Acknowledgements

First and foremost, I would like to thank Prof. Stefano Russo for the words of wisdom and moral support I have received from him ever since I met him. His encouragement has helped me get through a number of difficult moments. We are all dwarfs on the shoulder of giants and Stefano is one of the giants in my life, and his advices I will cherish forever.

I would like to thank my brother Enzo whom I dearly love, and apologize to him for having to put up with my bad mood all the time, especially over the last few months. He has been a great source of strength for me, and I hope one day soon we will be able to spend some more quality time together.

I would like to thank my sister Palmira with whom I have shared more than just the stress of this last month, and whose great generosity always astonishes me. Also, I could not have gone through this experience without the reassuring presence and support of my parents who have always been by my side, even when I was not on theirs or indeed I did not deserve it.

There is a long list of people I would also like to acknowledge but I cannot forget my dear “socio” Cristiano di Flora, partner in crime in this experience, for being always so available to help me. Last but not least (as we always say), I would like to thank Domenico Cotroneo for his technical and moral advice, without which my job would have been much much harder.

Table of Contents

List of Tables

List of Figures

Abstract

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. 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. Much has been said with regards to security issues and requirements in mobile computing, and most of the current literature focuses either on the intrinsic limitations of mobile devices and environments, or propose more appropriate trust models and secure communication paradigms. However, with regards to SOAs, they do not always address security, and when they do, it is not done consistently. In other words, the notion is security is not a shared one. Most SOAs have not been designed with security in mind and only address security as an afterthought.

The contribution given by this work is twofold. First, it defines a number of security requirements for SOAs in mobile computing, giving a consistent notion of security that is used to evaluate existing SOAs and that can be used to develop new ones. The requirements defined follow a detailed analysis of all major SOAs existing in literature and the definition of new application scenarios with clear security issues. Second, but foremost in this work, the author proposes an architectural model, called SeNCA, for secure service provision in nomadic computing systems. The architecture allows the secure integration of existing SOA and provides extensibility to integrate with future service technologies. An O-O methodology is finally proposed that would allow the design of secure SOAs for nomadic computing.

Introduction

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 "course-grained, discoverable software entity that exists as single instance and interacts with applications and other services" (Brown et al., 2002). 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 technology and application domains. The notion of service has evolved along with the related discovery and delivery protocols giving birth to a wealth of different SOAs, which has further emphasized the heterogeneity of communication technologies and raised even more the traditional integration and interoperability issues of distributed computing. The actual development of SOAs has been driven by mobile computing revolution and the radical shift from a distributed computing paradigm characterized by multi-user systems to one characterized by multi-system users. The number of personal and mobile computing devices per person has increased and they all allow some form of interaction with the world we live in.

Of the different mobile computing paradigms, the one that best models the way people work and interact with each other every day is the one of nomadic computing Kleinrock (1996, 2000). Often called a hybrid model, nomadic computing is characterized by the existence of a service infrastructure, available through fixed nodes interconnected through a permanent network. Mobile devices, usually general purpose, move around freely, keeping a connection, either permanent or intermittent,

to the service infrastructure. Every day we travel from our home to our office, maybe taking a train or flying from our local airport. In other words we find ourselves moving with our laptop and other portable computing devices from one fixed location to another one, almost always in a position to access a fixed service and network infrastructure. With increased mobility though, also comes the need for discovery and delivery of available services anytime-anywhere. We want to be able to discover available services wherever we go and use them according to our needs. But in order to achieve such dynamic service availability we must first address the integration issues of exiting service oriented architectures. Given its characteristics, a nomadic computing domain will contain many of such architectures and play a natural role for the so needed integration. However, the rapid growth and diversification of service oriented technologies and protocols has added further complexity to the integration goal.

Increased mobility and new communication and interaction paradigms have also introduced new threats and security concerns, which must be addressed, even though traditional service usage scenarios, such as discovery of printers and film projectors, still fail to expose strong security requirements. However, the definition of service has recently become more encompassing as to include also people, who can now be thought as service providers (e.g. a doctor or a policeman). Thinking about people as service providers quickly changes the perspective on things and security becomes a stronger requirement to address. For instance one must address the privacy of personal information such as identity and location. In order to accommodate new application scenarios suggested by the increased mobility, a number of service oriented technologies are beginning to be used beyond the original intended purpose and this has in turn further stressed the implications of how such technologies have addressed the security requirements. Then, when trying to achieve integration of service oriented architectures one cannot escape the security requirements that must be met to provide a secure service. However what it is meant by *secure service* and what security requirements must be met is still not clear from current research literature.

In this work the author addresses the existing research gap regarding the security of service oriented architectures and their integration in the context of nomadic computing. Specifically, the research work presented here targets the following questions: 1) what does it mean for a service oriented architecture to be

secure? 2) Can existing SOAs support secure service provision as needed by modern application scenarios? And ultimately 3) is it possible to achieve secure service provision in a nomadic environment, characterized by a collection of legacy service technologies and architectures?

Much has been said with regards to security issues and requirements in mobile computing, such as (Jøsang & Sanderud, 2003) (Ravi, Raghunathan & Potlapally, 2002), 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 (Harbitter & Menascè, 2001), and proposed more appropriate trust models and communication paradigms, such as (English, Nixon & Terzis, 2002) (Zhang & Kindberg, 2002). However, much less has been said with regards to the actual security requirements of service oriented architectures, and too little if one considers the relevance of the SOA paradigm in modern mobile computing.

The first step towards providing some answers to the above stated questions is to clearly characterize the new mobile environment in which modern SOAs operate and to also identify the design requirements that may affect the way security requirements would be normally met. Second, one must be able to identify the application scenarios that drive the need for security and analyse existing SOAs to understand how security has been addressed. Based on such analysis one can then clearly define the security requirements that must be met in order to support the realization of such scenarios and evaluate how "fit to security" current SOAs really are. The purpose of the evaluation is also to clearly identify the integration issues that challenge secure integration. Based on the evaluation results and on a clear picture of the issues that must be faced, we can then address the third and last research question, that of the secure integration. The latter must be able to meet the security requirements identified and abide by the design goals the characterize the new computing paradigms. The research work being presented here has been structured following the above logical steps. Specifically, this document is organized into five chapters, summarized below.

In **chapter one** we introduce the research context, which is that of nomadic computing, and clearly identify the issues regarding the secure provision of services across the heterogeneous service and communication technologies that are available

in a nomadic domain. In order to stress the security concerns and need for security in service provision, we also introduce two application scenarios which present clear security requirements, and discuss the issues that challenge secure service provision in nomadic computing. In the final part of this chapter we discuss the design requirements of middleware for mobile computing with specific reference to the security issues. In particular we draw a comparison between traditional middleware for fixed distributed systems and middleware for nomadic computing, highlighting the differences in terms of security functionalities addressed and design goals.

In **chapter two** we review existing work with regards to Service Oriented Architectures and analyse the way security has been addressed. Our 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 **chapter three** we present the first research contribution of this work by identifying the security requirements that need to be met by a SOA in order to be considered secure. The definition of such requirements is affected by both the intrinsic functionalities of a SOA and the lack of clear use cases from which one can possibly infer the security concerns. Ultimately but foremost, we identify a problem with regards to the correct definition of service, which should be security aware, in order to make the SOA itself security-aware from the outset. Using the identified requirements we perform an evaluation of the technologies and architectures presented in chapter two, providing at the same time an in depth analysis of the issues that must be addressed to achieve secure integration.

In **chapter four** we present the second contribution of this research work, an architectural model for the secure integration of existing SOA in a context of nomadic computing, called Secure Nomadic Computing Architecture (SeNCA). A crucial part of the architectural model is the security-aware definition of service on which the whole design is based, making the proposed architecture also security-aware from the

outset. SeNCA models the nomadic environment through administration domains, which in turn contain one or more technology domains, one for each available service technology. Following the middleware design criteria identified in chapter one, SeNCA adopts the use of proxies, running on the infrastructure, to offload mobile devices of the computational complexity associated to cryptographic operations.

In **chapter five** we present an object oriented methodology for the design of secure SOAs for nomadic computing systems. The methodology uses UML extensions for the formalization of security requirements. In particular it is shown how the design of secure SOAs can address the new threat models described in chapter one. The methodology presented can be used to both design new secure SOAs and to evaluate existing ones. The methodology can also be generalized and applied to the design of all secure applications for ubiquitous computing.

Finally, in the **concluding chapter** we briefly discuss the lessons learned, and draw our conclusions

1 Security in Nomadic Computing Systems

In this chapter we introduce the context of our research, which is that of nomadic computing, and identify the issues regarding the secure provision of services across the heterogeneous service and communication technologies that are available in a nomadic domain. In order to stress the security concerns and need for security in service provision, we also introduce two application scenarios which present clear security requirements, and discuss the issues that challenge secure service provision in nomadic computing. In the final part of this chapter we discuss the design requirements of middleware for mobile computing with specific reference to the security issues. In particular we draw a comparison between traditional middleware for fixed distributed systems and middleware for nomadic computing, highlighting the differences in terms of the security functionalities that should be addressed and design goals that should be followed.

1.1 Modern Mobile Computing

In the last decade, the distributed computing paradigm has undergone major changes and diversifications that have radically affected the way we think about and develop distributed applications. Words like ubiquitous, pervasive and smart have become more and more common to the extent that they are sometimes used interchangeably. In particular we have witnessed a radical shift from a distributed computing paradigm characterized by multi-user systems to one characterized by multi-system users. The number of personal and mobile computing devices per person has increased and they all allow some form of interaction with the world we live in. A possible classification of distributed systems can be done according to a number of characterizing parameters, listed below.

- *Device* – we can define a device as a network node that has computational capabilities. A device in turn can be classified in terms of its mobility and level of embeddedness. Generally, fixed devices tend to have a higher computational power and be more general purpose. Similarly, with increasing mobility we tend to have reduced computing power and higher level of embeddedness, i.e. the devices are built for specific purposes (e.g. sensors, badges, microcontrollers etc.). Mobility will also affect the device availability

due to limitations in battery life and connectivity. Mobile devices for example tend to be turned off more often or be in areas with no or scarce network coverage

- *Network connection*, i.e. the quality of the physical connection between two network nodes, not the quality perceived through any communication layer/middleware. Roughly, a network connection can be either permanent or intermittent. The former is usually characterized by a larger bandwidth and lower error rate than the latter where several factors may affect the overall characteristics.
- *Context* – with this term we refer to any information that may affect the behaviour of the application, such as battery status, CPU load, memory usage etc. Context may also be affected by environmental factors such as network bandwidth or location of the device. A context can be more or less dynamic depending on how it is affected by the available information.

Having introduced the above parameters, we can better describe the different domains of distributed computing. In particular we can define:

Traditional distributed computing – collection of fixed and general-purpose devices communicating through a permanent connection, and where the contextual dependencies, usually defined upfront, tend to be rather static.

Pervasive computing – collection of special-purpose devices, both mobile carried by the individuals, or fixed such as sensors, RFID readers etc., all transparently interweaved into every day's appliances and applications. The goal of pervasive computing is in fact to be invisible and transparent. The type of connection is usually intermittent even though some pervasive devices may be connected and communicate through a fixed network infrastructure. The context is very dynamic and can change quite rapidly, affected by the interaction of highly mobile pervasive devices.

Ubiquitous computing – very similar to pervasive computing both in terms of dynamicity of context and transparency of use, but with less emphasis on invisibility.

Network connections are also intermittent but the bandwidth in assumed to be higher to support ubiquitous interaction between computing devices.

Nomadic computing – often also called a hybrid model, is characterized by the existence of a service infrastructure, available through fixed nodes interconnected through a permanent network. Mobile devices, usually general purpose, move around freely, keeping a connection, either permanent or intermittent, to the service infrastructure. The latter may also include some pervasive devices such as sensors, but the emphasis in nomadic computing is on the transparent access to services and general purpose computing from a mobile user. The context is also dynamic but it varies less rapidly than in pervasive computing.

Ad hoc mobile computing – contrary to the nomadic one, this domain is characterized by the lack of a fixed infrastructure of reference and the devices must be capable of some form of autonomy to both auto configure themselves and interact with other devices. The context is also dynamic and the connections are mostly intermittent although they can also be permanent in some other cases when using wireless LAN technologies.

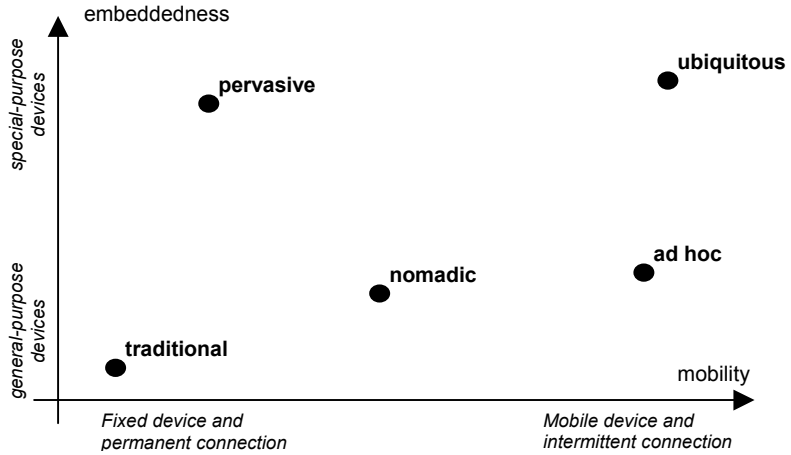

Figure 1-1 Modern Distributed Computing

Figure 1-1 shows a graphical representation of the above classification. In the context of this work we will mostly refer to ubiquitous computing to address the concept of anytime-anywhere computing and we will refer to nomadic computing to specify the presence of network and service infrastructures.

1.2 Nomadic Computing

This work focuses on nomadic computing Kleinrock (1996, 2000) as this better models the way people work and interact with each other. Every day we travel from our home to our office, maybe taking a train or flying from our local airport. In other words we find ourselves moving with our laptop and other portable computing devices from one fixed location to another one, almost always in a position to access a fixed service and network infrastructure. Wireless and transparent access to fixed networks is becoming more and more ubiquitous, either through more traditional 802.11 networks or through modern 3G communication technologies. The ubiquity of these network and service infrastructures is such that the times when the user will be "on his own" are few and far in between. The availability of such infrastructure and the increasing diffusion of mobile communication technologies such as Bluetooth is gradually fulfilling Weiser's vision (1999) of pervasive and ubiquitous computing and the realization of a number of application scenarios for office automation, smart homes and smart airports. A typical nomadic infrastructure will include a collection of communication and service technologies. Typically, a nomadic computing environment will include a number of different service oriented technologies, as shown in figure 1-2.

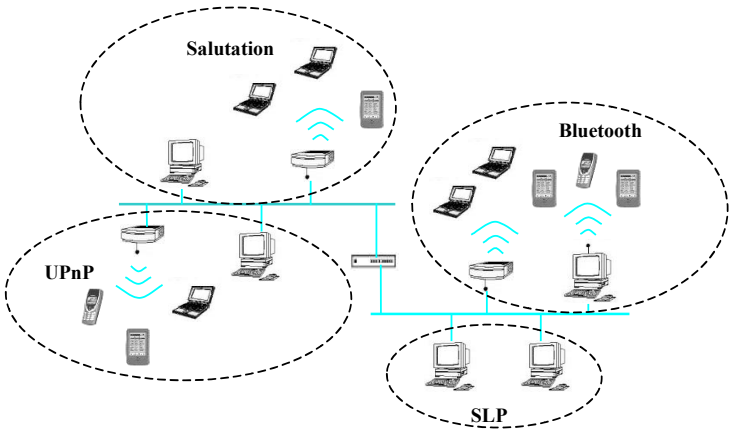

Figure 1-2 Nomadic computing architecture

With increased mobility though, also comes the need for discovery and delivery of available services anytime-anywhere. We want to be able to discover available services wherever we go and use them according to our needs. However, the multiplicity and heterogeneity of mobile technologies, both in terms of physical

communication and middleware, has greatly affected the goal of transparency and ultimately of service availability. Furthermore, mobility has also stressed the security issues. Even though in many cases a user will have access to a network infrastructure, the level of trust the user will place on it will vary according to the context. We may decide to trust our home network more than our office network. Contrarily, a work colleague may trust the corporate network more than our home network. Also, security is addressed differently across mobile technologies and providing end-to-end security in the nomadic domain is only limited to the lowest common denominator; we are only as strong as the weakest link in the chain. The research presented with this work addresses the problem of secure service provision in nomadic computing and in order to better appreciate the importance of security we are first going to present two application scenarios of nomadic computing which clearly highlight the need for security. Traditional scenarios, such as printer service discovery, tend to hide such need thus to mine the design-with-security-in-mind approach that is required.

1.3 Terminology

Before we delve into the discussion about the security issues in nomadic computing, it is useful to briefly introduce the terminology we will be using extensively throughout this work.

- We will use the terms *subject*, *principal* or *client entity* to identify the initiator of an action or service/resource request to which we can apply access control
- We will use the term *credentials*, based on the CORBA Security model (OMG, 2001), to address the security attributes associated to a subject (or principal or client entity), which are used to make access control decisions. Credentials may include the principal's identity, information about his role, associated privileges etc.
- We will use *object*, *service* or *resource* to refer to the target of any invocation or request from a subject. An object is protected through access control which makes decisions based on the subjects' credentials and some access control policies

With regards to basic security concepts such as authentication, integrity, confidentiality authorization and other, the reader can refer to (Stallings, 2003).

1.4 Secure Scenarios in Nomadic Computing

1.4.1 The Smart Campus

Professor Steve is meeting up with his PhD student Al for an update on Al's work and in preparation for next week conference in Berlin where Al will be presenting a paper on "Secure Ubiquitous Computing". The meeting is at the University coffee shop which is wireless-enabled allowing a connection to the University Intranet. Steve and Al have a Wi-Fi and Bluetooth enabled Smartphone and PDA respectively. They order a coffee and a bite to eat and they start discussing the paper; from time to time Al adds some new tasks onto his PDA to remind him of important things he needs to do while at the conference. After a bit, Steve suggests they should go through a formal rehearsal of the Power Point presentation using a projector. They have not booked a room and it is now 6 o'clock p.m. and there is no staff to do the booking for them. All the university offices are closed but they still would like to find an available room with a projector in it, and possibly a printer. Steve uses his 802.11 enabled Smartphone to locate the nearest room with a projector but he belongs to the faculty of Engineering and he can only have access to the available rooms in the engineering buildings. Unfortunately, no room is available from the faculty of engineering, but one room is available in the nearby faculty of Physics; so he reserves the room which instantly becomes unavailable for further bookings over the next hour. However, Steve is not really familiar with the location of the room. Fortunately for him, the on-line booking system also makes available a guide service and a local map is downloaded onto his PDA with both the current position and destination clearly marked, along with the directions. Alternatively, an active map is available which shows their current position as they move, using a combined Bluetooth Wi-Fi locationing service. Once reached the room with the projector, Steve and Al authenticate through the Bluetooth-enabled door and are able to get inside. The Projector is also Bluetooth-enabled and access controlled just like the door. Even though Steve is not from the faculty of Physics, the authentication and authorization has been handled transparently, thanks to administrative agreements between the two departments.

While they are working on the presentation, Steve does not want to be disturbed unless it is really urgent. Therefore he sets the access policy on his Smartphone to "do not disturb". This policy is also associated to a users profile so that

the phone will only let through calls or messages from the people contained in the profile. Additionally, as he himself doesn't quite know how do direct people to the current room where he's never been, he would also like the people associated with the above profile to be able to locate him. For everyone else he is utterly and definitely unavailable. However, Steve would also like to know who has been looking for him while he was busy with Al. Similarly, whoever looked for Steve might not want to let him know they were looking for him.

After working on the presentation, Steve and Al decide that maybe they should print some handouts to distribute at the conference so that attendees can get more out of the presentation. The room they are in doesn't have a printer so Steve looks for the nearest one, using the same service look up as before; this time though a wireless network is not available in the room and the service lookup is done using a Bluetooth network. It turns out that there is a UPnP printer available in the room next door and a Jini-enabled printer in the room down the corridor. Both printers are access controlled using the native technology and Steve is not recognized to use either of them. However, authorization is granted through the network infrastructure.

1.4.2 The Mobile Emergency Service

This scenario is based on Zhu, Mutka, and Ni's M911 scenario (2003). Rob is a general practitioner (GP) and he has a real dedication to his job. Recent advances in mobile technologies have made possible for him to offer his services also when out of duty in emergency situations. The same applies to Mark who is a law enforcement agent. Contrary to Rob though, Mark has been contracted to offer his services on demand even when formally off-duty. Both Rob and Mark's positions are monitored through the Neighborhood Positioning Service (NPS), which operates using a number of technologies, namely RFIDs, Bluetooth, GPS etc. It is the weekend and both Rob and Mark are out with their respective family for a nice meal in one of the new restaurants in town. As they enter the restaurant, their presence and a description of the potential services they may offer (i.e. doctor and policeman) are registered with the Neighborhood Service Infrastructure (NSI); if needed they can be located and contacted. While Rob and Mark are enjoying their evening, someone in the restaurant next door is committing a robbery and at the same time one of the customers, Luis, suffers a heart attack. Luis pushes a button on his M911 device which places an automatic call to the 911. In addition, the device sends an M911 request to find any

doctor in the vicinity, who may help. Similarly, another customer pushes the button on the M911 device to call for a police officer in the vicinity. Both Rob and Mark are close to the scene and they are located by the M911 service operating through the NSI. Once contacted by the M911, Rob and Mark are given automatic direction to the scene of the crime. Further information is also available to Rob regarding Luis' clinical records available through Luis' smart phone. Luis doesn't want his medical record exposed to total strangers but only to trusted doctors. Trust is managed through the NSI, which recognizes Rob as a trusted person to access Luis' medical records. Ultimately, if no trusted doctor is in the vicinity, Luis is happy to share his medical records with any doctor available in the area.

1.5 Issues in Achieving Security in Nomadic Computing

The first consideration that is possible to make by reading the above scenarios is that they introduce a new notion of service compared to the one we are traditionally more used to. But on the actual definition of service we will concentrate in chapter four. In this instance we can say that both scenarios above present a number of security related issues. For example, we do not want everybody to know how many rooms there are around with projectors or printer or other devices. This is to prevent enumeration attacks for the purpose of theft (for example) or other. Similarly, we may want to limit the visibility of the number of doctors and law enforcement agents available in a specific area. Alternatively, criminals may decide to look for "safer" areas where a robbery can be carried out successfully. We then have issues regarding the disclosure of private information. John, as a person, may provide a service (e.g. resolve an urgent problem) or pick up a phone call from a designated person or group or people. As a service, John may decide to limit or control his availability according to his needs and according to the company policy (e.g. it must be always possible to locate John). John could set the privacy control policy on his PDA or smartphone and such policy may change with the context (for instance based on John's location). As mobility increases the number of potential services, there is also an issue of scalability, which can also be addressed by performing careful access control to services. Availability is another issue to address; if the service is discoverable via Bluetooth or via any other batter operated device, continually querying for the service would flatten the battery of the device, making the service unavailable. These are just

a handful of security-related issues and we will address them more thoroughly in chapter three.

Traditionally, security requirements have been addressed with the use of encryption algorithms and security engineered communication protocols. Achieving security in the same way in the context of nomadic computing is somewhat more difficult for a number of reasons.

1.5.1 Limited Resources

In 1965 Gordon Moore, co-founder of Intel, predicted that the number of transistors per square inch on integrated circuits would double for the foreseeable future (Gordon, 1965). So far that prediction, which is also widely known as Moore's law, has been maintained and Intel expects that it will continue at least through the end of this decade. Unfortunately, Moore's law does not apply to **battery life** and, even though battery technology has advanced substantially in the past years, these advances have not been as outstanding as in microprocessor development. Low power consumption is very important in mobile and wireless devices, because a good percentage of the battery power typically needs to be reserved for the radio transmission activities. ABI Research (2003, cited in Paulson, 2003) reveals how common nickel-cadmium and lithium-ion batteries, generally used in laptops, cellular phones, PDAs etc., have only increased their energy by 10 to 15 percent per year, and are capable of providing only another 15 to 25 percent. This will not keep up with the increasing power demands of mobile devices with fast processors, high resolution displays, games and other intensive applications, such as cryptographic operations.

Another limitation is the **lower processing power**. Cryptography still is the main vehicle to achieve all security requirements (unless we are prepared to use security through obscurity). However, cryptographic operations, especially public-key based ones, are CPU-intensive. A common public-key cryptographic algorithm such as RSA using 1024-bit keys takes 43ms to sign and 0.6ms to verify on a 200MHz Intel Pentium Pro (Weiner, 1998) and it takes much longer on mobile devices that fit even smaller CPUs (Ravi,Raghunathan and Potlapally, 2002). With regards to CPU limitations research is currently pursuing several routes. In some cases traditional protocols have been adapted to the requirements of the wireless environment (Gupta and Gupta, 2001)(Harbitter and Menascé, 2001). Lots of effort is being put onto developing alternative public-key cryptosystems such as the Elliptic Curve

Cryptosystems (ECC) (Koblitz, 1987) and the lattice based cryptosystems (Hoffstein, Pipher and Silverman, 1998). Some security protocols can also be made to adapt their encryption policies based on the content of the data being encrypted. Video encryption algorithms such as those proposed by (Tosun and Feng, 2001) focus on protecting the more important parts of a video stream thereby reducing the total amount of data encrypted. Some researchers (Ravi, Raghunathan and Potlapally, 2002) even stress the existence of a *processing gap*, claiming that it will not be filled if not by the development of new hardware and software designs. The same researchers propose a platform called MOSES which can also be used as a coprocessor in a handheld device to accelerate security-specific computation.

Security is also affected by the **lower storage memory** for a number of reasons. First, the limited size of storage memory also limits the access control data structures (a.k.a access control lists or ACL) that we can store in them. Second, lower storage capacity also accounts for the increased difficulty to support multiple security protocols, and the flexibility demanded for the support of multiple encryption algorithms. Access control itself is still an unresolved issue in nomadic computing. In fact, a single point of access control is not sufficient because mobility may make that single point unavailable. On the other hand, keeping access control structures on the single device is not a viable solution either; first because of the limitation of storage memory which would make it not scalable, and second, because the administration associated to such an approach would be unbearable; it would entail operating on each individual device in order to modify access rights. For this reason, some form of distributed access control solution is required, which unfortunately increases the exposure to attacks.

We then have **usability**; protecting and securing small mobile devices is definitely harder than for traditional devices, characterized by handy keyboard for inputting user credentials. For starters, the limited size and poor user interface of some mobile devices already pose problems for implementing user-friendly applications. Such limitations affect even more the achievement of reasonable security. Issues of HCI (Human Computer Interaction) come into play, and securing a small device in a way that it is still user friendly becomes a great challenge. Whitten and Tygar (1999) argue that effective security for mobile devices requires a different usability standard, and that it cannot be achieved through the user interface techniques appropriate to other types of consumer software. With regards to authentication,

Jansen (2003) presents an extensive surveys of ways to achieve authentication in handheld devices.

Another limitation is given by the **communication channels**. While in the traditionally static environment we can afford expensive protocols which introduce large overheads and which are computationally expensive, not the same can be said for the mobile environment often characterized by lower and less reliable bandwidth.

Limited resources are not only dictated by technological constrains but more so by **economies of scale**. Typically, personal consumer devices such as cellular phones are manufactured in extremely large quantities and are sold to price-conscious consumers at prices that are very low. To improve their profit margins, device manufacturers want to keep the per-unit costs of the devices as low as possible. Additional processing power or precious dynamic memory will not be added unless the consumers are willing to pay for the extra capabilities. Also, the increased cost given by added general-purpose features and capabilities would have to be well-justified from the viewpoint of the target market and the consumer.

On the whole we can sum up the limited resources issues by saying that they impose constraints on the practicality of applying standard safeguards. Even if such constraints weren't there though, some mobile devices' small size and mobility would inevitably lead to greater exposure to theft or misuse in the field.

1.5.2 Flexibility

Everyone agrees that security is an end-to-end requirement and that we are only as strong as the weakest link in the chain. A Nomadic domain will contain a number of different communication technologies (bearers) on top of which numerous communication and application protocols are layered. From a modelling point of view, we can say that security can be addressed at four distinct layers:

1) *Communication technology*– Security in this domain relates to physical medium and the protocols involved in the establishment of physical link (wired or wireless) between communicating parties. It also deals with access control that lets only authorized devices connect to the physical network
2) *Network* – This layer includes the traditional network and transport protocols running on top of the existing communication technologies. Security at this layer

relates to the provision of a secure communication layer, independent of the underlying communication technology.

3) *Service* – Security in this domain relates to service oriented technologies and addresses service discovery and delivery protocols.

4) *Application* – Security here relates to application between distributed nodes. Applications are meant to be developed on top of the service technologies and protocols or directly on the network layer

While all other layers are more neatly defined, the service layer is the one that is most problematic to place in the model. As we will see in chapter three, existing service oriented protocols cannot be placed precisely in any one layer; some operate directly on top of the network layer, some others can be found in the application layer.

In the traditional distributed scenario security is achieved mostly through protocols at the network and transport layers (the ones we have identified as *network* only), using IPSec and SSL/TLS respectively. The new wireless communication technologies though, have introduced a number of other "layer one" (in the above model) security protocols. In the new distributed scenarios now, mobile communications can be secured by employing security protocols from any combination of the above layers, with each protocol addressing security in its own way and using a different combination of encryption algorithms (symmetric or asymmetric). Consequently, moving around in a nomadic environment, a mobile device would be required to execute multiple distinct security protocols at once. In turn, this requirement calls for flexibility. Unfortunately the latter requirement is much harder to achieve in today's mobile devices, characterized by limited processing and storage resources, and not suitable to run traditional middleware, through which in the past flexibility could be achieved.

1.5.3 New Threat Models

The diffusion of mobile technology has also greatly affected the number and types of threats to which we are all used from the traditional distributed computing. A nomadic environment will contain numerous wireless technologies and therefore inherit their threats and security challenges. The threat model for the traditional distributed computing is based on the notion of a *castle,* which contains all the resources and

information that we use and that we want to protect. The castle is surrounded by a security perimeter, defended by a system called *firewall*. The assumption on which a firewall works is that everything inside the perimeter is trusted, while not everything outside is. For this reason, a firewall cannot protect against internal threats or from malicious users from the inside. The trust model is mainly static and pre-evaluated. Traditional firewalls have then evolved into distributed architectures (Bellovin, 1999), which allow the use of firewalls on individual machines inside the defence perimeter, with a security policy dictated and administered by a central server. Both traditional and distributed firewalls though work on the assumption that the threats come from the outside and do not take into consideration the mobility of users and devices, nor the existence of alternative communication paths.

Wireless technologies allow for easier deployment of unauthorized equipment, as no wiring is required, and traditional protection mechanisms such as firewalls and IDS (Intrusion Detection System) can be more easily bypassed. If devices can be installed without permission there is also the danger that an attacker may set up unauthorized services and steal legitimate identities, and credentials in general, in the process.

Serious security concerns stem from the variety of ways in which mobile devices can interact with other, potentially malicious, computing resources, which can *infect* the device. Since PDA-enabled, application-level malware cannot typically be blocked by corporate firewalls, a device like the PDA may serve as a back channel through which network vulnerabilities can be exploited. Also, most mobile devices today incorporate various communication technologies (802.11, Bluetooth, GSM etc.) over which traditional defence mechanisms can exercise only very limited control, if any whatsoever.

Another cost of mobility is the higher risk of loosing the devices or having them stolen. Theft is more likely to occur with wireless devices because of their portability. Also, it is easier to lose a PDA than a laptop, both storing potentially the same security-sensitive type of information. For this reason, tamper-proof solutions become very important for such devices, a requirement which in turn must come to terms with the usability of the device, already hindered by its physical limitations.

Compared to the traditionally static threat model then, the new computing paradigms must account for mobility and wireless technology. In particular we can define three possible threat models, described below. Our classification is based on

the idea of Ostrovsky and Yung (1991) who first introduced the notion of a mobile adversary, later refined by Herzberg et al. (1995). The threat models can be applied to complex systems as well as to simple devices we wish to protect.

- *Traditional adversary* – This is the traditional model, based on the notion of a security perimeter; everything inside this perimeter is trusted and everything outside the perimeter is not trusted. A traditional adversary has considerable resources at its disposal (bandwidth, processing power, time to plan an attack, etc.). As an attack can be carried out from the distance, a traditional adversary is likely to be anonymous (at least the real attacker).
- *Mobile adversary* – This threat model considers the threats introduced by malicious entities (either an attacker or a compromised device), who are mobile and can therefore move freely inside the security perimeter. Compared to a traditional adversary, a mobile adversary must come into closer contact with the intended victim so it will be more exposed (more difficult to be anonymous). A mobile adversary is assumed to have fewer resources than the traditional one (limited to what can be carried around to perform the attack); time is also likely to be less.
- *Mobile victim* – this model takes into the consideration the threats a mobile device exposes itself to with its mobility. The model considers the likely application scenarios that the device will be involved in, trying to anticipate the possible threats that are associated to each scenario. Albeit similar, the mobile victim model is different from the mobile adversary model as it only considers the threat situations caused by the victim. The mobile adversary can be modelled as the user's *alter ego*, who will perform actions that may threaten the system, such as leaving the laptop unattended while in a public place. In particular, one could model the actions of a mobile victim in terms of the set of possible actions that may be enabled (or rights that may be granted) to either a mobile or traditional adversary.

The use of these threat models can better help address all the possible threats in a scenario of nomadic computing. In chapter five we will present a methodology for

modelling these threats using a combination of misuse cases and extensions to the UML modelling language, called UMLsec.

1.6 Security-Aware Middleware

Mobility, along with wireless technologies and heterogeneity of computing devices has also greatly affected the way middleware is supposed to address the security requirements. The concept of middleware as we know it traditionally is no longer valid. The paradigm shift in distributed computing has also brought about a radical rethinking of the definition of middleware as the traditional design requirements are greatly affected by physical mobility of modern devices and new wireless technologies (Mascolo et al., 2004). Existing middleware technologies have been developed so that distributed application developers wouldn't have to explicitly deal with problems related to distribution, such as heterogeneity, scalability, resource sharing and security. The main driving principle of traditional middleware, such as transaction-oriented, message oriented or object oriented middleware, has been the one of transparency, achieved through the use of higher levels of abstraction.

Modern middleware on the other hand is required to be *adaptable*, *available*, *secure*, *modifiable* and *performing* (Raatikainen et al., 2002). These qualities are dictated by the needs of new application scenarios which are enabled by the diffusion of wireless communication technologies and mobile devices. However, how modern middleware is supposed to address all of these qualities is still an issue. Mascolo et al. in (Mascolo et al., 2002) present a survey of existing middleware for mobile computing and introduce a reference model for the characterization of existing middleware solutions, based on *computational load*, *communication paradigm* and *context requirements*. Mascolo's reference model clearly shows the differences between the traditional middleware for fixed distributed systems and the various middleware solutions for the mobile computing paradigms.

In order to better discuss the characteristics of modern middleware with regards to the security requirements, we will first describe how security has been handled traditionally in middleware for fixed systems.

1.6.1 Security-Aware Middleware for Fixed Distributed Systems

To date CORBA (OMG, 2002) represents the best example of traditional security-aware middleware and we can use it as a reference model to better discuss the issues

regarding security-aware middleware for mobile computing. Figure 1-3 (OMG, 2001, p. 30) below depicts the model for CORBA secure object systems. All object invocations are mediated by appropriate security functions to enforce policies such as access controls.

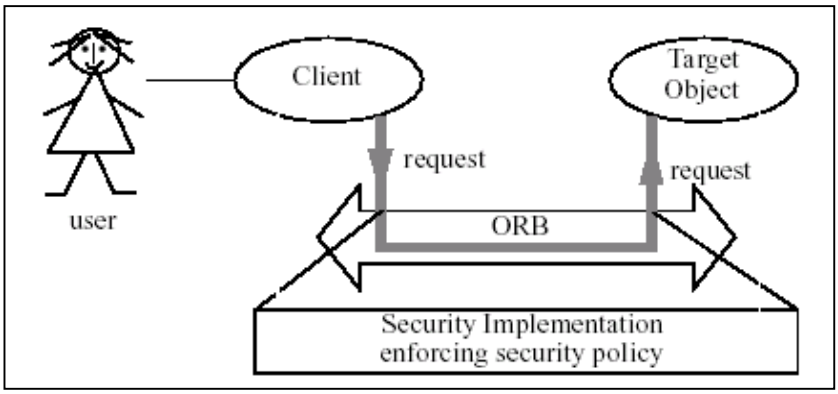

Figure 1-3 CORBA security model

CORBA security model envisages two possible security levels and a CORBA ORB (Object Request Broker) must provide at least one of these levels before it can claim to be a Secure ORB:

- **Level 1**: This provides a first level of security for applications which are *security-unaware* and for those having limited requirements to enforce their own security in terms of access controls and auditing.
- **Level 2**: This provides more security facilities, and allows applications which are *security-aware* to control the security provided at object invocation. It also includes administration of security policy, allowing applications administering policy to be portable.

In particular, CORBA security specifications (OMG, 2001) provide functionalities for:

- *Identification and authentication* of principals (human users and objects that need to operate under their own rights) to verify they are who they claim to be
- *Authorization and access control* – required to control access of principals to secured objects

- *Security Auditing* – required to make users accountable for their security related actions
- *Security of communication between objects* – required as communication between objects often occurs over insecure lower layer communications. Secure communication may include mutual authentication of client and server along with protection of the integrity and confidentiality of messages exchanged (message protection).
- *Administration of security information* – this is required to administer and apply security to a group of objects and applications (a domain), regardless of whether the applications are security aware.
- *Non-repudiation (optional)* – required for the creation, transmission and storage of irrefutable evidence that can be associated to an action performed by a principal, against subsequent attempts to falsely deny the receiving or sending of the data.
- *Delegation* – clients can authorize intermediate targets to user their identity or privileges, optionally with some restrictions.

All such functionalities rely on the use of other security functionalities such as cryptography, which are though transparent and not directly accessible by any application objects. In particular, a number of security-related architectural goals were defined, even though their achievement was not mandatory for the purposes of compliance to the security specifications. The goals set were simplicity, consistency, scalability, usability for end users, administrators and implementers, flexibility of security policy, independence of security technology, application portability, interoperability, performance, and ultimately but most important of all was the goal of object orientation. The end result (OMG, 2001) is a complex security architecture, described in over 400 pages thus failing the "first" stated goal of simplicity.

The most remarkable results have been achieved with regards to security interoperability. The latter is addressed through the CSIv2 architecture (Common Security Interoperability) (OMG, 2002) whose objective is to define the format and rules to send the security information from a client to a server, between CORBA to CORBA objects, J2EE to J2EE objects and between CORBA and J2EE objects. The CSIv2 architecture defines three layers as shown in figure 1-4.

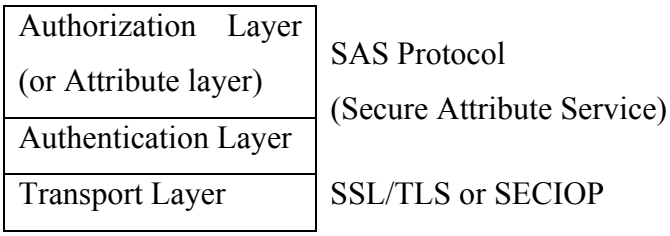

Figure 1-4 CSIv2 Architecture

As we can see from the figure, the core of the CSIv2 architecture is represented by the SAS protocol, intended to be used on top of a secure transport layer protocol (SSL/TLS support is mandatory), in charge of providing message protection (i.e. integrity and or confidentiality) and server-to-client authentication[1]. The *authentication layer* is used to support client authentication that may not be supported or not sufficiently supported by the underlying transport protocol. The *attribute layer* may be used by a client to deliver security attributes (identity and privilege) to a target where they may be applied in access control decisions.

SAS is a secure connection-oriented protocol based on the concept of a *security context*, which needs to be established before a client can access a target object on a server. For each object that needs to be secured, the server specifies a number of security requirements, for each one of three layers defined in the CSIv2 architecture. Before accessing the remote object, a client will check to see whether the target's requirement can be met and whether these in turn meet the client's own security requirements. Both client and target's security requirements are expressed in terms a defined set of *security associations* (e.g. Integrity, Confidentiality, EstablishTrustInTarget, EstablishTrustInClient etc.).

1.6.2 Security-Aware Middleware for Nomadic Computing

Before discussing the issues regarding the support of security requirements in middleware for nomadic computing, one should first try to identify the security functionalities that are needed by such a middleware, and then set the architectural goals that should drive its design. In the previous section we listed the security functionalities that were sought for in a mature middleware for fixed distributed

[1] One drawback with SAS is that target-to-client authentication is done at the transport layer and is not supported by neither the authentication layer nor the attribute layer.

systems such as CORBA. While all the functionalities defined for the traditional middleware still hold, we cannot say the same for all of the architectural goals. However, we can make some considerations with regards to the security functionalities for a security-aware middleware for mobile computing.

For starters, **secure administration** is equally important if not more than for traditional distributed computing. Nomadic computing brings together a number of heterogeneous technologies along with associated services; mobility in turn increases the possibility to access such services. Security administration is vital if one wants to manage and control access to a vast number of services from a dynamic set of moving entities. This is particularly true if an encompassing definition of service, as we will see later on in this work, makes the number of available services highly dynamic and countless.

The second consideration regards the functionality of **access control**, which is required to restrict the use of services only to authorized entities. This functionality is greatly affected, as discussed earlier, by the limited resources of a number of modern computing devices. However, in this context limited resources means that, compared to traditional fixed devices, smaller mobile devices are less likely and not meant to host numerous objects that need securing. Furthermore, the size and type of services running on such devices will not need the same access control granularity that is justified for traditional fixed devices. For the same reason it is not expected that the access control functionality will provide mechanisms for the support of service or application specific factors, nor support for both security-aware and security-unaware applications or services running on the device, as in CORBA. Also the **audit** and **delegation** functionality, as for the access control, are not likely to be performed by a small device, but more so by the service infrastructure that must manage the integration of technologically heterogeneous and administratively independent domains.

As regards the functionality of **secure communication**, the middleware must take advantage of the security services implemented by the underlying bearer technologies and, as the user moves across different technologies, it must also adapt to them. In other words, the secure communication functionality must be implemented in a technology-aware, and adaptable fashion.

When it comes to designing a middleware that has all the above functionalities, the most difficult part is doing it while meeting the architectural goals or design

requirements, which are dictated bythe context of application, in this case being the nomadic computing. Also, when looking at security-specific architectural goals, one must bear in mind that traditional design principles for middleware do not still hold in the context of mobile devices, as discussed earlier, while new ones find their way in. The following is a list of security-related design principles or goals which the authors believe should be considered in a middleware for nomadic computing devices.

- *Simplicity* – This goal, defined for traditional middleware, is even more applicable here and refers to a security model that should be simple to understand and administer.
- *Scalability* – With increased mobility, also comes a higher number of available services and service users. Scalability is therefore a dominant design goal. However scalability cannot be addressed solely or primarily by the middleware on the devices, due to their limited resources.
- *Independence of security technology (transparency)* – This is still required so that applications can be largely unaware of the underlying security technology. However, the same level of abstraction achieved in traditional middleware is not suitable for the new middleware. Traditional in fact, the diversity on underlying technologies was flattened by the common communication substrate (TCP/IP) and the problem was that of establishing a secure communication context between a client and a server (e.g. CSIv2). In a mobile context such an approach would be too cumbersome and unsuited given the disconnected communication paradigm. Ideally, the middleware should be modular so that different underlying security services could be used without needing the middleware layer being too thick, adopting the concept of *just enough middleware* (Subramonian and Gill, 2002) or some sort of *security profiles* specialized for type of devices, following the approach taken by the design of the Java 2 Micro Edition platform (J2ME).
- *Consistency* – This goal refers to the possibility to provide consistent security across a distributed system. Achieving this goal is going to be very challenging as security is already addressed inconsistently across existing service oriented technologies (we will show this in chapter three). Consistency refers also to fitting with existing environments, providing end-to-end security even when using communication services, which are inherently insecure. Given the intrinsic

resource limitations across the range of mobile devices, the middleware will not be able to bear the mechanisms required to support such consistency; this is yet another challenge which must be addressed with the help of the nomadic service infrastructure.

- *Usability for end users, administrators and implementers* – This is one of the most fundamental goals that must be met by the middleware. However, as we discussed earlier, security and usability together are more difficult to achieve in mobile devices. Another problem here is related to traditional approach to usable security, which tends to make security services and mechanisms as transparent as possible. However, if security is totally hidden from the user it can lead to system misuse or misplaced trust. A user might for example assume that the system is working securely (transparently) while it is not, and allow successful attacks to remain undetected. Some security awareness must then be assumed from the user and the challenge is to determine what type of evidence is really necessary and present it to the user in an intuitive and intelligible way.
- *interoperability* – This is an even bigger requirement than for traditional middleware given the number of already existing heterogeneity of service oriented architectures and communication technologies which address security in different and non-interoperable ways, as we will see in chapter three. Also, this design goal is considerably more difficult to achieve than it was traditionally, as the complexity that it entails cannot be absorbed by the middleware running on the device.
- *performance* – This design requirement is much stronger than for traditional middleware, which could benefit from powerful general purpose fixed machines.

The following architectural goals, on the other hand, cannot be considered anymore applicable to middleware.

- *Application Portability* – Traditionally, one of the architectural goals was to support application portability, i.e. the fact that an application object should not need to be security-aware, so it could be ported to environments that enforced different security policies and use different security mechanisms. This is no longer

a design requirement as applications tend to be more device-specific or device-bound and slim lined to fit the resource limitations of current devices.

- *Flexibility of security policy* – The middleware no longer needs to have flexible and granular security policy at device levels. First because, as we said, the type of application and services running on mobile devices will not require such policies; second, the analysis and enforcement of highly flexible policies would greatly affect the performance requirement, again due to the limitation of resources. Flexibility of security policy must addressed though at infrastructure level, where the required complexity can be born by the available computing resources, in order to manage access control for a large set of services and users.

Ultimately, the requirements for mobile applications are considerably different from the requirements imposed by fixed distributed applications and, as Mascolo et al. (2002) point out, the approach adopted in some cases so far, is to rely on the application to handle all the services and deal with the non-functional requirements as the security ones. However, this approach is a no-middleware solution, as it completely relies on application designers for the solution of the non-functional requirement middleware should provide.

In some other cases, such as MICOsec, object oriented middleware has been adapted to mobile computing to allow interoperation with traditional fixed networks. However, there are recognized problems associated to the physical resources required by the middleware and the assumptions of fixed connectivity, which no longer apply; synchronous connectivity cannot be given for granted in most of the mobile computing scenarios.

The current middleware scenario is very rich, characterized by a strong heterogeneity and lack of standardization. In the chapters that follow we will address the security requirements imposed by current and future service-based application scenarios analyzing current technologies to see how they have addressed security. Based on such analysis we will then evaluate each technology and identify the challenges that must be addressed to achieve secure interoperability.

2 Related Research

Having introduced the main concepts and definitions with regards to security and middleware for ubiquitous computing, in this chapter we review existing work with regards to Service Oriented Architectures and analyse the way security has been addressed. Our survey has been structured into three parts addressing standard protocols, integrated architectures and ongoing research, respectively. Standard protocols cover both the middleware and the transport layer. 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 concentrate on the security aspects and comment on their effectiveness and design choices. Some other surveys already exist in literature, which do not address though the security aspects (Rakotonirainy & Groves, 2002) (Lee & Helal, 2002).

2.1 Standard Protocols

2.1.1 UpnP

UPnP (Microsoft Corporation, 2000) 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. Several UPnP products are in the market today, but so far they have been limited to small office or home network scenarios only, not large enterprise wide networks.

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. Event messages are expressed in XML and formatted using the UPnP General Event Notification Architecture (GENA). 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 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 relies on TCP/IP to provide network connectivity between devices and on HTTP for its discovery and delivery protocols. UPnP uses the Simple Service Discovery Protocol (SSDP) for resource discovery and advertisement. SSDP uses HTTP over Multicast UDP (HTTPMU) and Unicast UDP (HTTPU) for its operations. When joining the network, a device that offer services sends an advertisement multicast message (using HTTPMU), called *ssdp:alive*, to advertise its services to control points. Similarly, when a device with control point functionalities is added to the network, it can send a search multicast message, called *ssdp:discover*, to search for devices/services of interest. Any device that hears this multicast message responds with a unicast response message (using HTTPU) to the control point. Both presence announcements (ssdp:alive) and unicast device response messages contain basic information about the device such as an identifier and the type of device, but most importantly, a pointer to the location of an XML file, called *device description document*. The latter contains information on the set of properties and services supported by the device (including embedded devices) and must be retrieved by a control point before it can begin any sort of interaction with the device.

UPnP security architecture, described in (Ellison, 2003b), addresses security from the viewpoint of the SOAP control messages exchanged between control points and devices. Each device has a password and a public key pair. The private key is kept secret on the device while the password and the hash of the public key, also called the device *SecurityID*, are generally made publicly visible to external users (either via a display or having them printed on a label attached to the device). Both the password and the key pair can be generated by the device on power up if the device has some source of randomness, or they can be assigned once and for all by the device manufacturer.

UPnP security architecture 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). A device which includes the DeviceSecurity service is called a *secured device*, and it can be discovered in a non-secure way, using SSDP, or securely

using a new defined protocol or *ceremony*[2] described in (Ellison, 2003a). Also, when new secured devices become available, they are announced as generic devices, and are not described in any detail except in response to requests from authorized Control Points.

The DeviceSecurity service is responsible for implementing access control for itself and for other services on the same device it runs on. Access control is enforced using either an ACL (Access Control List) or authorization certificates. Each ACL entry specifies the SecurityID (or a group of SecurityID) the entry refers to, and what that SecurityID (or group) is allowed to do. Associated to each ACL entry is also a validity period. Because the secured device might not have enough memory space to store an ACL, the SC can empower the various CPs by issuing them certificates that grant permission to the CP. These certificates can then also be cached, memory permitting, by the secured device for efficiency. An authorization certificate differs from an ACL entry in that the issuer needs to be specified as does the target device. Therefore, an authorization certificate is the equivalent of a signed ACL entry. It is important to note that no ACL entry or authorization certificate can overwrite some other entry's permissions. Revocation of grant of rights is accomplished differently with ACL entries and certificates. When a permission is granted by ACL entry, one can revoke that permission by editing or deleting that entry. When a permission is granted by certificate, the permission can be revoked only by having the certificate expire and not be renewed.

While it is the device that enforces access control, it is the SC that defines the access control policy by either editing the ACL or issuing the authorization certificates. A Security Console (SC) can in turn be used to administer other devices (for example an entire household network) but its use is not mandatory the device can do without it. In particular, the ACL can only be edited by the owners of the device, which are listed, with their SecurityID, in a special table called *owner list*. A SC can only take ownership of a device if the device does not have an owner yet, i.e. if the *owner list* is empty. Once owning the device, the SC is permitted to do everything on that device, including granting and revoking ownership (they cannot remove owners

[2] The concept of ceremony is introduced in UPnP to identify a protocol which refers to messages among computers, people and possibly the environment, as opposed to a network protocol which specifically refers to messages between network nodes only.

they have not granted themselves though). Until a device is not owned it will not be able to perform any access control as the ACL does not contain any entry to grant authorization. In order to take ownership of a device, a SC must have the device's public key and password, and sign the invocation request its private key. A timestamp is also used against replay attacks. Other methods of taking ownership, for example via physical contact through cables have not been standardized and left open to manufacturers.

Before a CP (or a SC) can invoke any secured actions on a device (except TakeOwnership and SetSessionKeys), session keys must be established, using an SSL-like handshake. The session keys are set by the control point and sent to the device encrypted with the device's public key. There are two keys for the device and two for the control point; one key is used for encryption and the other one for signing. Once a session key has been established, a sequence number is introduced for replay prevention.

Confidentiality can only be achieved if supported by the service provider. Before invoking a secure action, the control point invokes the GetAlgorithmsAndProtocols action on the target device. If GetAlgorithmsAndProtocols permits a "NULL" encryption algorithm, the CP does not need to encrypt all actions, otherwise it needs to do so. Confidentiality is achieved through a DeviceSecurity action called *DecryptAndExecute*, which allows a caller to encrypt the action invocation and to receive the reply also in encrypted form. The payload of the DecryptAndExecute message, as well as the reply from the device, are encrypted using one of the sessions keys established by the CP. Even if no encryption is required for the requested action, the CP must still digitally sign it in order to prove authorization. SOAP messages are digitally signed using the session key previously generated by the CP.

With regards to security, the UPnP working group also discussed the problem of *stealth*, i.e. hiding the nature of the device, including the services it offers, from all but carefully authorized parties. The initial design was to allow devices to advertise themselves as secured devices, allowing only authorized CPs to obtain the device's description document. This design choice was eventually dropped as it was believed that there were too many other ways for a device's nature to be betrayed..

ACL or authorization certificates may be used in a secured device to assign permissions to specific CPs to use the services on the device. Such permissions

though do not define whether the services running on the secured device are meant to be discoverable by every CP on the network or only by authorized ones, leaving the secured device exposed to enumeration attacks where it is possible to discover not only the services it offers but also those that are secured. If a CP wants to access a service, it tries to do so. The action might reply that the CP is not authorized, in which case the CP learns that it needs to be authorized.

Also, UPnP secure discovery is only meant to assure that a security console associates with the correct device(s) and that each device on the network associates with the correct SC. It doesn't guarantee any confidentiality or authorizations. Besides, UPnP secure discovery is not automatic and involves human intervention. First the user must read the target device's SecurityID (for example reading it from the device's display); then the user's SC discovers the target device in the normal ways using SSDP and obtains its public key by invoking the GetPublicKey action. Finally the SC computes a SecurityID from the so obtained public key and matches with the one the user obtained in the first step. If there is a match, the user can arbitrarily name the device, which from now on will be identified with that name

Even though secured devices are not described in any detail except in response to requests from authorized Control Points, it is still possible for a determined attacker to take an inventory of a network just based on timing and length of messages. The latter was one of the reasons why securing SSDP was considered a low priority by the UPnP working group until significant new research results in anonymity protection were reached.

2.1.2 Jini

Jini (Sun Microsystems, 1998) is an extension of the Java programming language and was introduced by Sun Microsystems in 1998. It is a distributed system that aims to form a federation or a community of devices and resources, with the aim to make a network a dynamic entity that better reflects the dynamic nature of the workgroup by enabling the flexibly addition or deletion of services. Jini relies on the existence of reasonable speed connecting the devices on the network, and also the presence of Java Virtual Machine (JVM) on each device that uses or offers services. Latency is also required to be small. 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.

A further aim was to simplify the task of building, maintaining, and altering a network of devices, software, and users.

The most important concept within the Jini architecture is that of a *service*. A service is an entity that can be used by a person, a program, or another service; it may be a computation, storage, a communication channel to another user, a software filter, a hardware device, or another user. Services in a Jini system communicate with each other by using a service protocol, which is a set of interfaces written in the Java programming language. The base Jini system defines a small number of such protocols which define critical service interactions. A set of Jini-based computing devices is called a *djinn*.

The heart of the Jini system is a trio of protocols called discovery, join, and lookup. Discovery is used by the service provider to look for a lookup service with which to register its services. The lookup service provides a central registry of services available within the djinn. This lookup service is a primary means for programs to find services within the djinn, and is the foundation for providing user interfaces through which users and administrators can discover and interact with services in the djinn. A service provider locates a lookup service by multicasting a request on the local network for any lookup services to identify themselves.

Join occurs when a service has located a lookup service and wishes to join it; a service object for the service is then loaded into the lookup service, along with a set of attributes describing the service, and a lease is issued by the lookup service. If the service provider fails to renew the lease then, when the lease expires, the lookup service removes the entry for it, and the service is no longer available. The service object loaded into the lookup service contains the Java programming language interface for the service including the methods that users and applications will invoke to execute the service, along with any other descriptive attributes.

After a service has been registered with a lookup service it can be found using the lookup protocol. A service can be found by its interface type (written in the Java programming language) and possibly, other attributes that describe it. To interact with the lookup service, the client must first obtain a *service registrar* object via the discovery protocol. The service registrar object, which implements the net.jini.core.lookup.ServiceRegistrar interface, enables the client to interact with the lookup service. To find desired services, clients build a ServiceTemplate, an instance of class net.jini.core.lookup.ServiceTemplate, and pass it to the service registrar

object, whose lookup() method performs the query and returns matching service objects to the client. The client can then select an object and use the requested service by invoking on the service object the methods declared in the service interface.

In situations where no lookup service can be found, a client could use a technique called *peer lookup* instead. In such situations, the client can send out the same identification packet used by a lookup service to request service providers to register. Service providers will then attempt to register with the client as though it were a lookup service. The client can select those services it needs from the registration requests it receives in response and drop or refuse the rest.

Jini Security Architecture is the outcome of Sun Microsystems' Davis project[3], which sets out to provide a new programming models and infrastructure needed for a secure Jini. The Jini security framework though, adds very few changes to existing Jini services, mainly defining security as a deployment-time option to offer some level of protection to security-unaware services.

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. In order to apply such constraints, each proxy in Jini Security, must implement the net.jinni.constraints.RemoteMethodControl interface (in addition to the application specific interfaces). Both the client and the server can then apply their respective constraints by invoking the method setConstraints(MethodConstraints constraints). It is important to note, as the name of the above method suggests, that the granularity of the constraints injected is the method calls. When invoked, the setConstraints method returns a copy of the proxy, with the set of constraints injected into it. Both the server and client can inject constraints on a service proxy: the server, before exporting the proxy, and the client, after retrieving the proxy from the network. Some constraints express requirements, which can be also specified as *preferred* i.e. optional, whereas others might convey restrictions. When making decisions based on a proxy's constraints, all required constraints must be satisfied and both the client and server must understand a proxy's

[3] The Davis Project homepage http://davis.jini.org/

constraints. If either encounters an unknown constraint type, the method invocation fails. Also, conflicting constraints indicate a discord between the client and server's requirements and the method call will fail.

Besides security-related requirements, the invocation constraint mechanism allows to specify additional constraints related to quality of service such as the maximum amount of time to wait for a proxy to establish a network connection to the remote service.

Jini security also introduces a mechanism for a client to ascertain that the downloaded Jini service proxy is genuine, based on a trusted *bootstrap proxy*. In fact, if the code for that Jini service proxy originated from a malicious source, the proxy might not follow the constraints placed on it by the client. The bootstrap proxy is instantiated by the service proxy itself upon request from the client, which then checks to verify that the bootstrap proxy only contains local code, and that is therefore trusted. The client then uses the bootstrap proxy to contact the service provider – whose identity is known – as it is the only one who can possibly verify the trustworthiness of the downloaded service proxy. The service provider sends back a *verifier* object which verifies the downloaded service proxy. Jini provides a default verification algorithm where the verifier object sent to the client contains a canonical service proxy (trusted) against which the local service proxy is compared. The verifier can however adopt any algorithm to verify the service proxy. Ultimately the client relies on the service provider to ascertain the trustworthiness of the downloaded proxy, regardless on how this is done. As an extreme case, it is also possible to configure the system not to trust any downloaded code (Sun Microsystems, 2003).

The client must also make sure that the verifier itself is trusted, and to this aim, Jini introduces a special form of HTTP URL, called HTTPMD, where the MD stands for the message digest of the verifier. SSL is not used by Jini both because it is considered an overkill and because of the trust infrastructure it would require. Once the verifier vouches for the integrity of the downloaded proxy, a Jini service client can decide whether to trust that proxy, and subsequently grant permissions to it according to the trust placed in the service provider. Access to services is based on the notion of principals and access control lists. All Jini services are accessed on behalf of some entity, called *principal*, which generally traces back to a particular user of the system. Services themselves may request access to other services based on the identity of the

object that implements the service. Whether access to a service is allowed depends on the contents of an access control list that is associated with the object.

Jini Security architecture is still in its infancy and little is know about its weaknesses. However, we can make a couple of considerations. First, security in Jini cannot be enforced by the client if the service has not been implemented to support the application of constraints, i.e. if the service object does not implement the RemoteMethodControl interface. The integrity check of the verifier object is then based on the trust of the service provider's identity, but such trust is not really justified. Most important though is that fact that the secure service discovery and delivery requirements have been left out of the new security framework.

2.1.3 Bluetooth

Bluetooth (Bluetooth SIG, 2001) is a technology developed by a consortium of companies to provide wireless interconnection between small mobile devices and their peripherals. The aim of the Bluetooth though was not to be yet another WLAN technology, but rather a wireless replacement for cables carried by mobile travellers, in terms of cost, security, and capabilities. The Bluetooth protocol stack contains a Service Discovery Protocol (SDP) which is used to locate services provided or available to a Bluetooth device. Bluetooth defines service any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software. SDP supports searching by service type, attributes and browsing, without a prior knowledge of the service characteristics. SDP involves communication between an SDP server and an SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request. If the client, or an application associated with the client, decides to use a service, it must open a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes (including associated service access protocols), but it does not provide a mechanism for utilizing those services (such as delivering the service access protocols). There is a maximum of one SDP server per Bluetooth device. (If a Bluetooth device acts only as a client, it needs no SDP server.) A single Bluetooth device may function both as an SDP server

and as an SDP client. If multiple applications on a device provide services, an SDP server may act on behalf of those service providers to handle requests for information about the services that they provide. Similarly, multiple client applications may utilize an SDP client to query servers on behalf of the client applications.

Service discovery in Bluetooth begins with the inquiry procedure, used to discover nearby devices, which are in a discoverable mode. A device can also be non -discoverable mode so that it cannot be found, or in a limited-discovery mode, which means that it can be discoverable only for a limited period of time, during temporary conditions or for a specific event. Even if a Bluetooth device is non discoverable, enumeration tools exist[4] that allow to find out its MAC address, the class of device, the type of the device and its name.

The design of the Bluetooth system has considered the security issue from the very beginning. The Bluetooth security architecture includes a set of cryptographic protocols to achieve authentication, integrity and confidentiality. Bluetooth Generic Acces Profile (Bluetooth SIG, 2003) defines three different modes of security. Each Bluetooth device can operate in one mode only at a particular time. This first mode is a nonsecure mode as no security whatsoever is provided, and is intended for applications for which security is not required. With the second mode, also called *service-level security,* 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. Whether a security procedure is initiated or not depends on the security requirements of the requested channel or service. For this security mode, a security manager, as specified in the Bluetooth architecture, controls access to services and to devices. The centralized security manager maintains polices for access control and interfaces with other protocols and device users. Varying security polices and “trust” levels to restrict access may be defined for applications with different security requirements operating in parallel. Therefore, it is possible to grant access to some services without providing access to other services, based on authorization. The third security mode is instead called *link-level security* as the Bluetooth device initiates security procedures before the physical channel is

4 Redfang v2.5 is an application that finds non-discoverable Bluetooth devices by brute-forcing the last six bytes of the device's Bluetooth address and doing a read_remote_name().

established, regardless of any application layer security that may exist. This mode supports authentication (unidirectional or mutual) and encryption.

All security operations are based on a 128-bit secret key called *link key* which is shared between two or more parties. A link key is exchanged securely between two devices using the *initialization key*. This is derived in both devices from a random number (sent form one device to the other), a Bluetooth hardware address and a PIN code. The PIN may be a fixed number provided with the devices, or it can be selected by the user, and then entered in both devices that are to be matched (if the devices have a user interface). This can be a problem when the two devices are unknown to each other. After the devices have performed the link key exchange, the initialization key shall be discarded. Once a link key has been exchanged, the two devices are said to be *paired* and the link key can be used to perform authentication and confidentiality. Bluetooth authentication is in the form of a challenge-response where the *verifier* (the device that validates the identity of another device) verifies the knowledge of the link key from the claimant (the device being authenticated). The challenge consist in a random number that the verifier sends to the claimant (the claimant address is also used in the process to avoid impersonation) and is designed to be different on every transaction.

It is important to note that service level security (security mode 2) is not part of the Bluetooth specifications and only described as an implementation reference in a white paper (Muller, 1999). Here, devices can be either trusted or untrusted; a trusted device, which has a fixed relationship (paired), is trusted and has unrestricted access to all services. An untrusted device on the other hand has restricted access to services The trust relationship is established during the pairing procedure, following link-level authentication. If the outcome of the authentication procedure is positive the device is marked as trusted, using the *trusted flag*. Trust relationship is maintained through the *device database*, which also stores the device name, its address and the link key. If a device doesn't have an entry in the device database it is treated as unknown and the default security policy is to apply authentication and authorization to all incoming connection, and authentication to all outgoing connection

Access to services is regulated by the information stored in the *service database*. Each service record contained in this database specifies the security requirements for accessing the service based on the values of three record attributes:

- *Authorization Required* – Access is only granted automatically to trusted devices or untrusted devices after an authorization procedure. Authorization always requires authentication to verify that the remote device is the right one.

- *Authentication Required* – Before connecting to the application, the remote device must be authenticated.

- *Encryption Required* – The link must be changed to encrypted mode, before access to the service is possible.

Combined together, the device and service databases contain the necessary information to perform access control in Bluetooth. Details about these databases can be found in (Muller, 1999).

The Security manager is the key component in the whole service-security architecture and has the responsibility of registering services and devices in their respective databases. All applications must register with the security manager before becoming accessible and the information could be stored in non-volatile memory or the services could just register at start-up time. Using the information in the service and device databases, the security manager decides whether the connection requests from client devices can be accepted and what kind of security functions should be applied. The service security architecture can also support link-level security (i.e. mode 3). The security manager can in fact command the link manager to enforce authentication before accepting a link connection

With regards to the service discovery itself, SDP is a simple client-server protocol with only two types of messages, that is a request and a response, and is not inherently a secure discovery protocol. Also, the Bluetooth specifications do not address service registration because a standardized approach to service registration was not required for ensuring interoperability of Bluetooth devices from different manufacturers. Consequently, the mechanics of service registration were left for Bluetooth stack implementations to define. For this reason, we cannot consider service registration secure in Bluetooth, especially if we consider examples where service may be made available by desktop or laptop computers endowed with a Bluetooth dongle. SDP can however take advantage of Bluetooth security architecture

and run on secure links. In order for SDP to achieve a certain level of security both devices (running the SDP client and the SDP server) must be running in Security Mode 3 or 2, which means that before an L2CAP channel is established, both devices have authenticated and negotiated an encryption key (if confidentiality is required).

In the end, Bluetooth specifications address authentication, authorization and confidentiality, but only with regards to devices and not users. The Bluetooth security architecture (through the security manager) allows applications to enforce their own security policies. The link layer, at which Bluetooth specific security controls operate, is transparent to the security controls imposed by the application layers. Thus it is possible to enforce user-based authentication and fine-grained access control within the Bluetooth security framework. Some weaknesses in Bluetooth cryptographic protocols have also been identified (Jakobsson & Wetzel 2001)

2.1.4 Salutation

The Salutation Architecture is a framework developed by the Salutation Consortium (1999) to solve the problems of service discovery and utilization among a broad set of equipment. The architecture consists of salutation managers (SLM) and Transport managers. The Salutation managers (SLM) are service brokers, isolated by transport managers from the details of specific network transport protocols. SLMs do the job of service brokering and services are required to register their capabilities with them and clients that need service query the SLMs. SLMs sit on the transport managers which are responsible for providing a reliable communication channel irrespective of the underlying network transports. The transport-independent interface available to server and client applications (SLM-API) includes service registration, discovery and access functions. Each SLM also discovers other SLMs and their registered services. So when a client requests a service, all the managers coordinate to perform the search. Salutation lite, also developed by the Salutation Consortium (1999b) is a lightweight flavor of Salutation targeted at mobile devices. It lends itself well to low bandwidth network such as IR and Bluetooth.

Network Entities in a Salutation network may be subdivided by meaningful functionality called Functional Units. The functionalities offered by the Functional Units are defined through sets of attributes grouped into Functional Unit Description Records. Each Attribute describes an aspect of the capability of the Functional Unit. A Functional Unit may be an entire Client or a part of a Client. Additionally, a

Functional Unit may be an entire Service or part of a Service. For example, the [Print] Functional Unit may make up an entire Print Service or be grouped with a [Scan] Functional Unit and [Fax Data Send] Functional Unit to form a Fax Service. Commands, responses, and data may be exchanged between a Client and a Functional Unit, a Functional Unit and a Service, or a Functional Unit and another Functional Unit. The collection of Functional Units within a Network Entity defines the Services available. Each service is described by a Service Description Record, which is a set of zero or more Functional Unit Description Records.

Each SLM contains a Registry to hold at least information about Services that reside in the local Salutation Equipment and optionally that are registered in other Salutation Managers. This way the registry maintains a 'directory' of Salutation Equipment that is important to the local environment. The limit on Registry implementation is the size of the storage reserved for the Registry function. Services are registered providing the SLM with the Functional Unit Description Records that describe the capabilities of the functional unit. Service registry is therefore distributed and service discovery is achieved through the cooperation of SLMs

In order to discover a service, a client or functional unit uses the Salutation capability exchange protocol to determine the attributes, or capabilities, of the service's Functional Unit(s). The Service may reside in the same Equipment as the locating Client/Functional Unit, or it may be located in other Equipment.

Before a client can use a service it has found, the Salutation Manager must establish a virtual data pipe between a Client and a Service, called service session. The latter is operated in one of 3 different modes: native mode, emulated mode, and salutation mode. In the native mode, messages are exchanged through a native protocol and Salutation Manager is never involved in message exchange. In the emulated mode, the Salutation Manager Protocol is used to carry messages between client and service but the Salutation Manager doesn't inspect the contents. In the salutation mode, Salutation Managers not only carry messages, but also define the message formats to be used in the session.

Salutation V2.0 Architecture (Salutation Consortium, 1999) addresses security from the point of view of users since some services may only be available to selected users. Specifically, the identification and authentication requirements are addressed, defining two parameters, called *credential* and *verifier*. The former is used to identify a particular user; the latter is used by the target Functional Unit to authenticate the

credential. Using these parameters it is possible to specify how user authentication is to be performed. The only two Authentication Flavors defined in the current specifications are NULL and UserID-Password. The former corresponds to no authentication while the latter is a simple UserID-Password mechanism with only a special user, the administrator, who is allowed to register/unregister UserIDs. Registration of Administrator's UserID-Password is out of the Salutation V2.0 scope. Implementation may make appropriate design for the function.

When a Client requests a service from a Functional Unit, it specifies the Credential and Verifier parameters. These parameters are passed to the target Functional Unit through the Salutation Manager(s) and used to authenticate the user associated with the Client. This way, authentication is performed solely on information provided by the client. A certain level of trust in the Client is therefore implied by the target Functional Unit. Nowhere in the specifications is this trust justified. Neither is described how authentication is to be performed other than using the credential and verifier parameters. Access control for the services is implicitly left to the target Functional Unit.

With regards to the registration/deregistration procedure, this in not protected as any entity is able to register and deregister a service. However, when a Functional Unit Description Record is registered with the Salutation Manager two security related attributes are also added, which are the authentication flavour and the user ID. The authentication flavour specifies the types of authentication supported by the Functional Unit and they can be No Authentication, Optional or Mandatory. In the first case no authentication is required to access the services; in the second case the client can choose to authenticate to the remote Functional Unit if it chooses to do so and in turn the Functional Unit can choose to restrict available services if the client does not to authenticate. Finally, when mandatory authentication is specified, the client must authenticate. Whenever a client uses authentication it does so by specifying the User ID and Password of the user associated with the Client.

Service discovery is also not secure because with the capability exchange protocol, the client only specifies the service description record it is looking for and no authentication nor authorization is performed. Any client can discover any service. Security only comes into play at delivery time when a Client tries to access a service. When a service is found, a Client will know whether authentication is required and what flavour and if so it will provide the required user ID and password.

Salutation also addresses the availability requirement. The Functional Unit can ask the local Salutation Manager to periodically check the availability of a specific Functional Unit, registered at either the local Salutation Manager or a remote Salutation Manager. When the Salutation Manager finds that the specified Functional Unit is no longer available, it will sent out a notification message. Unfortunately, the communication is not authenticated and therefore the availability requirement cannot really be considered reliable.

2.1.5 SLP

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 as an RFC 2165 (Veizades et al., 1997) and later developed into SLP v2, (Guttman, Perkins & Veizades, 1999). 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.

The SLP protocol is implemented by three agents called *User Agent* (UA), *Service Agent* (SA) and *Directory Agent* (DA). User Agents search for a service on behalf of the user or application; Service Agents provide the location and description of a service; Directory Agents work as a central repository for SLP information.

Service are registered and deregistered with the DA by the SA. When registering the service the SA specifies the location of the service through a URL (Uniform Resource Locator) and a set of attributes that describe the service. Once registered, a service can be found by a UA running on the user device by sending a service request to the DA. Optionally, the user can specify some attributes that the service should have (e.g. color printer). If the DA finds a service in its repository that matches the request, it returns a reply message that informs the user about the URLs of the found services.

The address of DAs to which send service requests and service registrations messages (and deregistrations) can be discovered actively or passively. In the active mode, UA and SA send a DA discovery message to an SLP multicast address. All DAs in the network listen to this address and reply with a unicast message to the requesting agent. DAs can also advertise themselves by sending unsolicited multicast messages to announce their presence. This discovery method is called passive DA

discovery. Finally, it is also possible to obtain the address of a DA through static discovery via DHCP (Dynamic Host Configuration Protocol).

The use of Directory Agents is not mandatory in all SLP implementations and only recommended for large networks with numerous services. When a DA is not present, SAs can advertise themselves to the multicast group and respond with unicast messages to service requests.

Like most of the other service discovery protocols, SLP is administratively scoped. This means that the protocol locates resources (devices and services) available in a network within an administratively defined network domain. Users belonging to a certain scope may only discover services that are offered in this scope; other services are hidden by the DA.

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.

SAs can include a digital signature with all their registration messages thus making registration and deregistration secure. Unfortunately SLP does not prevent replay attacks and service registration (and deregistration) messages can be replayed with potential threats of DoS attacks or other. Vettorello et al. (2001) propose a solution to this problem based on the inspection of the timestamps used in SLP messages. Signatures can also be included in unicast service reply message addressed to User Agents, which can in turn verify it and trust that the service location (URL) is trustworthy. 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.

2.1.6 JXTA

The open-source Project JXTA is the industry leading peer-to-peer (P2P) platform originally conceived by Sun Microsystems Inc. JXTA technology (Gong, 2001) is a set of simple, open peer-to-peer protocols that enable any device on the network to

communicate, collaborate, and share resources. Each protocol is defined by one or more messages exchanged among participants of the protocol. Each message has a pre-defined format, and may include various data fields. The following protocols are currently defined:

- Peer Discovery Protocol
- Peer Resolver Protocol
- Peer Information Protocol
- Rendezvous Protocol
- Pipe Binding Protocol
- Endpoint Routing Protocol

The Peer Discovery Protocol is the default discovery protocol for all peer groups and enables a peer to find advertisements on other peers, and can be used to find any of the peer or peer group. A JXTA advertisement is an XML-structured document that names, describes, and publishes the existence of a resource, such as a peer, a peer group, or a service. A peer in turn is defined as any entity that can speak the protocols required of a peer. As such, a peer can manifest in the form of a processor, a process, a machine, or a user. Importantly, a peer does not need to understand all the six protocols given above. Each Peer within a Peer Group has its own identity. A device participating within multiple Peer Groups will maintain separate identities, one for each peer instance. Peer discovery can be done through the peer discovery protocol specifying a name for either the peer to be located or the group to which peers belong. A name does not need to be specified, in which case all advertisements are returned.

Delivery in JXTA refers to the delivery of protocol messages and not to any service that may provided using JXTA as the overlay network. For this reason we do not find any service delivery protocol as such in JXTA, but rather pipe (JXTA asynchronous and unidirectional virtual channel) binding and routing protocols for the creation of channels between peers onto which messages can be exchanged.

In the JXTA virtual network, any peer can interact with other peers, regardless of location, type of device, or operating environment — even when some peers and resources are located behind firewalls or are on different network transports. Thus, access to the resources of the network is not limited by platform incompatibilities or the constraints of a hierarchical client-server architecture.

A network of JXTA devices is secured by implementing authentication, authorization and auditing services within each Peer Group and by restricting the types of information which may be published within each Peer Group. Associated to each Peer is a set of Credentials, used to prove a Peer's membership within the Peer Group, to uniquely identify the Peer, and may be used to grant or restrict access to services or content available within the Peer Group. Every peer is then protected by a peer ID and password, which are used to decrypt the private key to a user's personal security environment. This is quite a standard protection mechanism to defend against an attacker having physical access to the machine running the JXTA Peer). The choice of authentication and authorization schemes and Credential formats used to secure a Peer Group are left up to the Peer Group implementer.

Currently, security support in JXTA is provided through TLS, which allows to set up secure communication channels between peers. Each peer acts as its own Certification Authority (CA) and generates its own root certificate, whose related private key is used to sign service certificates that the peer issues for each service that it supports. The root certificate is distributed along with the peer's advertisement. Therefore, every other peer can always verify that an advertisement is indeed from the peer who claims to have issued it. Public Key Infrastructures that support this security model can be implemented in JXTA either with or without the use of commercial CAs. Alternatively, participation within Peer Groups may be also secured through an ad-hoc distributed trust model such as Poblano (Altman, 2003).

The latter is a reputation trust model where the trust associated with each peer is based on the three parameters which are the confidence in the Peer, confidence in the Codat (this is JXTA's term to define at the same time code and data), and risk, intended as the reliability of the peer from a performance viewpoint (accessibility, throughput, integrity of data) - does this peer distribute virus for example).Trust values in Poblano scale from Complete Distrust [-1] to Complete Trust [4]. Initial Trust values are specified by action or policy. Other reputation-based models for JXTA include the work by Damiani et al. (2002).

From a security stand, P2P systems such as JXTA have certain intrinsic advantages. Confidentiality is better achieved since messages are sent between two peers without going through a centralized server. Also, as content can be replicated non-deterministically anywhere on a P2P network, there is no central point of knowledge which can be the target of denial-of-service attacks. Finally, when

searching for things, a peer will always ask another peer in its local domain first. As a result, bad behaviour is limited to neighbours or direct contacts as opposed to using a central server which, if contaminated, will affect all its client.

2.2 Integrated Architectures

2.2.1 Ninja's Secure Service Discovery Service

The Secure Service Discovery Service (SSDS) was developed as part of the Ninja project (Czerwinski et al, 1999) (Hodes et al., 2002). SDS is a scalable, fault-tolerant, and secure information repository providing clients with directory-style access to all available services. The SDS can store many types of information, including descriptions of *unpinned services*, which are available for execution, and *pinned services*, which instead run at specific hosts. The SDS supports both push-based and pull-based access; the former allows passive discovery, while the latter permits the use of a query model. Service descriptions and queries are specified in XML.

Privacy and authentication is achieved through encryption using a hybrid symmetric-asymmetric 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 trusted services are discovered. Access control to services is achieved through 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.

SDS organizes the service area into domains, defined as a list of CIDR address ranges that can change with time, and each domain is managed by a particular SDS server. If a particular SDS server is overloaded, a new SDS server can be spawned on a nearby machine (if available), assigned to be a child of the overloaded server, and allocated a portion of the network. Each server is responsible for sending

authenticated messages containing a list of the domains that it is responsible for on a well-known SDS multicast channel. These domain advertisements contain, among other things, the multicast address that service providers will use for sending service announcements, and the desired service announcement rate (i.e. how often services are required to announce themselves). The messages are sent periodically using an announce/listen communication model. Once an SDS server has established its own domain, it begins caching the service descriptions that are advertised in the domain.

Services are announced using the *secure one-way service broadcast protocol* that provides service description privacy and authentication. With this protocol the service description is encrypted using a symmetric key, which is in turn encrypted with the SDS server's public key. Once decrypted with its private key, the server can then cache the symmetric key for future service advertisements. After decrypting the service description the SDS server adds it to its database and updates the description's timestamp. Services which do not send new announcements within a defined time limit are flushed.

Service discovery is authenticated using the Authenticated RMI protocol (ARMI) (Welsh, 1999) developed within the framework of the Ninja project itself. ARMI allows authentication using public key certificates and encryption through an agreed symmetric key. Once contacted the server, the client will submit a query in the form of an XML template along with the client's capabilities (access rights). Depending upon the type of query, the SDS server returns either the best match or a list of possible matches. In those cases where the local server fails to find a match, it forwards the query to other SDS servers based on its wide-area query routing tables.

Two main security components are present in the SDS architecture and they are the *Certificate Authority (CA)* and the *Capability Manager (CM)*. The former issues public-key certificate for the various principles and also encryption-key certificates, which bind a short-lived encryption key to a principal. Such encryption key is signed using the principal's public key. The Capability Manager instead, is used to support the use of capabilities for access control. Each service contacts the CM, and after authentication, specifies an access control with the name of principals allowed to access the service's description. The CM then generates the appropriate capabilities and saves them for later distribution to the clients. Capability distribution itself can be done without authentication because capabilities, like certificates, are securely associated with a single principal, and only the client possessing the appropriate

private key can use them. To aid in revocation, capabilities have embedded expiration times.

2.2.2 Project Centaurus

The Centaurus project (Kagal, L. et al., 2002) 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. Centaurus framework has then continued under the name of Centaurus2 (Undercoffer, J. et al., 2003) and Vigil/Secure Centaurus. Project Centaurus is responsible for maintaining a list of available services, and executing them on behalf of any user requesting them, which minimizes resource consumption on a user's device by avoiding the need to have the services installed on each device that wishes to use them, which is advantageous for most resource-poor mobile Clients.

Centaurus does not distinguish between users and services and they are both represented as *Clients* within the framework. This equality of users and services allows a Client to access services while at the same time providing some of its own services to others. Services are defined as objects that offer certain functionality to clients. Access to services is mediated by two architectural components called *Communications Manager* and *Service Manager*, using a language based on XML, called CCLM (Centaurus Capability Markup Language) as data exchange format. The Communication Manager provides a communication gateway between a Client device and a Service Manager. Its sole purpose is to abstract and translate communications protocols. The *Service Manager* brokers requests between registered user-Clients and service-Clients. Centaurus services are Java based but non-Java services can also be supported as long as they can use CCML and either communicate via sockets with the Service Manager or with the Communication Manager through some native protocol.
Clients talk to the Communication Manager and register themselves with a Service Manager. On registration, the Client receives a list of current services (updated dynamically); it can then select a service from the list and access the service functionalities. Centaurus does not envisage service delivery in the strict sense of term as no real service is actually delivered to the client. Once services are found they are instead executed on behalf of the client that sends execution commands and receive service updates via the Service Manager. The latter validates these commands (i.e. it performs access control) and forwards them to the intended service. Command

execution causes the service to change its state, and so it sends a status update to the Client via the Service Manager. In this sense we can say that service delivery refers to the delivery of service commands and service updates.

Centaurus security is based on public key encryption, using a simplified PKI. 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 (Service Managers and Clients) in the system 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. Instead of maintaining a CRL (Certificate Revocation List), the Capability Manager(s) has an entry for each valid user on the system. The absence of an entry blocks that entity from any and all accesses to the system. The absence of CRLs is the reason for Centaurus' simplified PKI.

When a client (either user or service) registers with a Service Manager 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 from the Service Manager upon registration depends on the client's groups membership. Generally, a user-Client will send an empty access list indicating that no other Client may be granted access to it, whereas a service-Client will include a list of groups where membership in one of the listed groups is required for access to that Client's service. As part of the registration, the Service Manager queries the Capability Manager to find out the group memberships of the registering Client. This membership information, along with the access list and the client's certificate is then stored in the Service Manager database of Client profiles.

After the registration, the service relies upon the Service Manager to which it is registered to enforce security, access control, and to broker requests for the service. Each Service Manager determines the entity's access rights based upon group membership (obtained from the Capability Manager) and forwards requests to Clients based on those rights. The Client, however, has ultimate jurisdiction on responding to those commands and may, for some reason, choose to ignore the service request.

Service de-registration is secured via public key encryption and so is service discovery, achieving both authentication and integrity. Public key encryption is only used in Centaurus to meet the authentication and access control requirements. No

consideration is given to confidentiality and all protocol messages are sent unencrypted, albeit digitally signed. Therefore no confidentiality is achieved in service discovery nor in service delivery

2.2.3 Proxy-based Security Protocols

This architecture was designed and implemented by Burnside et al. (2002a) and is based on the use of proxies. 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 architecture envisages three component types: devices, proxies and servers. A device refers to any type of shared network resource, either hardware or software. A resource may be location-aware and has an associated trusted software proxy. A proxy runs on a network-visible computer and its primary function is to execute commands on behalf of the resources it represents, and to allow communication between devices. The servers provide naming and discovery facilities to the various devices. Finally a one-to-one correspondence is assumed between devices and proxies. Each device communicates with its own proxy over the appropriate protocol for that particular device. Thus the device-side portion of the proxy must be customized for each particular device.

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 who can manually change it at a later stage if desired. Anti-virus scanning software can also be installed on the proxies to scan code before it is downloaded onto the device.

In this system, each user wears a special badge which identifies the user and is location aware (it knows the wearer's location within a building). User identity and location information is securely transmitted to the user's software proxy using the device-to-proxy protocol.

The resource discovery used in this architecture is similar to the one used in Jini. When a device comes online, it instructs its proxy to repeatedly broadcast a

request for a server to the local subnetwork. Such requests however, are neither authenticated nor encrypted. When a server receives one of these requests, it issues a lease to the proxy by adding the device's name/IP-address:Port-number pair to its directory. The lease request must be continuously broadcast lest the expiration of the lease. The proxy and the device communicate through a secure channel that encrypts and authenticates all the messages. HMAC-MD5 is used for authentication and message integrity; a modified version of RC5 is used for encryption. The proxy and the device share a 128-bit secret key.

Resource discovery in this architecture is based on the Intentional Naming System (INS) (Adjie-Winoto et al., 1999). The Intentional Naming System (INS) is a resource discovery and service location system intended for dynamic networks. INS provides users with a layer of abstraction so that applications do not need to know the availability or exact name of the resource (as for the DNS) they are looking for but only their characteristics.

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). SPKI/SDSI provides fine-grained access control using a local namespace architecture and a simple, flexible, trust policy model. Interaction between devices (i.e. services) is mediated by the proxies which forward the requests and process the replies. Resources on a server device are either public or protected by SPKI/SDSI access control lists. An SPKI/SDSI ACL consists of a list of entries. Each entry has a subject (a key or group) and a tag which specifies the set of operations that that key or group is allowed to perform. To gain access to a resource protected by an ACL, a requester must include, in his request, a chain of certificates demonstrating membership of a group in an entry on the ACL. If a requested resource is protected by an ACL, the principal's request must be accompanied by a "proof of authenticity" that shows that it is authentic, and a "proof of authorization" that shows the principal is authorized to perform the particular request on the particular resource. The proof of authenticity is a signed request, and the proof of authorization is a chain of certificates. The principal that signed the request must be the same principal that the chain of certificates authorizes.

Requests to services are initially sent unauthenticated and unauthorized. If the requested resource is protected, the server responds with the ACL protecting the resource, and a *Tag* formed from the client's request. The client then uses the received

ACL and Tag to produce a certificate chain, using the SPKI/SDSI certificate chain discovery algorithm. It then sends the certificate chain and the signed request in a second request to the server proxy. The signed request provides proof of authenticity, and the certificate chain provides proof of authorization. Requests are also time-stamped by the client to avoid replay attacks. The details of how access control is achieved using SPKI/SDSI are described by Burnside in (Burnside et al, 2002b).

The architecture proposed by Burnside et al. (2002a) has then been improved by Raman et al. (2003) who have adapted INS to store ACLs in the INS name-trees so that access decisions can be made at search time (i.e. discovery). In the new architecture the proxy presents the user's authorization rule (i.e. the chain of certificates) with the user's query. When INS has completed its search and returned the addresses of the resources found, the proxy can use a secure authentication and authorization protocol to contact the resource. While the security requirement of this new architecture is an obvious advantage, making the access decision at discovery time also improves scalability and efficiency especially when the number of available resources is high. Experimental results by Raman et al. show a reduced time for resource retrieval, which is paid though in terms of additional storage needed for the ACLs.

2.3 Ongoing Research

2.3.1 Splendor

Zhu et al. (2003) propose 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 security protocols in Splendor enable all parties to mutually authenticate each other, no matter if users have accounts in the network infrastructure systems or not. 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 may ask proxies to handle service registration, and key management for them. Alternatively, mobile services may do all the work for

themselves. Proxies also manage mobile devices stability problems (they may have poor wireless connection, or people may just turn them off) and cache soft state information of mobile services. At the same time services represented by proxies are stored as hard state in directories. This means that proxies explicitly register and unregister services with directories.

Splendor is inspired by the HP's Cooltown project, which introduces the idea of tagging things, people and places and associate them with their "web presence". Similarly, the Splendor service discovery protocol uses tagging to allow location-aware discovery, so that based on where they are, users can discover the relevant services they are interested in.

Splendor's security is based on the classic hybrid model of symmetric and asymmetric encryption, where public-key encryption is used to exchange computationally more convenient symmetric session keys. In particular, Splendor assumes that existence of a PKI with a CA that signs public key certificates for all four components of the architecture (i.e. clients, directories, proxies, and services).

Directories are used to store service description and for look up operations. Service registration and look up operations are both secured through encryption. Each directory periodically sends an advertising message to a well-known multicast address specifying the directory's certificate, the unicast address to be used for further communication and a timestamp (against replay attacks). When mobile clients or services move to a new place, they may solicit directories for announcements. When a service provider has a service to be registered it does so via the proxy. First, the proxy and the directory, using their respective public-key certificates, authenticate each other and exchange two session keys; one key is set by the proxy while the other is set by the directory. During the service discovery phase the client and the directory authenticate each other using their respective public key certificates. In response to a service request by the client, a directory sends a list of matching services along with the certificates of the proxies representing the service. The client chooses a service and verifies the corresponding proxy's certificate. Before the service can be accessed, the client and the proxy mutually authenticate and exchange a session key, generated by the proxy. For non-repudiation purpose, data is hashed and the hash is signed before encryption. Access control to services is performed by the proxy. On the downside, the client must perform all proxies' certificate validation, which is known to be a computationally intense activity, as discussed in chapter one. Also, the authors

of Splendor do not really consider access control, neither in terms of policies (i.e. who is in charge of setting them and how it is done), nor in terms of how it is achieved. For the latter, the protocol specifications seem to assume the use of simple access control lists placed on the Proxies. Zhu et al. (2004) also propose a protocol for the secure ad hoc discovery of services in public environments.

3 Security Requirements for SOAs in Nomadic Computing

The main reason why security is so difficult to address in pervasive computing is the lack of a formal definition of the security requirements that need to be met. The definition of such requirements is affected by both the intrinsic functionalities of a SOA and the lack of clear use cases from which one can possibly infer the security concerns. Ultimately but foremost, we identify a problem with regards to the correct definition of service, which is security unaware in the great majority of examples studied, making the SOA itself not security-aware from the outset.

In this chapter we identify a number of security requirements for SOAs in ubiquitous computing and evaluate the way these requirements are met by current architectures and technologies. Among these, we pay special attention to the privacy requirement which is critical for the success and adoption of ubiquitous computing. In particular we review the existing literature in the area of information privacy for ubiquitous computing.

3.1 Security in Service Discovery and Delivery Protocols

Traditionally, security is identified with a number of basic requirements such as authentication, confidentiality and integrity. With regards to service discovery and delivery protocols, the above requirements are two low-level to reflect the threats and the need for security in service-oriented architectures. Saying that we need authentication, confidentiality and integrity does not say anything about what we are really trying to secure and which threats we aim to mitigate. As we described earlier, the three basic mechanisms underlying a SOA are service registration, service discovery and service delivery. Associated to each one of such mechanisms there are a number of security issues, which need to be addressed and which can be better visualized if referring to the sample scenarios presented in chapter one

3.1.1 Service Registration and 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. If a new service is to be registered 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 being registered maintains its integrity (while it is being registered) and that only the intended repository is communicated the registered service (confidentiality). No other entity should know about the registered service if not through the discovery phase, which must be secured in its own way. The same applies to when a service is deregistered

With regards to service registration and 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 attack (DoS). 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. Similarly, a service may be made available outside allowed times.

Security of the registry must also be addressed. In fact, failing to address the security of the registry may invalidate the security addressed for service registration/deregistration.

The requirement for secure service (de)registration is less stringent where the service registry is embedded into the device, as in Bluetooth, because little or no communication takes place between the service provider and the registry itself. Conversely, the requirement is much more important when services are registered on external repositories.

3.1.2 Service Discovery

Service discovery is the primary service of ubiquitous computing onto which all other services are layered. If the discovery phase is compromised, then the security of all other relying services is compromised too. Secure discovery can be related to both devices and services in the sense that we can have secure discovery of devices but not of services or vice-versa. Here though, we will disregard the security of devices and concentrate on the secure discovery of services. We will assume that security of the device is already addressed by the security of the service registry that is hosted by the device. Secure service discovery can be divided into the following areas:

Discovery Authentication –. This is intended as mutual authentication between the entity performing the discovery and the registry responding to the search query. The former can thus verify the identity of the service registry while this can authenticate the querying entity to better control the services that it can discovery.

Authorized Discovery – It is important that only authorized entities are allowed to discover existing services. Only authorized entities must be allowed to use the discovery protocol/and or the registry where the services are registered. This is required to prevent malicious users from building an inventory of available services (e.g. the number of policemen in the area) and devices. If authentication is performed, this could be used to enforce authorization or a default authorization policy may be defined for non-authenticated entities.

Controlled Discovery – This is a special case of authorized discovery where the discovery is controlled so that not every service is discoverable by every entity. Entities 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 way controlled discovery is achieved may depend on the way access control is performed. The limitation achieved with this requirement 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 allowed to discover a service but not the device and/or service provider that offers it.

Confidential Discovery – Service discovery requests should be confidential to the device/registry which services the requests. For instance, if services are held in a registry which devices can query to retrieve services, only the registry should know about that particular query and not other devices too, unless specified. For example, any discovery protocol which uses unencrypted broadcast messaging suffers from lack of confidentiality as attackers may in principle eavesdrop the communication and know which services are sought; 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. Confidential

discovery is easier to achieve with discovery protocols that use a central registry to which discovery requests are sent. This way no other entity on the network, apart from the registry, will know when or what service is being requested.

Genuine Discovery – 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 just simply corrupted. It is important to note that the trustworthiness we refer to here is related to the service description and not the service itself.

It is also important to distinguish between the two possible ways a service discovery protocol may operate. In the first case, a service discovery query returns just a list of service descriptions but the actual service is not delivered until the client has made a choice. Alternatively, second case, when performing a service discovery, the client is returned a list of matching services from which it chooses one or more. In our study, where the discovery protocol behaves as in the latter way (e.g. Jini), we will assume the genuine service requirement to be addressed by the secure delivery requirement. In other words, secure discovery will mean secure delivery of the discovered service, where secure delivery entails integrity and authenticity to ensure that the service has not undergone any changes and that it is authentic.

Anonymous Discovery – 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 devices which is issuing the service discovery request. The latter refers instead to the anonymity of the entity requesting the service.

3.1.3 Service Delivery

After a service has been found, it can be delivered securely provided a number of requirements are met. Delivery of a service may be a very delicate issue depending on the type of service being delivered. Again, here it helps visualizing real people (e.g. a doctor) as possible service providers, rather than just devices offering standard services. As for the discovery phase, also service delivery must be secured, by addressing the requirements listed below.

Delivery Authentication – the recipient of the service and the provider need to mutually authenticate. Through authentication the service provider can make sure that the service is delivered to the intended recipient. This in turn can make sure that the service comes from a known source.

Authorized Delivery – As for the discovery phase, one may want to control the delivery of services only to authorized entities. In some instances such requirement may overlap with the authorized and controlled discovery requirements. In fact two possible approaches are possible here. First, one may assume that if an entity is allowed to discover a service, that entity is also allowed to use that service. Alternatively, an entity may be allowed to discover a service but it may be limited in how to use the service, depending on its set of credentials.

Delivery Confidentiality – This requirement guarantees that the service is only delivered to the intended recipient. This is an important requirement especially if the service is a commercial one.

Delivery Integrity – Even if the communicating parties have authenticated each other and the service is delivered in a confidential way, 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.

Anonymous Delivery – This requirement ensures the anonymity of the entity to which the service is delivered. As for the discovery phase anonymity is also referred to location anonymity and identity anonymity.

3.1.4 Application Security

After a service has been discovered it is then delivered. In many instances the service delivered is supposed to run on the end device and/or interact with the device's operating system. The service may for example be a device driver and device configuration for the use of services available from nearby peripherals or even a small application required to interact with the service provider. Much of the security issues related to application and more generally to services running on the end device is catered for by other security requirements, which make sure that the service is genuine and has not been tampered with. This prevents us from downloading a rogue drive which, in addition to being a driver, also installs malicious software that could compromise our privacy.

When applications run on the target device it is important to grant them rights based on their credentials, rather than, based on the user (of the device) privileges. However, limiting the privileges of the running application so that, for instance, a downloaded printer driver cannot also read the user's address book and send it to a remote device, depends on the way a service is defined and the device operating environment.

3.1.5 Availability

One important function of a service oriented architectures is to quickly react to faults. A service for example may not be available anymore due to server's failure or user mobility. Also, the user may simply switch off the device on which the service is running in order to save the battery life. 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, typically using all the available resources. In the context of security of a SOA then, we want to ensure that a malicious user cannot prevent legitimate users from having reasonable access to existing services. It is in this respect that the term Denial of Service (DoS) is used, which can be defined as the prevention of authorized access to resources or delay of time critical operations (ISO, 1989). The issue of availability is very much

felt in the mobile computing domain where devices may adopt a power conservation strategy, trying to sleep as often as possible to conserve the already limited energy of their batteries. Keeping a device awake until its energy runs out can be an effective DoS attack. Once the battery is flat, the attacker can walk away, leaving the device disabled and unable to provide any service. Authentication may help preventing such attacks, but not always. In certain traditional cases, such as for a Web server, applications cannot refuse to serve unknowns and identifying repeat offenders doesn't help either, both because source information can easily be faked, and because a villain might subvert multiple "innocent" principals into cooperating in the attack (the so-called distributed denial-of-service attack or DDOS). Alternative approaches may be based on the resolution of cryptographic puzzles, which would be easy to solve only to authorized entities.

3.2 Protecting Access to Services

3.2.1 Access Control

In a SOA, service provision must be protected and requests by a client entity should only be honoured if the client possesses sufficient access rights for that requests. The process of verifying a client entity's rights to access a specific service is referred to as *access control*. The process of granting access rights is referred to as *authorization*. With regards to SOAs for nomadic computing though, the issue of access control is more complicated than in traditional distributed computing. As we discussed earlier, in a SOA for ubiquitous computing, not only we need to secure access to services, which is what have referred to as delivery, but also service registration/deregistration and service discovery. These also needs to be protected through access control.

Another issue is the granularity level of both the client entity and the service being requested. With regards to the client, in some cases, such as Bluetooth and UPnP, the access control model identifies clients with devices. In some other cases instead, such as in Jini, access control is performed on a human or software subject running on the device; so we could have more client entities per device. In traditional distributed computing, the adopted granularity level is the latter one. With regards to services, the granularity level highly depends on the way the service has been defined. In the context of a SOA, a service can be generally defined as a course-grained and self contained software entity interacting with applications and other services (Brown,

Johnston & Kelly, 2002). In a traditional distributed computing and with regards to SOA for Internet applications, a service can offer a number of functionalities, each of which may need access control. With regards to ubiquitous computing instead, services tend to offer a single functionality and therefore the level of access control needed by each single service is lower. This is especially true for small devices.

In order to better discuss the issues of access control we can introduce a widely accepted model of access control which envisages three distinct components, which are the *subjects*, i.e. the client entity issuing the service request, the *services* and the *reference monitor*. The latter is also known as policy-enforcement-point (PEP) or resource-manager and is in charge of controlling access to the service from unauthorized subjects. When a subject issues a request to a service, the request is mediated by the reference monitor which records which subject may do what, and decides whether a subject is allowed to access the service, i.e. obtain information, have an action performed or control a resource. This monitor is called each time a service is invoked. Consequently it is important that the reference monitor is itself secure and in particular tamperproof. In large distributed systems a policy enforcement point is often associated to a Policy Decision Point or PDP. In this extended architecture the PEP collects information about the subject, the request, the target service, and some context attributes and passes it onto the PDP which in turn makes the access decisions. In small systems PDP and PEP are usually merged into what we referred to earlier as reference monitor or resource manager. However, different security models exist which differentiate on where the PDP and PEP are placed with regards to the application and the middleware (Beznosov, 2002).

With regards to the access control itself, several models have been proposed to address the requirements of distributed applications. Traditional access control models are broadly categorized as discretionary access control (DAC) and mandatory access control (MAC) models. New models such as role-based access control (RBAC) or task based access control (TBAC) models have also been proposed to address the security requirements of a wider range of applications.

DAC models control the access to the information explicitly specifying the authorization for each subject to each object in the system. The access control is at the discretion of the object's owner or anyone else who is authorized to control the information object's access. The most traditional way of implementic DAC is by using an *access control matrix* (Naldrug & Campbell, 2003) where each subject is

respresented by a row and each object is represented by a column. The entries in the matrix contain the subjects access rights to the corresponding objects. There are two main variations with regards to the access control matrix, referred to as *access lists* and *capability lists*, respectively Krager (1978). Access lists store lists of (subject, method) pairs for each object and define what methods subjects can use on a specific object. Each object is therefore associated to its own ACL of the access rights of subjects that want to access the object. Capability lists on the other hand, store lists of (object, method) pairs for each subject and correspond to a list of objects and methods that the subject can access. The (object, method) pair is also referred to as a *capability*, and equates to an entry in the access control matrix. In this model when the subject makes a request on the target object, it includes the required capability. The server where the object resides is not anymore interested in whether it knows the client or not; the capability says enough and not having a capability for a specific object means that the subject has no right for that object. Other variations of the DAC model are based on how subjects grant and revoke access to the objects they own to other subjects (Krager, 1978) and (Osborn et al., 2000)

In MAC, access control decisions are made beyond the control of the individual owner of the object. A central authority determines what information is to be accessible by whom, and the users cannot change the rights. The advantages of MAC models derive basically from their suitability to some kinds of environment (usually military) in which the users and objects can be assigned to a security clearance or a security level.

In large distributed systems, the number or objects and subjects can be extremely high. Access control lists (or capability lists) may become enormous in size and their maintenenace would be difficult and costly. The RBAC framework (Sandhu et al., 1996) solves this problem by introducing the concept of roles. Users are grouped into roles, and they access objects using the access privileges associated to their role. The notion of role is an enterprise or organizational concept, i.e., a role represents a job function in an organization and embodies a specific set of authorizations and responsibilities for the job. System administrators can create roles, grant permissions to those roles, and then assign users to roles on the basis of their specific job responsibilities (Ferraiolo & Kuhn 1995). RBAC greatly reduces the complexity and cost of security administration and simplifies the management of access rights.

3.2.2 Authorization Management

While access control takes care of ensuring that services are only accessed by authorized entities, rights must first be granted and then managed properly through a process also known as *authorization management.* In mobile computing, authorization management is complicated by the fact that each client entity may potentially have access to services that are spread across a number of geographically distributed devices. If one were to use a traditional approach to granting access rights to entities, one would have to modify the access control matrix on each device. This would pose scalability problems with the number of users and devices often not known in advance. Furthermore, each device would need to host an access control matrix capable of holding information on possibly an extremely large number of client entities. A simple and often used approach is based on the use of a central server, where a single account is created. The server is consulted each time a user accesses certain resources or devices. Two better approaches are based on *capabilities* and *attribute certificates*.

A capability, introduced earlier, can be better defined as an unforgeable data structure for a specific resource (or service), specifying exactly the access rights that the holder of the capability has with respect to that resource. A capability can be described as containing two parts (Levy, 1984):

- an identifier or name for an object
- some access rights or privileges to that object

By giving a user capabilities only to the objects we wish he/she to have access to, finer control over the processes/rights that can be performed on the object is obtained. The fact that capabilities can specify what actions can be performed on an object allows greater control over the users access rights to an object, limiting the users access where necessary. Capabilities can be managed through a special Capability Manager, examples of which have been described in chapter two.

A generalization of capabilities that is used in modern distributed systems is the *attribute certificate*. Compared to traditional public key certificates, attribute certificates are used to list attribute-value pairs that apply to a specific entity. In particular, attribute certificates can be used to list access rights that the holder of a

certificate has with respect to the identified objects. Like other certificates, attribute certificates are handed out by special certification authorities, called *attribute certification authorities*.

3.2.3 Trust Management

Trust management addresses the problem that a secure SOA should not just provide authentication and authorization for known entities but also allow them to delegate their rights to other non-registered entities and have a flexible mechanism for this delegation. In other words it should be possible to trust foreign and not directly known entities. Blaze, Feigenbaum & Lacy (1996, p. 1) first addressed the trust management problem defining it as the "*..intellectual framework needed for the study of security policies security credentials, and trust relationships*". Humans use trust every day in their interaction with people and other entities. However, it's commonly agreed that trust is subjective and based on available evidence, it is contextual dependant (i.e. it depends on the situation) and it is also dynamic (i.e. it evolves as new information becomes available). This is particularly true in ubiquitous computing; trust in one environment does not transfer to another environment.

Existing trust models for ubiquitous computing typically represent trust using a security policy which explicitly permits or prohibits actions. These policies are not well suited to dynamic environments, in which participants have only partial trustworthiness, and trust assessments must constantly change. To avoid this, a better approach is based on the use of recommendations. English et al. (2002) for instance, describe a dynamic trust model for ubiquitous computing which would give interacting entities the ability to operate and make autonomous decisions. The model combines recommendations consistently, by formally ordering them according to information content.

Trust in dynamic environments such as that of nomadic computing is an on-going research topic. However, it is worth questioning, as Langheinrich (2003) does, the actual benefits of introducing the concept of trust into computational frameworks, and should maybe leave trust assessment in ubiquitous computing environments up to humans.

3.3 Privacy and Anonymity

Technologies for locating and tracking individuals are becoming increasingly commonplace, and they will become more pervasive and ubiquitous in the future. Such technologies allow the development of location aware applications, required for the fulfilment of the ubiquitous vision. Unfortunately, the location of an individual can be used to infer highly private information. Traditionally, privacy of personal location information has not been a critical issue, albeit an important one, but with the development of location tracking systems capable of following user movement 24 x 7, location privacy has become important. In a nomadic computing scenario, location-based applications track people's movements so they can offer various useful services. Some of these services can be accessed in complete anonymity, such as "alert me when I walk past a shop which sell the shoe model I am looking for". Some other services instead do require some form of identity to work, either real or fictitious (pseudonym), such as a presence service that notifies other people of our location.

Information privacy can be defined as "*..the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others*" (Westin, 1967). Privacy information regards primarily the identity and the location of an individual. Further to that, Beresford and Stajano (2003) define location privacy as "the ability to prevent other parties from learning one's current or past location". However, other more traditional properties of an individual such as interests, behaviour, or communication patterns could lead to the identity and location by inference or statistical analysis.

In the literature there exist several approaches to protect the privacy of a user, most of which try to prevent disclosure of unnecessary information by explicitly or implicitly controlling what information is given to whom, and when. Some solutions as the one proposed by Myles et al. (2003) provide identity anonymity through pseudonyms, either short-term or long-term ones. However, Beresford and Stajano (2003) stress how long-term pseudonyms cannot really offer sufficient protection against location privacy. In fact, certain locations such as an office desk, act as a "home" space for the user and an application could identify users by following the footsteps of a pseudonym to or from such a home area. Instead, they propose a solution based on *mix zones* and *application zones* (Beresford & Stajano, 2004). Mix

zones are unregistered geographical regions where no application can trace user movements; because applications do not receive any location information when users are in a mix zone, the identities are effectively mixed. The degree of anonymity a mix zone offers depends on the size of the anonymity set, which is defined as the "the set of all possible subjects who might cause an action" (Pfitzmann & Köhntopp, 2000). Users might then decide to refuse to provide location updates to an application until the mix zone offers a minimum level of anonymity.

The *pawS* system (Langheinrich, 2002) is a privacy awareness system for ubiquitous computing environments, based on the Platform for Privacy Preferences (P3P) (Cranor, Langheinrich, Marchiori, 2002). When a user enters an environment in which services are collecting data, a privacy beacon announces the privacy policies of each service in the environment. A user's *personal privacy proxy* (similar to P3P's user agents) checks these policies against the user's predefined privacy preferences; if the policies agree, the services can collect information and users can utilize the services. If the policies don't agree, the system notifies the user, who can choose not to use the service in question or, in extreme cases, leave the area in which the information collection is occurring.

Myles et al. (Myles et al., 2003) propose another architecture, shown in figure 3-1 (Myles et al., p. 4), based on location servers, validators and policies, also expressed using the P3P specification.

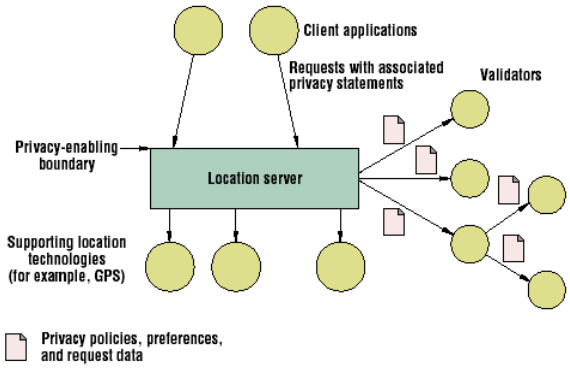

Figure 3-1 Myles et al's Anonymity Architecture

In order to obtain location-aware services, users must register with a location server, which in turn answers applications' queries concerning users' location. Upon registration, users also specify their privacy requirements by registering a special component called *validator*, which can either reside on the location server itself as a

self-contained middleware service or be available externally. A user can specify a privacy policy to limit the release of privacy information, according to time bounds (e.g. only make the information available at certain time of the day), or location bounds (i.e. the user is happy to let a shopping center track his/her position while s/he is shopping there but not when the user leaves the center). A user may even want to limit the level of accuracy that location information is released to an application (e.g. it is ok to know we are in shopping center but not in which shop we actually are). When an application requests the location or location related information about a specific user, it does by providing its privacy policy. In turn, the location server invokes the validator which, based on the application's policy and on the user privacy preferences, determines whether the requested information can be released. The validator can also protect the user identity (if requested by the user) so that it is not disclosed if the system receives a location request that requires it to divulge it. Depending on the user's policy the validator will return the user's long-term identifier, a short-term identifier associated with the user, or a new randomly generated identifier. Such a scheme however does not provide complete anonymity because applications might be able to deduce the mapping between temporary and permanent identifiers by observing movement patterns.

A more proactive approach to anonymity is achieved in the Cricket Location-Support System (Priyantha, Chakraborty and Balakrishnan, 2000). The latter uses radio and ultrasonic waves to determine distance and thus location. Unlike in other systems, such as AT&T Active Badges (Want et al., 1992) or the Cambridge Bat (Ward, Jones and Hopper, 1997), mobile devices never transmit their position at all; instead, they passively listens to the environment. In this way, the device knows its location, while the environment does not.

More examples of work in location awareness and privacy-aware initiatives can be found in (Beresford & Stajano, 2003), (Gorlach, Heinemann, Terpstra, 2004). The most interesting one is by Juels, Rivest and Szydlo (2003) which addresses the privacy issues associated to Radio Frequency Identification (RFID) systems which, thanks to manufacturing advancements are slowly being introduce as "smart" replacements for omnipresent optical barcodes. People carrying objects which contain RFIDs might not even be aware of the existence of these devices because of their size[2] and their passive nature. Also, due to their inability to do any computation (e.g. encryption), RFIDs require their own measures for privacy protection. The above

authors propose the development of a special RFID tag, called *Blocker Tag*, which can selectively block attempts of readers to identify it.

3.4 Technologies and Protocols Evaluation

Having clearly defined the security requirements for Service Oriented Architectures in nomadic computing, it is now possible to better evaluate the technologies and architectures presented in chapter two. The results of our evaluation are summarized on table 3-1.

The first observation that is possible to make is that no technology addresses security with respect to all the above defined requirements. This does not come as a surprise in as much the same way that perfect security does not exist and that the most secure computer is one which is not connected to a network and locked in an underground bunker, protected by three sets of steel doors; a computer which is not much use either, by the way. However, it is possible to make a number of important observations which are listed below.

3.4.1 Inconsistent approach

Security is not addressed consistently among existing SOAs, and each one perceives security in its own way. Some SOAs such as Bluetooth have been designed with security in mind more than others and therefore address it more thoroughly. However, not all SOAs address the same security requirements discussed earlier and even when they do, it is not to the same extent. In Bluetooth for instance, secure discovery is limited to the actual devices but once a device becomes discoverable so are all the services it offers. In other words one can perform an inventory of available services and carry out a selective attack on the devices which offer a particular service, such as a dial-up networking. Looking at SSDS, while service discovery is more thoroughly dealt with, the protocol fails to address the genuine service requirement. As the outcome of the service discovery is a list of matching service description records it is important to make sure that these have not undergone accidental or intentional modification. For other self-explanatory examples the reader can refer to table 3-1.

An inconsistent approach is not a problem per se, and it is to be expected that each technology would only address those security requirements that arose from the threat modeling. Unfortunately, for none of the technologies presented in chapter two was such modeling made explicit in the sense that it is not clear why certain security

		UPnP	Bluetooth	Salutation	SLP	Jini	JXTA	SSDS	Centaurus	Proxy-Based Security	Splendor
SSR/SSD		NO	NO	NO	YES	NO	NO	YES	NO	NO	YES
RP		YES	NO	N/A	NO	NO	NO	N/A	YES	NO	YES
Secure Discovery	AD	NO	YES	NO	NO	NO	YES	YES	YES	NO	YES
	ContD	NO	NO	NO	NO	NO	YES	YES	YES	YES	NO
	ConfD	NO	YES	NO	NO	NO	YES	YES	NO	NO	YES
	GD	NO	NO	NO	YES	NO	NO	NO	YES	NO	YES
Secure Delivery	ADe	YES	YES	YES	NO	NO	YES	NO	YES	NO	NO
	ContDe	YES	YES	NO	NO	NO	YES	NO	NO	NO	NO
	ConfDe	YES	YES	NO	NO	NO	YES	NO	NO	NO	NO
	GDe	YES	NO	NO	NO	YES	NO	NO	YES	NO	NO
Information Privacy		NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
Application Security		NO	NO	NO	NO	YES	NO	NO	NO	NO	NO

RP = Replay Prevention
AD = Authenticated Discovery
ContD = Controlled Discovery
ConfD = Confidential Discovery
GD = Genuine Discovery

SSR/SSD = Secure Service Registration/Deregistration
ADe = Authenticated Delivery
ContDe = Controlled Delivery
ConfDe = Confidential Delivery
GDe = Genuine Delivery

Table 3-1 Security evaluation of SOAs

requirements were addressed over some others. Besides, even for those requirements that were addressed it has not been justified why certain aspects were left out (e.g. the genuine discovery)

3.4.2 Service Registration

Apart from SLP no other standard technology addresses secure service registration and deregistration which means that any entity could register and deregister potentially malicious services. Besides, even standard SLP suffers from replay attacks so that the same service registration (service deregistration) messages can be replayed thus enabling (disabling) a previously disabled (enabled) service.

Secure service registration is not necessarily a problem in those cases where the service registry is embedded in the service provider (e.g. Bluetooth) and an implicit trust exists between the device offering the services, and the embedded registry. However, in those very same instances where service registration is trusted is not always possible to verify that trust via out-of-band means (e.g. human physical presence). With Bluetooth for instance, where service registration is trusted, it is usually possible to verify the trustworthiness of a service by visual contact with the device offering the service, in turn carried by a trusted entity (i.e. a known person). In some cases though, such verification is not possible and there is no way of telling a malicious service from a genuine one.

3.4.3 Service Discovery and Delivery

While service discovery and delivery may be secured in their own way by each different technology, one important consideration can be drawn by looking at table 1. All standard technologies address security from the point of view of service delivery rather t han service discovery. This is extremely worrying as it raises two problems. First, by eavesdropping the service discovery protocol it is possible to obtain an inventory of available services. Second, insecure service discovery also violates the privacy of service seekers. If one looks at what type of services those technologies were designed to support (e.g printers, projectors, networked appliances etc.), one may also fail to appreciate fully the relevance of these security threats. However, there are application scenarios, as those described in chapter one, where information

privacy is of utmost importance, and we would not want to allow an inventory of available services. If we think of a police agent as a service, who is registered in a *nomadic neighbourhood*, we can then appreciate the importance of keeping the confidentiality and anonymity of service discovery requests, as well as the responses (i.e. how many policeman there are in the neighbourhood)Still, from a more traditional viewpoint (where services are still offered by either software or hardware) achieving secure delivery on top of insecure discovery is also a contradiction; secure delivery of service discovered insecurely cannot be considered secure. This is for instance the case of UPnP where the secure discovery is based on the assumption that all available hardware on the network is legitimate and that there are no false UPnP devices.With regards to secure discovery we must also address the difference between the discovery of devices and the discovery of services that may be available on those devices. In UPnP for instance, as just highlighted, as well as for other cases, devices are assumed trusted, and a misplaced trust can undermine the security of service discovery.

3.4.4 OSI Mismatch

Not only security is addressed differently by the various SOAs, it is also addressed at different layers of the OSI network model. As we can see from the table below, service discovery and delivery protocols run on top of various transport mechanisms, dictated by the design of the specific architecture. In particular, most protocols are layered directly on top of TCP/IP, with the exception of UPnP and JXTA which use HTTP for their protocols. At the end of the scale we find Bluetooth, whose discovery and delivery runs on a proprietary protocol, which is positioned below the TCP/IP layers and can provide confidentiality and authentication directly at the physical layer of the OSI networking model.

	UPnP	Bluetooth	Salutation	SLP	Jini	JXTA	SSDS	Centaurus	Proxy	Splendor
Discovery	HTTP	L2CAP	Any	TCP /IP	Any	TCP/IP or HTTP	TCP/IP	TCP/IP	TCP/IP	TCP/IP
Delivery	HTTP	L2CAP	Any	TCP /IP	Any	TCP/IP or HTTP	TCP/IP	TCP/IP	TCP/IP	TCP/IP

Table 3-2 OSI mismatch in SOAs

3.4.5 Anonymity

While research on information privacy is producing its first results, it is also developing in parallel and not in integration with service discovery and delivery protocols. None of the SOAs examined addresses anonymity neither at discovery time nor at delivery time so that privacy of information can be provided. As ubiquitous devices permeate the every-day lives of ordinary people, privacy protection measures will have increasing impact on their lives. Even though the privacy issue can be dealt with from a technical viewpoint, it is also the subject of a still open social debate. In actual fact the debate on privacy has been open ever since the term privacy was introduced and chances are it will never end. However, it is the author's opinion that the ability to support information privacy will be a discriminating factor in the success of future service oriented architectures and it should be therefore considered as one of the guiding design criteria of any SOA.

On the other hand, even if clever anonymization technology for ubiquitous computing becomes readily available as it is already for the current and traditional distributed computing, one should question its implication on our social interactions. Unless we want to forsake our current way of interacting with the world and deal only behind digital pseudonyms in virtual reality with each other, we must realize that our real-world presence cannot be completely hidden, nor perfectly anonymized.

3.4.6 Service Definition

All SOAs are built around the definition of service, which represents the starting point for the design of the discovery and delivery protocols. In the context of a SOA, a service is defined as a "*...course-grained, discoverable software entity that exists as single instance and interacts with applications and other services*" (Brown, Johnston & Kelly, 2002). In other words a service is *"..well-defined, self-contained, and does not depend on the context or state of other services*" . Ultimately, service descriptions are required to provide discovery, selection, binding, and composition of services. Service descriptions are used to describe a service with regards to its capabilities, interface, behaviour, and quality (Papazoglou & Georgakopolous, 2003). In particular, the service capability description states the conceptual purpose and expected results of the service (by using terms or concepts defined in an application-specific taxonomy). The service interface description publishes the service signature (its input/output/error

parameters and message types). The expected behaviour of a service during its execution is described by its service behaviour description (for example, as a workflow process). Finally, the Quality of Service (QoS) description publishes important functional and non-functional service quality attributes, such as service metering and cost, performance metrics (response time, for instance), security attributes, (transactional) integrity, reliability, scalability, and availability.

Initially developed for Internet-scale environments, SOAs have been gradually adopted in more and more domains and the initial definition of service has gradually evolved and been re-defined slightly by the various architectures emphasizing the already existing integration issues. Service availability across coexisting SOAs greatly depends, among other factors, on the service definition itself. Existing definitions of services vary from strict ones to very encompassing ones.

From looking at the service definitions of the service architectures presented in chapter two, it is clear to see how the initial concept of service as a software entity has in some cases evolved into the more general concept of entity that can provide information or perform an action. A service is not necessarily a software entity anymore but it can be of hardware or even human nature. We could for example define a mobile medical service provided by doctors on the move in shopping centres, as described in first scenario presented in chapter one. The service is the one provided by the doctor, but we would still need a SOA to locate the service and deliver it (i.e. the doctor must be able to find the patient).

It is also surprising that all definitions but one, used in Salutation, do not addresses security directly. In other words security is added on top and achieved through the discovery and delivery protocols independently of service definition. However, if we were to define a service regardless of security and only deal with security afterwards we would be making a classic software engineering mistake. Security is a process and not a patch and must be taken into consideration right from the outset of any software design. 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 service(s) that the SOA provides. Therefore, if we want to design a secure SOA, we must start from a security-aware definition of service. But security is just one aspect of service definition; one more issue relates to defining the correct level of service granularity. While this is a fairly subjective thing, we can make a distinction between component or atomic services and composite services. We can define atomic a service

that is self-contained, that performs an atomic action and that does not depend on any other service. A composite service, on the other hand, involves and orchestrates the functionality of other services, which can in turn be atomic or composite. The level of granularity will depend on how low level is the functionality provided by an atomic service.

Another issue in defining a service relates to whether the service is pinned or unpinned. An unpinned service is a service which can be executed or used on any host. For instance, an unpinned service might be a device driver or software agent or mobile code which provides a service by running on the local host. A pinned service is instead located and running on a specific (remote) host. Examples of pinned services may be a coffee dispenser or vending machine for the purchase of candy bars or concert tickets, an interactive map running on a portable device etc.

3.4.7 Access Control

When looking at access control, the heterogeneity of existing SOAs really comes to light. We have analyzed all the architectures presented in chapter two and compared the way they address access control. The results of our evaluation are summarized in the following table 3-3. In particular, we have focused on four evaluation parameters, listed below.

- *Access Control Type* – Approach followed to perform the access control. As we discussed earlier, access control can be achieved in many ways.
- *Policy Enforcement Point (PEP)* – This is where access control is performed, and it is either by the device offering the services or by an external entity.
- *Policy Maker* – This is the entity in charge of defining the access control policy, which is then enforced by the PEP. A policy maker is not to be confused with the more traditional policy decision point (PDP), described earlier.
- *Granularity* – This is the granularity of the access control systems and refers to both the type of resources we can control, such as services and the devices from which the services are available, and the entities trying to gain access to those resources, such as simple users or just other services.

	UPnP	Bluetooth	Salutation	SLP	Jini	JXTA	SSDS	Centaurus	Proxy-Based Security	Splendor
ACT	**ACL and Capabilities**	**ACL**	**ACL**	**N/A**	**ACL**	**ACL**	**Capability**	**Capability**	**ACL**	**ACL**
PEP	**Device**	**Device**	**Device**	**N/A**	**Device**	**Device**	**SDS Server**	**Service Manager**	**Proxy**	**Proxy**
PM	**Security Console**	**Device**	**Device**	**N/A**	**Device**	**Device**	**Device**	**Device**	**Proxy**	**Proxy**
ACG	**Device**	**Device**	**User**	**N/A**	**Object**	**JXTA Peer**	**Device (SDS Component)**	**Client/Service**	**User**	**User**

ACT = Access Control Type
PEP = Policy Enforcement Point
PM = Policy Maker
ACG = Access Control Granularity

Table 3-3 SOAs access control models

The first observation we can make regards the access control types, which for all standard protocols, made exception for UPnP, is based on the use of access control lists (ACL) alone. Even though such ACLs are implemented differently among the various SOAs (from simple userID and password in Salutation up to authorization certificates in UPnP), policy enforcement is always performed by the devices themselves. This fact alone poses great computational constraints on the device, which in some cases need to perform public-key certificate verifications in order to control access to their services. Furthermore this architectural choice limits the scalability of access control to only a limited number of known entities. As the number of potential service users increases, so does the size of the ACLs and the authorization management associated to it, as we must add/delete entries to the ACL on each device to account for new/past users. Also, still for the standard technologies, access control policy is decided at the device level. The only technology that differs in that respect is UPnP which allows for better management of multiple devices through the *Security Consoles* (SC).

Finally, also the granularity level is different. In the majority of cases, access control is applied to devices, apart from Salutation where it addresses users specifically. Only with Jini does granularity get down to the object level, thus being very fine and detailed. The general approach however is quite coarse and it stops at the device level, following the assumption that in a mobile world most devices are personal and can be associated to specific users. Only the Centaurus architecture (along with Jini as already mentioned) addresses also other services as entity against which to apply access control.

It is only when we analyze the integrated architectures such as Centaurus and SSDS that a capability approach to access control is used, apart from Splendor, which is under-specified in that respect, and from the Proxy protocol, which addresses scalability through its ACL-controlled INS-based discovery protocol. Still for both Centaurus and SSDS, the access policy is defined by the device even though it is implemented by a third party, called Service Manager and SDS respectively. As seen in chapter two, both architectures use *capability managers* to perform authorization management. The use of capabilities is particularly useful as it allows scalable access control without having to refer to a remote authorization server. A capability can be

issued by a capability server and verified by a reference monitor (PEP). With UPnP for instance, capabilities are verified locally.

3.5 New Application Domains

The fact that none of the analyzed SOAs addresses all the defined security requirements is not necessarily due to bad design. Some technologies such as Bluetooth, which were designed with security in mind, did not aim to address certain security threats in the first place. Bluetooth in fact, simply aimed to be a wireless replacement for cables and therefore it addressed the security issue so that it could offer a cable-like type of security. Similarly, other service-oriented technologies were conceived with specific application domains in mind, each characterized by specific types of users and threats. While such technologies have become more widespread, personal devices have also developed to the point of enabling more advanced application scenarios of anytime-anywhere type. At the same time, new uses of the same technologies have brought about new security threats, as the same service technology is now being used beyond its original design context. For instance, while it is reasonable to expect that a Bluetooth-enabled phone only belong to one person, it is not similarly acceptable to expect that more general purposes and powerful Bluetooth-enabled devices are also associated to a single entity; the same assumption of trust no longer holds. Similarly, with mobility comes the need of anytime-anywhere access to service technologies, which were only conceived to operate with more static environments.

While we cannot stop the adaptation of existing technologies for the realization of more and more advances and ubiquitous application scenarios, we must also not disregard the new security threats that we introduce by doing so.

3.6 Integration Issues

As service oriented technologies become more widespread and the number of available services increases, integration will become the key to everywhere-anytime computing. Also, but foremost, integration is dictated by the diffusion of the nomadic computing paradigm which also works as the environment in which all standard and bespoke technologies can coexist and communicate with one another.

One way of reaching out to all available services would be for each device to support multiple discovery and delivery protocols. This approach strongly clashes

with the design principles of modern middleware, as it would entail a fat mobile platform. Support for heterogeneous service technologies could be achieved by either having multiple client platforms running on the same device or just one, which integrates all existing protocols. The complexity however, in this latter case would be far more than we could expect from most modern mobile devices.

As a matter of fact, integration of existing service technologies is still beyond reach and only experimented within the framework of a handful of research initiatives. The only wide-scale service oriented architecture that are around have "reinvented the wheel" by introducing yet another set of protocols, which do not interoperate with existing ones. For this reason, while the need for integration is evident one must also highlight the issues that make it difficult to achieve. And when we look at end-to-end integration of such SOAs we cannot escape the security implications, especially with regards to application scenarios as the ones presented in chapter one.

The security issues discussed so far with regards to existing SOAs have further emphasized the existing heterogeneity. In particular, we can list the following integration issues with regards to secure service provision in an integrated domain:

- *Different access control models* – As we have seen, different SOAs use different access control model, usually a choice between ACLs and capabilities. And there are also implementation differences. As a result, the same credentials cannot be used across different SOAs.
- *Different access control granularity* – This diversity creates problems too as the credentials must support both device and user information in order to cater for all types of granularity.
- *Scalability* – As users move around in an integrated environment they expect to have access to a wider range of services. Unfortunately, most current SOAs are not scalable from the point of view. As access control is performed locally by the device using ACLs, there is a physical limit, due to the nature of the device, to how many entities we can support.
- *OSI mismatch* – Security is addressed at different layers of the OSI model, from the physical one up to the application one. The same security requirements identified earlier can be therefore met in various ways, which again, creates

mapping problems from a single set of credentials to be used in an integrated environment.

- *Inconsistent approach to security* – When integrating security solution, the overall security is limited by the weakest link. As security is addressed inconsistently, the end result can be very poor.
- *Different authorization management* – Access rights are granted to users in different ways, which means that if we want to allow service access to a new user we must do it in as many ways as required.
- *No existing security-aware definition of service* – Current definitions of service are not capable of modelling future nomadic scenarios which present security requirements (as presented in chapter one). A service is the building block of any SOAs and correct service definition should be address from the outset; a security-aware definition must also include support for information privacy. Besides, current definitions are not even consistent across SOAs to promote interoperability.
- *Need of a security administration infrastructure* – An integrated architecture will be characterized by a large number of mobile devices, services and people. Security administration is of utmost importance. For instance, while in many SOAs the access control policy is currently defined at device level, an integrated environment should provide for better policy management according to contextual information.

4 SeNCA: a Secure Nomadic Computing Architecture

As discussed in chapter one, integration and interoperability of existing SOAs is more and more needed to support current and future needs of nomadic scenarios. The two sample scenarios, also discussed in chapter one, emphasize the need for the integration to be also secure. However, as concluded in chapter three, secure integration of existing SOAs is challenged by a number of factors, first of which the high level of heterogeneity of the security models supported by each SOA. In this chapter we present an architectural model, called Secure Nomadic Computing Architecture (SeNCA) for the secure integration of existing SOA in a context of nomadic computing. A crucial part of the architecture design is the security-aware definition of service and related data structure for its description. The whole architecture is based on such definition and, as the definition of service is security aware, the SOA model we are presenting is also security-aware.

4.1 Architectural Goals

SeNCA assumes that different users will be carrying different mobile devices. Each one of such devices will support one or more service oriented technologies for the discovery of services in the nomadic environment. An example of such nomadic environment would be a university campus scattered throughout the town, with user moving from one site or department to another. When moving around, users will be looking for services, which are available using the service technologies supported by their own devices. User may even be interested in discovering services which are available via other technologies; for instance, the user might be carrying another device which is currently switched off but, if turned on, could allow the use of a particular service only available through that device. As a result, users will need access to a service discovery mechanism, which is technology-independent but is at the same time aware of the technologies available to the user. All this must be provided while assuring a number of security requirements.

SeNCA does not aim to provide security where it not natively supported by the service technology. For instance, if the discovery protocol doesn't provide confidential discovery, SeNCA will not be able to provide it. Interoperability cannot

provide security. We cannot translate from one technology to another otherwise we lower security to lowest common denominator. However we aim for integration with the nomadic environment. This way we support the security of existing service technologies and add security administration and scalability on top of it. Following on from the requirements defined in chapter three, The Secure Nomadic Computing Architecture (SeNCA) aims to meet the following:

- *Secure service registration of heterogeneous service technologies* – For instance is should be possible to securely register Bluetooth or UPnP services, which do not support secure registration natively
- *Secure service discovery of heterogeneous service technologies* – The discovery is meant to be secure from the point of view of authentication, authorization, confidentiality and anonymity. However these requirements will only be supported as far as the local technology allows. SeNCA discovery will be service technology-independent but device technology aware. In other words, a search query for a printer will discover both Bluetooth and Jini enabled printers but it will be know that the user device only supports Bluetooth
- *Scalable access control* – SeNCA will provide scalable access control, beyond the limitations of the various service technologies.
- *Security administration of policy and trust* – This is required to manage large number of devices and services through properly managed policies, and also to add trust and authorization management to limited-resources devices.

The following architectural goals are also considered:

- *Scalable discovery* – As integration increases the number of possible services that can be discovered, the architecture must be scalable to cope large numbers of services
- *Non invasive* – SeNCA does not aim to change any of the service technology protocols that are being integrated. Nor does it aim to develop software components that need to be installed on every client device.

Having defined the needed security functionalities and architectural goals, these must be met taking into consideration at the same time the middleware design goals identified in chapter one.

4.2 Architectural Overview

Integration introduces new technologies, new services and adds overall complexity to the discovery process. In order to manage the increased complexity and also address issues of scalability it is useful, if not mandatory, to use a hierarchical approach.

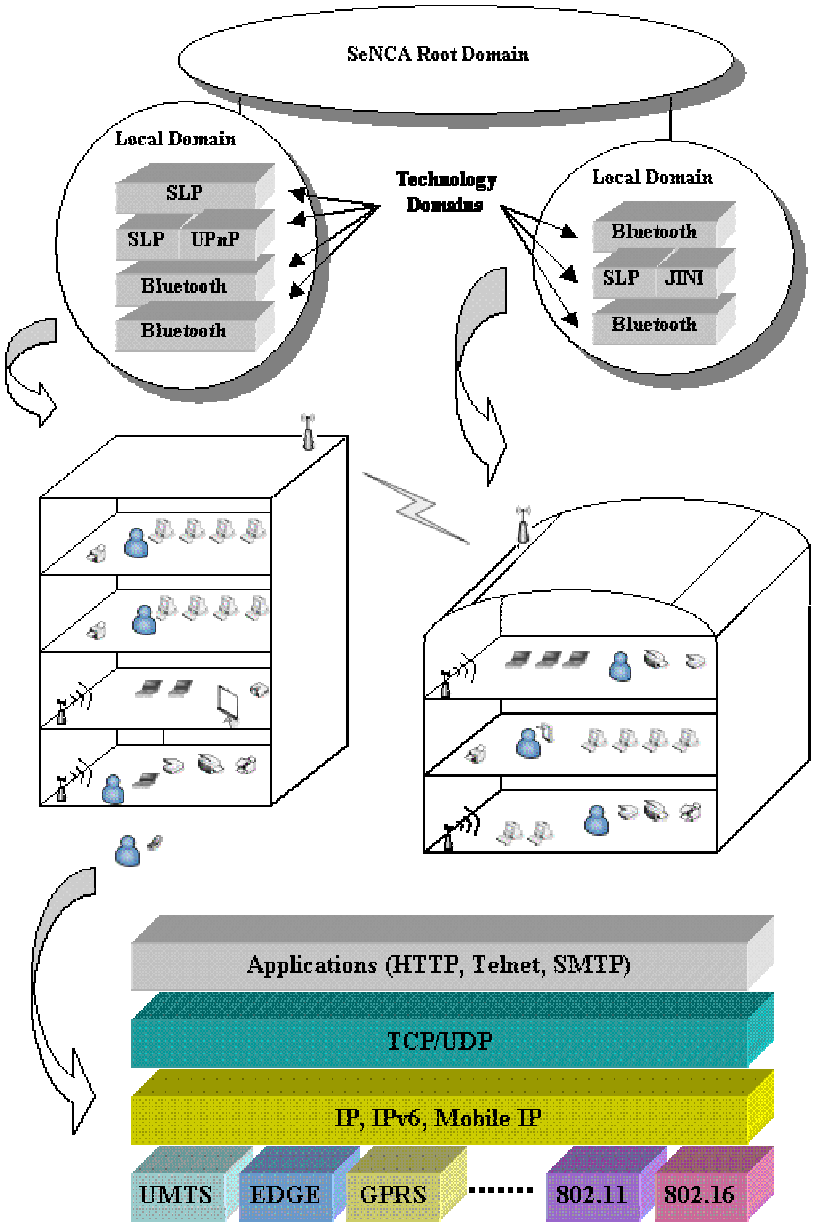

Figure 4-1 SeNCA architectural model

Examples of useful hierarchies include those based on administrative domains, network topology, network measurements (e.g. bandwidth and QoS) and geographic locations. SeNCA hierarchies are dictated by technology constraints, scalability problems and security requirements. In particular SeNCA identifies two categories of domains, shown in figure 4-1.

- *technology domain* – this type of domain represents a specific service oriented technology such as Bluetooth, JINI etc. Each technology domain is under a specific administration, which in turn can be responsible for more than one technology domain.
- *administrative domain* – this type of domain identifies a geographical area (e.g. a building) under a single administrative authority. SeNCA adopts a hierarchical administration model with two levels of administration domains. The topmost level is called the *root domain*, and it contains one or more *local domains*. A local domain can contain one or more technology domains, which are all under the same administration. There is only one root domain and it serves as a trusted domain for communications between local domains.

Compared to administration domains, technology domains can overalap (figure 4-1 shows some building floors with two technology domains). Administrative domains cannot overlap as each service can only be under one administration. When moving around in the nomadic environment, it is assumed that a user will have access to one or more wireless communication technologies, which will allow the user to connect to the SeNCA network infrastructure. The latter in turn, is based on IP and all SeNCA architectural components are assumed to have an assigned IP address. In particular, a local administration domain will also represent a range (not necessarily continuous) of IP addresses.

4.2.1 Architectural Components

The following is a detailed description of SeNCA architectural components:

- *Secure Technology Domain Proxy (STDP)* – Performs local discovery and import services into SeNCA (proactive discovery). STDP have two communication

interfaces, one towards the technology domain (e.g. Bluetooth) and one to the SeNCA domain to communicate with the SLDP. Each STDP performs a semi-automated import procedure in the sense that before a service can be registered it must be verified. As we said, each technology domain is associated to a specific administration, which specifies the rules for verifying and trusting the services discovered (technology dependant). This way the service is securely registered. An STDP also manages access control for local services, applying the security policies dictated by the SLDP and mapping credentials from the user profile to technology-specific credentials.

- *Secure Adaptation Agent (SAA)* – This component sits on each technology domain device and manages access control on the device. For instance, a SAA may be running on a Bluetooth device and change the content of the *device database* of the device in order to grant trust to a new device that wants to gain access to the services provided.
- *Secure Local Domain Proxy (SLDP)* – This component maintains a registry of all the services available in the local domain, interacting with all SLTPs in the local domain. Each SLDP then runs the SeNCA Discovery protocol (SDP) and forwards service requests from the local domain to other SLDPs in SeNCA; it also responds to service discovery queries coming from other SLDPs. It then manages the security policies for the local domain communicating them to the STDP that enforces them on the local technologies.
- *Secure Local Client Proxy (SLCP)* – A SLCP is a Web agent, accessible by the user, which performs queries on behalf of the user and bears all the complexity required to perform secure discovery, based on the use of cryptographic operations. User-to-SLCP communication is protected through a hybrid encryption system (symmetric-asymmetric).
- *User Profile Registry (UPR)* – This is a registry of user profiles defined in the root administration domain. User profiles are registered at root level, for example using an LDAP directory. Local administrative domains on the other hand, decide the access control policy for local services.

4.3 SeNCA – A Three-Phase Service Oriented Protocol

SeNCA must allow a discovery of services, which is at the same time technology and location independent within a set nomadic environment. Certain service technologies such as Bluetooth only allow local discovery and are not geographically scalable (because of technology limitations). Other technologies such as SLP may also be locally scoped (because multicast-based).

Given the heterogeneity of service technologies it is not possible to use one technology-specific discovery protocols to discover all available services. Two possible approaches could be followed then, which allow the discovery of services independently from the technology and the location (which is our aim). The first approach is to achieve interoperability by importing services of one technology into other technologies. However, this approach cannot work for all combination of services and technologies and has strong implications for the security requirements we want to meet, unless we are prepared to modify the legacy protocols[5]. The second approach, which is the one followed by SeNCA, is based on service technology integration. Figure 4-2 shows and architectural overview of SeNCA.

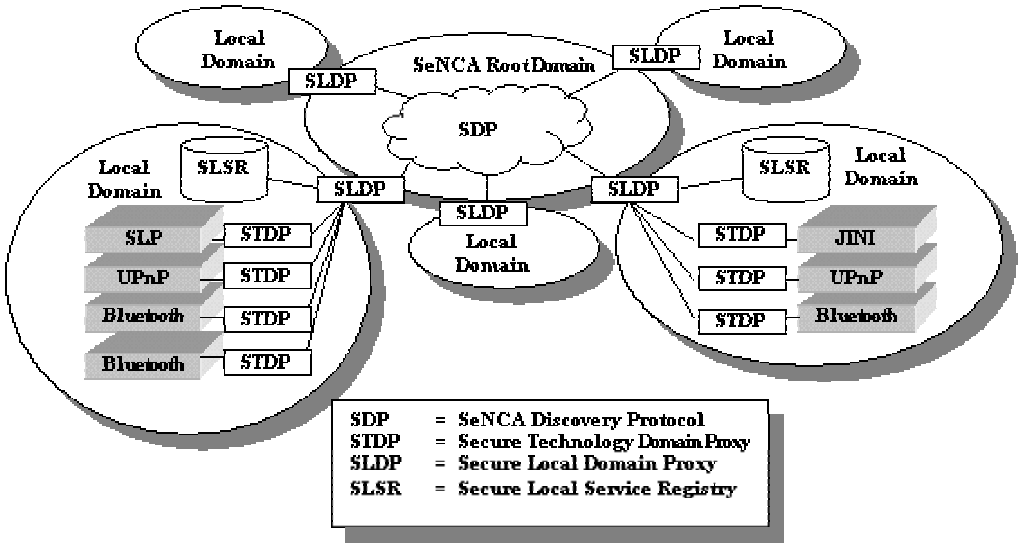

Figure 4-2 SeNCA architectural components

[5] Example: we want to import services available from a Bluetooth domain into a JINI domain. However, in Bluetooth we can only discover services if we are an authorized Bluetooth device. Importing Bluetooth services into JINI means that we either lower the security by allowing all JINI client to discover the imported services, or that we modify the JINI discovery protocol to only allow authorized clients.

In order to discover all existing services, these must all be published on the SeNCA registry before we make explicit service requests. SeNCA adopts a proactive service discovery paradigm. In particular, associated to each technology domain is a *secure technology domain proxy* (STDP), which performs a service discovery using the specific domain technology discovery protocol (e.g. Bluetooth service discovery). All discovered services are exported to SeNCA but kept in the local administrative domain. In particular, each local administrative domain contains a *secure local service registry* (SLSR), maintained by a *secure local domain proxy* (SLDP), which receives service discovery updates by all STDPs in that domain, and communicates with other SLDPs of SeNCA using the SeNCA Discovery Protocol (SDP). This way SeNCA maintains a distributed service registry. The discovery protocol itself is a hybrid proactive-reactive one. It is proactive as services are discovered by the STDPs and exported to SeNCA; it is reactive because users only discover the actual services at search time, as the query is sent across the NSCA network. In order to offload user devices from the computational complexity required by cryptographic operations, SeNCA adopts Secure Local Client Proxy (SLCP), which are entities trusted by the user device. Each local administrative domain defines the access control policy for its services and may decide to grant access to users as they move around the nomadic environment. SeNCA manages authorization through special components, called Secure Adaptation Agent (SAA). Finally, all secure discovery and delivery operations are based on the use of user profiles, centrally managed and stored in the User Profile Registry (UPR). User profiles contain information regarding the user available devices, and personal privacy preferences, needed for anonymous service discovery and delivery.

4.3.1 Service Registration

Service registration is a semi-automated process, which involves the STDP and the SLDP. The following steps describe the service registration:

1. The STDP performs a local discovery of devices using the specific discovery protocols of the technology domain.

2. Each discovered service is verified or automatically trusted according to a *service acceptance policy*, which is set by the technology domain administrator, who has the responsibility of vouching for services in that domain.
3. After the service has been verified the STDP builds a *SeNCA Service Description Record* (SSDR), based on service information obtained from the local discovery, and adding descriptors to specify the security requirements of the service. Such security requirements are defined in a *service security policy*, which the STDP inspects before creating the SSDR.
4. The STDP sends a secure service announcement message to the SLDP (authenticated and confidential and guaranteed for integrity), containing a list of the SSDRs of the services found.
5. The SLDP verifies the service announcement and stores the SSDRs in the *service registry* of the local domain

Figure 4-3 shows a sequence diagram of the SeNCA registration protocol.

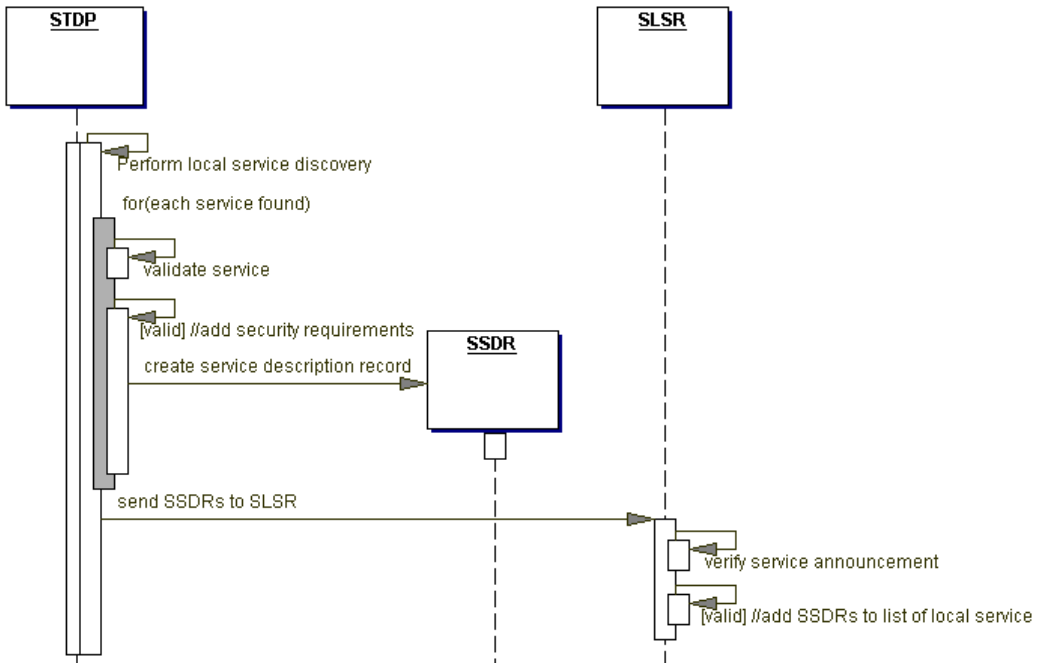

Figure 4-3 SeNCA service registration

In particular, the following security functionalities are achieved:

- Only trusted services are registered in the SeNCA distributed registry
- Secure registration and deregistration is achieved using the secure service announcement between the STDP and the SLDP. Even if a new service is registered in the local technology domain, it will not be registered in SeNCA automatically.

For some of the service technologies, the STDP must be authorized to perform local discovery. SeNCA assumes that each STDP is authorized using the technology-specific mechanism, either manually or programmatically. For instance, in a Bluetooth domain, the STDP will need to be authorized by the Bluetooth devices in the domain to discover the services they offer. Also, before registering a service, the STDP must make sure that each service offered locally has a different name. For example, it would be quite legitimate for two Bluetooth devices to have the same name and be offering the same service. However, given SeNCA two-phase discovery protocol, it is not secure to allow the latter possibility.

4.3.2 Service Discovery – First Phase

In order to achieve secure service discovery, SeNCA uses a two-phase discovery protocol. The main components involved in the first phase of the discovery are the *Secure Local Client Proxy* (SLCP), the *User Profile Registry* (UPR) and all the SLDPs of the SeNCA domain. The following steps, shown also in figure 4-4, describe the first phase of the discovery protocol:

1. User logs on the SLCP; this is a Web proxy available in every local domain. User and SLCP mutual authentication is achieved through SSL and the user device must only be able to verify the public key certificate sent by the SLCP. A symmetric key is also agreed in the process, which is going to be used to secure all communications between the user and the SLCP. In order to perform user authentication the SLCP retrieves the user profile from the UPR and compares the credentials given by the user with those kept in the user profile.

2. A user searches for services (e.g. a printer). Optionally, the user can specify to only discover services that are available through the user device-supported service technology (e.g. Bluetooth).

3. The SLCP filters user requests on the basis of the personal privacy preferences contained in the user profile. For instance, the user may only want to discover services which do not require any user information (identity anonymity). A service discovery request message is then sent to the local SLDP along with a partial user profile (based on the privacy preferences), and from there to all other SLDPs in the SeNCA domain.

4. Each SLDP that receives a service discovery request message performs a matching between the user credentials and the security requirements of the services registered in the local registry. The user profile is also needed to filter service matches against the service technologies available to the user. Each SLDP is authorative for the services held in the local service registry and decides independently whether to allow the service discovery for a particular user.

5. All matching records are sent to the SLDP of the local domain where the query was initiated; here they are collated and sent to the SLCP and finally to the user, ordered according to service technologies available to the user device.

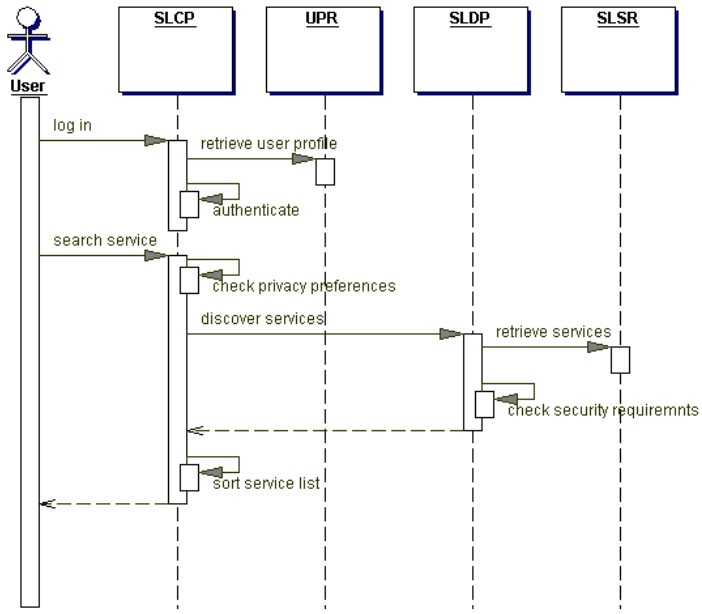

Figure 4-4 SeNCA service discovery phase one

SeNCA first phase discovery protocol allows to achieve the following security functionalities:

- *Confidential discovery* – all service discovery messages can be sent encrypted from the SLCP to the SLDPs does achieving the required confidentiality
- *Authenticated discovery* – this is also achieved through encryption for the communication between the SLCP and the SLDP and between the local SLDP and all other ones.
- *Controlled discovery* – By looking at service security requirements in the SSDRs, each SLDP can control the discovery of service to authorized users only, and limit visibility according to time restrictions, for instance, or other parameters.
- *Information privacy* – The user profile contains the user's privacy preferences. If the user wants to maintain anonymity, the SLCP will forward anonymous service discovery messages by just sending a cut-down version of the user profile
- *Service availability* – Once a service has been registered it can be found using the first phase of SeNCA discovery protocol. This approach also improves service availability as we do not perform technology-specific service discovery, which in some cases such as in Bluetooth may deplete the battery of the device. We only perform local discovery when we are ready for the delivery.

4.3.3 Service Authorization

Services found during the first phase of the SeNCA discovery are not necessarily local services and they may be available in other administrative domains to the one where the discovery was initiated. Services in SeNCA are delivered using the native delivery protocol and therefore they must be ultimately discovered through the native discovery protocol. The second phase of SeNCA discovery protocol in fact uses the discovery protocol of the technology domain itself. However, depending on the specific technology domain, the service discovery might be natively secured and need specific credentials. Therefore in order to be able to perform a local discovery, the user must obtain some credentials that can be used locally. The following steps, shown in figure 4-5, explain the service authorization procedure.

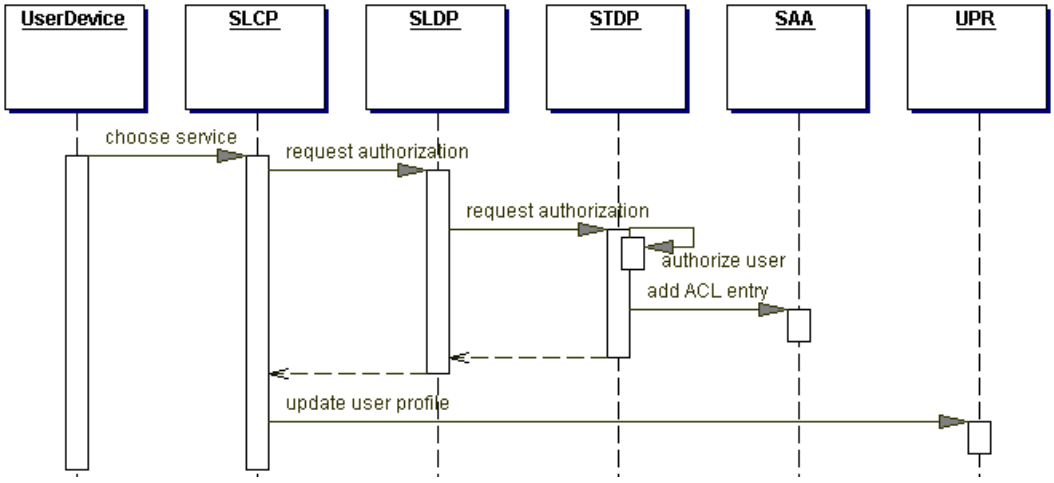

Figure 4-5 SeNCA service authorization

1. The user chooses one or more services from the list of matches returned by the SLCP.
2. Upon choosing a service, the SLCP begins an authorization request procedure on behalf of the user. This procedure is needed to obtain technology-specific credentials that can be used for the local discovery (if this is natively secured) and eventually for the delivery. The SLCP forwards authorization requests to all relevant SLDP (according to the chosen service).
3. Each SLDP that receives an authorization request, forwards it to the relevant STDP
4. Upon receiving an authorization request, each STDP does three things. The first is to create a technology specific set of credentials for the remote user, using information contained in the user profile (e.g. device network identifier); second it instructs the SAA on the device offering the service to add a new user to its ACL (or similar depending on the access control model). Finally, the STDP sends the new credentials to the SLCP, which in turn adds them to the user profile, for later use during the second phase of discovery and/or the delivery. In other words the STDP takes care of authorization management.

It is important to stress that in order to use a service, the user must still possess a device enabled with the technology specific service technology. This is because, SeNCA does not provide service interoperability but service integration; services from one technology are not imported into other technologies and therefore it is not

required to perform credentials mapping between different service technologies. However, service authorization is likely to generate some tokens such as an authorization certificate or a PIN. Such tokens need to be delivered to the user and are therefore stored in the user profile.

4.3.4 Service Discovery – Second Phase

Once the user has chosen the service to be delivered, his/her profile is updated with the needed credentials. In order to use the intended service, the user must move to the physical location where the service is available. Directing the user to the exact location may be part of another service offered via SeNCA or service technology available within it. Here we will therefore assume that the user knows how to get to the location where service is going to be delivered. We will also assume that SeNCA will be able to monitor the user movements from one local domain to another, using for example RFIDs or other technologies. The following is the list of steps involved in SeNCA second phase of the discovery, also represented in figure 4-6.

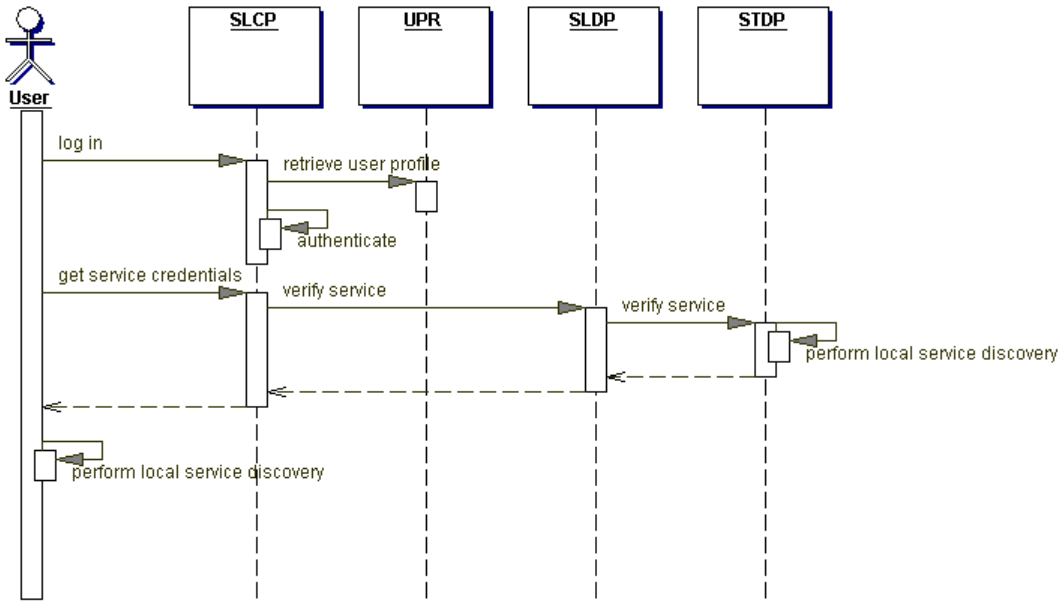

Figure 4-6 SeNCA service discovery phase two

1. The user moves to the physical location where the service is available.
2. The user logs onto the local domain's SLCP to obtain the needed credentials.
3. The SLCP retrieves the user profile
4. The user selects the service that he wants to use

5. The SLCP displays the associated credentials to the user. Before doing that though, the SLCP makes sure that the service is still available. In fact, between the first and second phase of the discovery, an attacker might have placed a new device in the local technology domain, offering a service with the same name. Therefore, the SLCP asks the SLDP to verify the validity of the service. The SLDP then instructs the relevant STDP to perform a new local discovery as the one performed at registration time. Each local service must have a different identifier so that, if performing the new local search, the STDP discovers two services with the same name and it will know that something is wrong and issue an error. Similarly, an error is also issued if a service with the right name is found but coming from a different service provider[6].
6. The user performs a local discovery using the new set of credentials if this is needed; the local discovery in fact may not need any credential.

SeNCA second phase discovery protocol achieves the following security functionalities:

- *Scalable access control* – It is possible to achieve a controlled service discovery beyond the initial set of trusted client entities.
- *Discovery of trusted service* – SeNCA performs behind-the-scene checks to verify that the service found with the local discovery is still trusted from when it was discovered with phase one.

In practice, the SeNCA second phase discovery protocol does not add any security to the native discovery protocol. If confidentiality is not natively supported it will not be achieved. The same applies for other requirements. However, this is because SeNCa does not add anything or modify the technology specific discovery protocol.

[6] Two Bluetooth services with identical names, offered by two different devices that also have the same name, will appear identical to a user who has no way to tell which one is the service discovered during phase one of the protocol

4.3.5 Service Delivery – Third Phase

Once a service has been found it will need to be delivered. This is the stage where most service technologies address the security requirements and this is where then, we find most of the integration issues. As we saw in chapter three, secure delivery can be addressed from many angles such as authentication and authorization. However, secure service provision can be achieved in two possible ways. The first is by following the principle of least authority. In other words, client entities are only allowed to discovery what they are allowed to use, which means that authorized discovery also implies authorized delivery. The other approach is to allow service discovery first and to perform authorization at delivery time. The first approach can be followed to prevent enumeration attacks that could be performed to find out what services are available in the environment. The second approach can be adopted for less critical services, which still have security requirements but do not need to be hidden from users, such as m-commerce services. In the latter case the service provider would want the service to be found but only deliver it to end users who could meet the specified security requirements. We call the second approach *post-discovery access control*. SeNCA can support both approaches by using appropriate service descriptors.

In SeNCA service delivery assumes that the user has already performed secure discovery and is now prepared for the delivery. Therefore the secure integration of service technologies is actually achieved when the user chooses the services after phase one of the discovery. With regards to the secure delivery requirements SeNCA does not achieve any more that already supported by the native technology. The added secure functionality regards the scalable access control as a local service could be also provided to client entities from other administrative domains. Also, it could be used in principle to provide anonymous delivery if the local credentials mapping does not require knowledge of the user identity.

4.4 SeNCA Services

The possible services that SeNCA aims to cater for are all the service types that are available through the various technology domains. Besides, SeNCA aims to support the extended notion of *people services*, i.e. services offered by human entities.

However a more formal definition of service is required which, for the reasons discussed in the previous chapter, must be security aware.

Service definition in SeNCA follows a capability approach. In their research on capability-based security Miller & Shapiro (2003) define an object as the "*.. finest-grain unit to which separate direct access rights may be provided, such as a file, a memory page, or another subject, depending on the system*". Similarly, restricted access to an object is modelled as access to an object whose behaviour embodies the restriction. Without loosing in generality we could apply Miller and Shapiro's definition of an object to define an atomic service, or simply a service, as the finest-grain service to which separate access rights may be provided. In the context of SeNCA we will use the following definition:

> *A service is any self-contained entity, pinned or unpinned, to which specific security requirements may be applied, which can provide information, perform an action, or control a resource on behalf of another entity. A service may be provided by either a software, hardware, or a human entity or by any combination of such entities.*

As we can see, the above definition is security-aware in the sense that no service can be regarded as such if no specific security requirements can be applied to it. In other words, we can only have a service if we can protect it. The reader should also notice how encompassing the above definition is; not only does it cater for traditional type of services but also for those provided by hardware and human entities.

4.4.1 SeNCA Service Description Record (SSDR)

A SeNCA Service Description Record (SSDR) must allow the description of services so that we can perform secure discovery and delivery. The structure of the SSDR aims to describe:

1) *legacy services* – First an SSDR must allow us to describe any type of legacy service that is available through the various technology domains
2) *people services* – Service description should also support the concept of services offered by human entities rather than just the traditional types

3) *security requirements* – This is needed to both capture the security requirements of native service technologies, and to enable secure service discovery at SeNCA level (discovery phase one). Where the service being described is a *people service*, this field will also contain the personal privacy preference of the person offering the service.

In order to meet the first two design goals the SSDR must contain enough information to be able to represent any type of legacy service. A simple but inefficient approach to follow would be to include in the SSDR a union of all possible legacy service descriptors. Alternatively one could try to extract semantically equivalent information across different service descriptors and only include that in the SSDR. Based on the way services are described by the various SOAs presented in chapter two, we can define the structure of the SSDR to include the following three fields.

- *Basic Information (BI)* – This field contains attributes (mostly mandatory) that uniquely identify and describe the service, such as an identifier, a name and some keywords. This field must include support for people services, and to this end SeNCA should contemplate integrating with the vCard standard (Howes, Smith and Dawson, 1998)
- *Service Profile (SP)* – This field contains attributes, some mandatory and some optional, which specify the class (or category) the service belongs to, such as printing services or video services. This field can be also used to specify the category of people services.
- *Device Capabilities (DC)* – This field contains attributes describing the physical requirements of the client device onto which the service will be running, such as minimum memory requirements, CPU type, and I/O devices. In particular, this field is used to capture the name of service technology through which the service is available (e.g. JINI, SLP etc.). This field should not normally be used for people services which are not meant to be delivered on any client device

In order to describe the security requirements of a service SeNCA defines a fourth field in the SSDR, called *Service Security Requirements (SSR)*. In particular, SeNCA follows the approach adopted by the CSIv2 architecture, where associated to each

secured objects is a data structure called *CSI compound security mechanism*, which describes the security requirements that a client must meet on all three of the CSIv2 layers, so that it can access the object. As for CSIv2, also for the SSR, the security requirements are structured in a number of layers. However, the layers identified must take into consideration the mobile context of application. Therefore SeNCA identifies a new layering structure called *Service Secure Interoperability* (SSI), shown in figure 4-7. SSI layer structure is dictated by the design goal to support different types of service and not only legacy ones. Future services will be layered on top of diverse communication technologies, using different transport protocols and delivered using new protocols.

Service Security Interoperability
Service Layer
Transport Layer
Communication Layer

Figure 4-7 Service Secure Interoperability

Compared to the CSIv2 architecture, which starts from the transport layer, the SSI architecture includes the lower wireless communication technologies, which compared to traditional layer one technologies address security requirements. Using the SSI structure allows SeNCA to support post-discovery access control. The service secure interoperability information is contained in the SSR field, whose structure is shown in figure 4-8.

Service Security Requirements
Service Security Interoperability
Availability
Information Privacy Preferences

Figure 4-8 SeNCA service security requirements field

Besides the service security requirements, the SSR field also contains information regarding service time availability, i.e. times when the service is available. In fact a service may be available for users in the local domain but not to users coming from

other administration domains apart from certain times of the day. If a service is not available it means that the authorization procedure performed by the STDP cannot be applied. Last in the SSR data structure, we find the service *information privacy preferences*, which allow to specify restrictions on the SSDR that can be sent back to the user as outcome of the discovery. In other words, a user may be allowed to discover a service but only have limited visibility of the service description record. This is particularly relevant to people services where a person may want to restrict disclosure of personal information. A doctor may allow to be found by potential patients but may not like to disclose his personal telephone number. Figure 4-9 shows the high level structure of the SeNCA Service Description Record.

SeNCA Service Description Record
Basic Information
Service Profile
Device Capability
Service Security Requirements

Figure 4-9 SeNCA Service Description Record (SSDR)

4.5 SeNCA Client Profiles

Client profiles are a key component of the SeNCA model as they are used to achieve secure service discovery and delivery. So far we have used the term user profile to identify the data structure describing a human client entity and the associated security requirements and credentials. However, a client entity is not necessarily human and a user profile must be in fact more generally defined so to encompass also other types of client entities. In particular, the design of the SeNCA client profile is driven by the following goals:

1) *capture general information of a client entity* – A profile must include general information about a user or device or any other type of client entity
2) *capture information privacy* – The profile must allow to capture privacy preferences so that just enough private information is disclosed to achieve service

discovery and delivery. The degree of information disclosure may depend on the service

3) *store client credentials* – Access control to service discovery is performed based on the client credentials such as the local administrative domain the client belongs to, the role etc. Credentials also relate to the specific communication technology, transport and service protocol.

As regards the first design goal, the profile must allow for the description of different types of client entities. As we saw in chapter three, access control granularity varies across service technologies, ranging from devices to subjects invoking a software object. For this reason the client profile must contain information that identifies not just a human entity but also another service or software agent.

Client profiles in SeNCA should follow the approach taken by W3C with their Composite Capabilities/Preference Profiles (CC/PP) recommendation (W3c, 2004), and by the WAP forum with their User Agent Profile (UAProf). A CC/PP profile provides description of device capabilities and user preferences that can be used to guide the adaptation of content presented to that device. The WAP User Agent Profile is concerned with capturing classes of device capabilities and preference information, including (but not restricted to) the hardware and software characteristics of the device as well as information about the network to which the device is connected. Compared to CC/PP and UAProf though, SeNCA profiles could happily lose content preference information and be more "service oriented". Also, SeNCA profiles are held in a central repository in the nomadic infrastructure and not by the devices themselves. This choice is based on two motivations. First, a SeNCA client profile is much more dynamic in nature as it can store other information than just device capabilities, such as personal preferences and temporary authorization tokens required for service access. Second, if a user loses a device there is no loss of sensitive information and no authorization leakage.

Client profiles are pre-registered in the SeNCA domain; there is one profile per user. Each profile should contain a general section with the user credentials, and a section describing the capabilities of the devices used by the user. Figure 4-10 shows the structure of the SeNCA Client Profile

SeNCA Client Profile
Client Basic Information
Device Capabilities
Client Credentials
Privacy Preferences
Service Security Interoperability

Figure 4-10 SeNCA Client profile

The first field contains basic information about the client, and varies according to the type of client entity. For human entity it is conceivable to use a vCard data structure. For most user devices, the second field could be populated with information taken by UAProf profiles. Generally speaking, the device capabilities field mirrors the homonymous field in the service description record and is used by the SLDP to filter the service description records that match the client requests.

The *Client Credentials* part of the profile contains technology-specific credentials, related to the device and/or supported service technologies; also contained here are other client security attributes that are used by the SLDP to make access control decisions.

The privacy preferences part is interpreted by the SLCP when sending service discovery queries to the SLDPs. Depending on the content of the privacy preferences field, the SLCP will only send a sub-part of the client profile. For instance, using this field it would then be possible to achieve identity anonymity.

Finally, the SeNCA client profile also contains a Service Security Interoperability part (SSI), which mirrors the SSI field in the service description record. Here, the SSI field is used to capture the client security requirements, which must be met by a service provider in order for the client to accept the service delivery. A security requirement may be for instance that of authentication and confidentiality at the communication technology layer. When performing service discovery, the SSI field of the client profile are matched against the SSI field in the service description record.

4.6 Implementation strategies

SeNCA represents an architectural model for the secure integration of service oriented technologies in nomadic computing systems. The model is based on the use of information rich descriptors of services and client entities, also known in other contexts as metadata or data about data. The use of metadata to improve information discovery is extensively recognized. In particular, the authors have experimented with Quality of Service (QoS) metadata to improve wide area information discovery services (Graziano et al., 2002); figure 4-11 shows a high level diagram of the architecture for wide area information discovery and delivery.

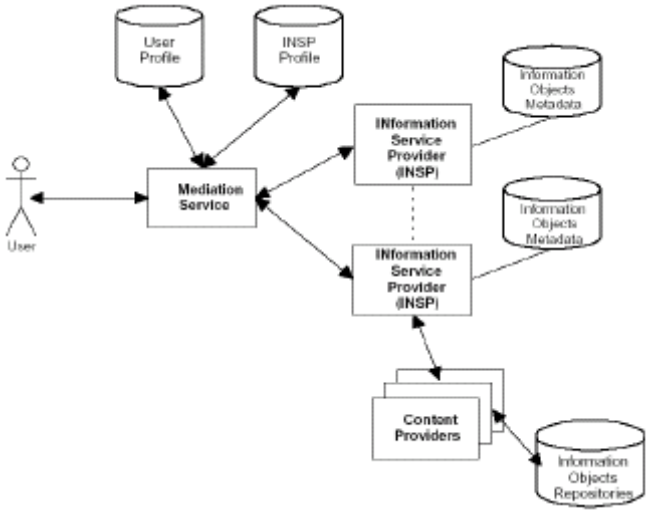

Figure 4-11 QoS-enhanced information discovery architecture

Here too we find user profiles, and information objects are described using metadata enriched with QoS information to describe the delivery requirements. User profiles on the other hand contain a description of the device capabilities and user preferences, using CC/PP. In the resource discovery phase, search engines and/or mediators acting as information brokers use QoS metadata to customize search capabilities according to the user preferences and needs, matching these against the characteristics of both the service provider and of the digital resources. In the delivery phase, distributed multimedia applications, that are part of a QoS-aware information management system, exploit the extra QoS-metadata to negotiate the quality-of-service with network equipments, according to QoS network protocols, such as the Resource ReserVation Protocol (RSVP).

The above architectural model presents itself as an ideal candidate for developing SeNCA. In fact, service discovery in SeNCA is for all intent and purposes a problem of information discovery over a wide area, whose result is a set of service description records. Also, SeNCA itself does not aim to develop a platform for service delivery, which is instead left to the specific service technologies.

With regards to the actual security, this can be achieved using public key cryptography with X509.v3 certificates, similarly to other secure discovery protocols. However, compared to these very same protocols, SeNCA does not use public key certificates to identify each individual user and service in the system but rather to identify only the architectural components of the nomadic domain that are supposed to communicate securely and trust each other. Service discovery is performed by proxy components which also manage the registration of services. Therefore, only these components need to be trusted as they vouch for the security of services and the users they represent. Choosing to associate public key certificate only to the architectural components would greatly improve the scalability and implementation issues, traditionally associated to the use of PKIs.

With regards to the actual SeNCA service protocols (registration and discovery), these can be authenticated and encrypted using a classic hybrid encryption model, with session keys established between communicating proxies (possibly cached for improved performance). In particular, all communication could be administratively scoped using multicast. Each SLDP in a local administrative domain sends authenticated messages on a well-known multicast address, containing the multicast address that STDPs will use for sending service announcements, the range of IP addresses that SLDP is responsible for, and the desired service announcement rate (i.e. how often services are required to announce themselves). The messages are sent periodically using an announce/listen communication model. Once an SLDP has established its own local domain, it begins caching the service descriptions that are advertised by the STDPs in the domain. The message sent by the STDP can be encrypted with the SLDP public key to guarantee confidentiality; service registration messages must also be secured against replay attacks.

Finally, also user devices would be required to perform a bare minimum of public key operations when connecting to the SLCP. However, such operations are not required each time as caching of the symmetric key for limited period of time could be used.

5 Designing Secure SOAs for Nomadic Computing

Security requirements in pervasive computing depend on the types of application we want to address and is contextually dependent on both the physical and organizational (human) environment. The physical environment dictates the technological infrastructure and limitations in which the mobile device operates. The organizational environment on the other hand, dictates the trust level among the various entities. Both contextual factors affect the way the identified security requirements are met.

In this chapter we present an object oriented methodology for the design of secure SOAs for nomadic computing systems. The methodology uses UML extensions for the formalization of security requirements. In particular it is shown how the design of secure SOAs can address the new threat models described in chapter one. The methodology presented can be generalized and applied to the design of secure applications in general, and for the evaluation of existing service oriented architectures. The first part of the chapter gives an overview of existing methodologies for secure software and system engineering, reflecting upon their applicability to the ubiquitous domain.

5.1 Methodologies for the Development of Secure Systems

As widely discussed so far, the requirement for security is both expanding and changing. Again, the expansion is due to the increasingly interconnected nature of systems, and new types of distributed system, especially pervasive ones, that use this interconnectivity to support more forms of social collaboration. As the requirement for security has evolved and become more pervasive, the challenge is still on as to how to integrate security, or more generally non-functional requirements, into the requirements development process and into the practical software engineering context. A number of methodologies exist which allow the integration of security with the design process. Here we list the state of the art in security requirements engineering and security design

5.1.1 Misuse Cases

Use cases (Jacobson, 1992) have proven helpful for the elicitation of, communication about and documentation of requirements. Some research (Weidenhaupt et al., 1998) also indicates that they may be more successful in capturing the user needs than

textual requirements. On the other hand, because they typically describe functions that the system should be able to perform, use cases are better suited for the so-called functional requirements than they are for the extra-functional ones, such as security.

Recent work has applied use case modelling to requirements analyses other than the purely functional ones. In particular, McDermott (1999, 2001) and Sindre & Opdahl (2000) propose abuse and misuse cases as means to capture and analyse security requirements. McDermott & Fox (1999) define an abuse case as a "specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders in the system". Figure 5-1 (Sindre & Opdahl 2000, p. 2)

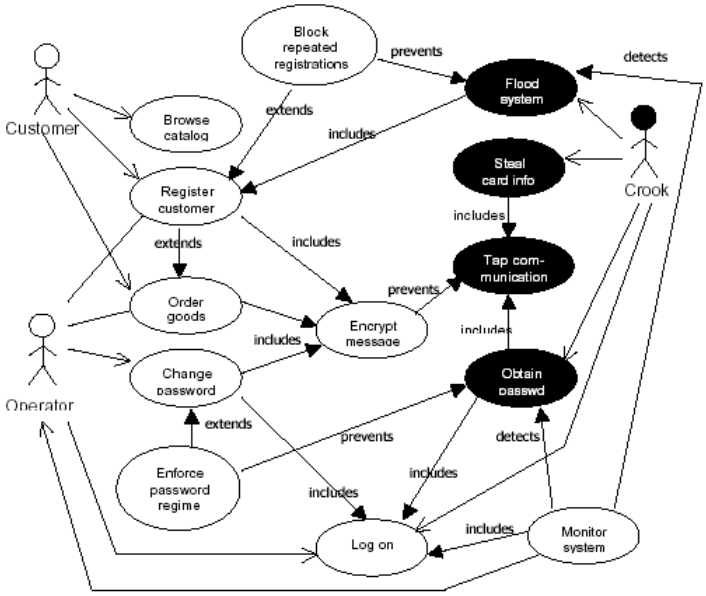

Figure 5-1 Misuse cases

In order to be complete, an abuse case must produce harm. Misuse cases can be described in the same say way as traditional use cases, i.e. using use case diagrams and use case descriptions. With regards to the latter, Sindre & Opdahl (2001) propose a specific template that supports representation of information relevant for security considerations during requirements determination. With regards to misuse cases, Sindre, Firesmith & Opdahl (2003) also propose a reuse-based methodology for misuse case analysis and the subsequent specification of security requirements. Development with reuse involves identifying security assets, setting security goals for

each asset, identifying threats to each goal, analysing risks and determining security requirements, based on reuse of generic threats and requirements from the repository.

5.1.2 CORAS

The EU-funded CORAS project (IST-2000-25031) has developed a new UML profile (Houmb et al., 2002) and a tool[7]-supported model-based methodology for UML-based security (risk) analysis. The profile has also been recently standardized by the Object Management Group (OMG) as part of the "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms". CORAS profile is based on a meta-model, whose classes are mapped to UML modelling elements as stereotypes. It also introduces a number of new icons for representing the stereotypes in UML diagrams.

The metamodel of the profile is divided into six submodels that support different stages of the risk management process. The security assessment starts with identifying the context of the assessment. A strengths, weaknesses, opportunities and threats (SWOT) analysis may be part of this. After the context has been established, the reminder of a risk assessment can be divided into the identification and documentation of unwanted incidents, risks and related treatments. The unwanted incident model is concerned with organising and documenting the threats and vulnerabilities that open for incidents that may harm the system. The modelling of a threat is inspired by already existing work on misuse cases, and uses threat agents (the malicious actor) and threat scenarios. An additional submodel, called *ThreatAgent*, is used to model threats through a number of well-known threat agents, namely human threats, system threats and malicious software. The risk model quantifies unwanted incidents with respect to the reductions of asset value that they may cause. A risk is an unwanted incident that has been assigned consequence and frequency values. Similar risks are categorized into risk themes. The treatment model supports documenting ways of treating the system and quantifying the effect of treatments with respect to reducing the potential harm of risks. Compared to other UML profiles targeting security, i.e. UMLsec (Jurjens, 2002) and Secure UML (Lodderstedt et al., 2002), CORAS UML profile provides specialized support for security risk analysis.

---

[7] The CORAS platform can be downloaded from http://coras.sourceforge.net/ and is distributed under the Lesser GNU Public License (LGPL)

5.1.3 UMLsec

Jan Jürjens (2002) defines a number of extensions that allow expressing security-related information within the diagrams in a UML system specification. The proposed extensions are defined through a number of new s*tereotypes* and *tags*; *constraints* are also defined to give the criteria that determine whether the security requirements are met by the system design. All such extensions are grouped into the UMLsec profile.
With UMLsec Jürjens defines a formal semantics for a simplified fragment of UML, which includes the most common diagram types (class, statechart, sequence, activity and deployment) and the subsystems component (a certain kind of package). Although simplified, the choice of the given subset of UML has proven to be more than adequate for modelling in several industrial case studies, as described in (Jürjens, 2003, 2004) and (Houmb & Jürjens, 2003).

Use case diagrams, along with collaboration and component diagrams are instead not formalized in UMLsec as they are deemed to present information that could already be contained in the chosen subset of UML. For some of the constraints used to define UMLSec extensions, a precise semantic has also been defined.

UMLsec defines a model to specify the types of adversaries of a system and the threats that they can pose. Possible threats envisaged by the UMLsec model are *delete*, *read*, *insert* and *access*. Threats arise from the specification of the physical layer of the system under consideration using deployment diagrams, and are only applicable to links or nodes. The types of possible threats depend on type of link (node), which in turn is defined through a set of stereotypes. UMLsec defines four stereotypes for the link and they are *Internet*, *encrypted*, *LAN*, *wire*; three stereotypes are instead defined for the node and they are *smart card*, *POS device*, *issuer Node*. A node can only be subject to the *access* threat; a link can be subject to the *delete*, *read*, *insert* threats.

Using the defined formalism, each component is associated to an input/output state machine, and communicates with other system components by exchanging messages, through input/output queues. The behaviour of an adversary then is also modelled as a state machine, which can alter in a non-deterministic way the contents of the link queues of the system. The execution of each object is performed in parallel, subject to a possible interference from an attacker. This interference is

determined by the threat scenario arising from the physical environment of the system and is expressed in UML as a deployment diagram.

Based on the defined formalism, a tool has also been developed for the analysis and validation of a model developed with UMLsec. The tool offers several possible degrees of functionality, depending on which stereotype should be verified. First, it is possible to verify security properties included as stereotypes in structure and deployment diagrams. The, it is also possible to validate the security requirements defined on the behaviour level through the statechart and sequence diagrams. Finally it is also possible to validate complex dynamic features; the UMLsec model describing dynamic behaviour is translated into the input language of an analysis tool (such as a temporal logic formula in the case of a model-checker), and thus can be verified against even subtle dynamic properties.

5.1.4 Model-driven security

Model-driven software development is an approach where software systems are defined using models and constructed, at least in part, automatically from these models. A system can be modelled at different levels of abstraction or from different perspectives. The syntax of every model is defined by a metamodel. Many enterprises are implementing service-oriented architecture (SOA) using Web services, and are designing those services according to the principles of Model Driven Architecture (MDA). Because the UML used to express MDA lacks model elements for indicating the security needs of business processes, system architects are forced either to ignore security concerns in their models, or to indicate their intentions in ways that are implementation-specific. The integration of security engineering into a model-driven software development approach has the advantage that the security requirements can be formulated and integrated into system designs at a high level of abstraction. In this way, it becomes possible to develop security aware applications that are designed with the goal of preventing violations of a security policy. Two initiatives are worth mentioning with regards to model-driven security. The first one is called SecureUML and has been developed by Lodderstedt et al. (2002). SecureUML is based on the role-based access control model with additional support for specifying authorization constraints and it can be used in a model-driven software development process to automatically generate complete access control infrastructures. It is based on the RBAC model with additional support for specifying authorization constraints.

SecureUML defines a vocabulary for expressing different aspects of access control, like roles, role permissions and user-role assignments. A permission represents the authorization to a user to execute an operation on one or more protected objects or resources. Users can have one or more roles. A role is a job or function within an organization and combines all privileges needed to fulfil the respective job or function. Privileges are expressed as permissions. SecureUML has been successfully used to implement a prototype generator for the component architecture Enterprise JavaBeans (EJB) (Lodderstedt et al., 2002). Epstein and Sandhu (1999) also propose a UML based notation for access control, which is not intended though for a model-driven software development process.

Still in its infancy is then Johnston's proposed UML profile (Johnston, 2004), which contains yet still a limited number of security-related intent elements, expressed as stereotypes, that business users and software architects can apply to capture business requirements. Specifically, the proposed security intents are: *audit*, *authenticate*, *authorize*, *private*, *signed*, *tamperproof* and *trusted*. It is then assumed that the high-level models are transformed into implementation-specific models through an MDA approach, In particular, Johnston proposes the development of patterns that represent technology-specific implementations for each of the intents in the model. Ideally, one should be able to choose different implementation patterns from a catalogue, based on different characteristics such as performance.

5.2 Methodologies evaluation

When looking at methods for security requirements engineering, we can surely say that they have traditionally not been use case based (Kirby, Archer & Heitmeir, 1999), and looking at misuse and abuse cases, these have yet to be put into large-scale industrial use. Nevertheless, Alexander (2003) claim that they can be applied to elicit security requirements, and to elicit safety requirements from failure cases; they have been successfully used in trade-off requirements analysis (Alexander, 2002). However, given that the success criteria for a misuse case is a successful attack against an application, misuse cases are highly effective ways of analyzing security threats but are inappropriate for the analysis and specification of security requirements.

One of the most pressing shortcomings of current design methodologies is the possibility to model malicious users or adversaries and reflect the threats they pose in

the design model so that we can design secure systems. In this regard UMLsec presents itself as the best candidate for solving the problem, with its capability to model adversaries. On the other hand though, UMLsec alone does not suffice, as it is only a modelling language with extensions for dealing with security requirements and a validation tool for assessing the validity of the design. In other words UMLsec does not provide for a way of feeding threats into the adversary models so that we can then specify and address the right security requirements. As discussed earlier in this work, mobile computing is potentially exposed to different types of threats and in particular to different types of adversaries. Traditional secure design methodologies are based on a static adversary model and therefore are unsuitable for modern mobile computing systems. Achieving security means to be able to cope with such threats and address them at design stage. Another problem with UMLsec, and in fact with the other methodologies examined, is the fact that the same adversary model is applied to the whole design. However, when designing systems in mobile computing, especially complex ones, the adversary models is dynamic and contextually dependant, which then means that the systems security requirements are also contextually dependant. This fact alone has tremendous implications in the system design and must be addressed somehow.

Maintaining that misuse cases are more adequate to analyze and specify threats, Firesmith (2003) proposes instead the use of *security use cases* to specify a system security requirements, as shown in figure 5-2 (Firesmith 2003, p. 54). In a security use case the actor is still the intended user and the success criteria is that application succeeds (not that the misuser succeeds). Still, security use cases are only a piece of the puzzle and do no address secure design and validation.

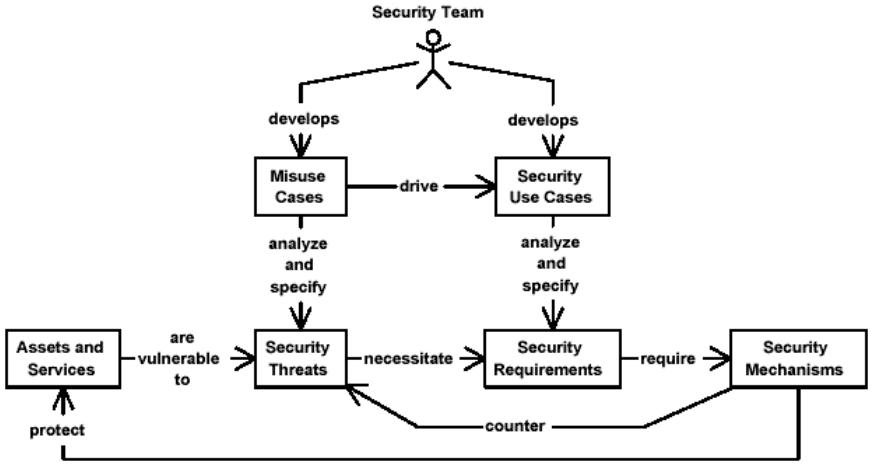

Figure 5-2 Misuse cases vs Security use cases

Out of the solutions evaluated in this work, the one the best of all addresses the modelling of adversaries and related integration in the design process is UMLsec. Not only it allows to model adversaries but also to address the security requirements at design level. However, UMLsec lacks the possibility of modelling threats to the system, which instead can be done by applying misuse cases.

5.3 SDOOM – Secure Design O-O Methodology

Based on the consideration made in the previous section we can better characterize a potential methodology that was suitable for the design of secure systems in ubiquitous computing. Firstly, the methodology would need to address the issue of dealing with different threat models; we proposed a classification in chapter one and we maintain that any other possible classification would include more than just the traditional static model. Second, the methodology would need to address contextual dependencies and the fact that as users move with their devices, the threats to which they are exposed may change. Third, but most important of all for certain aspects, the methodology would need to be able to feed changes in the threat models back into the design in order to identify possible design changes and in particular, to assess whether the system can be still considered secure under the new conditions.

The proposed methodologies is an O-O one and draws upon exiting tools and methodologies. Figure 5-3 shows a diagram model of the methodologies. As we can see, the methodology adopts misuse cases to analyze and specify the possible threats to the system. The novelty here is that the notion of *mobile adversary* and *mobile victim* are forced into the misuse cases as possible misactors. The attentive system designer may autonomously and intuitively includ support for *mobile misactors*. However, the purpose of a design methodology is to address the needs of a more common, and possibly less expert, designers.

Each misactor is modelled using the UMLsec adversary model; for each link or physical node we can define the possible actions or rights of the misactor. In other words for a misactor to succeeds in his misuse case it must perform a certain action, and in order to perform that action it must be able to either *access*, *read*, *insert* or *delete* on the links and nodes defined in our system.

We can then proceed onto designing our secure system using UMLsec to formalize the security requirements that we think the system should address in order to face the threats posed by the various adversaries. The last step of the methodology

involves the use of the UMLsec validation tool to verify that the system we have designed does indeed protect from the identified threats.

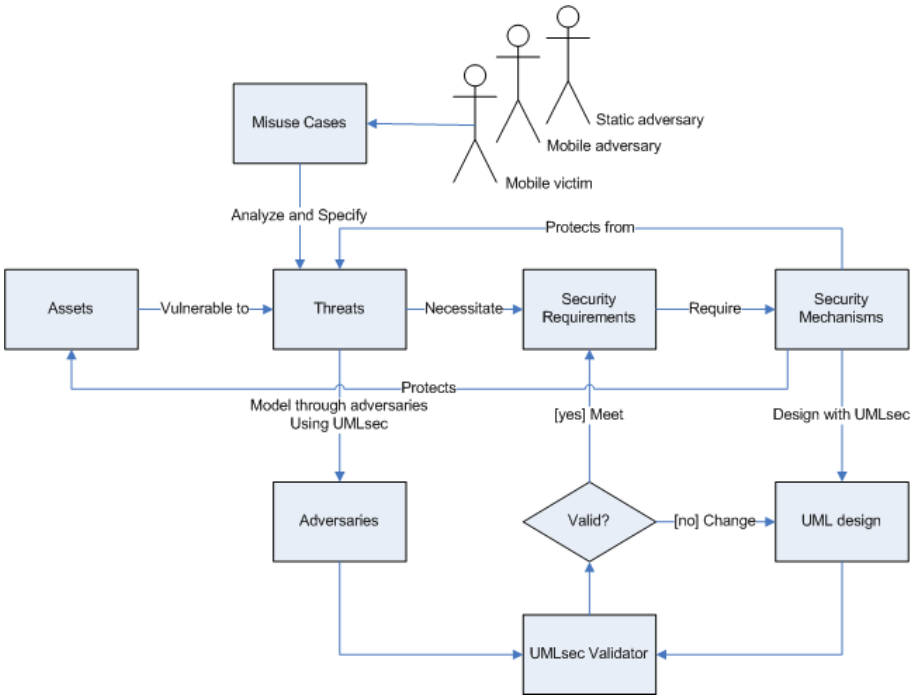

Figure 5-3 Secure design methodology

When performing the design validation, the UMLsec tool reveals possible flaws in the design by listing the threats which are not covered by the current design.

The proposed methodology addresses different types of threat models and feeds them directly into the system design. SDOOM also addresses the third characteristic of a suitable methodology. If the context of application changes or the adversary capabilities change raising new threats, it is quickly possible to verify if the current design still holds and/or identify which parts of the design are now flawed. Similarly, it is also possible to assess the extent to which we can use our systems, which was initially designed to address specific threats. The latter is an extremely valuable capability; as discussed in chapter three, new communication paradigm and application domains are driving the use of legacy technologies and systems beyond the initially intended purpose. Assessing the system security in changing context has become more than ever necessary.

Unfortunately, the methodology shown in figure 5-3 does not address contextual dependency well. This is because the UMLsec validation tool applies the

same adversary model to the whole design, while it would be desirable to apply different threat models to different parts of the system. This is not really a limitation as one could use SDOOM to parts of the system design which are exposed to the same threats. However, that requires an iterative use of SDOOM; improvement to the validation tool, in the way it associates threat models to system design, would greatly improve the usability of SDOOM.

Conclusions

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 can be defined as a course-grained, discoverable software entity that exists as single instance and interacts with applications and other services. 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 technology and application domains, driven by the mobile computing revolution; along with such widespread adoption, the notion of service has evolved too with the related discovery and delivery protocols. Specifically, the definition of service has recently become more encompassing as to include also people, who can also be thought as service providers (e.g. a doctor or a policeman).

Of the different mobile computing paradigms, the nomadic computing one best models the way people work and interact with each other every day. Often called a hybrid model, nomadic computing is characterized by the existence of a service infrastructure, available through fixed nodes interconnected through a permanent network. Given its characteristics, a nomadic computing domain will also contain many communication and service oriented technologies, which a mobile user can have access too. When moving around in a nomadic environment, a user will require the ability to discover and use available services, making integration of existing service oriented protocols a clear requirement. However, the rapid growth and diversification of such service protocols has added further complexity to the integration goal. Thinking about people as possible service providers also changes the perspective on things and security suddenly becomes a stronger requirement to address in a SOA.

In this work the author addresses the existing research gap regarding the security of service oriented architectures and their integration in the context of nomadic computing. Specifically, the research work presented here targets the following questions: 1) what does it mean for a service oriented architecture to be secure? 2) Can existing SOAs support secure service provision as needed by modern application scenarios? And ultimately 3) is it possible to achieve secure service provision in a nomadic environment, characterized by a collection of legacy service technologies and architectures?

In order to answer the first question it was deemed necessary to first address the lack of suitable application scenarios. The literature of ubiquitous computing is full of application scenarios which range from simple "find the nearest printer" scenario to the more futuristic ones where the people involved hardly press any button and everything is transparently and intelligently guessed by the pervasive environment. Therefore two scenarios where introduced, which show the relevance of security requirements when providing services in a nomadic domain.

The state of the art of SOAs has then been thoroughly investigated to understand what secure service provision means for different SOAs and whether an established notion of secure SOA existed. The results of the analysis performed are quite revealing but also consistent with the way security is traditionally addressed. In fact, as for traditional distributed systems, security means different things to different people. This is particularly true for SOAs in ubiquitous computing where not only a common concept of secure provision does not exist, but also security is addressed at different layers of the traditional OSI networking model. The results of the analysis have shown a number of issues that undermine the secure integration required in a nomadic environment. Specifically, SOA differentiate with regards to the access control models, the level of granularity to which access control is applied and way authorization is managed. Also some SOAs have been designed to address scalability more than others, which also has security implications.

Based on the above extensive analysis, a number of security requirements were defined that would allow secure service provision. The requirements address the security of service registration and that of the discovery and delivery phases. Another requirement which is also discussed is that of anonymity. The above SOAs were then evaluated in the light of the defined requirements. From the evaluation it has been possible to make a number of interesting observations. First, the great majority of

SOAs does not address secure service registration and deregistration which means that any entity could register and deregister potentially malicious services. Service delivery seems to have been given a preferential treatment with regards to security, disregarding the in most cases the discovery phase. This is extremely worrying as it raises two problems. First, by eavesdropping the service discovery protocol it is possible to obtain an inventory of available services. Second, insecure service discovery also violates the privacy of service seekers. If one looks at what type of services those technologies were designed to support (e.g printers, projectors, networked appliances etc.), one may also fail to fully appreciate the relevance of these security threats. However, future nomadic application scenarios will envisage a more encompassing definition of service, which includes *people services*. As a result, none of the SOAs examined can be considered suitable for the secure provision of future "enhanced" services. Finally, the evaluation reveals how none of the existing SOAs, made exception for one, addresses security from the point of view of the service definition. In other words security is added on top and achieved through the discovery and delivery protocols independently of service definition. However, security must be addressed from the outset, when defining a service because this in turn drives the design of the whole SOA. If the service definition is not security aware neither will the SOA be security aware.

The second part of this work addresses the research issue of achieving secure service provision in a nomadic computing characterized by a number of heterogeneous service oriented architectures. A solution is presented in the form of an architectural model, called SeNCA. The model is based on the use of information rich descriptors of services and client entities, also known in other contexts as metadata or data about data. In particular, SeNCA proposes a novel three-phase discovery-delivery protocol which allows the enforcement of a number of security requirements, identified earlier in the work. When trying to achieve secure service provision in nomadic computing systems, we must address the integration of legacy technologies and protocols, which deal with security differently and in a way which is insufficient for the needs of future application scenarios. Therefore, the "patches" that need to be applied to each individual technology to make them more security-aware would be different in each case. Security though, is a process and not a patch, and improving the security of existing technologies and protocols would actually need to go through a complete redesign. For this reason, SeNCA proposes a non-invasive approach to

security-enhance legacy protocols, while at the same time providing a secure SOA for the support of future scenarios. The model proposed with SeNCA is also extensible to cater for new SOAs that might need to be integrated in the future nomadic environment. To date, SeNCA is the first architectural model that addresses secure provision of services while also addressing the integration issues with legacy technologies.

Finally, while providing some implementation strategies for the development of the SeNCA architectural model, we also address the issue of how to best design secure SOAs in nomadic computing systems. New communication, collaboration and working paradigms have brought about fundamental changes in the assumption made about systems and their users. This has translated into new threat models, of which we propose a categorization, and a need for reappraisal of current systems to assess how they meet the new changes. Specifically, the security requirements that must be addressed depend on the types of application we run and are contextually dependent on both the physical and organizational (human) environment. In order to address the design of secure SOAs we propose an O-O methodology called SDOOM (Secure Design O-O Methodology). The methodology uses UML extensions for the formalization of security requirements and allows feeding new threat models into the design of secure systems for nomadic computing. The methodology presented can be generalized and applied to the design of secure applications in general, and for the evaluation of existing service oriented architectures.

Bibliography

Adjie-Winoto, W., Schwartz, E., Balakrishnan, H. & Lilley, J. (1999) *The Design and Implementation of an Intentional Naming System*. Operating Systems Review, 34(5):186-301, December 1999

Alexander, I. (2002) Initial Industrial Experience of Misuse Cases in Trade-Off Analysis. *In* Proceedings of IEEE Joint International Requirements Engineering Conference, 9-13 September 2002, Essen, pp. 61-68

Alexander, I (2003) Misuse Cases: Use Cases with Hostile Intent. IEEE Software, Vol. 20, No. 1, pp. 58 – 66

Beresford, A. & Stajano, F. (2003) Location Privacy in Pervasive Computing. *IEEE Pervasive Computing*, 2(1), pp. 46-55

Beresford, A. & Stajano, F. (2004) Mix Zones: User privacy in location-aware services. In *Proceedings of the First International Workshop on Pervasive Computing and Communication Security, March 14, 2004*. IEEE Computer Society Press, pp. 250-253

Bluetooth SIG (2001). Specification of the Bluetooth System - core and profiles v. 1.1, 2001

Bluetooth SIG (2003) Bluetooth Core Specification v1.2, 05 November 2003

Bondavalli, A. et al. (2001) Dependability analysis in the early phases of UML based system design. Journal of Computer Systems Sciences and Engineering, Vol 16, pp. 265-275

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

Burnside et al., (2002a) *Proxy-Based Security Protocols in Networked Mobile Devices*. Proceedings of the Symposium on Applied Computing (SAC'02), March 2002

Burnside et al., (2002b) *Distributed SPKI/SDSI-Based Security for Networks of Devices*. CSG Technical Report, December 2002

Chen, R. & Yeager, W. (2001). Poblano - a distributed trust model for peer-to-peer networks. JXTA Security Project White Paper, 2001 [Online] Available from http://www.jxta.org/docs/trust.pdf [Accessed on 22 May 2004]

Cortellessa, V., Singh, H. & Cukic, B. (2002). Early reliability assessment of UML based software models. In Workshop on Software and Performance, pp. 302-309

Cranor, L., Langheinrich, M. & Marchiori, M. (2002). A P3P preference exchange language 1.0 (APPEL1.0). The World Wide Web Consortium (W3C) [online] Available from http://www.w3.org/TR/P3P-preferences, [Accessed 12th March 2004].

Czerwinski, S.E., Zhao, B.Y., Hodes, T., Joseph, A.D., Katz, R. (1999) *An Architecture for a Secure Service Discovery Service*. Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), Seattle, WA, August 1999

Damiani, E. et al. (2002). Peer to peer networks: A reputation-based approach for choosing reliable resources in peer-to-peer networks. *In* Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, USA, 2002, pp. 207-216

Ellis, C. (2003a) UPnP Security Ceremonies Design Document for UPnP Device Architecture 1.0. Intel Corporation, November 17, 2003

Ellis, C. (2003b) DeviceSecurity:1 Service Template for UPnP Device Architecture 1.0. Intel Corporation, November 17, 2003

English, C., Nixon, P. & Terzis, S. (2002). Dynamic trust models for ubiquitous computing environemnt. *In* Proceedings of the Ubicom 2002

Epstein, P. & Sandhu, R. (1999). Towards a UML based approach to role engineering. *In* Proceedings of the fourth ACM workshop on Role-based access control, pages 135–143. ACM Press, 1999.

Firesmith, D.G. (2003) Security Use Cases. Journal of Object Technology, Vol. 2, No. 3

Gong, L. (2001) Project JXTA: A Technology Overview. Sun Microsystems, Inc., 1 April 25, 2001 [Online] Available from http://www.jxta.org/project/www/docs/TechOverview.pdf [Accessed on 22 May 2004]

Gorlach, A., Heinemann, A. & Terpstra, W (2004). Survey on Location Privacy in Pervasive Computing. Presented at SPPC: Workshop on Security and Privacy in Pervasive Computing, April 20, 2004.

Graziano, A. et al. (2002). Metadata Models for QoS-Aware Information Systems. The Fourteenth International Conference on Software Engineering and Knowledge Engineering, Ischia, ITALY,July 15-19, 2002 (ACM)

Gupta, V. and Gupta, S. (2001). KSSL: Experiments in Wireless Internet Security. Sun Microsystems ed. TR-2001-103, November 2001 [Online] Available from http://research.sun.com/techrep/2001/abstract-103.html [Accessed on 20th April 2003]

Guttman, E., Perkins, C. & Veizades, J. (1999) Service Location Protocol, Version 2. RFC 2608, June 1999

Harbitter, A. & Menascè, D. (2001). The performance of public key enabled kerberos authentication in mobile computing applications. *In* Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8), November 2001

Herzberg, A., Jarekci, S., Krawczyk, H. and Yung, M. (1995). Proactive secret sharing or: How to cope with perpetual leakage. *In* Proceedings of CRYPTO 1995, the 15th Ann. Intl. Cryptology Conference, August 1995, pp. 339 – 352.

Hodes, T., Czerwinski, S.E., Zhao, B.Y., Joseph, A.D., Katz, R.H. (2002) *An Architecture for Secure Wide-Area Service Discovery*. ACM Wireless Networks Journal, special issue. Volume 8, Issue 2/3, March/May 2002, pp. 213-230

Hoffstein, J., Pipher, J. and Silverman, J.H. (1998) NTRU: A Ring-Based Public Key Cryptosystem. Lecture Notes in Computer Science, Vol. 1423, pp 267- 288

Houmb, S.H., den Braber, F., Lund, M.S. & Stølen, K. (2002). Towards a UML profile for model-based risk assessment. In Proc. UML'2002 Satellite Workshop on Critical Systems Development with UML, pages 79-91, Munich University of Technology, 2002.

Houmb, S.H. & Jürjens, J. (2003) Developing Secure Networked Web-Based Systems Using Model-based Risk Assessment and UMLsec. *In* Proceedings of the10th Asia-Pacific Software Engineering Conference (APSEC 2003), Chiangmai (Thailand), 10–12 December 2003

Howes, T., Smith, M. and Dawson, F. (1998) A MIME Content-Type for Directory Information. RFC 2425, September 1998

Huszerl, G. et al. (2002) Quantitative analysis of UML Statechart models of dependable systems. The Computer journal, Vol. 45, num. 3, pp.260-277

ISO (1989). Information processing systems -- Open Systems Interconnection -- Basic Reference Model -- Part 2: Security Architecture

Jacobson, I. et al. (1992) Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992

Jakobsson, M. & Wetzel, S. (2001) Security Weaknesses in Bluetooth. *In* Proceedings of the Cryptographer's Track at the RSA Conference (CT-RSA 2001), LNCS 2020, Springer, 2001

Jansen, W.A. (2003) Authenticating Users on Handheld Devices. *In* Proceedings of the RSA Conference 2003 [Online] Available from http://www.rsaconference.com/rsa2003/europe/tracks/pdfs/implementers_w15_jansen.pdf [Accessed on 4th March, 2004]

Johnston, S. (2004). Modelling security concerns in service-oriented architectures. IBM, 25 June 2004 [Online] Available from http://www-106.ibm.com/developerworks/rational/library/4860.html [Accessed 30th June 2004]

Jøsang, A. & Sanderud, G. (2003). Security in mobile communications: Challenges and opportunities. *In* Proceedings of Australian Information Security Workshop, February 2003

Juels, A., Rivest, R. L. and Szydlo, M. (2003). The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy. *In* V. Atluri, ed. 8th ACM Conference on Computer and Communications Security, 2003. pp. 103-111

Jurjens, J. (2002). UMLsec: Extending UML for secure systems development. LNCS Vol. 2460

Jürjens, J., Popp, G. & Wimmel, G. (2003). Use Case Oriented Development of Security-Critical Systems. Information Security Bulletin, Vol. 8, Feb. 2003, p. 51–56

Jürjens, J.(2004) Developing High-Assurance Secure Systems with UML: A Smartcard-based Purchase Protocol. *In* Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE '04), 25–26 March, 2004, Tampa Florida

Kagal, L. et al. (2002) Centaurus: An Infrastructure for Service Management in Ubiquitous Computing Environments. *In* Wireless Networks, Vol. 8, Num. 6, pp. 619-635, Novemeber 2002. Kluwer Academic Publishers.

Kirby, J. Jr, Archer, M., Heitmeyer, C. (1999) SCR: A Practical Approach to Building a High Assurance COMSEC System. *In* Proceedings of ACSAC ’99, Phoneix AZ, 6-10 December 1999

Kleinrock, L. (1996). Nomadicity: Anytime, Anywhere In A Disconnected World. *In* Mobile Networks and Applications, Vol. 1, No. 4, pp. 351-357

Kleinrock L. (2000). Nomadic Computing and Smart Spaces. *In* IEEE Internet Computing, Vol. 4, No. 1, pp. 52-53

Koblitz, N (1987) Elliptic Curve Cryptosystems. Mathematics of Computation. Vol. 48, num 177, pp.203 – 209.

Langheinrich, M (2002) *A Privacy Awareness System for Ubiquitous Computing Environments*. Proceedings of Ubicomp 2002. Ubiquitous Computing: 4th International Conference, pp. 237-245, Springer-Verlag LNCS 2498, September 2002

Lee, C. & Helal, S. (2002). Protocols for service discovery in dynamic and mobile networks. International Journal of Computer Research, Vol. 11, No. 1, pp. 1–12, 2002

Lodderstedt, T., Basin, D. & Doser, J (2002). SecureUML: A UML-based modeling language for model-driven security. LNCS, Vol. 2460

Mascolo, C., Capra, L. and Emmerich, W. (2002) Middleware for Mobile Computing (A Survey). *In* Advanced Lectures in Networking. Ed. E. Gregori, G. Anastasi, S. Basagni. Springer. LNCS 2497. 2002

Mascolo, C., Capra, L. and Emmerich, W. (2004). Principles of Mobile Computing Middlewar. *In* Middleware for Communications, Wiley Ed. 2004.

McDermott, J. (1999) Abuse Case Models for Security Requirements Analysis. Proceedings of the 15th Annual Computer Security Applications Conference, p.55, December 06-10, 1999

McDermott, J. & Fox, C. (1999) Using Abuse Case Models for Security Requirements Analysis. *In* Proceedings of the15th IEEE Annual Computer Security Applications Conference, Phoenix, Arizona, December 06 - 10, 1999

McDermott, J. (2001). Abuse-Case-Based Assurance Arguments. *In* 17th Annual Computer Security Applications Conference, December 10-14, 2001

Microsoft Corporation (2000) *Universal Plug and Play Device Architecture. Version 1.0, 08 Jun 2000* [online] Available from http://www.upnp.org/download/UPnPDA10_20000613.htm [Accessed 18th April 2004]

Moore, G. (1965) Cramming more components onto integrated circuits. Intel Corporation [Online] Available from ftp://download.intel.com/research/silicon/moorespaper.pdf [Accessed 20th May 2004]

Muller T. (1999) Bluetooth Security Architecture: Version 1.0, Bluetooth White Paper [online] Available from http://www.bluetooth.com [Accessed May 10th 2004]

Myles, G., Friday, A. & Davies, N. (2003) *Preserving Privacy in Environments with Location-Based Applications*. IEEE Pervasive Computing, Vol. 2, Num. 1, January-March 2003, pgs. 56-64

OMG (2002) Common Object Request Broker Architecture: Core Specification. Ed. The Object Management Group, Inc. (OMG), Version 3.0, December 2002

OMG (2001) Security Service Specification version. Ed. The Object Management Group, Inc. (OMG), Version 1.7, March 2001

OMG (2002) Security Interoperability. Chapter 24 of Common Object Request Broker Architecture: Core Specification. Ed. The Object Management Group, Inc. (OMG), December 2002

Ostrovsky, R. & Yung, M. (1991) How to withstand mobile virus attacks. *In* Proceedings of the 10th ACM Symposium on Principles of Distributed Computing, 1991, pp. 51-59.

Paulson, L. D. (2003) Will Fuel Cells Replace Batteries in Mobile Devices? *In* IEEE Computer, Vol. 36, No. 11, pp. 10-12

Pfitzmann, A. & Köhntopp, M. (2000). *Anonymity, Unobservability and Pseudonymity—A Proposal for Terminology*. Designing Privacy Enhancing Technologies: Proc. Int'l Workshop Design Issues in Anonymity and Observability, LNCS, vol. 2009, Springer-Verlag, Berlin, 2000, pp. 1–9

Priyantha, N. B., Chakraborty, A. and Balakrishnan, H. (2000). The Cricket Location-Support System. *In* Mobile Computing and Networking, Vol. 2 num 3, pp. 32 – 43

Raatikainen, K., Christensen, H.B. & Nakajima, T. (2002) Application Requirements for Middleware for Mobile and Pervasive Systems. *In* ACM SIGMOBILE Mobile Computing and Communications Review, Vol.6, No. 4 pp. 16 – 24

Rakotonirainy, A. & Groves, G. (2002) 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

Ravi, S., Raghunathan, A.& Potlapally, N. (2002). Securing wireless data: System architecture challenges. *In* Proceedings of International Symposium on System Synthesis (ISSS 2002), October 2002

Raman, S et al. (2003) *Access-Controlled Resource Discovery for Pervasive Networks*. Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), March 9-12, 2003, Melbourne, FL, USA

Salutation Consortium (1999) Salutation Architecture Specification (part-1), Version 2.0c, June 1, 1999 [online] Available from http://www.salutation.org/ [Accessed on May 20th 2004]

Salutation Consortium (1999b) Salutation-Lite, Find-And-Bind™ Technologies For Mobile Devices, Salutation White Paper [online] Available from http://www.salutation.org/ [Accessed on May 20th 2004]

Sindre, G., Firesmith, D. & Opdahl, A. (2003) A Reuse-Based Approach to Determining Security Requirements. *In* Proc. 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03), Klagenfurt/Velden, Austria, 16-17 Jun 2003

Sindre, G. & Opdahl, A.L. (2000). Eliciting Security Requirements by Misuse Cases. *In* Proceedings of TOOLS Pacific 2000, pp 120-131, 20-23 November 2000

Sindre, G. & Opdahl, A.L. (2001) Capturing Security Requirements through Misuse Cases. *In* Proceedings of Norsk informatikkonferanse - NIK'2001. Trondheim, Norway: Tapir; 2001. s. 219-230

Stallings, W. (2003) Network Security Essentials. Ed. Prentice Hall, ISBN: 0-13-016093-8

Subramonian, V. and Gill, C. (2002) Just Enough Middleware. 2nd TAO Workshop, Arlington, VA, USA, July 19, 2002

Sun Microsystems (2003) Jini Architecture Specifications, Version 2.0, June 2003 [Online] Available from http://wwws.sun.com/software/jini/specs/jini2_0.pdf [last accessed on July 23rd 2004]

Tosun, A.S. and Feng, W. (2001). Lightweight Security Mechanisms for Wireless Video Transmission. *In* Proceedings of International Conference on Information Technology and Computing, April 2001, pp. 157-161

Undercoffer, J. et al. (2003) A Secure Infrastructure for Service Discovery and Access in Pervasive Computing. *In* Mobile Networks and Applications Vol. 8, num. 2 pp.113–125. Kluwer Academic Publishers

Veizades, J., Guttman, E., Perkins, C. & Kaplan S. (1997) Service Location Protocol. RFC 2165, June 1997

Vettorello, M. et al. (2001) Some Notes on Security in the Service Location Protocol Version 2 (SLPv2). *In* Proceedings of the Workshop on Ad Hoc Communications, held in conjunction with the 7th European Conference on Computer Supported Cooperative Work (ECSCW'01), Bonn, Germany, September 2001

Want, R., Hopper, A., Falcão, V. and Gibbons, J. (1992). The Active Badge Location System. *In* ACM Transactions on Information Systems, Vol. 10, num. 1, pp. 91-102.

Welsh, M. (1999) NinjaRMI: A Free Java RMI. Paper [online] Available from http://www.eecs.harvard.edu/~mdw/proj/old/ninja/ninjarmi.html [Accessed on May 22th 2004]

Weidenhaupt, K., Pohl, K., Jarke, M. &Haumer, P. (1998) Scenario Usage in System

Development: A Report on Current Practice. IEEE Software, Vol. 15, No 2, pp. 34-45

Weiner, M. (1998). Performance Comparison of Public Key Cryptosystems. RSA Laboratories' CryptoBytes, Vol. 4, num. 1

Weiser, M. (1999). Some computer science issues in ubiquitous computing. Mob. Comput. Commun. Rev. Vol. 3, num. 12.

Westin, A. (1967). Privacy and Freedom, New York: Atheneum

Whitten, A. & Tygar, J. (1999). Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. *In* Proceedings of the 8th USENIX Security Symposium

World Wide Web Consortium (W3C) (2004). Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0 W3C Recommendation, 15 January 2004 [Online] Available from http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/ [Accessed on 20th July 2004]

Zhang, K. & Kindberg, T. (2002). An authorization infrastructure for nomadic computing. *In* Proceedings of the seventh ACM symposium on Access control models and technologies, pages 107–113, 2002

Zhu, F., Mutka, M., Ni, L. (2003) *Splendor: A Secure, Private, and Location-aware Service Discovery Protocol Supporting Mobile Services*. Proceedings of the 2003 IEEE Annual Conference on Pervasive Computing and Communications (Percom 2003), March 2003

Zhu, F., Mutka, M. & Ni, L. (2004) Facilitating Secure Ad hoc Service Discovery in Public Environments. Journal of Systems and Software (in press)