SECURE AGENT-BASED MANAGEMENT FOR PERVASIVE ENVIRONMENTS

G. Navarro, J.A. Ortega-Ruiz, J. García and S. Robles

Dept. of Information and Communications Engineering,
Univeristat Autònoma de Barcelona,
08193 Bellaterra (Spain)
{gnavarro,jao,jgarcia,srobles}@ccd.uab.es

ABSTRACT

A typical pervasive computing scenario may consists of a wide range of devices interconnected through ad-hoc networks. One of the problems that pervasive computing introduces is the management and interaction between these devices, as well as the security implications of this management. We present in this paper an architecture, which provides initial mechanisms to solve these problems. Inspired by trust management systems, our architecture is built on top of a multi-agent system. However, our proposal is sufficiently open to allow the integration with other technologies. Results of our work can be easily applied to existing pervasive computing software and associated technologies.

1. INTRODUCTION

The current development of computer systems are leading to a situation where the number of processors and computer networks is becoming more and more pervasive. Nowadays, there are processors embedded in lots of everyday devices. From personal computers, laptops, PDAs, and mobile phones, to refrigerators, heaters, coffee machines, or toasters. Furthermore, these devices can be interconnected through computer networks. The increased research on wireless and mobile networks is making possible to have cheap networks at home, at the office or even at the streets.

One of the problems that pervasive computing introduce is the management of all those devices interacting one with another [10], and the security implications of this management. A desired property of pervasive computing systems is self-management. Self-management is the ability for those systems to manage themselves with a minimum human intervention. For example, to tune and set up the configuration to make the system work optimally, to adapt the system to changing workloads, to detect potential attacks against the system itself, etc.

This work has been partially funded by the Spanish Government Commission CICYT, through its grant TIC2003-02041.

This is one of the reasons why multi-agent systems are becoming very popular in pervasive computing. A software agent is an autonomous entity that can interact and perceive the context of its own execution. Hence, it is a clear candidate to build self-managing systems in pervasive computing. A problem of multi-agent systems is that sometimes they present too much complexity for embedded devices. Most of the mechanisms used, for instance, to make up coalitions in agent systems, are quite complex and may not be suitable for some constrained environments.

In this paper we present an architecture for pervasive computing, which provides some basic mechanisms to introduce multi-agent systems in pervasive environments. It is a trust management inspired system, which provides a simplistic approach to allow the management of pervasive computing systems. The main idea is to be build on top of multi-agent systems, but it is enough open to allow the integration with other technologies.

The architecture we present is called SAMAP, *Secure Agent-based MAnagement for Pervasive computing*. It is originally based on a mobile agents platform, although as we will show it is not restricted to mobile agents or agents in general. The system has been inspired by some other work on pervasive computing such as [11, 7, 3]. Nevertheless, we think that SAMAP provides a novel approach to deal with agent systems in pervasive computing, in a very simple way.

Section 2 provides an overview of SAMAP. In Section 3 we present the main entities that conform the system, their role, and the naming schema used in SAMAP. Section 4 introduces the management protocols. Finally, Section 5 summarizes our conclusions and further work.

2. ARCHITECTURE OVERVIEW

Although SAMAP does not require any specific base technology, we have built a prototype in top of JADE (*Java Agent DEvelopment Environment*)[12]. JADE is a popular open source multi-agent platform implemented in Java and supports the FIPA (Foundation for Intelligent Physical Agents) specifications[6]. Although, JADE does not sup-

port agent mobility we have extended it to support it [1]. We have also provided several security mechanisms for mobile agents[9], including resource access control, itinerary protection, ...

Another reason for the election of JADE is the possibility of using JADE-LEAP (JADE Lightweight Extensible Agent Platform). A platform based on JADE, which is intended to run on top of J2ME MIDP (Java 2 Micro Edition, Mobile Information Device Profile) or PersonalJava for small devices such as PDAs or cell phones. This platform is provided as an add-on to JADE.

An important feature of SAMAP is the communication procedures. We rely in the FIPA Agent Communication Language (ACL), which is highly used and supported in multi-agent systems. It can be extended or combined with lots other languages such as RDF, FIPA Semantic Language (SL), KIF (Knowledge Interchange Format), DAML (Darpa Agent Markup Language), or the current OWL (Web Ontology Language). There are also bindings to use FIPA ACL on top of most common network protocols such as HTTP, IIOP, SMTP, or TFTP.

In general, we consider an *entity* or *SAMAP-entity* any software entity with autonomy to run by itself, or in other words with its own thread of execution. Although most times a SAMAP entity will be a software agent, we have not restricted a SAMAP entity to be a software agent because we think it is too restrictive. For example, one can think of a simple temperature sensor which runs a very simple daemon to answer queries of the current temperature. Section 3 provides a description of the main SAMAP entities.

SAMAP provides a simple naming system, which is somehow a simplified local name system based on the SPKI/-SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure). This naming system is a good base to allow the creation of groups that can be self-managed by agents.

It also provides some basic protocols for self-management of entities. This protocols can be seen as very basic step towards the management of pervasive computing environments. They are based on the idea that entities can hold other entities to coordinate tasks

3. SAMAP ENTITIES

We classify these entities in four categories: Simple Entity, Control Station, Directory Data Station, and Environment Data Station. This categories can be seen as *roles*, an entity can act in one or more of these roles at the same time.

Simple Entity (SE) This is the most simple and basic entity. The minimum requirements for an SE is to be able to communicate with other entities. Although not required, an SE can be a mobile entity, such as a mobile agent, or generic mobile code with enough autonomy to be considered an entity.

Control Station (CS) The Control Station, is an entity able to manage and control other entities. A CS will normally need enough processing resources to perform cryptographic operations. They can be autonomous or interactively controlled by a user or another entity. A CS can own other entities and use them.

Directory Data Station (DDS) The DDS is an entity that stores directory information. It can be seen as a very simplified UDDI or DNS server for SAMAP. The DDS maintains a directory listing services and the entities associated to the services, as well as single entities and groups of entities. It can be consulted by other entities to find services and other entities location.

Environment Data Station (EDS) An Environment DataStation is an entity which provides context information. Such entities can be sensor devices providing information from the environment, such as location information used to track users on a room, temperature, movement, ...

It is important to note that there may be some possible constraints on the mobility of entities due to secure requirements. Although entities will normally run in a controlled environment, this may not be case for all the situations. For instance, if we consider a mobile agent environment, where there is not a complete trust relation in all the entities, security operations may be very dangerous. A first approach is to impose the constraint to a CS to be a static entity, which will run always in a trusted platform. Obviously this restriction can be bypassed when there are no security threads or entities are running in a closed environment.

An important notion in SAMAP, is *ownership*. Each non-CS entity has a CS *owner* associated to it (a CS can or cannot have owner). The ownership is an static and immutable property of the entities. It makes reference to the origin of the entity, which will normally be the creator of the specific application or service supplier. The owner of an entity is the main responsible for the entity. If an entity misbehaves or produces some erratic execution due to bugs, the owner can be made responsible for it. The owner has also to take care of the execution of its entities, ensuring that an idle entity does not run forever idle, providing a potential denial of service. This is accomplished by a simple heart-breath protocol, where the owner can get the status of its entities every given period of time.

Beside the owner, there is the *holder*. Each entity can have one or several holders, or none if it is idle. A holder is a CS which is using the entity for an specific application or service and normally for a temporary period of time.

3.1. SAMAP Identity

In SAMAP, each entity has to be uniquely identified. The identity is a very important concept in the SAMAP framework, since it is the base for all the trust relations and interactions.

We use a cryptographic approach to provide security related features, including trust management, to SAMAP. In SAMAP each entity has a pair of cryptographic keys (a public key and a private key), and the public key acts as an identifier of the entity. Given the properties of common public key algorithms we can assume public key uniqueness in practice. This is a common approach of trust management systems such as SPKI/SDSI [4] or KeyNote [2].

In order to simplify the management of public keys, we define an SAMAP identifier, denoted as *SID* as the *hash* of a public key. In our current implementation we use a MD5 functions, which gives a 128 bits SID. As a side effect it results compatible in size with the 128 bits UUID (*Universal Unique Identifier*) identifier [8], which is widely used in distributed environments.

The SAMAP identity is complemented with one or more optional *resolvers*. Resolvers provide network addresses to locate the entity or to locate some directory entity, which can resolve the SID to a network address. It is important to note that in most cases, resolvers won't be used, given the pervasive environment and characteristics of the applications of SAMAP, the most common way to locate an entity is through service discovery services provided by Directory Data Stations.

3.2. SAMAP Entity and Service Discovery

In order to provide a fully distributed environment, we relay on current service discovery algorithms to locate not only services but also concrete entities. Since we base our system in a multi agents system, we have opted for adapting the current specification for agent discovery services from FIPA [5].

The FIPA Agent Discovery Service provides a simple agent discovery service, with its associated ontology. We have used the same ontology, but we have adapted the specification to SAMAP. In our case the discovery service is provided by Directory Data Stations. Each DDS can perform service and entity location by using a simplified gnutella-like protocol. An we can *discovery* not only agents but services (in an UDDI fashion), and individual entities. Also, entities can advertise themselves to DDSs.

For example we can have some SAMAP-entities, which can control the ambient conditions of the rooms when we enter in a building. They adjust the lights, the temperature, the humidity, etc. of the room we enter in. We can get one of these entities to work for us by using our personal Control Station to control it, but first of all we have to find it. We can use a DDS to ask for the *ambient adjuster* service, which will return to us the identity and location of entities that we can use for that purpose.

Summarizing, DDSs offer a discovery service to other entities to look for entities, groups, or services:

- *SAMAP-entity-discovery*: returns the location (network address) of a given entity if the parameter is a SID.
- *SAMAP-service-discovery*: returns a list of entities that can offer a given service.

4. SAMAP MANAGEMENT PROTOCOLS

SAMAP provides some simple protocols to manage entities, based on authorization or trust management. The main idea is to provide protocols as simple as possible, that can be extended to support more complex interactions. Here we describe the main basic protocols, which deal mainly with the management of the SAMAP entities, and more precisely with the *possession* of entities, that is, how to become a *holder* or other entities, and related actions. These protocols are normally specified through simple FIPA ontologies.

4.1. Take Possession

The `take-possession` protocol allows a CS to become the *holder* of another entity. This is achieved in a two step protocol where both entities interchange their public keys.

1. The CS sends a *possession-request* together with its public key, and an ontology corresponding the service or task requested to the entity, say SID_2.
2. SID_2 replays with a *possession-granted* with its public key if it accepts the possession. This normally means that the entity is in an *idle* or *engaged* state.
3. If the possession is not accepted, the entity responds with a *possession-denied* message, which optionally may include the reason. The reason to reject a possession will normally be due to the entity being in a *busy* or *transient* state, or because the entity cannot handle the ontology required by the *holder*.

4.2. Delegate Possession

A CS can delegate the possession of an entity to another CS. This is very useful in situations where there are complex interactions between several CSs and entities. CSs can exchange their entities.

1. CS_1 sends a *delegation-request* to SID_1 providing the identifier CS_2. Note that SID_1 cannot deny the delegation, this is because the operation is fired by its

(or one of its) holder. Even if SID_1 is busy it has to accept the delegation. After the delegation SID_1 can revoke the possession of CS_2 if needed.

2. SID_1 contacts CS_2 with a *possession-notify*, which warns CS_2 that it has become its holder. The message includes SID_1 public key.

3. CS_2 can accept or deny the delegation. If accepted, it sends a *possession-accepted* message together with its public key, if not, it sends a *possession-denied*.

4. Finally, SID_1 notifies CS_1 the result of the possession notification. If CS_2 does not accept the possession, CS_1 continues to be the holder of SID_1.

4.3. Terminate and Revoke Possession

Since the possession of an entity is required and initiated by a CS it has to be terminated by the same CS. This terminations is done by a simple *possession-termination* notification message from the CS to the entity. Nevertheless, there are some situations where the entity may initiate the termination of the possession. This situations does not correspond to the normal operation between the holder and the entity, thus we refer to them as *revocation* of possession. The revocation can occur because the entity is detecting a malfunction, has to stop doing its tasks, is going to be stopped (shutdown, killed, ...), or by direct indication of the owner.

The protocol is also a simple notification message, *possession-revoked*, which may contain the reason.

5. CONCLUSIONS

In this paper we have presented an overview of the *Secure Agent-based MAnagement for Pervasive computing* (SAMAP) system. SAMAP provides a trust management inspired architecture to support multi-agent systems in pervasive computing. The main idea behind SAMAP is to provide a very simple framework that can be adapted and used in a wide range of applications and environments.

SAMAP provides support for several roles interacting in a pervasive environment. A key issue of SAMAP is the use of a naming schema that provides considerable benefits. It allows the easy creation of groups, in a very scalable and robust way. On the other hand it also simplifies the interactions between entities, and the protocols can be safely simplified. The naming schema identifies each entity as its public key gaining all the advantages that this kind of naming schemas (used by some trust management systems) have.

We have also designed some protocols for the management of entities. A required characteristic of pervasive computing is self-management. These protocols, given its simplicity and robustness allows the management of entities by other entities, in an easy and efficient way.

Currently we have a prototype implementation of SAMAP, which runs on top of a modified version of JADE (a popular open source multi-agent platform) supporting mobility and several security features. And we are working towards other platforms and languages.

6. REFERENCES

[1] J. Ametller, S. Robles, and J. Borrell. Agent migration over fipa acl messages. In *Mobile Agents for Telecommunication Applications (MATA)*, 2003.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System. RFC 2704, IETF, September 1999.

[3] C. Ellison. UPnP Security Ceremonies Design Document v.1.0. UPnP Forum, October 2003.

[4] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, IETF, September 1999.

[5] FIPA. Fipa agent discovery service specification. Foundation for Intelligent Physical Agents (FIPA), TC Ad Hoc Preliminary Specification, August 2003.

[6] FIPA. FIPA Specifications. Foundation for Intelligent Physical Agents (FIPA) `http://www.fipa.org`, 2003.

[7] L. Kagal, T. Finin, and A. Joshi. Trust-based security in pervasive computing environments. *Computer*, 34(12), 2001.

[8] P. Leach, M. Mealling, and R. Salz. A UUID URN Namespace. Interet draft: `draft-mealling-uuid-urn-03.txt`, January 2004.

[9] G. Navarro, S. Robles, and J. Borrell. Role-based access control for e-commerce sea-of-data applications. In *Information Security Conference 2002*, 2002.

[10] M. Sloman. Will pervasive computers be manageable? Keynote Talk HP OpenView 2001, New Orleans, June 2001.

[11] Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols*, 2000.

[12] TILAB. Java Agent DEvelopment Framework: JADE. Telecom Italia Lab (TILAB) `http://jade.tilab.com/`.