A Secure and Auto-configurable Environment for Mobile Agents in Ubiquitous Computing Scenarios*

Javier López, Antonio Maña, and Antonio Muñoz

GISUM group, Computer Science Department, E.T.S.I. Informática
University of Malaga, Spain
{jlm, amg, amunoz}@lcc.uma.es

Abstract. The increased heterogeneity and dynamism of new computing paradigms and especially of ubiquitous computing models is boosting the need for auto-configurable systems. In these new scenarios, heterogeneity and dynamism are inherent properties and applications are built by aggregating distributed information and services that are not under the control of a single entity. The current trend towards distributed computing poses important problems related to the transmission of large amounts of data between the nodes of the computing system; the control over the information; and the flexibility to adapt to heterogeneous client requirements. These characteristics, difficult to manage by traditional computing models, are making the mobile agent paradigm to gain momentum and increasing the interest of researchers and industry in this paradigm. In this paper we present a solution to provide a secure and auto-configurable environment for mobile agents in ubiquitous computing scenarios, based on two main building blocks: trusted platforms and profiles.

Keywords: Security, Agents, Profiles, Trusted Computing, Ubiquitous computing.

1 Introduction

Personalization and ubiquity are key properties for on-line services, but at the same time, they challenge the development of these systems because of the complexity of the required architectures and the security concerns they introduce. In particular, the current infrastructures for the development of personalized ubiquitous services are not flexible enough to accommodate the configuration requirements of the various application domains. To address such issues, highly configurable infrastructures are needed. An intimate relationship exists between auto-configurable systems and ubiquitous environments due to the nature of these environments, in which a device interacts with the context and adapts itself to it, performing auto-configuration.

In these new scenarios, heterogeneity and dynamism are inherent properties and applications are built by aggregating distributed information and services that are not under the control of a single entity. Furthermore, the current trend towards distributed

* Work partially supported by E.U. through projects Ubisec (IST-506926) and SERENITY (IST-027587) and by Spanish Ministry of Science and Education through research grant PR2005-0175.

J. Ma et al. (Eds.): UIC 2006, LNCS 4159, pp. 977 – 987, 2006.

computing poses important problems related to the need to transmit large amounts of data between the distributed nodes of the computing system; the control over the information; and the flexibility to adapt to heterogeneous client requirements. These characteristics are difficult to manage by traditional computing models. For these reasons, the mobile agent paradigm is gaining momentum and the interest of researchers and industry in this paradigm is increasing.

The mobile agent paradigm uses the network to carry software objects that are to be executed in the sites of service providers. A client orchestrates the work of a server by sending to the server an agent that is responsible for performing all of the required actions related to the services and data offered by the server. For instance, consider the case of a digital library server offering several Tb. of information. In order to allow the remote processing of this information we can find different schemes:

- *The server offers only basic services in order to access the data*. In this case clients need to download the data and to process it locally. This is the scheme of many traditional client-server systems. This scheme reduces the processing power required in the server, but has three main drawbacks: (i) the system is very inefficient due to the overhead caused by data transmission and redundant storage; (ii) the server looses control over the information; and (iii) in cases of very dynamic information sources, the transmission delays can cause problems of synchronization between the copies of the data in the client and the server. This is especially true when clients can modify the data because, in this case, there are important problems caused by concurrent access by many clients.
- *The server offers advanced information processing services*. This scheme solves the problem of data transmission and redundancy, as the data is processed locally by the server. However, the flexibility of the system is limited, because it is very difficult for the server to foresee all processed that may become necessary for the clients. The main problems arise when clients need to access many heterogeneous information servers, because it is difficult that all servers offer the same services with the same access mechanisms. This is the approach followed in service-oriented computing. Clients need to download only the partial results produced by the server, in order to compute locally the final results. For this reason, the control of the server has more over the information than the previous case, although not complete.
- *The server offers the possibility of sending agents to perform the data processing*. In this case, the server offers only basic access services, which are easier to manage from the points of view of efficiency and security. The flexibility is very good because agents are then responsible for implementing the complex data processing required by the client. Only results need to be sent to the client. The control of the server over the information is complete.

The agent paradigm can represent a valuable model for the interaction of applications and devices in ubiquitous environments. Each element in the ubiquitous scenario can act both as client (sending agents to other elements) and as server (allowing other elements to send agents to it).

The main motivation of our work is to define a secure and adaptable execution environment for mobile agents based on the use of profiles and taking advantage of the new Trusted Computing architectures. We define a profile as a repository

materialized as a structured data object, containing properties and features, as well as present and past status about an entity, in our case an agent. Profiles are normally used to convey the properties of an entity to other entities, not being intended as information storage for internal use by the entity itself. Therefore, profiles are conceived with the objective of being shared with others. Our approach includes mechanisms for the secure management of profiles, which are key elements for the auto-configuration of the hosts for the execution of the mobile agents.

We use mechanisms for guaranteeing that execution environments (a.k.a. agencies) provided by the hosts are trustworthy. This is achieved by using the mechanisms provided by hardware devices known as Trusted Platform Modules (TPMs). TPMs are the core elements of the secure computing model defined by the Trusted Computing Group (TCG) [1]. We also use the remote attestation capability provided by the TCG model. In the combination of these two central elements, we highlight the intrinsic auto-configurability of agent systems and we enhance it by using profiles.

The rest of the paper is structured as follows. Section 2 provides an overview of relevant related work. Section 3 describes the proposed solutions for achieving our objectives. Finally, section 4 presents conclusions and describes some ongoing work.

2 Background and Related Work

The growing interest in ubiquity, dynamic adaptability, profiling and auto-configuration is evident in the literature [2-5]. In the emerging computing paradigms, this interest is augmented by the need to provide dynamic autonomous responses to ever-changing contexts and unforeseen interactions.

Profiles have been used for capturing information about users and preferences. Some authors describe profiling as "the process of inferring a set of characteristics (typically behavioural) about an entity and then treating that entity (or other entities) in the light of these characteristics" [6]. Based on this definition Pearson describes a method based on the use of trusted agents for self-profiling in e-commerce scenarios by which customers can have greater control over their related profiles [7]. However, the objective of our profiles is to inform agencies about the needs of the agents and at the same time to provide tools for agencies to control and monitor the behaviour of agents. The real value of profiles depends on the accuracy of the information they contain. Therefore, the protection of these profiles is an important aspect to consider.

Regarding the processing and description of profiles, RDF [8] provides a way to define a generic data model so facilitating a multi purpose mechanism to describe resources. Another approach, CC/PP [9] proposes a framework for the management of information about devices capabilities and user preferences. This framework, based on the RDF approach, has proved useful for content customization. UAProf [10] provides a solution to define a specific vocabulary concerning device information. Finally, FIPA [11] defines device ontologies for the communication of devices.

Many proposals use profiles to characterize users' behaviour while they browse through the Internet [12]. The information obtained is used to construct a transient navigation profile, which is useful to anticipate future actions of the user. Reference [13] follows the same approach, using Bayesian networks as a tool for creating profiles of visitors in a museum in order to customize the information.

2.1 Security in Agent-Based Systems

The inherent complexity of information security is increased in agent-based ubiquitous systems. In fact, securing these systems requires protecting any element from every other. Some of the general software protection mechanisms can be applied to the protection of agents. However, the specific characteristics of agents mandate the use of tailored solutions. First, agents are most frequently executed in potentially malicious platforms. Then, from the point of view of platforms, agents are potentially malicious pieces of software. Therefore, we can not simplify the problem as is done in other scenarios by assuming that some elements of the system can be trusted.

Then, the security of an agent system can be defined in terms of many different properties such as confidentiality, non repudiation, etc. but it always depends on ensuring the correct execution of the agent on agent servers (a.k.a. agencies) within the context of the global environments provided by the servers [14].

Finally, conflict management, communication, intelligence and negotiation are important components of collaborative multi-agent activity. Thus, a collaborative agent must be able to handle situations in which conflicts arise and must be capable of negotiating with other agents in order to fulfil its goals. These capabilities are especially relevant for the security of the agent.

Several mechanisms for secure execution of agents have been proposed in the literature with the objective of securing the execution of agents. Most of these mechanisms are designed to provide some type of protection or some specific security property. In this section we will focus on solutions that are specifically tailored or especially well-suited for agent scenarios. More extensive reviews of the state of the art in general issues of software protection can be found in [15, 16].

Some protection mechanisms are oriented to the *protection of the host system against malicious agents*. Among these, SandBoxing is a popular technique that is based on the creation of a secure execution environment for non trusted software. In the agent world a sandbox is a container that limits, or reduces, the level of access its agents have and provides mechanisms to control the interaction among them.

Another technique, called proof-carrying code, is a general mechanism for verifying that the agent code can be executed in the host system in a secure way [17]. For this purpose, every code fragment includes a detailed proof that can be used to determine whether the security policy of the host is satisfied by the agent. Therefore, hosts just need to verify that the proof is correct (i.e. it corresponds to the code) and that it is compatible with the local security policy. In a variant of this technique, called proof-referencing code, the agents do not contain the proof, but just a reference to it [18]. These techniques share some similarities with the constraint programming technique; they are based on explicitly declaring what operations the software can or can not perform. One of the most important problems of these techniques is the difficulty of identifying which operations (or sequences of them) can be permitted without compromising the local security policy.

Other mechanisms are oriented towards *protecting agents against malicious servers*. Sanctuaries [19] are execution environments where a mobile agent can be securely executed. Most of these proposals are built with the assumption that the platform where the sanctuary is implemented is secure. Unfortunately, this assumption is not applicable in our scenario. Several techniques can be applied to an

agent in order to verify self-integrity in order to avoid that the code or the data of the agent is inadvertently manipulated. Anti-tamper techniques, such as encryption, checksumming, anti-debugging, anti-emulation and some others [20, 21] share the same goal, but they are also oriented towards the prevention of the analysis of the function that the agent implements. Additionally, some protection schemes are based on self-modifying code, and code obfuscation [22]. In agent systems, these techniques exploit the reduced execution time of the agent in each platform.

Software watermarking techniques [23, 16] are also interesting. In this case the purpose of protection is not to avoid the analysis or modification but to enable the detection of such modification. The relation between all these techniques is strong. In fact, it has been demonstrated that neither perfect obfuscation nor perfect watermark exists [24]. All of these techniques provide short-term protection; therefore, in general they are not applicable for our purposes. However, in some scenarios, they can represent a suitable solution, especially, when combined with other approaches.

Many proposals are based on checks. In these systems the software includes software and hardware-based "checks" to test whether certain conditions are met. However, because the validation function is included in the software, it can be discovered using reverse engineering and other techniques. This is particularly relevant in the case of agents. Theoretic approaches to the problem have demonstrated that self-protection of the software is unfeasible [25].

In some scenarios, the protection required is limited to some parts of the software (code or data). In this way, the function performed by the software, or the data processed, must be hidden from the host where the software is running. Some of these techniques require an external offline processing step in order to obtain the desired results. Among these schemes, function hiding techniques allow the evaluation of encrypted functions [26]. This technique protects the data processed and the function performed. For this reason it is an appropriate technique for protecting agents. However, it can only be applied to the protection of polynomial functions.

The case of online collaboration schemes is also interesting. In these schemes, part of the functionality of the software is executed in one or more external computers. The security of this approach depends on the impossibility for each part to identify the function performed by the others. This approach is very appropriate for distributed computing architectures such as agent-based systems or grid computing, but has the important disadvantage of the impossibility of its application to off-line environments

Finally there are techniques that create a *two-way protection*. Some of these are hardware-based, such as the Trusted Computing Platform. With the recent appearance of ubiquitous computing, the need for a secure platform has become more evident. Therefore, this approach adds a trusted component to the computing platform, usually built-in hardware used to create a foundation of trust for software processes [27].

3 Secure Execution of Agents in Ubiquitous Computing Scenarios

The main goal of this paper is to provide a secure and auto-configurable environment for mobile agents in ubiquitous computing scenarios. In order to achieve this goal we will base our approach on two main building blocks: trusted platforms and profiles.

On the one hand, in order to enhance the security of the execution environment our approach uses the concept of Trusted Platform. Because we are focusing on ubiquitous scenarios where we consider every device to be able to act as agency, and where the interaction with other previously unknown devices and applications will be the frequent, the security must be based on mechanisms that allow one party to verify the trustworthiness of the others. The idea behind the Trusted Computing paradigm was introduced in 1997 by Arbaugh, Farber and Smith [28]. The technology currently known as Trusted Computing has been developed by the Trusted Computing Group (TCG) on the basis of the specifications developed by the Trusted Computing Platform Alliance (TCPA). According to the TCG documentation, "the distinguishing feature of TCG technology is arguably the incorporation of 'roots of trust' into computer platforms." The TCG technology is not only for personal computers. In fact, it can be applied to most computing devices, such as PDAs, mobile phones, etc. The basic idea in our approach is to use the services provided by the TCG architecture in order to allow agents to verify that the agencies where they will be executed are trustworthy. In particular, we will check that the agencies run on top of trusted hardware and software configurations and that they have not been tampered with.

On the other hand, in order to facilitate auto configuration of the agencies, we will use secure profiles (i) for informing the agencies about the security requirements of the agents; (ii) for facilitating auto-configuration of the agency; and (iii) for supporting advanced monitoring of the behaviour of the agent.

3.1 Trusted Computing Support

The basic idea behind the concept of Trusted Computing is the creation of a chain of trust between all elements in the computing system, starting from the most basic ones. Therefore, the chain starts with a tamperproof hardware device, known as *Trusted Platform Module* (TPM), which analyses the BIOS of the computer and, in case it is recognized as trusted, passes control to it. This process is repeated for the master boot record, the OS loader, the OS, the hardware devices and finally the applications. This process can be tailored to suit the boot sequences of other devices such as routers, PDAs, mobile phones, etc. In Trusted Computing scenarios trusted applications run exclusively on top of protected and pre-approved supporting software and hardware.

One of the features of the Trusted Computing model is that it makes possible for both the platform owner and arbitrary third parties to obtain evidence about the integrity and configuration of a platform by measuring the platform components and comparing such measures to predefined values. The TPM component of the TCG architecture can also securely store secrets such as cryptographic keys and platform configuration measures in special shielded memory locations known as *Platform Configuration Registers* (PCRs). Additionally, TPMs can perform security relevant operations such as encryption and production of digital signatures. The process of obtaining metrics of those platform characteristics that affect its security and dependability; and storing and putting digests of those metrics in shielded locations is known as *integrity measurement*. *Integrity reporting* is the process of attesting to the contents of integrity storage. This is done by digitally signing specific internal TPM data using an *Attestation Identity Key* (AIK). These features are designed to allow platforms to enter any state, including those not identified as secure, but to prevent

that a platform can lie about states that is was or was not in. An independent process may evaluate the integrity state(s) and determine appropriate responses.

Remote attestation is another interesting feature. In the typical scenario where Alice and Bob are communicating, Alice can take advantage of the remote attestation feature in order to determine whether the current configuration of Bob's platform is safe. This is possible for Alice because the Trusted Computing technology provides mechanisms for her to measure (obtain a cryptographic hash) of the configuration of Bob's platform. If this configuration is altered or modified, a new hash value must be generated and sent to Alice in a certificate. These certificates attest the current state of Bob's platform and allow her to accept or reject the communication.

In particular, in our scenario we use this remote attestation mechanism between the TPMs of the platforms where each agency runs in order to verify that the agency software has not been tampered with and that it is running over appropriate software, firmware, and hardware configurations. The agency where the agent is currently running (source agency) is responsible for using remote attestation procedures in order to verify that the next agency in the agent itinerary (destination agency) is also trustworthy. In this way, assuming that the agent is started in a trusted agency (the home agency), we can be sure that the agent runs only on trusted agencies. Attestation is carried out by the TPM of the source agency in order to ensure that destination agencies provide secure and dependable execution environments for the agent.

Fig. 1 shows an interaction diagram that illustrates how the verification of the destination agency is done with the help of the TPMs of both agencies. For the sake of simplicity we include only two agencies. It is straightforward to extend this process for multiple agencies in the case of multi-hop agents. The process starts when the source agency (*A1*) receives from an agent (*ag1*) a request to verify the configuration of the destination agency (*A2*) according to some requirements (*A2req*). This explicit notification is necessary because the verification is done by the source agency with the help of the local TPM (*tpm1*). There are some cases in which agents can carry out this process without the intervention of the source agency. However, this second alternative requires the agents to be able to access the services of the local TPM directly, which is not likely to be allowed.

Then, the source agency TPM uses the remote attestation mechanism in order to obtain the configuration of the destination agency (*A2conf*). This process requires the collaboration of the destination agency TPM (*tpm2*). If the required configuration is successfully verified, the agent can safely migrate to the destination agency. In this scenario we assume the simplest case where the requirements from the agent can be directly compared by the TPM (e.g. requirements are expressed in the form of TPM measurements). In some cases, it is possible that the process of checking the conformance between the agent requirements and the destination agency configuration requires some more complex processing. In these cases, we foresee that the source agency will be responsible for carrying out such process. An example of such process is the so called *semantic attestation*, which provides enhanced flexibility to the attestation mechanism at the cost of more complex processing.

In summary this scheme provides guarantees of the security and dependability of the destination agency before the agent runs on it. The scheme is simple and efficient and can be successfully applied to multi-hop agents.

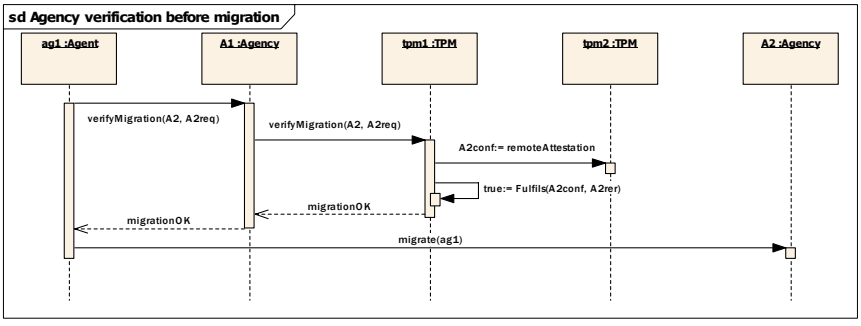

Fig. 1. Verification of the destination agency before migration

3.2 Profiles and Auto-configuration

In the approach proposed by Ghosh et al [29], profiling the behaviour of a program is performed through the study of the set of system calls of the program. We use the same idea, though with agents. The main idea is to make the profile contain a declaration of the task that the agent intends to perform, following an approach analogous to the proof-carrying code [17]. However, we propose that, additionally to this information, the profile contains information concerning the agencies visited together with the set of relevant operations executed in each agency. This information will be added in the profile dynamically by each agency. This is noteworthy for the case when an agent can contain malicious code, a code that can not be detected by agent execution in only one agency but a malicious code segmented and executed in the different agencies visited by the agent. Thus, detection needs a global supervision.

Fig. 2 shows how the complete agent migration process is performed, from a source agency (*A1*) to a destination agency (*A2*). In this case the interaction includes the transmission of the agent profile and the auto-configuration of the destination agency. In the first step, the agent (*ag1*) sends a request for migration to the source agency (*A1*). As in the previous example, it is necessary to specify the destination agency (*A2*) and the configuration requirements (*A2req*). Then, the source agency sends a message requesting the verification of the destination agency to the local TPM (*tpm1*). This one will carry out a remote attestation with the remote TPM (*tpm2*). In case this attestation is successful, and the destination host configuration fulfils the agent requirements, the local TPM requests the public key from *tpm2*. This public key is used by the encryption service of the local TPM to encrypt the agent profile (*ag1.profile*). Then, source agency sends the encrypted profile to destination agency. Finally, the encrypted profile is decrypted by the TPM of the remote agency.

At this point the destination agency verifies the agent profile with regards to its local policy. In case the profile is accepted, the destination agency auto-configures itself in order to receive and execute the agent and notifies this circumstance to the source agency. Finally, the agent can migrate to the destination agency.

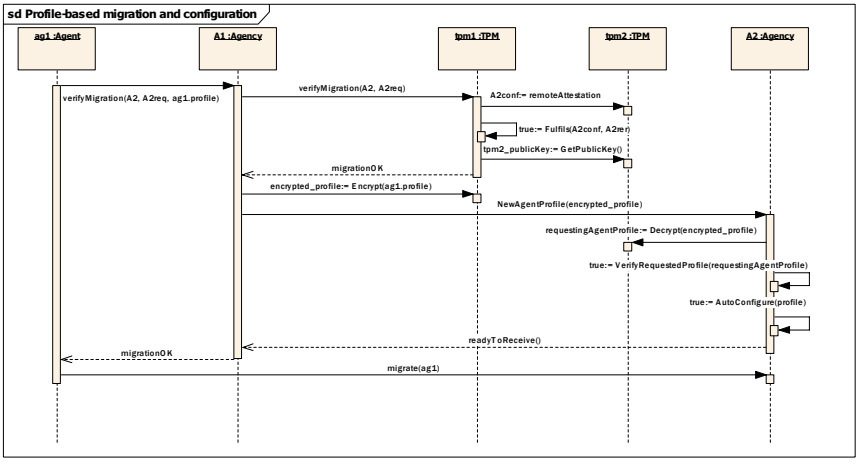

Fig. 2. Auto-configuration of the destination agency based on the agent profile

Once the fundamentals for providing a secure run environment for mobile agents has been described in detail, we follow with the description of their self-configurability. As mentioned, we will use the agent profile. To better understand what an agent means, Fig. 3 shows an example of an agent profile in XML format. In this example we have structured the information of the agent in five main blocks. Firstly, we have the information related to general requirements of the agents, like amount of memory required, the type of communication, etc. Secondly, we have established the security requirements for the execution of the agent. In the example we have information like the cryptographic mechanisms used, the access to TPM, and the certification of the platform. Then, we have established another section with the platform configuration where we can indicate if the agent requires the agency to run in trusted mode or not. The next section contains the dynamic part of the profile. Each agency visited will include new information on this section. We have highlighted the monitoring information, as it provides interesting features to our scheme. As it can be seen, the final section is left open for any extension required for specific cases.

Coming back to the monitoring, this scheme is interesting because it allows agencies to include information inside the agent profile, so that information could be used for later supervision tasks. This would be useful in the cases when an agent performs something suspicious but not detectable in the partial execution of an agent in an agency. This mechanism introduces the possibility of supervising the global behaviour of the agent by all agencies it passes through.

Another interesting aspect is that the profile can be certified by the agency with the help of the TPM. But, moreover, the profile could include a history of all agencies the agent passed by, and these could certify its operation. This scheme takes advantage of the Trusted Computing model as well as the possibilities of rich profiling, providing a secure execution environment for agents. Besides, we allow a self-configuration of the agent execution environments (agencies) prior to its arrival to the host.

```
<?xml version="1.0" encoding="UTF-8" ?>
<agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="D:\informacion\agentProfile.xsd">
   <generalReq>
      <memoryReq>128kb</memoryReq>
      <communication>none</communication>
<!-- more general requirements -->
   </generalReq>
   <securityReq>
      <cryptoMechanisms>
         <cryptoMechanism>
            <type>AsymmetricEncryption</type>
            <algorithm>RSA</algorithm>
            <keyLength>512</keyLength>
         </cryptoMechanism>
      </cryptoMechanisms>
      <tpmAccess>Yes</tpmAccess>
      <platformCert>Yes</platformCert>
<!-- more security requirements -->
   </securityReq>
   <platformConf>
      <trustedMode>
         <required>Yes</required>
         <confID>'71386cb5c2ed63f855d2533cc264bc2e'</confID>
      </trustedMode>
<!-- more platform configuration values -->
   </platformConf>
<!- dynamic part of the profile containing
    platform-generated information -->
   <platformInfo>
      <monitoring>
         <visit>
            <host url="http://www.agentHome.org">
            </host>
            <actions>
               <!-monitored actions -->
            </actions>
         </visit>
      </monitoring>
<!-- more platform-generated information -->
   </platformInfo>
   <extensions>
<!-for scenario-specific information -->
   </extensions>
</agent>
```

Fig. 3. Example of agent profile

4 Conclusions and Ongoing Work

Our main motivation is to provide a secure and auto-configurable environment for mobile agents in ubiquitous computing scenarios. In order to achieve this goal we have based our approach on two main building blocks: trusted platforms and profiles.

We have described how these two elements contribute to the proposed solution and have discussed some alternatives. We have also reviewed the advantages of the proposed approach and have shown how additional features are enabled by the use of this scheme. Among these additional features we must highlight the enhanced support for supra-agency monitoring. The importance of this feature is that it enables the detection of attacks that can not be identified by analyzing the actions performed in just one agency. Additionally we have illustrated the migration processes.

We are currently working on the implementation of agencies over platforms containing TPMs. We are also working on the FIPA-OS specifications in order to accommodate the new functionalities required by our system (e.g. access to the TPM).

Finally, we are also working on complementary protection mechanisms for the agents that do not involve the use of TCG technology [30].

References

1. Trusted Computing Group: TCG Specifications. 2005. Available online at https://www.trustedcomputinggroup.org/specs/
2. Resnick, P. and Varian, H., Eds.: Communications of the ACM: Special Issue on Recommender Systems 46. 1997.
3. Riecken, D., Ed.: Commun. ACM: Special Issue on Personalization 43. 2000.
4. Maybury, M., Ed.: Commun. ACM: Special Issue on News on Demand 43. 2000.
5. Maybury, M. and Brusilovsky, P., Eds.: Commun. ACM: The Adaptive Web 45. 2002.
6. Bygrave, L.: Electronic Agents and Privacy: A Cyberspace Odyssey 2001, Intl. Journal of Law and Information Technology, vol 9, no 3, p 280, Oxford University Press, 2001.
7. Pearson, S.: Trusted Agents that Enhance User Privacy by Self-Profiling. Proceedings of the AAMAS Workshop (Special track on privacy). 2002.
8. W3C: Resource Description Framework (RDF): Concepts and Abstract Syntax. 2004.

9. W3C: CC/PP: Structure and Vocabularies 1.0 January 2004. Available online at http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html
10. Wireless Application Protocol Forum: Wireless Application Group User Agent Profile Specification. Nov. 1999.
11. Foundation for Intelligent Physical Agents: FIPA Device Ontology Specification, December 2002. Available online www.fipa.org.
12. Chi, E.H.: Transient User Profiling. Proceedings of the workshop on User Profiling. 2004.
13. Sparacino, F.: Sto(ry)chastics: a Bayesian Network Architecture for User Modelling and Computational Storytelling for Interactive Spaces. Proceedings of the Fifth International Conference on Ubiquitous Computing. 2003.
14. Berkovits S, Guttman J, Swarup V.: Authentication for Mobile Agents. In Mobile Agents and Security volume 1419, pages 114-136. Springer-Verlag. 1998.
15. Maña, A.: Protección de Software Basada en Tarjetas Inteligentes. PhD Thesis. University of Málaga. 2003.
16. Hachez, G.: A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards. PhD Thesis. Universite Catholique de Louvain. 2003.
17. Necula G.: Proof-Carrying Code. Proceedings of 24th Annual Symposium on Principles of Programming Languages. 1997.
18. Gunter Carl A., Homeier Peter, Nettles Scott.: Infrastructure for Proof-Referencing Code. Proceedings of the Workshop on Foundations of Secure Mobile Code. March 1997.
19. Yee, Bennet S.: A Sanctuary for Mobile Agents. Secure Internet Programming. 1999.
20. Schaumüller-Bichl1, I., Piller, E.: A Method of Software Protection Based on the Use of Smart Cards and Cryptographic Techniques. Proceedings of Eurocrypt'84. Springer-Verlag. LNCS 0209, pp. 446-454. 1984.
21. Stern, J. P., Hachez, G., Koeune, F., Quisquater, J. J.: Robust Object Watermarking: Application to Code. Proceedings of Info Hiding '99, Springer-Verlag. LNCS 1768, pp. 368-378. 1999.
22. Collberg, C., Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. University of Auckland Technical Report #170. 2000.
23. Wayner, P.: Dissapearing Cryptography. Information Hiding, Stenography and Watermarking. Morgan Kauffman. 2002.
24. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of Obfuscating Programs. Proceedings of CRYPTO `01. Springer-Verlag. LNCS 2139. pp. 1-18. 2001.
25. Goldreich, O.: Towards a theory of software protection. Proceedings of the 19th Ann. ACM Symposium on Theory of Computing, pp. 182-194. 1987.
26. Sander, T., Tschudin C.F.: On Software Protection via Function Hiding. Proceedings of Information Hiding '98. Springer-Verlag. LNCS 1525. pp 111-123. 1998.
27. Pearson, S., Balacheff, B., Chen, L., Plaquin, D., Proudler. G.: Trusted Computer Platforms. Prentice Hall. 2003.
28. Arbaugh W., Farber D., Smith, J.: A Secure and Reliable Bootstrap Architecture. Proceedings of the 1997 IEEE Symposium on Security and Privacy, pp 65-71. 1997.
29. Ghosh, A., Schwartbard, A., Schatz M.: Learning program behavior profiles for intrusion detection. Proceedings of the Workshop on Intrusion Detection and Network Monitoring, Usenix. 1999.
30. Maña, A., Muñoz, A.: Mutual Protection for Multiagent Systems. Proceedings of the Third International Workshop on Safety and Security in Multiagent Systems (SASEMAS '06). 2006.