Security Requirements in Service Oriented Architectures for Ubiquitous Computing

Domenico Cotroneo(*), Almerindo Graziano(*+), Stefano Russo(*)

(*)Dipartimento di Informatica e Sistemistica
Università degli studi di Napoli, Federico II
Via Claudio 21, 80125 Naples- Italy
(+)School of Computing and Management Sciences
Sheffield Hallam University, Howard Street, S1 1WB, Sheffield(UK)
{cotroneo, agrazian, sterusso}@unina.it

ABSTRACT

This work presents a detailed analysis of the security requirements for Service Oriented Architecture in mobile computing, still missing in the current literature. The purpose of this work is twofold. First, to provide protocol architects and software engineers with a map of security requirements in ubiquitous computing, through the evaluation of existing protocols and architectures. Second, to highlight architectural issues, including technologies and trade offs, in the design and implementation of a secure service oriented architecture for ubiquitous computing.

Keywords

Security Requirements, Service Oriented Architectures, Service Discovery, Service Delivery

1. RATIONALE AND CONTRIBUTION

In recent years we have witnessed a paradigm-shift in distributed computing from middleware-oriented architectures to Service Oriented Architectures (SOAs). The concept of service is becoming crucial, since a distributed system consists of a collection of interacting services, each providing access to a well-defined set of functionalities. In the context of SOA, a service is defined as [1] "*course-grained, discoverable software entity that exists as single instance and interacts with applications and other services*". SOAs federate such services into a single, distributed system capable of spontaneously configuring itself upon service connections and disconnections. Services can in fact be added or removed dynamically composing a changing pool of available functionalities. A service can be implemented on a single machine, distributed on a local area network or even across several company networks. In all instances, a service must first be found and then it can be accessed. To this aim each SOA relies on two distinct infrastructures called

Service Discovery and Service Delivery. Initially developed for Internet-scale environments, SOAs have been gradually adopted in more and more domains with particular interests to ubiquitous computing in all its flavours (ad hoc, nomadic and pervasive). The plethora of SOAs available today has further emphasized the heterogeneity of communication technologies and raised even more the integration and the traditionally neglected security issues. A number of service discovery and delivery protocols found in literature present some security flaws. In the majority of cases, security is only addressed from the point of view of the service delivery and in some other cases it covers the discovery phase too. More often than not though such protocols fail to address other security requirements such as those related to the service itself which we may discover and delivery securely but may be not inherently secure. The service may have gone some accidental or, worse still intentional, modification which could make it insecure to be delivered to the end entity. We believe that much of the security shortfalls of existing protocols and more elaborated service oriented frameworks stem from a poor understanding of what the security requirements are with regards to service architectures in ubiquitous computing. Much has been said with regards to security issues and requirements, such as [2, 3] in mobile computing and most of the literature focuses on the intrinsic limitations of mobile devices and environments. A variety of research studies have produced secure communication protocols, often adapting them from the traditional wired equivalent versions, such as [4], and proposed more appropriate trust models and communication paradigms, such as [5, 6]. None of the protocols and frameworks examined in our investigation has actually justified the choice made with regards to the security solutions proposed which means that even though some security is achieved this is not often sufficient and in some cases useless. It is often the case that authors concentrate on specific security aspects such as a new trust model, or how to achieve secure discovery, without looking at the whole picture or reflecting on which security requirements are actually addressed and to what degree. While addressing each and every requirement may not be possible or necessary, we believe that a clear vision of the problem area is necessary. Using risk management techniques it would then possible to make informed choices as to which requirements are to be addressed and what type of security they deserve (e.g. computational, probable etc.). This work represents a first step towards the design and the implementation of a service

oriented Secure Ubiquitous Computing Architecture. More precisely, we present a detailed analysis of the security requirements in mobile computing which is still missing in the current literature. The purpose of this work is twofold. First, to provide protocol architects and software engineers with a map of existing security requirements in ubiquitous computing, through the evaluation of existing protocols and architectures. Second, to highlight architectural issues, including technologies and trade offs, in designing and implementing a secure service oriented architecture for ubiquitous computing. In section 2 we present a survey of security issues which have been addressed in the context of standard protocols, integrated architectures, and shade some lights on ongoing research. In section 3 we present a detailed analysis of the security requirements of a SOA for Ubiquitous Computing, and evaluate the surveyed solutions against these requirements.

Section 2 presents a survey of security issues which have been addressed in the context of standard protocols, integrated architectures, and shades some lights on ongoing research. In section 3 we present a detailed analysis of the security requirements of a SOA for Ubiquitous Computing, and evaluate the surveyed solutions against these requirements.

2. SURVEY OF SECURITY ISSUES IN SOA FOR UBIQUITOUS COMPUTING

This survey has been structured into three parts addressing standard protocols, integrated architectures and ongoing research, respectively. Standard protocols cover both the middleware (e.g. UPnP) and the transport layer (e.g. Bluetooth). Integrated architectures include more complex service architectures which have been developed in the framework of research projects or commercial initiatives. Finally we also include ongoing research work. In describing the various protocols we will not describe them in detail but only address the security aspects. Some other surveys already exist in literature, which do not address though the security aspects [7, 8].

2.1 Standard Protocols

UPnP stands for Universal Plug and Play and it has been developed by a consortium of companies formed in 1999 lead by Microsoft. UPnP is designed to support zero-configuration, *invisible* networking, and automatic discovery for a breadth of device categories. The main components of a UPnP network are devices, control points (CP) and services. A UPnP device acts as a container of services and nested devices. A service exposes actions and models its state with state variables. Changes of state may in turn trigger events which the service publishes to interested subscribers. A control point is a software entity capable of discovering devices and invoking actions on the services they offer in the form of control requests to the device. A control request is a Simple Object Access Protocol (SOAP) message that contains the action to invoke along with a set of parameters. The response is also a SOAP message and it contains the status, return value and any return parameters. A device can incorporate both services and control point functionality. UPnP addresses security from the viewpoint of the SOAP control messages exchanged between control points and devices. UPnP security architecture, described in [9], introduces two new entities called the DeviceSecurity service, and the Security Console (SC). The former provides the services necessary to secure the UPnP SOAP actions (authentication, privacy etc.); the latter provides the administrative human interface for controlling the device, and it can be a UPnP Control Point (CP).

Jini is an extension of the Java programming language and was introduced by Sun Microsystems in 1998. Jini was developed with the aim of enabling users to share services and resources over a network, and providing users easy access to resources anywhere on the network while allowing the network location of the user to change. Jini Security Architecture is the outcome of Sun Microsystems' Davis project, which sets out to do provide a new programming models and infrastructure needed for a secure Jini. The Jini security framework adds very few changes to existing Jini services, mainly defining security as a deployment-time option. Using the new JSK (Jini Starter Kit) it is in fact possible to deploy an existing service in a secure way. In this respect then, Jini security is similar to J2EE security. In the Jini Security Framework both the client and the service provider can impose constraints on the service object (or proxy). For instance, once a service's proxy has been downloaded, it is possible to restrict which client (on the same device) can invoke which proxy's methods. Similarly, the client may impose certain constraints on the service provider such as that it authenticates and that it achieves integrity and confidentiality.

The design of the *Bluetooth* system has considered the security issue from the very beginning [10]. The Bluetooth security architecture includes a set of cryptographic protocols to achieve authentication, integrity and confidentiality. Bluetooth Generic Access Profile defines three different modes of security. Each Bluetooth device can operate in one mode only at a particular time. The first mode is intended for applications for which security is not required. We then have Security Mode 2, also called service-level security because the Bluetooth device does not initiate any security procedure before a channel establishment request has been received or a channel establishment procedure has been initiated by itself. For this security mode, a security manager controls access to services and to devices according to varying security polices and trust levels. The last mode is called Security Mode 3 or link-level security because the Bluetooth device initiates security procedures before the physical channel is established. This is a built-in security mechanism, and it is not aware of any application layer security that may exist. This mode supports authentication (unidirectional or mutual) and encryption.

The Salutation Architecture is a framework developed by the Salutation Consortium to solve the problems of service discovery and utilization among a broad set of equipment. Salutation V2.0 Architecture [11] addresses security from the point of view of user authentication. Two architectural parameters are defined and they are called credential and verifier. The former is used to identify a particular user; the latter is used by the target to authenticate the credential. Both parameters specify the authentication flavour, i.e. how the authentication is performed. Currently only a simple password-userID is specified.

The service location protocol (SLP) has been designed and developed by the SrvLoc working group of the Internet Engineering Task Force (IETF). It was first published in 1997

(RFC 2165) and later developed into SLP v2 (RFC 2608). One of the motivations for developing SLP was to develop a solution that allowed clients to look for a certain service based on its type and obtain the software without having to prompt the user for further inputs. The new protocol also needed to be scalable. Security is an optional feature in SLPv2 and is based on digital signatures using public key cryptography. The sender of an SLP message includes a digital signature, which is calculated over selected parts of SLP messages. The trust relationship between SLP agents is established by the network administrator who supplies the agents with the correct public and private keys. This ensures the mutual trustworthiness between communicating agents. The scope of the trust relationship is also defined by the administrator by defining the distribution scope of the public and private keys. SLP does not address access control to services which is left to higher-level protocols. SLP security is only for authenticating service advertisement messages and not the services that are advertised.

2.2 Integrated Architectures

The Secure Service Discovery Service (SSDS) was developed as part of the Ninja project [12]. SDS is Ninja's information repository providing clients with directory-style access to all available services. General security is based upon a hybrid symmetric-asymmetric encryption model where a long-lived asymmetric key is used to deliver a per-session symmetric key. Associated with every component in the SDS system is a principal name and public-key certificate that can be used to prove the component's identity to all other components. Services can be signed by such principals and clients can specify the principals that they both trust and have access to, and when they pose queries, a SDS server will return only those services that are run by the specified, trusted principals. This way only genuine services are discovered. Discovery of services is also controlled using a capability model where services can specify which *capabilities* are required to learn of a service's existence. Capabilities are signed messages indicating that a particular user has access to a class of services. Whenever a client makes a query, it also supplies the user's capabilities to the SDS server. The SDS server ensures that it will only return the services for which the user has valid capabilities.

The Centaurus project [13] was carried out at the University of Baltimore (MD) with the main design goal of developing a framework for building portals to services using various types of mobile devices. The Centaurus framework is responsible for maintaining a list of available services, and executing them on behalf of any user requesting them. Centaurus security is based on public key encryption, using a simplified PKI without CRLs (certificate revocation lists). The security architecture is based on two components. The first one is the Certificate Authority, which is responsible for generating X.509v3 digital certificates for each entity in the system (users and services) and for responding to certificate validation queries from Service Managers. The second component is the Capability Manager, which maintains a database of the group membership of entities in the system and answers requests for group membership. When a client (either user or service) registers with system, it transmits along with its certificate, an access list containing the group memberships that are required for access to the client. Also, the list of accessible services that a client receives upon

registration depends on the client's groups membership.

Burnside et al. [14] propose a protocol which is based on the use of proxies. Resource discovery in this architecture is based on the Intentional Naming System (INS) [15]. All objects in the system, e.g. appliances, wearable gadgets, software agents and users have associated trusted software proxies that either run on the appliances hardware or on a trusted computer. Security and privacy is enforced using two protocols: the first one is used for secure device to proxy communication; the second one is used for proxy to proxy communication. In the case of the proxy running on an embedded processor on the appliance, it is assumed that device to proxy communication is inherently secure. The proxy's primary function is to make access control decisions on behalf of the device it represents. Proxies are grouped into farms for ease of administration. When a new device is added to the farm, the device's proxy is automatically given a default ACL by the administrator. The core protocol of the architecture is the proxy to proxy protocol, which is based on SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure). Access control is in fact performed by a proxy using SPKI/SDSI-based ACLs. The architecture proposed has then been improved, adapting INS to store ACLs in the INS name-trees so that access decisions can be made at search time (i.e. discovery) [16].

2.3 Ongoing Research

Splendor [17] is a location-aware architecture with support for user privacy and non-repudiation. The specific problem scenario that Splendor addresses is one where services may be discovered, but mobile users may not have accounts in the infrastructure systems. Typical examples are represented by public environment such as shopping malls. The service discovery model adopted by Splendor is based on the use of a client-service-directory model with the addition of a fourth component, the proxy, used to achieve privacy for service providers, offload mobile service's computational work, and facilitate the processes of authentication and authorization. Mobile services authenticate with proxies and ask proxies to handle service registration, and key management for them. Alternatively, mobile services may do all the work for themselves without contacting the proxies. Proxies are not used though on the client's side, which is still left with the burden of cryptographic operations. Splendor assumes that existence of a PKI with a CA that signs X509v3 public key certificates for all four components of the architecture (i.e. clients, directories, proxies, and services).

3. SECURITY REQUIREMENTS

Traditional security requirements include authentication, authorization and confidentiality. With regards to service oriented architectures though security needs to be defined in terms of services themselves, the way they are dynamically added and removed, and the way they are discovered and delivered. Last but not least we must also address the availability of services.

3.1 Service Registration and Deregistration

The first set of requirements to be addressed regards secure service registration/deregistration. Before services can be discovered and delivered, they must be registered, either on a central repository or on the device where the service

resides. Where the registry is embedded into the device (e.g. Bluetooth) such requirement is less stringent as little or no communication takes place between the service provider (for example a doctor providing his/her service while in a shopping center) and the service registry; in this case stronger trust assumptions can also be made. Secure service registration is instead much more needed when external repositories (to the quering device) are used. If a service is to be registered/deregistered it is important that authentication, authorization, integrity and confidentiality are maintained. In other words only authorized service providers should be allowed to register and deregister a service from the repository. It is important that the service maintains its integrity while it is being registered and that only the intended repository is communicated the registered service (confidentiality). This is to avoid enumeration attacks which allow to perform an inventory of available services and resources. Once in the registry, service integrity must be also secured. With regards to service registration/deregistration replay prevention is also important. An attacker may eavesdrop a registration message and replay it later after the service has been deregistered, or vice-versa, thus achieving some form of denial of service (DoS) attack. Replaying an authentic, but old, service registration request may also restore an old, and deemed not secure, service after this has been patched by the original service provider and just recently registered.

3.2   Secure Discovery

The second set of requirements regards the discovery of services. If the discovery phase is compromised, then the security of all other relying services is compromised too. As for service registration, mutual authentication is also required between the entity searching for the service and the entity responding to the query. Most important is the need to control the discovery of services to authorized entities. Only authorized entities must be allowed to use the discovery protocol/and or access the registry where the services are described. This is once more required to prevent malicious users from building an inventory of available services and devices. In this sense we can introduce a requirement of authorized discovery which is also needed to control the services that are discoverable by each entity. Users may be authorized to perform a service discovery but they may only have a controlled visibility of available services, based on their set of credentials. In the more general sense, controlled discovery refers to controlling the information that is retrieved while discovering services. The limitation achieved may be one of service description (e.g. only retrieve service description of certain services), time availability (e.g. certain services can only be discovered at specific times) or even identity (e.g. it might be possible to discover a service but not the device and/or service provider that offers it).

Service discovery requests should then be confidential to the device/registry which services the requests. For example, any discovery protocol which uses unencrypted broadcast messaging suffers from lack of confidentiality as attackers may in principle eavesdrop the communication and find out which services are requested (and maybe perform a man in the middle attack); the attacker could also perform an inventory of the services available on the network, based on responses by the service registry. For the latter reason, also responses to service requests need to be confidential.

We then have a requirement we can call genuine discovery, which is to ensure that the service being discovered must be genuine, i.e. it must be as intended by the service provider and the client. Ultimately, what the genuine discovery requirement means is that the service that we find must be trustworthy. This requirement prevents the client from discovering phoney services which may compromise the security of the client. This is an important requirement because, failing that, it would be then pointless to seek the secure delivery of the service. This requirement is related but different to the one of secure service registration. A service may be securely registered but undergo accidental or voluntary changes by the time it is discovered. Where a central registry is used, this might be tampered with or more simply corrupted. It is important here to distinguish between the two possible outcomes of a service discovery process. The first one is just a list of service descriptions but the actual service is not delivered until the client has made a choice. Alternatively, when performing a service discovery, the client is returned a list of matching services. For the purposes of our evaluation, where the discovery protocol behaves as in the latter way (e.g. Jini), the genuine service requirement will fall into the secure delivery requirement. In other words, secure discovery will mean the secure delivery of the discovered service. In our case though genuine discovery will refer to the discovery service description and not to the service itself. The final requirement with regards to the discovery phase is that of anonymity. This requirement ensures the anonymity of the entity performing the service discovery. Anonymity can be intended either in terms of location anonymity or in terms of identity anonymity. The former refers to the anonymity of the location of the entity which is issuing the service discovery request. The latter refers instead to the anonymity of the entity requesting the service.

3.3   Secure Delivery

After a service has been found, it can be delivered securely provided a number of requirements are met. First of all, the recipient of the service and the service provider need to mutually authenticate. Through authentication the service provider can make sure that the service is delivered to the intended recipient, who can in turn make sure that the service comes from a known source. Delivery confidentiality must also be met so that the service is only delivered to the intended recipient(s). Finally, as for the discovery phase, we need to make sure that the service being delivered is genuine, i.e. it is authentic and that integrity is maintained. In fact, even if the communicating parties have authenticated each other and the service is delivered in a confidential manner, the service can still be tampered with or be subject to accidental modification before it reaches the intended recipient. The integrity requirement assures that the service is delivered as intended by the source, without accidental or active (by an attacker) modifications. Anonymity can also be required with regards to service delivery, and again with reference to location and identity. However, anonymity is not always a requirement as some services cannot be accessed anonymously, i.e. they require some form of identity, even if just a pseudo identity. The same applies to anonymous discovery.

3.4   Availability

One important function of a service discovery/delivery ar-

chitecture is to quickly react to faults. A service for example may not be available anymore due to server's failure or mobility for example. Availability can be defined as the property of a system which always honours any legitimate requests by authorized entities. It is violated when an attacker succeeds in denying service to legitimate users (for example using all the available resources). An emerging example of DoS attack in mobile computing emerges from the relationship between security and power conservation that some devices have. If a device has limited battery energy and tries to sleep as often as possible to conserve it, keeping it awake until this energy runs out can be an effective and selective attack. Once the battery is flat, the attacker can walk away, leaving the victim disabled. Authentication may help preventing such attacks, but not always.

3.5   Evaluation and Open Issues

In the light of the security requirements just detailed, we have evaluated the protocols and architectures surveyed. Figure 1 summarizes the results obtained. For the purposes of our discussion we will simply talk about architectures and not distinguish between protocols and architectures as we have done so far. The first consideration to make is that in most cases service registration/deregistration is not secured which translates to high risks of DoS through replay attacks. A malicious user may for instance enable a simple printer service outside permitted hours by replaying a legitimate registration message.

Our evaluation also shows how genuine discovery and delivery are underestimated. In some cases we find secure delivery but no secure discovery and there is no rationale behind this obvious contraddiction. No assumptions are made which justify trust at discovery time. Most architectures envisage some form of authentication, either simple, based on userID-password, or strong, based on public key certificates. With regards to authorization, this is also required to secure almost all protocol operations. In most cases though, authorization is only performed at delivery time to control access to services and not at discovery time to control the visibility of services. Again here, the choice made is generally not justified. Also evaluated in our study was the support for application security. In some cases the service delivered is supposed to run on the end device and/or interact with the device operating system. The service may for example be a device driver and device configuration for the use of services available from nearby peripherals. In most cases though, meeting the other security requirements can be deemed sufficient as we can make sure that the service is genuine, i.e. authentic and not tampered with. With regards to limiting the privileges of the running application so that, for instance, a downloaded printer driver cannot also read the user's address book on the device and sent to a remote device, this lays with the device specific middleware and running environment. None of the architectures addresses all the security requirements listed in the previous section. This comes to no surprise as in most cases we would expect some sort of risk assessment and contextual dependences analysis to limit the number of requirements addressed (and their relevance). However, very rarely in our study did we found an analysis of the security requirements and of the context of application. The evaluation done has also highlighted the use of different trust and authorization management models. While a discussion of the

merits of one approach over the other is beyond the scope of this paper, an issue currently unsolved is that of a suitable trust model for the management of foreign entities. Authorization management is also done differently by the various architectures, some of which use a capability model while other use traditional ACLs. Such differences emphasize the integration problems between standard technologies. We ask ourselves whether/how it is possible to achieve an integration of existing standards meeting the above defined security requirements. The alternative would be yet another custom-designed integrated architecture such as those of Ninja and Centaurus. None of the architectures offers support for anonymity or addresses the issue of integration with current research in the field. We believe that future SOAs should address the anonymity issue and address it early in the design phase. The major flaw is though the fact that none of the architectures listed here has a security-aware definition of service or a clear definition of service all together. The reason why security must be addressed when defining a service is that the design of a SOA is driven by the definition of the services that the SOA provides. Many of the listed technologies have been developed to target specific environments and/or applications. Addressing security in pervasive computing is undoubtedly harder than in traditional distributed computing because of mobility and the authors believe that a comprehensive secure SOA, suitable for all these environments at once, is a clear utopia. First, there exist too many inherent computing constraints to allow a pervasive deployment of crytpographically-secured protocols. Second, security requirements are contextually dependant, much more than other requirements could possibly be. Such dependability is greatly emphasized by mobility. We may use our PDA much more securely at home than when we are in our office, or working on the move at an airport. Addressing the security requirements for all kinds of environments and context would be too expensive or just not possible on tiny computing devices. What the authors do believe is possible though, is to have a secure SOA for clearly identified categories of use cases, and with known limitations and applicability to other contexts. In order to have such a secure SOA we need to have a clear understanding of the services that are offered and the security requirements that must be met.

4.   CONCLUSIONS

This work presented a detailed analysis of the security requirements for Service Oriented Architecture in mobile computing which is still missing in the current literature. We evaluated current solutions, in order to understand the requirements to be met when developing a SOA for Ubiquitous Environment. The security requirements detailed in this paper have been highlighted by the evaluation of the architectures surveyed. However, we believe that the difficulty in defining security requirements stems from the difficulty in clearly defining use cases for mobile computing, many of which are often too futuristic. Future work will aim at designing and implementing a service oriented Secure Ubiquitous Computing Architecture, which is based on a security-aware definition of service.

5.   ACKNOWLEDGMENTS

Requirement		UPnP	Bluetooth	Salutation	SLP	Jini	SSDS	Centaurus	Proxy-Based Security	Splendor
SSR/SSD		NO	NO	NO	YES	NO	YES	NO	NO	YES
RP		YES	N/A	N/A	N/A	NO	N/A	YES	N/A	YES
Secure Discovery	AD	NO	YES	NO	NO	NO	YES	YES	NO	YES
	ContD	NO	NO	NO	NO	NO -	YES	YES	YES	NO
	ConfD	NO	YES	NO	NO	NO	YES	NO	NO	YES
	GD	NO	NO	NO	YES	N/A	NO	YES	NO	YES
Secure Delivery	ADe	YES	YES	YES	N/A	NO	N/A	YES	N/A	N/A
	DC	YES	YES	NO	N/A	NO	N/A	NO	N/A	N/A
	GDe	YES	NO	NO	N/A	YES	N/A	YES	N/A	N/A
Privacy	LP	NO	N/A	N/A	N/A	NO	N/A	N/A	N/A	N/A
	IP	NO	N/A	N/A	N/A	NO	N/A	N/A	N/A	N/A
AS		NO	N/A	N/A	N/A	YES	N/A	N/A	N/A	N/A

SSR = Secure Service Registration	**ConfD** = Confidential Discovery	**GDe** = Genuine Delivery	**LP** = Location Privacy
SSD = Secure Service Deregistration	**AD** = Authorized Discovery (implies authentication and/or default authorization policy)	**ADe** = Authorized Delivery (implies authentication and/or default authorization policy)	**IP** = Identity Privacy
RP = Replay Prevention	**GD** = Genuine Discovery	**DC** = Delivery Confidentiality	**AS** = Application Security
	ContD = Controlled Discovery		

Figure 1: Security Requirements Evaluation

6.   REFERENCES

[1] A.W. Brown, S. Johnston, and K. Kelly. Large-scale, using service-oriented architecture and component-based development to build web service applications. *Rational Software White Paper TP032*, 2002.

[2] S. Ravi, A. Raghunathan, and N. Potlapally. Securing wireless data: System architecture challenges. *in proc. of Internatitonal Symposium on System Synthesis (ISSS 2002)*, October 2002.

[3] A. Jøsang and G. Sanderud. Security in mobile communications: Challenges and opportunities. *in proc. of Australasian Information Security Workshop 2003*, February 2003.

[4] A. Harbitter and D. Menascé. The performance of public key enabled kerberos authentication in mobile computing applications. *in proc. of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, November 2001.

[5] C. English, P. Nixon, and S. Terzis. Dynamic trust models for ubiquitous computing environemnt. *In porc. of the Ubicom 2002*, 2002.

[6] K. Zhang and T. Kindberg. An authorization infrastructure for nomadic computing. *in proc. of the seventh ACM symposium on Access control models and technologies*, pages 107–113, 2002.

[7] A. Rakotonirainy and G. Groves. Resource discovery for pervasive environments. *Lecture Notes In Computer Science of On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE*, pages 866–883, 2002.

[8] C. Lee and S. Helal. Protocols for service discovery in dynamic and mobile networks. *International Journal of Computer Research*, 11(1):1–12, 2002.

[9] C. Ellison. Device security:1 service template for upnp device architecture, November 2003.

[10] Bluetooth SIG. *Specification of the Bluetooth System - core and profiles v. 1.1*, 2001.

[11] Salutation architecture specification (part-1), version 2.0c, 1999.

[12] T. Hodes et al. An architecture for secure wide-area service discovery. *ACM Wireless Networks Journal, special issue*, 8(2/3):213–230, 2002.

[13] L. Kagal et al. Centaurus: An infrastructure for service management in ubiquitous computing environments. *Wireless Networks*, 8(6):619–635, 2002.

[14] M. Burnside et al. Proxy-based security protocols in networked mobile devices. In *Proceedings of the Symposium on Applied Computing (SAC'02)*, March 2002.

[15] W. Adjie-Winoto et al. The design and implementation of an intentional naming system. *Operating Systems Review*, 34(5):186–201, 1999.

[16] S. Raman. Access-controlled resource discovery for pervasive networks. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC)*, March 2003.

[17] F. Zhu. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *Proceedings of the 2003 IEEE Annual Conference on Pervasive Computing and Communications (Percom 2003)*, March 2003.