Vigil : Providing Trust for Enhanced Security in Pervasive Systems

Lalana Kagal, Jeffrey Undercoffer, Filip Perich, Anupam Joshi, Tim Finin and Yelena Yesha
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
email : {lkagal1, junder2, fperic1, joshi, finin, yeyesha}@cs.umbc.edu

Abstract

Computing today is moving away from the desktop, becoming diffused into our surroundings and onto our personal digital devices. Moreover, ad-hoc networks such as Bluetooth provide for spontaneous connectivity between computationally enabled devices within proximity to each other. In such *pervasive computing* environments, users expect to access resources and services at any time from anywhere. This expectation results in serious security issues, since devices are constantly interacting with others outside of their "home" environments. We describe the security challenges in pervasive computing, explaining why traditional security mechanisms fail to meet the demands of these environments. We use an agent-oriented paradigm to model the interactions between computationally enabled entities in such dynamic environments, and present an infrastructure that combines existing authentication features like Simple Public Key Infrastructure (SPKI) with notions of *policy driven interaction* and *distributed trust*, in order to provide a highly flexible approach for enforcing security policies in pervasive computing environments. We present an implementation of the system on a variety of handheld/laptop devices using Bluetooth/802.11, and include an ontology to describe principals, credentials and policies.

Index Terms

Distributed Trust, Digital Certificates, Service Discovery and Delivery, Delegation of Rights.

I. Introduction

Computing today is becoming pervasive. The computational capability of commonly used devices, like cellular phones and handhelds, and their ability to wirelessly communicate with other devices, are the forces driving the progress of pervasive computing. Computing capability will increasingly be embedded in all manner of manufactured artifacts, from roads and bridges, to home appliances, to even the clothes we wear. Wireless technologies are playing an important role in allowing the multitude of devices to spontaneously form short-range, short-term, *ad-hoc* networks. The various flavors of the IEEE 802.11 standard and Bluetooth are the technologies du jour, and their variants will likely dominate the near future. In order to support and maintain these networks, efficient routing and transport protocols are required. Traditionally, much of the research in academia

This research was supported in part by the IBM EECOMS program, the DARPA DAML program under contract F30602-97-1-0215,and by NSF through awards CCR0070802, IIS9875433

and industry has been, and continues to be, focused on the development and improvement of these protocols.

The next and perhaps most important step, from a user's perspective, is for useful applications to exist on the devices forming these networks, so that users can access these resources and services anytime and anywhere[38], [35]. Much of the work in this space has focused on allowing applications built for the wired world to run in the wireless domain. This work (typically based on a client–proxy–server model) has been developed by academia over the last five–six years in contexts such as web access from mobile platforms (for instance[17], [23], [5], [25], [30], [2], [18]) or transaction support for database access[8]. Variants of this approach are now emerging from several commercial companies in the form of systems that allow phone based "microbrowsers" or PDAs to access domain specific Internet content (headline news, stocks, sports etc.) from designated portals. Such an approach, however, is not suitable for the pervasive environment, where devices are peers and can be both providers and consumers of services. In other words, service provision is not a "one-way" affair from wired servers to wireless clients. Our research seeks to move beyond this model and provide a framework which uses the power of mobility and ad-hoc wireless connectivity to enable novel applications.

In these loosely coupled and highly dynamic environments users will be able to move around and remain connected to the devices that happen to be in their vicinity. These devices may be carried by other users or may be embedded in the infrastructure around the user. The resources they can access will thus change and be dependent on their current location as well as the composition of their vicinity. Moreover, these ubiquitous devices will be ever present and in constant communication, dynamically organizing themselves to cooperatively to do their user's bidding.

A very challenging problem is posed in those cases where the number of such devices is large, and the environment is unknown. In order to satisfy user requests, applications on a device *discover* services or information on other devices in their vicinity[1]. In some situations, information or services that are available on multiple devices are *composed* together before being presented to the user[7]. Users rarely *query* the system in the traditional database sense, but often declaratively specify things they are interested in. The system in turn needs to query data sources, most of which are volatile in as much the vicinity of a device keeps changing with movement thus changing available data sources[32]. Moreover, much of this interaction with other devices happens outside of a user's home environment, which means that devices in the vicinity cannot by default be assumed to be trustworthy[19]. The eBiquity group (*http://research.ebiquity.org*) at the University of Maryland Baltimore County, is involved in the design, development and evaluation of a framework that addresses these issues[7], [1], [32], [19]. This paper focuses on the security challenges and our proposed solution. We seek to address security issues in pervasive computing environments through the use of agent methodologies and distributed trust. We present *Vigil* as the security infrastructure for such pervasive computing environments. Our current implementation operates in a heterogeneous environment consisting of Bluetooth, Infrared, 802.11b and Ethernet.

Vigil is the extension and culmination of our two previous projects: *Centaurus* [22] and *Centaurus2* [36]. The main design goal of the Centaurus project was 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. Centaurus was designed for a single location such as a room in an office.

The Centaurus2 project extended the concepts presented in Centaurus by providing secure access to services across an enterprise, where the enterprise is connected by a logical hierarchy organized along the lines of departments and offices. 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, Centaurus2 provides a bridging mechanism so that a user in one enterprise can reach a service in another enterprise provided that the user is known to the system and has the appropriate permissions.

This paper is organized as follows: Section II provides a motivation for our infrastructure and we demonstrate, with an example, the usefulness of Vigil. Section III presents related work and discusses other research and technologies, briefly comparing Vigil to similar projects. Section IV details the design and architecture of the Vigil system. Section V explains important functionality of the system. Section VI details our implementation from both an operational and a security perspective. Section VII provides an analysis of our security and applications protocols. Section VIII explains some of the experiments we conducted in order to test the effectiveness of Vigil. Section IX discusses our ongoing work and presents a brief summary of our work.

II. Background & Motivation

Traditionally, stand-alone computers, or those networked within a (small) organization, rely on user authentication and access control to provide security. These physical methods use system-based controls to verify the identity of a person or process, explicitly enabling or restricting the ability to use, change or view a computer resource. However, these mechanisms are inadequate for the new set of problems that the pervasive computing paradigm introduces. Some of these problems include heterogeneity, the lack of centralized control and presence of foreign users. In particular, the system may not, at a given point in time, know exactly which resources to protect from which users because both the users and resources available in a vicinity dynamically change, and not all of the users that a resource encounters are “pre-known” to it.

Such pervasive computing systems can be viewed as being composed of autonomous, pro-active and interacting agents, where users, devices, and resources are all perceived as agents[34]. These agents have different levels of autonomy and intelligence, different models of agency, and use varied modes of communication, making a pervasive system a multi-agent system, where heterogeneous agents must collaborate to fulfill their own goals. We associate the problem of security in pervasive environments with the security issues in dynamic systems of heterogeneous agents.

This paper describes the use of distributed trust, which is similar to the trust required for human co-operation. For example, we decide to co-operate with a person and make decisions about this relationship based on how much we trust her. The basis for trust differs from person to person, leading to trust being distributed, as every entity has its own parameters for establishing trust. It involves the exchanging of beliefs, the delegation of permissions, obligations, and credibility. In terms of security for pervasive computing, the system “proves” that a user has the right to access some resource, in the absence of a static access control list, solely through trust relationships and recommendations. An open environment is assumed, one in which agents must interact with other agents with which they are not familiar, so the system also includes proofs of justification for beliefs and security decisions. An ontology based on a semantic language (DAML+OIL[10]) is used to represent security information, including beliefs, proofs, and credentials. This ontology provides a powerful way to describe this knowledge, and helps agents to better understand the knowledge.

Motivation

Vigil provides security where traditional security architectures could fail or become overly cumbersome. To motivate our application consider the case of *Acme* a national company with offices located in most major metropolitan areas of the United States. Suppose John, a software developer, is an Acme employee assigned to Acme’s Los Angeles office where he only has access to systems and services in the Los Angeles office, in accordance with Acme’s security policy. Suppose John is temporarily assigned to a short-term project in Acme’s McLean, Virginia office. When John arrives at the Virginia office he has no access to systems and services, his smartcard enabled company badge containing an x.509 version 3 digital certificate will not even allow him to pass through the access control system governing the entrance to the McLean office.

If John needs to use any systems or services in the McLean office he must first be authenticated by the system. One way to grant John his needed access is for the system administrator to create a temporary system account with an associated user name and password setting the appropriate permissions. After John leaves, the system administrator will need to de-activate John's temporary account. Another solution, would be to allow company wide access to all locations. This solution, however poses vulnerabilities to security because access is granted to many people who have no need. Moreover, it is a highly centralized solution, which is not particularly scalable. One might be tempted to suggest *Role Based Access Control* (RBAC) as a possible solution. RBAC is a form of mandatory access control where access control decisions are determined by the roles individual users take on as part of an organization. RBAC tends to be modeled after the structure of an organization where functions are grouped into roles and users are permitted to one or more of these roles. The fundamental premise of RBAC is orthogonal to *Discretionary Access Control* [11], where access privileges to be left to the discretion of the individual users allowing them to grant or revoke access to any authority under their control without the intercession of a system administrator. Consequently, in most implementations of RBAC, users cannot sub-delegate access permissions on to other users at their discretion. Hence, a RBAC solution is unworkable in a pervasive computing environment.

Vigil, provides an alternative. Vigil enables the McLean office to to define a policy which confers upon John axiomatic rights associated with whatever role John fulfills in his temporary assignment. Additionally, Vigil allows John to be delegated certain other rights for limited time periods and to request additional rights and permission on an ad-hoc basis from duly authorized Acme personnel. For example John can electronically ask Susan, a manager in McLean, to allow him to access the color printer. In turn, Susan may delegate this right to John for some limited time period. While this particular delegation is valid, Vigil allows John to access the color printer. Consequently, John's rights are restricted strictly and the system does not need to be modified in any way to change access rights of particular entities within the system.

Now, let us consider a slightly more complex scenario. Let *Zylon* be a company that is providing Acme with contractual consulting services on the same temporary project that John is assigned to. Moreover, suppose that Alice and Mary are employees of Zylon and are to report to John in McLean for the duration of the project. Furthermore, assume that Alice and Mary possess Id's comprised of x.509 digital certificate that are issued by and signed by Zylon's root certificate. This example necessitates that John will need to delegate certain rights to Acme systems and services to Alice and

Mary for the duration of the temporary project.

Our example scenario takes place in a pervasive computing environment where numerous, casually accessible and often mobile computing devices are embedded in a ubiquitous network structure. Therefore, the basic requirements of a security system to enable this example scenario are:

- The system should allow foreign entities to access entities within the system.
- The system cannot follow a strict Role Based Access Control model.
- Rights need to be tailored for each entity.
- The system needs to be easy to maintain.
- Certain rights should be *delegate-able* and *re-delegate-able*.
- Only entities with the right to delegate should be able to delegate.

Vigil uses a *distributed trust* model that is more flexible than traditional *Role Based Access Control* and is better suited than traditional security mechanisms in most pervasive scenarios. Vigil is able to provide the right amount of security with little or no additional overhead or maintenance. In Vigil, Alice and Mary will continue to use their Zylon issued Id's once John validates them, by asserting that they are to be given the basic role of a *outside-contractor* in the system. In turn John may delegate any additional right to Alice and Mary which he has been granted permission to *re-delegate*. Alice and Mary's rights will be valid for the most restrictive of the following:

- For the period that John specifies.
- Until John revokes the rights.
- Until John's rights expire.
- Until John's rights are revoked.

III. Related Work

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

A. *SmartSpaces : Related Work*

Some of the projects dealing with *SmartSpaces* are the joint Unisys Corporation/Orange [31] experimental house in Hertford, England, UC Berkeley's Ninja project [12] [14], the University of Washington's Portolano project [9], and Stanford's Interactive Workspaces Project [6].

As demonstrated by the Unisys/Orange project, the concept of SmartHomes transitioned from the purely 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 be paramount for success and public adoption of any system.

In the Centaurus project [21], the main design goal was 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. Vigil uses CCML as its sole form of data exchange.

As stated in Section I Centaurus2 extended Centaurus. Like Project Centaurus, Centaurus2 employs *Communications Managers*, *Service Brokers*, *Services* and *Users*. However, Centaurus2 merges Services and Users into *Clients* where a Client may be a producer, consumer or both. Centaurus2 introduces *Certificate Authority* and the *Capability Manager*. These architectural components will be detailed in Section IV.

In Centaurus a grouping of a Service Manager, Services and Users operate as an autonomous system at any given instance, while in Centaurus2 access 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.

In Centaurus2 communication between clients (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 enterprises. This bridge allows a user in their SmartOffice to reach services running in their SmartHome.

Security is paramount to the success of the ubiquitous computing paradigm. Accordingly, security is fundamental to Centaurus2, which employs the *Centaurus2 Certificate Authority* and the *Centaurus2 Capability Manager* in furtherance of security. 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.

As previously states, users and services are treated equally as clients in Centaurus2. This enables 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 arbiter as to who may utilize it.

Another important research project is UC Berkeley's Project Ninja [12], [14] which employs Group Controllers, Certificate Authorities, and a hierarchy of Service Discovery Service Servers. However, unlike Vigil, it does not delegate state management to the Services themselves nor does it allow the Service Broker 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. Vigil 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 Vigil 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 Vigil Group Controller(s). Moreover, while Vigil is protocol and communications medium independent, Ninja is not.

The Portolano [9] 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 Vigil this extensibility is accomplished by the ad hoc nature in which a service-Client can register with any Service Broker within its domain making its service available to all authorized clients regardless of location.

Stanford's Interactive Workspaces [6] 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 Vigil approach where XML is used as the sole format for data exchange.

B. Distributed Trust : Related Work

In traditional security systems, authorization includes some form of authentication followed by access control. Woo et al. describe various methods of authentication in distributed systems like Kerberos [39]. However these techniques do not scale well, and generally require a central server for authorization. They suffer from overhead in communication and also make several implicit trust assumptions. Access control in these distributed scenarios is also a problem because storing this information in a central location is not always feasible or desirable. Hence it no longer makes sense to divide authorization into authentication and access control.

We follow the trust management approach proposed by Blaze et al. [3], [4], [28], which formulates authorization as verifying whether an entity has the credentials that comply with the security policy governing the requested resource. These credentials include properties, and trust relationships of the agent. In this proposal, distributed trust is used as a way of authorizing users' requests based on security policy using trust relationships.

Trust is explained in terms of a relationship between a trustor, who trusts a certain entity, and a trustee, the trusted entity [13]. Based on his trust in the trustee, a trustor can decide whether the trustee should be allowed to access his resources and what rights should be granted. Therefore, trust plays an important role in deciding the access rights of a trustee. Using trust involves trust establishment, and trust management. The basis for trust differs from person to person, leading to trust being distributed, as every entity has its own parameters for establishing and managing trust. Trust establishment is the process of deciding what the trust relationship with another entity should be. Trust management involves using the trust information, including recommendations from other trustees, to reason about authorization requests. In dynamic systems, every entity enforces its individual policies so a trust management component cannot use global policies or trust relationships to evaluate requests for authorization. This causes trust pertaining to each entity to be handled specifically in terms of its own policies, which requires principles of distributed trust.

Blaze's PolicyMaker [29] is probably one of the first forays into *distributed trust*. Though the concept has its roots in Pretty Good Privacy (PGP) [40], Simple Public Key Infrastructure (SPKI) [16], Simple Distributed Security Infrastructure (SDSI) [33] and Role Based Access Control (RBAC) [26] [27].

PGP [40] 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 [16]. This solution, though simple and elegant, includes only a rudimentary notion of delegation, which is crucial to the developed of *distributed trust*.

PolicyMaker [29] 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 [26] [27] 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.

In *Vigil* we extend classical trust management, which is solely based on simple delegation, by incorporating conditional delegation - that allows conditions to be placed on execution of access right and re-delegation, negotiable delegation - an agent may receive a delegation but in turn has to carry out a certain obligation, prohibition - an agent can be prohibited from a right, and request - an agent can make a request for an ability. 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 [20]. 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 *re-delegatable* 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.

IV. Architecture

Our system is designed to provide security and access control in distributed systems, and has been optimized to work in *SmartSpaces*, where many of the clients are hand-held devices. Vigil can also be used in wired systems, but the focal point of our research is the security in dynamic, mobile systems. Vigil is designed so that clients can move, attach, detach, and re-attach at any point within the framework.

Vigil employs several agents such as the *Communication Manager, Certificate Authority, Capability Manager*, and the *Service Manager*. Agents that either request a service or provide one are viewed as *Clients* of the Vigil infrastructure. Vigil introduces the *Trust Agent* that is used for delegation of rights.

The Vigil system is divided into *domains*, and each domain is controlled by one or more *Service Managers*. The *Service Manager* agent, as the name suggests, finds matching services for users. It allows users and services to register and then provides brokering between them. The *Communication Manager*, provides a communication gateway between a Client and a Service Manager. Its sole purpose is to abstract and translate communications protocols and may contain several modules, one for each protocol [21]. The *Certificate Authority* is responsible for generating x.509 digital certificates for each entity in the system and for responding to certificate validation queries. The *Capability Manager* agent maintains a static capability matrix for system entities and responds to initial requests for access control and authorizations for delegation of rights. Each domain generally has one Capability Manager. The *Trust Agent*, manages the trust in the domain, receiving information about new access rights that are conferred on a user, and reasoning about the current rights of a user. Finally there are users and services that are treated equally as *Clients*.

All messages between the various entities in the Vigil system are in Centaurus Capability Markup Language (CCML) [21], which is an extension of Extensible Markup Language (XML) [37].

Figure 1 shows a overview of the Vigil System. 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

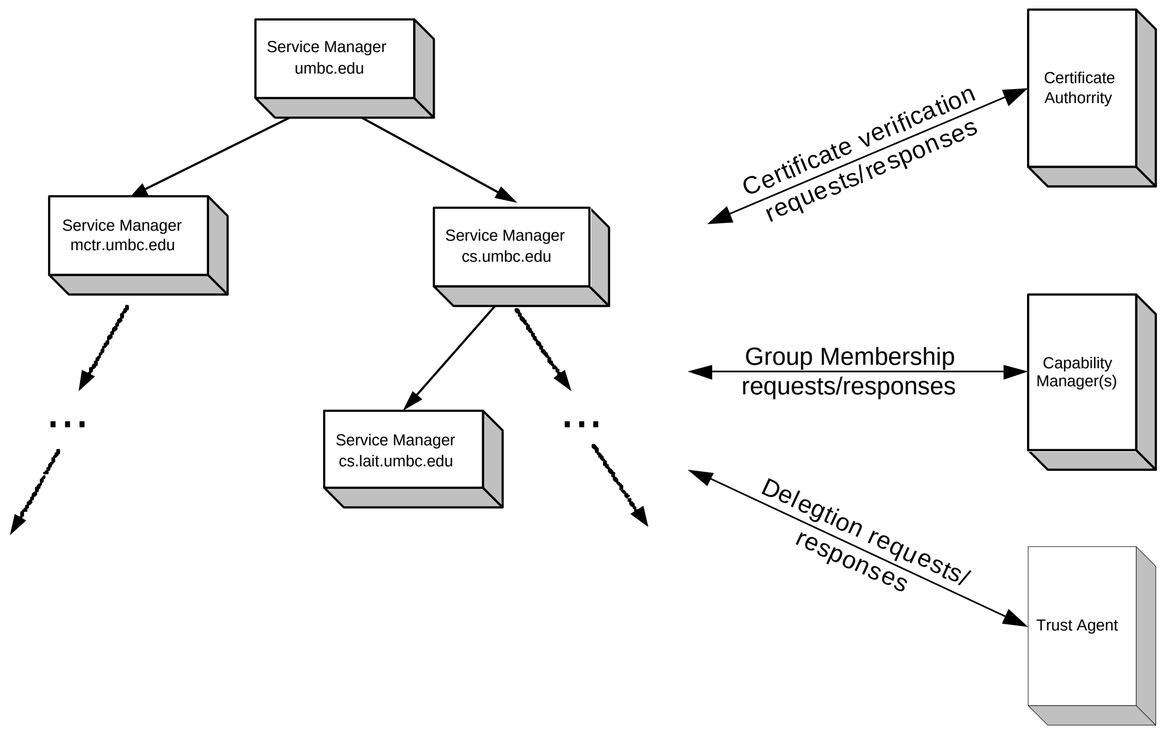
Service Manager umbc.edu
Service Manager mctr.umbc.edu
Service Manager cs.umbc.edu
Service Manager cs.lait.umbc.edu
Certificate Authorrity
Capability Manager(s)
Trust Agent
Certificate verification requests/responses
Group Membership requests/responses
Delegtion requests/ responses

Fig. 1. Vigil Framework

initialization, each Service Manager learns the location of its Capability Manager from a 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.

A. *Certificate Authority*

In Vigil, each entity receives an x.509 version 3 digital certificates [15]. Here, certificate request and issuance is ancillary to the system. Although accomplished "out of band" when a certificate request from a Vigil entity is filled, the entity receives its requested x.509 v3 certificate signed by the Certificate Authority and the Certificate Authority's self signed certificate. The Certificate Authority's self signed certificate is subsequently used to validate other entities' certificates. In Vigil, certificates may be stored and protected on a smartcard or stored in a PKCS#11 container.

Certificate generation and signing is typically a one time occurrence for any entity within the Vigil System. An entity's certificate is only available upon being presented by that entity. This is in contrast to a typical PKI where 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, Vigil uses a simplified PKI. In Vigil, each entity presents its certificate to its Capability Manager when it registers (the registration process is detailed in Section VI-D). 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 Vigil system. This precludes the necessity of maintaining a CRL, which must be signed by the Certificate Authority each time it is modified.

A 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 Vigil, 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 verifies the response according to Equation (1). If the test passes, then the copy of Certificate Authority's private key is valid and any object signed by that key is also valid.

$$\begin{aligned}\text{(CA's certificate SHA-1 message digest)} =& \\ &D_{publickey}(E_{privatekey}(\text{(verifier)})) \qquad (1)\end{aligned}$$

This 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.

B. Capability Manager

The Capability Manager is responsible for maintaining and communicating group membership(s) of all entities in the Vigil system. Entities include Service Managers and Clients. 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 Service "*filip.cadip.cs.umbc.edu*", which implies that only the named entity is allowed access.

When the Capability Manager is initialized it reads its x.509 v3 digital certificate and its PKCS#11 [24] wrapped private key from a secure file and stores it into local memory. It also reads and indexes the capability file containing the group memberships of all entities within the system, as well as storing the time stamp of the capability file.

When a group membership request is received from a Service Manager, the Capability Manager compares 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 dynamic administration, to include rights revocation, of the capability file.

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

C. Trust Agent

The Trust Agent is responsible for maintaining distributed trust in the Vigil system. It enforces the security policy of the organization or enterprise. It interprets the policy in order to provide controlled access to Services and uses distributed trust as a more flexible and easily extensible policy based mechanism. A policy includes rules for role assignment, rules for access control, and rules for delegation. This policy can be extended by the use of *delegation* by authorized entities to third parties. These delegations are only valid if the delegatee has the right to delegate. Revocation of rights is also possible.

The Trust Agent uses a knowledge base and sophisticated reasoning techniques to handle security and distributed trust. 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 user has the ability to access a service if the user has not been prohibited from accessing the service by an authorized entity and if it either has the 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.

Vigil uses a well defined ontology to represent security information, which provides agents with a better understanding of information and security mechanisms as well as providing a common understanding of the domain. Our trust ontology includes a rich set of constructs for concepts like principals (agent, entity and user), credentials (that an agent or object possesses), beliefs, permissions, delegations, proofs and policies.

We use DAML+OIL [10] as an ontology language. DAML+OIL is a semantic ontology language for resource markup that is being developed for the realization of the *Semantic Web*[*].

Figure 2 illustrates our trust and security ontology. The root of the ontology is dived into *State, Entity* and *Action*. State contains all information pertaining to the current state of an entity and has

[*]W3C's Ontology Wrapper Language (OWL) is based on DAML+OWL

one subclass, *Proposition*. Propositions are clauses that have a truth value and are further divided into *Permission*, *Obligation* and *Belief*, however Vigil does not currently use Obligation and Belief.

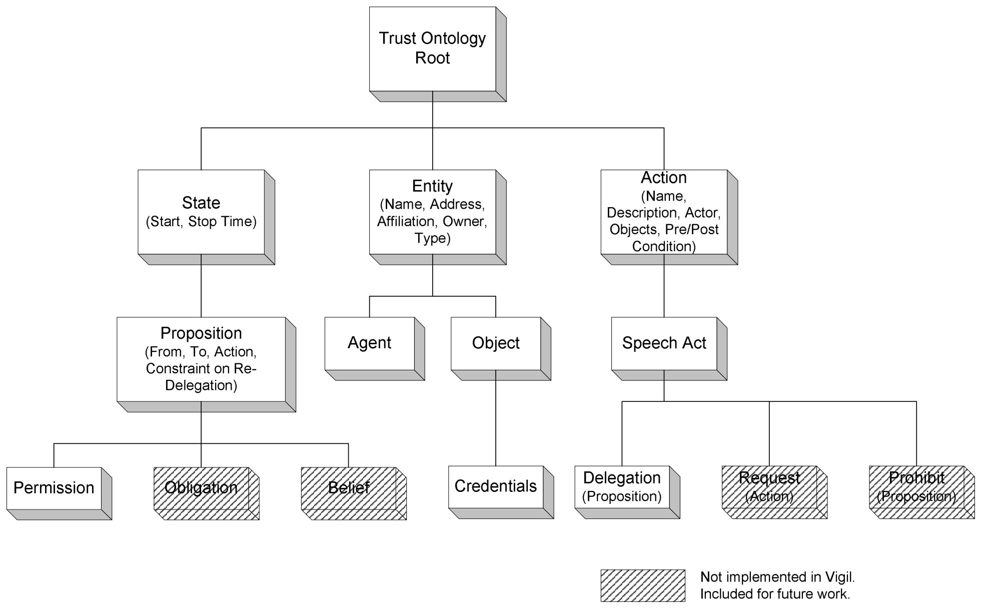
Trust Ontology Root
State (Start, Stop Time)
Entity (Name, Address, Affiliation, Owner, Type)
Action (Name, Description, Actor, Objects, Pre/Post Condition)
Proposition (From, To, Action, Constraint on Re-Delegation)
Agent
Object
Speech Act
Permission
Obligation
Belief
Credentials
Delegation (Proposition)
Request (Action)
Prohibit (Proposition)
Not implemented in Vigil. Included for future work.

Fig. 2. Vigil Trust Ontology Represented as a Class Hierarchy

An Entity may be either an *Agent* or an *Object*, where an Object can be extended to include resources such as credentials (x.509 version 3 digital certificates) files, printers, computers, etc. An Action is associated with a set of Objects or resources. *Speech Acts* such as *Requests* and *Delegations* are an extension of Actions.

The rules specifying delegation of access rights, validity of delegations, etc. are part of the policy. There is more inference using these rules, than simple access control lists. It is also more than just using *Role Based Access Control*, because access rights can be delegated, and these delegations are not random, they have to be from an authorized entity to another entity or group of entities, and should follow the security policy.

Figure 3 shows how the Trust Agent works to provide distributed trust. The steps in the process are given in the diagram. Revocation is propagated in a similar way. A user sends a revocation to the Trust Agent, which sends it to the Service Manager, next time it is queried. The user, whose right was revoked, is denied access to the service, by the Service Manager.

When a user needs to access a service that it does not have the right to access, it requests another user, who has the right, or a Service Manager or the service itself, for the permission to access the Service. If the entity requested does have the permission to delegate the access to the Service, the

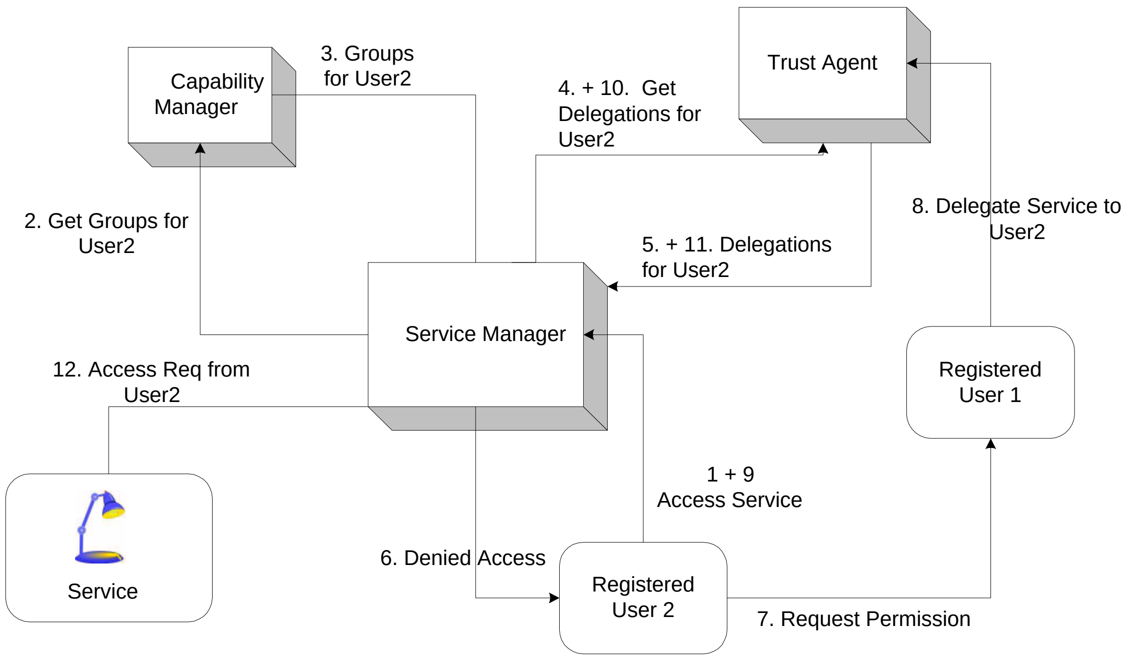

Fig. 3. Trust Agent Functionality

entity sends a delegate message, signed by its own private key, to the Trust Agent and the requester. The Trust Agent 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 Trust Agent has to reprocess the delegation. This is very useful in case 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, till everyone after the entity loses the ability. Consequently, the next time a Service Manager asks about the delegated rights of the client, the Trust Agent sends back all valid permissions.

A user can also *revoke* rights that it has delegated by sending the appropriate message to the Trust Agent. The Trust Agent removes the permission for the delegated entity, and when a Service Manager asks about the delegated entity, it is informed of the revoked right. So revocations progress rapidly through the system.

D. Service Manager

The Service Manager acts as a mediator or broker between the Services and the users. All clients of the system, whether they are Services or users, have to register with a 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 users to services, and for sending service updates to all subscribed users whenever the state of a particular service is modified.

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

All clients depend on their Service Manager to enforce security and to broker requests for services. Consequently, each client is only concerned with the trust relationship with its Service Manager. Service Managers in the tree establish trust relationships with each other. Consequently, trust between clients is transitive through the Service Managers. Figure 4 illustrates the relationship between a user, Service Manager, and service within the context of single Service Manager.

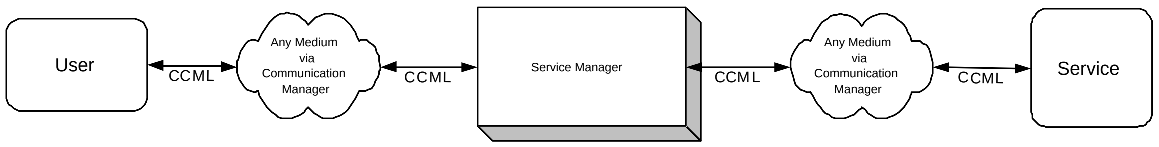
User
CCML
Any Medium via Communication Manager
CCML
Service Manager
CCML
Any Medium via Communication Manager
CCML
Service

Fig. 4. Domain of a Service Manager

When an Service Manager initializes, it reads its handle of the form: *servicename*, its parent's handle of the form: *ServiceManager.cadip.cs.umbc.edu*, its Capability Manager's address, the Certificate Authority's address, and the handle of Service Managers within its same level of hierarchy from a configuration file. Each Service Manager starts with its own digital certificate and corresponding private key, and the digital certificate of the Certificate Authority.

Upon start up the following sequence of events occur:

1) The Service Manager sends a certificate verification request to the Certificate Authority to ascertain that the local copy of the Certificate Authority's certificate is valid.
2) In response, the Service Manager receives a signed certificate verification response from the Certificate Authority. The Service Manager verifies the signature of the response according to Equation (1).
3) The Service Manager sends a certificate request to its Capability Manager.
4) The Service Manager receives a certificate response from the Capability Manager. It verifies that the certificate contained in the message was signed by the Certificate Authority and also verifies that the signature of the message is valid.
5) If the Service Manager is not the top most Service Manager of the domain $(handle \neq parent'shandle)$, register with the parent Service Manager by sending a CCML registration message that con-

tains a copy of the registering Service Manager's digital certificate that has been converted from its ANS.1 encoding to Base64 string. The entire message is signed according to Equation (2). The digital signature is also converted to Base64 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 Vigil system, and is a configurable parameter. When all steps are successfully performed, the receiving Service Manager registers the sending Service Manager in its table of pending Services and sends a group membership request to the Capability Manager.
7) Once the group membership response is received from the Capability Manager the 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.
8) Once a registration acknowledgment has been received the registering Service Manager accepts communications from registrant.
9) Service Managers at any level may register with Service Managers at any other level of hierarchy. When an Service Manager receives a CCML message destined for another Service Manager it forwards the CCML message according to the rules detailed under Security Protocol (Section VI-D).

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

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

The Service Manager maintains a table of profiles for all entities registered with it. Information contained in the profile table includes the entity's certificate, group memberships, location (the Service Manager to which it is immediately connected), name, and permissible access groups. Table I illustrates the contents of the profile table:

Figure 5 depicts the Client located at Service Manager *cs.umbc.edu* and registered with both *cs.umbc.edu* and *lait.cs.umbc.edu*. The user is accessing a service registered at *lait.cs.umbc.edu*.

Element	Contents
handle	the name of the service
parent	the name of the Service Manager to which it is connected
Address	network address
access groups	access groups to which the entity belongs
member groups	groups in which membership is required for access to the service
registered entities	a list of all entities registered with the particular service
crypto-storage	a class containing the entity's digital certificate its private key (populated only for the key holder), and functions for cryptographic operation

TABLE I
SERVICE MANAGER PROFILE TABLE

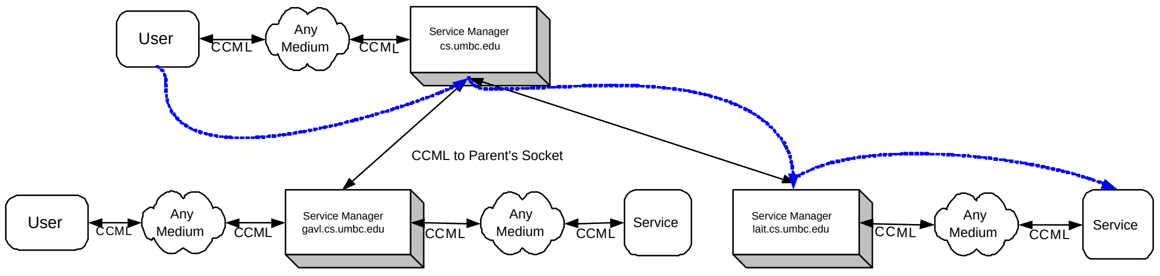

Fig. 5. Multiple Service Manager Registrations

E. Client

As mentioned earlier, all entities within the Vigil system possess both their own digital certificate as well as the Certificate Authority's certificate. The Certificate Authority's certificate is used to verify other entities' certificates.

All clients (agents that use the Vigil infrastructure) must connect to and register with a Service Manager. During registration, the client transmits its digital certificate, a list of groups which can access it, and a flag, *ShowAll*, indicating whether or not the Service Manager should publish the client's presence to other clients, that do not have permission to access to it. If the client has no services to offer (is solely a user) the list of access groups is empty. During registration, a service can inform the Service Manager how important the access information is. This reflects how often the Service Manager updates access information about the service. To get new access information, the Service Manager queries the Trust Agent. The Trust Agent will return current information about delegations and revocations to the Service Manager, which updates its knowledge.

On receiving the registration information, the Service Manager verifies the client's certificate, and sends the client a copy of its digital certificate, which the client verifies. This handshaking procedure ensures a trust relationship between the Service Manager and the client. After the client certificate is verified, its certificate is sent to the Role Assignment Module, which decides which roles the client can have, based on rules in the policy. For example, the policy could state that consultants from Zylon are software developers within the Acme enterprise. The client has all the rights associated with the roles it is assigned. Once a client is successfully registered, it is provided with an interface to the services, that are registered with the Service Manager and that it can access. The client is also shown all the services that it may not be able to access but that have their *ShowAll* flag set. The Service Manager also sends to the client an interface to all other Service Managers that the Service Manager is currently aware of. Using the interface sent by the Service Manager, the client can request a service.

This request for service is authenticated by the Service Manager, which makes sure that the client has the right to access the service. The Service Manager If valid, the request is forwarded to the service. A client can request permission to access a service from the Service Manager another client or the service itself. If the Service Manager or the client has the ability to delegate rights on the service, it delegates this ability to the client for a certain period. While this delegation is valid the client is allowed to access the service. After which it needs to make another request for access.

V. Functionality

In the previous section, we discussed the overall design of the system and the design of the individual components. In this section, we describe certain important functions of the system that were mentioned briefly earlier.

A. Entity Communication

Clients (users and services) are able to communicate irrespective of the underlying communications medium. Accordingly, to ensure communication through the hierarchy of Service Managers, Vigil 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.

B. Service Access

A user is given an interface to all those services that it can access and is also displayed services that it may not be able to access, but which have their *ShowAll* flag set. When a Service Manager gets an access request for a service from a user, it first makes sure that the user has the right to access the service. This validation for access rights is shown in Equation (4).

Then it queries the Trust Agent about any delegated rights for the user. If the user has the right to access the service by being in an access group of the service, or if the user has been delegated this right by an authorized entity, then the Service Manager forwards the request to the service.

$$\begin{aligned} & Client.groupmembership \cap \\ & \quad OtherClient.accessgroups \end{aligned} \tag{4}$$

C. Service Updates

Every time the state of a service changes, it sends a *Service Update* to the Service Manager that it is registered with it. The Service Manager sends this update to all users that have subscribed to this service.

D. Request for Permission

A user can see some services that it cannot access and it may also learn about some services from other users. If the user needs to access a service that it does not initially have the permission to access, it can request another user, or the service itself to delegate this ability to it. The user sends a CCML message to the appropriate entity asking for the permission.

E. Delegation of Rights

A user can delegate all the rights it has, as long it has the permission to redelegate these rights. Rights can be delegated to a user or a group and are generally valid for a certain period of time. The delegations are constrained, whether or not the delegation itself can be delegated, whether the right can be redelegated etc. When a user receives a *request for permission* message, it can decide whether to delegate the right or not. A user can also delegate a right, without being asked to. For example, a manager can decide to delegate the ability to access certain sensitive information to his assistant, in his absence. These delegations are sent to the Trust Agent. The Trust Agent reasons about them and makes sure that the delegations are valid. When a Service Manager asks the Trust Agent for information about a user, the Trust Agent goes through relevant delegations for the user and computes new delegated rights for the user. These rights along with the time period of their validity are returned to the Service Manager. The Service Manager maintains a list of delegated rights for each client along with their period of validity. Periodically, the Service Manager goes through the list and clears expired delegations. This list reduces the amount of communication between the Service Manager and the Trust Agent.

F. Revocation of Rights

In the same way as a right can be delegated, it can also be revoked. If for some reason, a user can decide to revoke a certain right that it delegated to a user or a group. The user sends a *RevokeRights* message to the Trust Agent, specifying the right and from whom the right should be revoked. The Trust Agent removes the permissions that have been generated as a result of the earlier delegations. When queried next by the Service Manager, it informs the Service Manager about the revoked right. The Service Manager removes the right from its list. When the user, from whom the right was revoked, tries to access the service, it is denied access. Now the user has to make another request for permission.

VI. IMPLEMENTATION

Our goal of instantiating security, specifically: authentication, authorization, non-repudiation, and anti-playback, as a primary component of Vigil was heavily influenced by the desire to make security as unobtrusive as possible. We also wanted to make it lightweight, to work in ubiquitous environments. We believe we have designed a system that is both highly secure, security that is nearly transparent to the user and light enough to work with mobile, hand-held devices easily. 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
6) Security Protocol

The sole action required of the individual entity (i.e., Service Manager, Service, 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.

A. Lightweight PKI

In a Public Key Infrastructure (PKI) system, certificates are made available in an on-line repository. Consequently, when a user needs an entity's digital certificate it can be retrieved from such a repository and is valid as long as the certificate chain to the top level CA is verifiable. 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 Vigil by its simplified PKI framework. As previously stated, each Vigil 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.

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 in the CRL indicates that a given certificate is, for one of many possible reasons, invalid. In Vigil, each entity presents its certificate to its Service 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 Vigil system. This precludes the necessity of maintaining a CRL, which must be signed by the Certificate Authority each time it is modified.

B. Smart Cards

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 Vigil Capability Manager, Services and those Clients that do not have smart cards, is stored in a PKCS #11 container in a regular File System.

C. Capabilities Matrix

The Vigil Capability Manager is responsible for responding to requests for group membership from Service Managers. Accordingly, the Capability Manager maintains a database of all entities (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.

D. Security Protocol

The Service Manager is responsible for ensuring the integrity of the Vigil system. As stated, each Service Manager has a copy of the Certificate Authority's certificate. As will be explained below, a Service Manager to which a client is registered is responsible for authorizing CCML messages destined to the Service. The following describes the security protocol implemented in Vigil by the Service Manager(s).

1) 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 Base64 string and insert it into the message. When verifying the message compare the message digest from Equation (5) with the decrypted signature from Equation (2), and verify the timestamp using Equation (3).

$$\text{SHA-1}\left(\text{CCML message}\right) \tag{5}$$

2) 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.

   a) If the message is a registration request and 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 Service 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.

   b) If the Service Manager is both the source and destination Service Manager, then the originator of the message is one of its immediately connected services. 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 Service profile storing the profile in its user database, and transmits a registration response message containing its digital certificate.

   c) If the Service Manager is the source Service Manager but is not the destination Service Manager, this indicates that the registering Service has already registered with its 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.

   d) 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.

   e) 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 Service database, resigns the registration response, and forwards the response to the Service.

3) Upon receipt of a message the Service Manager will verify the signature contained in the message, ensuring that it was signed by sender and will verify that the sender has the appropriate rights with the destination. If the signature and rights are valid it will resign the message and forward it to the destination, either up, down, or laterally based upon the destination address.

E. Operational Protocol

The following enumerates Vigil 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.
3) *Permission* From: Service Manager To: Distributed Trust Manager. When a client-service registers with a Service Manager, the client informs the Service Manager the requirements for access to that client. The Service Manager forwards these permissions to the Distributed Trust Manager.
4) *Delegations Query* From: Service Manager To: Distributed Trust Manager. When a client-user registers with a Service Manager the Service Manager queries the Distributed Trust Manager about any delegated rights for the client.
5) *Delegations Response* From: Distributed Trust Manager To: Service Manager. The Distributed Trust Manager sends a Delegations response as a result of a query from a Service manager or as the result of a new delegation regarding a particular client. The case of the later the response will be sent to all affected Service Managers.
6) *RequestForPermission* From : Client (user) To : Client(service) or another Client(user). A request to delegate some rights to access a service to it.
7) *Delegate Message* From : Client To : Trust Agent. A client may decide to delegate some rights to another client and tells the Trust Agent.
8) *RevokeRights* From : Client To : Trust Agent. A client could take back any delegation that it made. It sends a RevokeRights to the Trust Agent, specifying which group or user the particular right has to be revoked from.

9) *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.
10) *De-Registration* Response From: Service Manager To: Client. Signed by the sender, it is assumed that the destination does not receive this message.
11) *Access Request* From: Service Manager To: Capability Manager. Sent to the Capability Manager from a Service Manager requesting the group memberships of some entity.
12) *Access Rights Response* From: Capability Manager To: Service Manager. Signed by the Capability Manager. Contains group memberships of the subject.
13) *Service List Request* From: Client To: Service Manager. Signed by the sending user-Client requesting a listing of available services.
14) *Service List Response* From: Service Manager To: Client (user). Signed by the sender, informs the Client of available services registered with that Service Manager.
15) *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.
16) *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.
17) *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.
18) *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.

VII. Analysis

We have conducted an analysis of both the operational and security protocols employed by Vigil. In analyzing the security protocol we identified the critical steps and any associated vulnerabilities. The operational protocol analysis was conducted on the basis of time and space complexity and verified via experimentation which is reported in Section VIII.

Security Protocol

The Vigil security protocol employs digital certificates, digital signatures, and public key cryptography. The Certificate Authority's (CA's) public key, contained in the CA's digital certificate, is used to authenticate the digital certificate of each entity using the system. In turn, the public key of each entity is used to authenticate communications from that entity. It is assumed that the process of generating digital certificates is secure. Each entity starts with the CA's digital certificate and their own digital certificate which were distributed "out of band". No private keys are exchanged during the course of business.

The critical steps of the security protocol and their associated vulnerabilities follow:

1) Verifying the authenticity of the copy of the CA's digital certificate stored with an entity.

   If the CA is compromised and the root certificate is altered, or if the copy of the CA's certificate stored at the entity is altered, the authentication check as defined in Equation 1, will fail. This will result in the entity ceasing to function because it has no way of authenticating communications. This constitutes a denial of service.

2) At system initialization, a Service Manager obtains a copy of the Capability Manager's digital certificate, verifying the authenticity of that certificate.

   If the Capability Manager is compromised and its digital certificate is altered then the certificate verification process of Equation 6, will fail and the Service Manager will not trust any communications from the Capability Manager. This results in the Service Manager not functioning and constitutes a denial of service.

$$\text{Thumbprint}(\text{Certificate}) = D_{CA_{publickey}} \\ (\text{SHA-1}(\text{Certificate})) \tag{6}$$

3) Obtain an entity's access permissions from the Capability Manager.

   If the communication containing an entity's access rights is intercepted and altered the verifications according to Equations 2 and 3, will fail and the message will be discarded. This results in the Service Manager not being able to establish the entities permissions, consequently that entity will not be able to function within the system. This constitutes a denial of service.

4) During the registration process when an entity sends a copy of its digital certificate to the service manager. The authenticity of the entity and its digital certificate are verified.

If the registering entity is counterfeit, having intercepted a valid digital certificate, the counterfeit entity will not have a valid private key to sign the registration request as per Equation 2, consequently verification of that message using Equation 7, will fail.

$$\text{Message Signature} = D_{publickey}(E_{privatekey}(\text{SHA-1} \\ (\text{Original Registration Request CCML message}))) \quad (7)$$

Although there is the possibility of denial of service attacks, Vigil is secure from fictitious entities and is not vulnerable to messages being intercepted and altered between source and destination.

In the current implementation, communications between Vigil entities are not encrypted. However, if needed, this feature could be easily added. Table I depicts profile information stored by Vigil entities. Clients are only concerned with a one-to-one relationship with Service Managers, consequently a client only stores its own profile and that of its Service Manager. Service Managers store profiles for every entity to which it is connected. Recall that this profile contains a class called *crypto-storage*. To add message encryption, the registration process could be extended to include the selection of a session key generated by the Service Manager and transmitted to the client encrypted under the client's public key. The *crypto-storage* class could then be modified to contain the shared session key to be used with subsequent communications between the entities as well as some agreed upon symmetric encryption algorithm.

Operational Protocol

Message size in Vigil varies according to content, however, all messages contain a 128 byte digital signature Base64 encoded as a 172 byte string. Registration and Registration Response messages include the 1255 to 1291 byte digital certificate Base64 encoded as an $\approx$ 1700 byte string. Note: the Base64 encoding of binary data results in an expansion of the ratio 3:4. The space complexity for messages is shown in Table II below:

Messaging complexity (e.g: the number of messages required for a task) for entity to entity communications is:

$$(2 * \mid number\ of\ messages \mid)$$

because the Service Manager receives, verifies and forwards the message from source to destination. The best case messaging complexity for Service Manager to Service Manager is 1 where the source and destination are one logical level apart. The worse case messaging complexity from Service

Message	*Size in bytes*
Registration Request/Response message (including certificate)	3,328 bytes
All other Messages	700 - 800 bytes
Digital Certificate	$\approx$ 1,700 bytes

TABLE II
SPACE COMPLEXITY

Manager to Service Manager is given as follows:

Let *N* = the number of Service Managers.

Let *H* = the height of the logical Service Manager tree.

Let *M* = Messaging Complexity.

Assume that the branching factor of the Service Manager tree is ≥ 2.

$M = 2 * H$ where $H \leq \lceil log_2 N \rceil$

VIII. EXPERIMENTS

We used Java and Prolog in the development of Vigil. Prolog was used for reasoning in the Trust Agent and Java was used for all the other development. All Service Managers are initialized from the same class. All clients (services and users) are subclasses of a common client class that provides call-backs for event notification and methods for triggering events. At present, the agents do not use an ACL to communicate. In future, we would like to integrate our functionality into a platform such as LEAP(*http://leap.crm-paris.com*) that provides FIPA compliant agent functionality.

Vigil has been qualitatively tested for *usability, functionality,* and *scalability* by placing it into service as a *real-time* infrastructure in support of controller type services. Vigil functions in a heterogeneous environment in support of light controller services, audio controller services, temperature controller services, etc. User clients consist of PDA's using Infrared and Bluetooth, laptops using 802.11b and workstations using Ethernet.

The implemented framework is configured in a hierarchy approximating the Computer Science and Electrical Engineering Department at the University of Maryland Baltimore County. We have configured our framework to enable trusted communications between approximately 50 entities in the previously stated heterogeneous environment. In this configuration our framework functions without any performance degradation.

We conducted several tests to assure that Vigil fucntioned as specified. The specific tests and their purpose follow:

1) *Know User #1* registers with a Service Manager.
    a) Verify that the user is presented with an interface to authorized services.
    b) Verify that the user is not presented with an interface to to unauthorized services.
2) *Know User #2* registers with a Service Manager.
    a) Verify that the user is presented with an interface to authorized services.
    b) Verify that the user is not presented with an interface to to unauthorized services.
3) *Known User #2* requests *Know User #1* to delegate rights to a service to which *Know User #2* does not have rights and *Known User #1* has permission to delegate. *Know User #1* delegates those rights for a period of 3 minutes.
    a) Verify that *Know User #2* receives those rights and is given an interface to the service.
    b) Verify that *Know User #2* looses those rights after a period of three minutes.
4) *UNKnown User #1* registers with a Service Manager.
    a) Verify that *UNKnown User #1* is not given access or an interface to any service.
5) *UNKnown User #1* requests
6) *Known User #2* to delegate rights to a service to which *Known User #2* has been delegated rights but does not have the right to re-delegate.
    a) Verify that *UNKnown User #1* does not receive rights to that service.
7) *UNKnown User #1* requests
8) *Known User #1* to delegate rights to a service to which *Known User #2* has rights.
    a) Verify that *UNKnown User #1* receives rights to that service.
9) *UNKnown User #1* requests
10) *Known User #2* to delegate rights to a service to which *Known User #2* has been delegated rights to for a period of five minutes and has the right to re-delegate.
    a) Verify that *UNKnown User #1* receives rights and an interface to the service.
    b) Verify that *Known User #2* looses those rights after the 5 minute period ends.
    c) Verify that *UNKnown User #1* looses those rights after the 5 minute period ends.
11) *UNKnown User #2* registers with a Service Manager and requests *UNKnown User #1* to delegate rights to a service to which *UNKnown User #1* has been granted rights to and has permission to re-delegate.
    a) Verify that *UNKnown User #2* receives those rights and is given an interface to the ser-

Process	*Time in ms*
Average total time to process a Client Registration (via one Service Manager)	486 ms
Average total time to process a Client Registration (via chain of three Service Managers)	≈ 800 ms
Average time to process a command (via one Service Manager)	187 ms
Average time to process a command (via chain of three one Service Managers)	500 ms

TABLE III
TIME COMPLEXITY

vice.

Vigil performed as specified during our experiments. Additionally, during our experiments we measured the time required to process a client registration via 1 and 3 service managers and the average time required to process a command via 1 and 3 service managers. The results are reported in Table III.

IX. CONCLUSION AND FUTURE WORK

Pervasive computing environments have several characteristics that existing security protocols are not suitable for; such as presence of foreign users, dynamic resources in the vicinity, very large number of users and resources and no central control. This makes much of the existing research in the area of security of distributed systems inadequate for pervasive environments. Vigil builds on a multi-agent abstraction to provide security in pervasive computing environments by combining elements of authentication via SPKI, an ontology to specify security policies and distributed trust. This allows for greater flexibility in access control over distributed networks. With this framework it is relatively easy to allow a foreign entity to securely access a certain service or piece of information, doing so without opening up the entire system to the foreign entity. Vigil allows organizations to develop a security policy in terms of group memberships, rules for authorization, rules for delegations and rules for access control. This makes the policy very flexible, easy to control and easy to implement. But, along with being flexible, it is also possible to strictly restrict access control to services, without modifying the entire security policy. Vigil uses a lightweight PKI which is easier to implement than a traditional PKI and without a reduction in any functionality of PKI.

Now that the infrastructure is in place, we are working to improve the security and trust capabilities of Vigil. We are looking at *distributed belief* as a way for the Trust 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 Trust Agent would believe that entity1 had role1 in affiliation1, if two registered Clients said that this was the case.

REFERENCES

[1] Sasikanth Avancha, Tim Finin, and Anupam Joshi. Enhanced service discovery in bluetooth. In *IEEE Computer, June 2002*, 2002.

[2] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakyl. An Active Transcoding Proxy to Support Mobile Web Access. In *Proc. of the IEEE Symposium on Reliable Distributed Systems*, October 1998.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The keynote trust management system version. Internet RFC 2704, September 1999., 1999.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems. Secure Internet Programming, LNCS vol. 1603, Springer, Berlin, 1999, pages 185-210, 1999.

[5] E.A. Brewer, R.H. Katz, Y. Chawathe, A. Fox, S.D. Gribble, T. Hodes, G. Nguyen, T. Henderson, E. Amir, H. Balakrishnan, A. Fox, V. Padmanabhan, and S. Seshan. A network architecture for heterogeneous mobile computing. *IEEE Personal Communications Magazine*, 5(5):8 – 24, 1998.

[6] 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.

[7] Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy Finin, and Yelena Yesha. A reactive service composition architecture for pervasive computing environment. In *7th Personal Wireless Communications Conference (PWC 2002)*, October 2002.

[8] M. Dunham, A. Helal, and S. Balakrishnan. A mobile transaction model that captures both the data and movement behavior. *ACM/Baltzer Journal of Mobile Networks and Applications*, 2(2):149–162, 1997.

[9] 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.

[10] Ian Horrocks et al. DAML+OIL Language Specifications. http://www.daml.org/ 2000/12/daml+oil-index, 2001.

[11] David Ferraiolo and Richard Kuhn. Role based access control. In *Proceedings of the 15th National Computer Security Conference*, 1992.

[12] 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.

[13] Tyrone Grandison and Morris Sloman. A Survey Of Trust In Internet Applications. *IEEE Communications Surveys, Fourth Quarter 2000*, 2000.

[14] 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.

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

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

[17] A. Joshi. On proxy agents, mobility and web access. *ACM/Baltzer Journal of Mobile Networks and Applications*, 2000.

[18] A. Joshi, S. Weerawarana, and E. N. Houstis. On Disconnected Browsing of Distributed Information. In *Proc. of the Seventh IEEE Intl. Workshop on Research Issues in Data Engineering*, pages 101–108. IEEE, April 1997.

[19] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust-based security in pervasive computing environments. In *IEEE Computer, December 2001*, 2001.

[20] 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.

[21] 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. In *”Technical Report, TR CS-01-06”*, Department of Computer Science and Electrical Engineering. University of Maryland Baltimore County, Baltimore, MD, 2001.

[22] 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.

[23] R. H. Katz, E. A. Brewer, E. Amir, H. Balakrishnan, A. Fox, S. Gribble, T. Hodes, D. Jiang, G. T. Nguyen, V. Padmanabhan, and M. Stemm. The bay area research wireless access network (barwan). In *Proceedings Spring COMPCON Conference*, 1996.

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

[25] M. Liljeberg, M. Kojo, and K. Raatikainen. Enhanced services for world-wide web in mobile wan environment. http://www.cs.Helsinki.FI/research/mowgli/mowgli-papers.html, 1996.

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

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

[28] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized Trust Management. *Proceedings of IEEE Conference on Privacy and Security*, 1996.

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

[30] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R.Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, 1996.

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

[32] Filip Perich, Sasikanth Avancha, Dipanjan Chakraborty, Anupam Joshi, and Yelena Yesha. ”profile driven data management for pervasive environments”. In *13th International Conference on Database and Expert Systems Applications - DEXA 2002*, Lecture Notes in Computer Science, Aix-en-Provence, France, September 2–6 2002. Springer.

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

[34] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[35] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Communications*, 2001.

[36] Jeffrey Undercoffer, Andrej Cedilnik, Filip Perich, and Anupam Joshi. Centaurus2: A secure infrastructure for multi domain service discovery and utilization, June 2001.

[37] W3C. Extensible markup language.

[38] Mark Weiser. The world is not a desktop. *Interactions*, pages 7 – 8, January 1994.

[39] Thomas Woo and Simon Lam. Authentication for Distributed Systems. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996*. 1996.

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