Service Manager.

9. *Subscription Request* From: Client (user) To: Service Manager. Signed by the sending user-Client requesting subscription to a particular service. In addition to verifying the signature the Service Manager verifies that the user has access rights to the requested service.

10. *Subscription Response* From: Service Manager To: Client (user). If the user has access rights to the requested service a subscription response is signed and sent to the user.

11. *Command* From: Client (user) To:Client (service). A request to some Client to change state. Signed by the user and received by the services Service Manager where both the signature and the user's access rights to the service are verified. Re-signed by the Service Manager and transmitted to the Service for final arbitration.

12. *Update* From: Client (service) To: Client (user). Signed by the Service and sent to the user via the Service Manager. The Service Manager verifies the signature, re-signs the message and forwards it to the User. Additionally, the Service Manager sends an update to all other users subscribed to the Service.

4.5 Configuration

Centaurus2 was implemented using Java. All Service Managers are initialized from the same class. All clients are subclasses of a common client class that provides call-backs for event notification and methods for triggering events.

Within one domain there were six Service Managers were configured in a hierarchy approximating the laboratories in computer science department at the University of Maryland Baltimore County. A second domain was established using a Service Manager in a SmartHome. We had service-clients and user-clients registered to each Service Manager in this system. Service-clients consisted of light services with controls for power on/off, intensity, and hue, a music service with controls for selection and volume, and a weather service that broadcast current weather conditions.

The Centaurus2 system performed without error in this configuration.

5 Ongoing Work

In this section, we will describe our future system that incorporates our augmented distributed trust model at its core.

Every entity will have a web page that contains all its trust information, eg. the delegations that it has received, or certain rules that it wants to abide by, etc. Every certificate contains the URL of the entity's trust information.

Each domain follows a certain set of rules for authorization, delegation, controlling access to Services, etc. called a policy. This policy consists of authorization policies and delegation policies. Authorization policies deal with the rules for checking the validity of requests for actions. An example of a rule for authorization would be checking the identity certificate of an agent and verifying that the agent has an axiomatic right. Delegation policies describe rules for delegation of rights. A rule for delegation would be checking that an agent has the ability to delegate before allowing the delegation to be approved. A policy also contains basic or axiomatic rights, and rights associated with roles. We introduce the concept of primitive or axiomatic rights, which are rights that all individuals possess and that are stored in the global policy. For example, every student has the right to access the school library, and anyone who owns a database has the right to delegate the right to read from/write to that database. These are basic rights that are not often expressed, but used implicitly. All policies are described in Prolog. A policy can be viewed as a set of rules for a particular domain that defines what permissions a user has and what permissions she/he can obtain.

Users of the system will be assigned roles. A role is defined as a collection of rights and duties [11] [12]. Roles are arranged in a hierarchy, so that rights can be inherited. An entity has a right if it is mentioned in the policy or if the right has been delegated to it by another entity that has the ability to delegate. Delegations generally flow downwards in the role hierarchy, and are from a higher role to a lower role. However our framework does not strictly adhere to role based access, and allows rights and delegations to be assigned to individuals and groups. This overcomes the drawbacks of static Access Control Lists and Role Based Access Control.

The Capability Manager will be augmented by the *Access Control Agent*, which is primarily responsible for trust management in this system. On initialization, it reads the policy and stores it in a Prolog knowledge base. All requests are translated into Prolog, and the knowledge base is queried. The policy consists of permissions which are access rights associated with roles, and prohibitions which are interpreted as negative access rights. The policy also contains rules for delegation and role assignments. A Client has the ability to access a Service if the Client has not been prohibited from accessing the Service by an authorized entity and if it has the pre-defined access right, role based access right or if some authorized entity has delegated this right to it. An entity can only delegate an access right that it has the ability to delegate. For example, client1 could have the ability to use service1, but not the ability to delegate the right to access service1 to any other client. Another example could be a Client, client2, who cannot access service1, but has the right to delegate access to service1. These would be part of the delegation rules that are contained in the policy. The Service Manager sends the Access Control Agent a Clients certificate. The Access Control Agent gets the role of the Client and the URL of the webpage containing its trust information. It reads the trust information and validates and verifies this information before adding it to its knowledge base. After this the Access Control Agent will be able to decide which Services the Client has the right to access.

When a Client needs to access a Service that it does not have the permission to access, it requests another Client, who has the right, for the permission to access the Service. If the latter Client does have the permission to delegate the access to the Service, the Client can send a delegate message, signed by its own certificate, to the requester. The requesting Client, adds this delegate statement to its web page containing all its trust related information. Then the requesting Client sends its certificate to the Service Manager. The Service Manager sends the certificate to the Certificate Manager to be checked. Then the Service Manager sends the certificate to the Access Control Agent. The

Access Control Agent reads the role of the Client and the URL of the trust web page from the certificate. It reads all the delegate statements on the web page and validates and verifies them. It makes sure that the delegator has the right to delegate, then it adds the permission for the Client to access the Service, but sets a very short period of validity for the permission. Once this period is over, the Access Control Agent informs the Service Manager, which has to go through the entire process of validation and verification again. This is very useful incase of revoked certificates, delegations or rights. If any one entity in the delegation chain loses the permission, then it is propagated down the chain very quickly, until everyone appearing in the chain after the entity loses the ability.

We are also looking at *distributed belief* as a way for the Access Control Agent to garner the required trust information. It is not always possible for a agent or entity to know its role in advance. The policy could include rules for belief as well. For example, the Access Control Agent would believe that entity1 had role1 in affiliation1, if two registered Clients said that this was the case.

6 Conclusion

Centaurus2 extended Project Centaurus through the organization of a hierarchy of service management which enabled making services available across an enterprise. This extension was accomplished while retaining the goal of a distributed system where users and services may attach to the system at will. The process of making services available to a broad spectrum of users necessitated the addition of security architecture. Services, like any resource of value, are vulnerable to exploitation and misuse if access to them is not adequately governed.

Centaurus2 utilized a simplified public key infrastructure that minimized run time key and certificate administration while at the same time provided a high degree of assurance to the process of authenticating users and services. Additionally, the Centaurus2 Capability Manager was utilized so that once a Client was authenticated the Client's access would be limited to those services to which the Client was entitled.

We also described the model for enhancing our distributed trust management. We are in the process of implementing this portion of the system. The end result of this effort will be an infrastructure for a ubiquitous computing environment that incorporates *delegated trust* to control access to a dynamic set of Services by an ever evolving group of entities.

References

[1] Orange and unisys build the house that listens. http://www.unisys.com/news/releases/2001/feb/02127058.asp.

[2] George Candea and Armando Fox. Using dynamic mediation to integrate cots entities in a ubiquitous computing environment. In *Second International Symposium on Handheld and Ubiquitous Computing 2000*, pages 248–254, 2000.

[3] Mike Esler, Jeffrey Hightower, Tom Anderson, and Gaetano Borriello. Next century challenges: Data-centric networking for invisible computing. In *Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99*, pages 256–262, N.Y., August 15–20 1999. ACM Press.

[4] Ian Goldberg, Steven D. Gribble, David Wagner, and Eric A. Brewer. The ninja jukebox. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS-99)*, pages 37–46, Berkeley, CA, October 11–14 1999. USENIX Association.

[5] Steven D. Gribble et al. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks (Amsterdam, Netherlands: 1999)*, 35(4):473–497, March 2001.

[6] IETF. Simple public key infrastructure (spki) charter: http://www.ietf.org/html.charters/spkicharter.html.

[7] Lalana Kagal, Tim Finin, and Yun Peng. A framework for distributed trust management. In *To appear in proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.

[8] Lalana Kagal, Vladimir Korolev, Sasikanth Avancha, Anupam Joshi, Timothy Finin, and Yelena Yesha. Highly Adaptable Infrastructure for Service Discovery and Management in Ubiquitous Computing Technical Report, TR CS-01-06.

[9] Lalana Kagal, Vladimir Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Project Centaurus: A Framework for Indoor Services Mobile Services. In *Proceedings of International Workshop on Smart Appliances and Wearable Computing IWSAWC, in the The 21st International Conference on Distributed Computing Systems (ICDCS-21), 2001*, Department of Computer Science and Electrical Engineering. University of Maryland Baltimore County, Baltimore, MD, April 2001.

[10] RSA Laboratories. PKCS 11-Cryptographic Token Interface Standard. January 1994.

[11] E. Lupu and M. Sloman. A policy based role object model, 1997.

[12] E. C. Lupu, D. A. Marriott, M. S. Sloman, and N. Yialelis. A policy based role framework for access control, 1995.

[13] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized trust management. *IEEE Proceedings of the 17th Symposium*, 1996.

[14] W. Polk D. Solo R. Housley, W. Ford. RFC 2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile. Janaury 1999.

[15] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rumpsession, 1996.

[16] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.