Personalisation of Medical Services

D. Isern, D. Sánchez, A.Moreno, and A.Valls

Multi-Agent Systems Group (GruSMA)
Computer Science and Mathematics Department
Universitat Rovira i Virgili (URV)
ETSE. Av. dels Països Catalans, 26 (Campus Sescelades)
43007-Tarragona, Catalonia (Spain)
{disern,dsanchez,amoreno,avalls}@etse.urv.es

Abstract. In this paper we describe an application called HECASE, which is an agent-based system that provides medical services to its users (the citizens or the visitors of a city). This multi-agent system contains agents that have information about the medical centres, departments and doctors of a region. All these agents coordinate their tasks to provide a set of services to the user of the system, who can search for medical centres satisfying a given set of requirements, as well as access his medical record, or make a booking to be visited by a particular kind of doctor. Special care has been paid to the implementation of mechanisms that guarantee the confidentiality in the access and transmission of medical data, and provide personalised information using a dynamically updated user profile.

1 Introduction

The new information technologies allow people to access eficiently a wide range of data and services. Our aim is to use these tools to solve daily problems to improve the citizens quality of life. We focus our research in the health care domain using multi-agent systems technology (MAS) [15], that offers an added value in front of classical approaches such as web-based applications [3]. A MAS has the following interesting properties:

- *Modularity:* the different services or functionalities may be distributed among diverse agents, depending on their complexity.
- *Efficiency:* agents may coordinate their activities to perform complex tasks, so that several parts of the same process may be solved concurrently by different agents.
- *Reliability:* any distributed process is more reliable than its centralised counterpart, because there does not exist a single point of failure that may cause the crash of the whole system.
- *Flexibility:* agents may be dynamically created or eliminated according to the needs of the application. Negotiation and knowledge exchange allow the optimisation of shared resources.

I. Rudomín, J. Vázquez-Salceda, J. Díaz de León. (Eds.). e-Health: Application of Computing Science in medicine and Health Care. © IPN, México 2003.

- *Existence of a standard:* the FIPA (Foundation for Intelligent Physical Agents, [8]) is a non-profit foundation based on Geneve (Switzerland). Its main mission is to establish the rules that have to govern the design and implementation of a MAS in order to achieve interoperability among systems. Since 1997 it has been releasing specifications that have been slowly gaining acceptance and have turned into de facto standards in the agents community. Due to this fact, any of our agents is compatible with any other agent that follows the same specifications.
- *Existence of development tools:* JADE (Java Agent Development Environment, [13]) is a programming tool that contains a set of JAVA libraries that facilitate the development of FIPA-compliant MASs. Apart from providing low level agent management functionalities and graphical interfaces that ease development and debugging, it also provides an execution environment for agents. Recently, a JADE plug-in that provides certain security mechanisms, called JADE-S, has been released.

In this paper we describe the design and implementation of a MAS called HECASE that offers secure personalised health care services. The rest of the paper is organised as follows. First of all, in §2 we give a general overview of the HECASE system. Section §3 explains how the system adapts dynamically its behavior to the user's profile. In §4 we offer an overview about security issues. The paper finishes with a discussion of the work done and some future lines of research.

2 HECASE: Overview

HECASE is a MAS prototype that models the medical institutions of Catalonia [3]. The main design objectives are the following:

- To provide a decomposition of the problem that allows to model the real entities of the medical domain as agents, based on the structure of medical centres in Catalonia [3] (each centre has a set of departments, and each department has a set of doctors).
- To provide an ontology for the medical domain [10].
- To provide security measures that ensure the confidentiality and privacy of medical data such as Secure Socket Layer (SSL) for message ciphering and Public Key Infrastucture (PKI) and Hash functions for authenticating and user's access control [9].
- To make the developed agent services as reusable as possible by: (1) using standard languages - in this case FIPA-ACL ([7]) for agent language communication and Resource Description Framework (RDF, [4]) for content and representation of ontologies - and (2) providing detailed service models to describe the individual functioning and objective of each agent including descriptions of actions, protocols used and example messages [10].
- To implement mechanisms to allow the user to adapt his Personal Agent behavior based on his profile. This will be very useful to guide a negotiation or to make proposals according to the user's preferences (See §3).

- To provide ubiquitous access to the system from anywhere, anytime [10].

With these aims, we have implemented a MAS that offers the following features (more details in [10]):

- The user may request information about all the medical centres available in a particular geographical area.
- It is possible to book a visit to be examined by a doctor.
- As mandated by a Catalan law [2], the user has access to his medical record.
- It must be made sure that nobody can access the private medical information of the users of the system without proper authorisation.
- The user can manage his own timetable and a set of preferences in order to select the most appropriate proposal or sort a list of options in the booking negotiation protocol.
- The doctor can manage his working hours and patient's appointments. During an examination, he is able to access the patient's medical history and to update it with the result of the visit.

2.1 Architecture

The basic architecture of the MAS which has been developed in this work is shown in Fig. 1. This multi-agent system contains the following agents:

i) The user's personal agent (PA), that provides a graphical interface of the MAS to the user, allowing the access to the offered services (queries, bookings, agenda, preferences...).

ii) The broker agent (BA) is an agent that provides a gateway between personal agents and the rest of the agents in the system. It controls the access of users that are properly authenticated.

iii) The information of a medical centre is divided in three levels. For each medical centre there is one medical centre agent (MCA), several department agents (DEPs, one for each department of the centre) and many doctor agents (DAs, one for each doctor of each department). The MCA has the general information of the centre (e.g. its address and opening times). Each DEP has the knowledge of a certain department (e.g. all the information of the Ophthalmology department). Each DA maintains the schedule of a given doctor, and is aware of the doctor's visiting times. They also implement a graphical interface to the doctor that is used to manage the list of pending visits with patients, to introduce the results of a medical visit and to look up the patient's personal data.

iv) The database wrapper (DW) is the agent that controls the access to a database that contains the medical records of the users.

The agents are stored in several containers within a single platform [7]. The MCAs (with DEPs and DAs) and the DW are internal to the system and only the BA is accessible by external agents. The PAs are running in the user's machine in an external container. The internal agents could be deployed physically around several machines in a real setting, e.g. each MCA (with its hierarchy of associated agents) could be running in a separate machine.

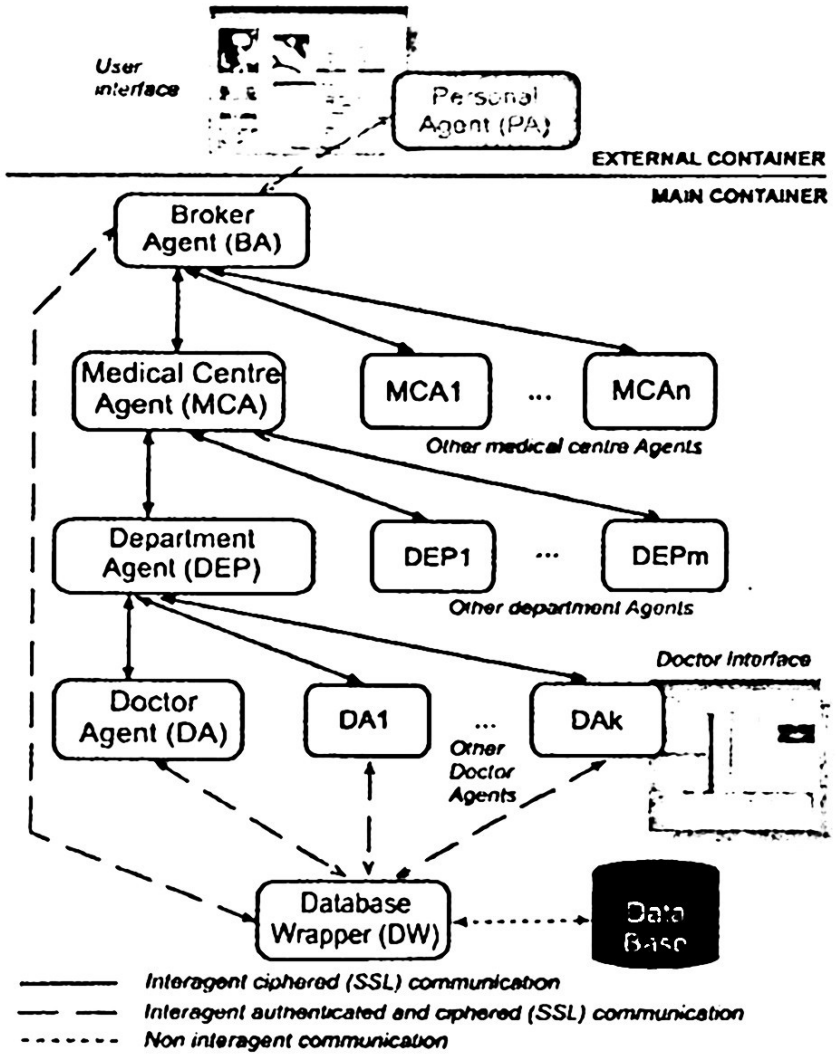

Fig. 1. Architecture of HECASE

3 Service personalisation

In order to provide more personalised services, each patient has his own profile which stores personal data (e.g. name, address, birth date, job), medical information (e.g. allergies, disabilities, medical antecedents), security elements (e.g. login, password), timetable (e.g. appointments, reminders) and preferences (e.g. ranking of doctors, medical centres and cities). The first three items are used to create the personal record that can be accessed by doctors or by the owner (after an authentication process, [9]). The timetable and the preferences are used to guide and personalise the booking negotiation process, as described in the following subsections.

3.1 Timetable plug-in

The *timetable plug-in* is a domain independent package for time management and scheduling visualisation. The list of features that this plug-in provides is the following:

- Management of appointments (an interval of time in which some kind of event occurs). Two types are distinguished: the *blocking ones* (shown on the left column of each day) that do not allow overlapping with others (like independent tasks) and the *non blocking ones* (shown on the right column) that can happen at the same time interval (like reminders).
- An intuitive GUI for user interaction (See Fig. 2) that allows introducing, checking, modifying and removing appointments as well as configuring the visualization properties (number of days, size, time intervals).
- The complete set of appointment configurations can be saved/loaded in a standard file format.
- Appointment booking negotiation that allows to reserve a set of intervals and confirm or refuse them later.
- A set of methods that allow searching the best free interval where an appointment fits, or retrieving the list of appointments contained in a given time range.

This plug-in has been included in both personal and doctor agents in order to manage their schedule during the booking negotiation. A detailed explanation of this process is made in §3.3.

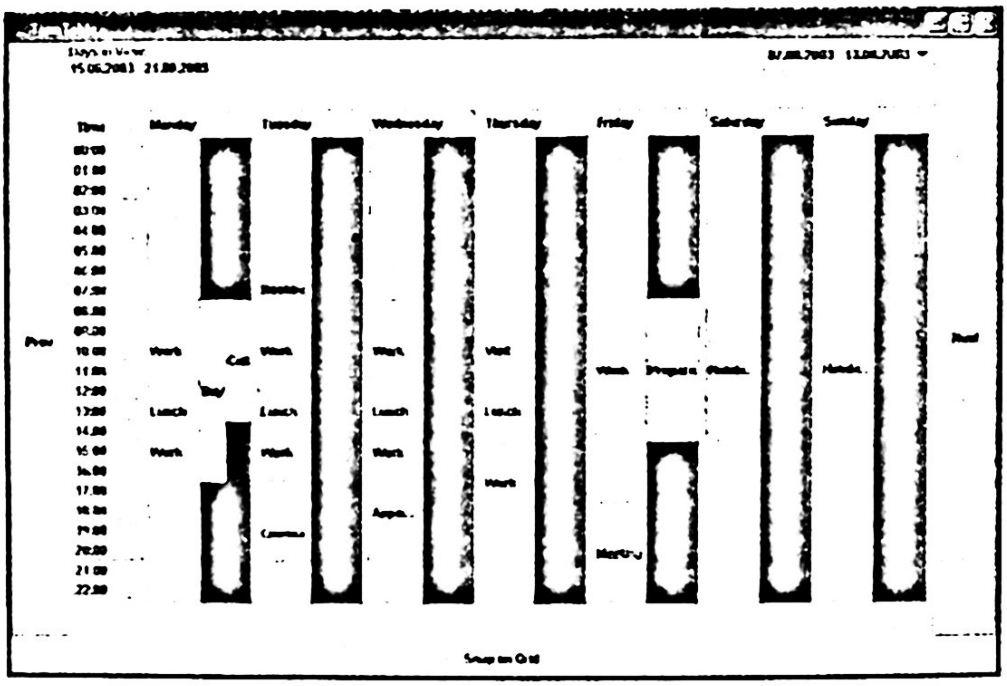

Fig. 2. Timetable plug-in GUI

3.2 Preferences plug-in

The *preferences plug-in* is a package which is able to manage multiple domains of preference. Its main goal is to allow to the user to obtain a ranking of a list of options (alternatives) according to the preferences of its corresponding domain.

Each domain (e.g. health care) has a set of criteria that corresponds to different properties or elements in the domain (e.g. doctors, cities). Each criterion can take different linguistic values to define the set of alternatives (e.g. doctors: Dr. Smith, Dr. Winston; cities: Tarragona, Barcelona). The user must express his preferences on these values. To do this, the user assigns a preference linguistic label to each of the possible values of the criterion (e.g. Tarragona=Best, Montblanc=OK, see Fig. 3). In addition, the user can indicate that one of the possible values of the criteria is not valid for him. For exanple, if the user does not want to be visited in a certain medical centre, he can assign a "Veto" label to this value (e.g. ConsultoriLocalAlforja=Veto).

Another feature of the plug-in is the possibility of giving different weights to each criterion. Moreover, some criteria can be temporally deactivated if the user decides not to take them into account for further decisions.

Once the user has specified his preferences, the plug-in can rank the alternatives taking into consideration all the criteria. To solve this problem a classical MCDM (Multi Criteria Decision Making, [14]) method has been implemented, which is based on multiattribute utility theory. This approach considers that each criterion is expressing a partial utility of each alternative. To aggregate these partial utilities, a weighted average is computed. Since the preferences are defined using linguistic values, a pre-processing stage is performed in which labels are translated into numbers. The result is a sorted and rated list of alternatives.

The plug-in has been integrated in the Personal Agent to sort the list of received proposals. They are sorted according to some criteria (e.g: place of the treatment, the name of the doctor, the medical centre) after a query for bookings has been performed.

As the user cannot know in advance all the possible values of the criteria (e.g: name of doctors or medical centres), the plug-in is able to discover new values each time that a query is performed. With this functionality the agent is able to learn from the user interaction and improve the quality of the rankings.

3.3 Booking negotiation

When the user needs the care of a doctor, he can book a visit using HeCaSe. The booking process requires the negotiation of many agents. So, this is the most complex process in the system due to its high level of customisation according to the user's profile and the number of agents that interact.

The protocol (see Fig. 4) is divided in several steps on which the data is processed from the most general to the most specific agent in order to achieve the best proposal that fits with the user's preferences:

1) Firstly, the user initialises a set of search constraints (medical specialty, urgency, number of desired proposals per doctor, number of days to consider

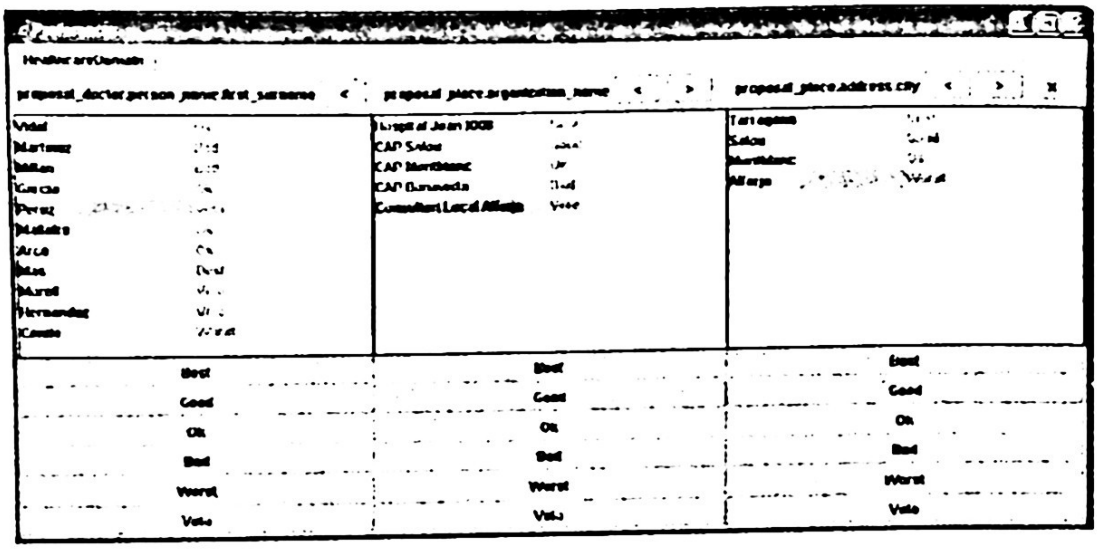

Fig. 3. Preferences plug-in GUI

and, optionally, the desired medical centre). Then the system analyses the user's timetable through the Timetable plug-in in order to find the best time intervals in which an appointment with a doctor could be set. To do this, it divides the following X days (the number has been given by the patient),into three intervals that are sorted according to the amount of free minutes inside them, preserving their natural order (position in time) in case of a tie. This represents the patient's wish to get an appointment in an interval that is as soon as possible and not already full with other appointments. With all this information, the Personal Agent (PA) constructs a message.

2) Secondly, the message is sent to the Broker Agent (BA) that broadcasts it to all Medical Centre Agents (MCA) or to the one selected by the user. The MCA fowards the message to the Department Agent (DEP) that fits with the selected medical speciality and finally, the DEP broadcasts it to all its Doctor Agents (DA).
3) Each DA evaluates the booking preconditions included on the received message and constructs a list of proposal appointments. In particular, the agent tries to fulfil the patient's wishes by matching the doctor's free time (determined by the timetable) with the sorted preferred intervals contained on the received message. If there is no possible slot in the most preferred interval or the user has requested more than one proposal, the doctor repeats the matching process with the next interval. The final proposed hours are reserved until the doctor agent receives the final accept or refuse message from the patient. This list of proposals of each doctor is returned to the DEP, which puts together all the doctors' proposals and sends them to the MCA. Then, all the MCAs involved on the protocol send their lists to the BA that joins them all and sends them to the user's PA.
4) Once this list is received, the PA evaluates the user's preferences (on the medical centres, cities and doctors). The Preferences Plug-in sorts and rates

the proposals (as described in §3.2) and the result is shown to the user. If some of the proposed appointments overlap with the appointments in the patient's timetable, the PA gives him a warning.

5) Finally, the user can select his preferred proposal or reject all of them. In the first case, the appointment is fixed on his timetable and an Accept message is sent through the medical hierarchy of agents to the desired doctor, and at the same time a Refuse one is broadcasted to the rest of involved agents; finally a confirmation is returned to the PA. In the second case, all agents receive a Refuse message.

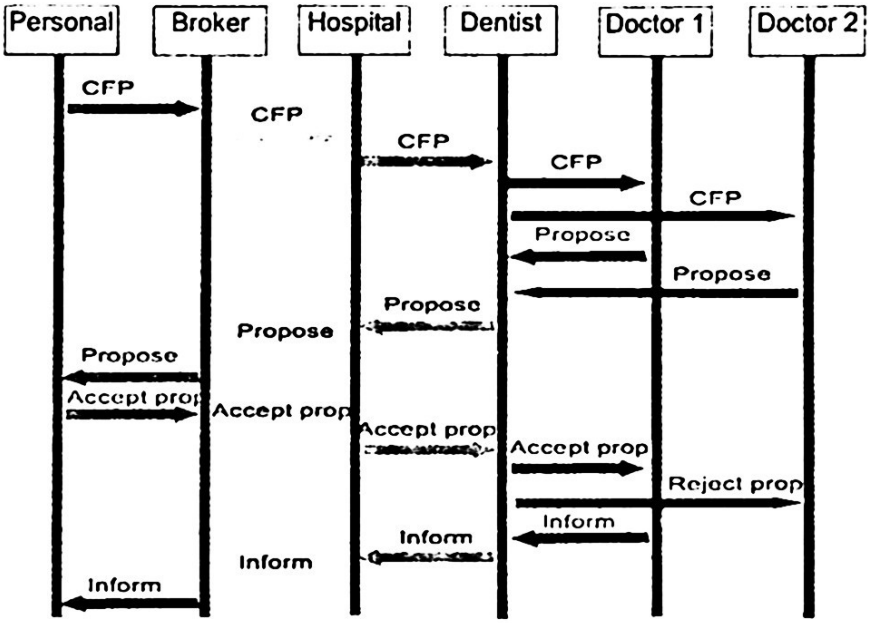

Fig. 4. 4-layer CFP Protocol for booking a visit

4 Security

In order to understand the role of security in the transmission of private and critical information (e.g. health records) through an open environment (e.g. Internet), it is necessary to define the concepts that determine the diverse security levels [6]:

- *Confidentiality*: is the property that ensures that only those that are properly authorised may access the information.
- *Integrity*: is the property that ensures that information cannot be altered. This modification could be an insertion, deletion or replacement of data.
- *Authentication*: is the property that refers to identification. It is the link between the information and its sender.

- *Non-repudiation*: is the property that prevents some of the parts to negate a previous commitment or action.

In the case of a MAS these properties are especially important, due to the autonomy and mobility of agents. A MAS without security support could not be used in an open environment such as Internet if it deals with critical data, because communications could be spied or the identities of the agents could be easily faked.

JADE-S [12] is a plug-in of JADE that allows to add some security characteristics in the development of MAS, so that they can start to be used in real environments. It is based on the Java security model ([11]) and it provides the advantages of the following technologies:

- JAAS (Java Authentication and Authorization Service)[1]: it allows to establish access permissions to perform certain operations on a set of predetermined classes, libraries or objects.
- JCE (Java Cryptography Extension) [2]: it implements a set of cryptographic functions that allow the developer to deal with the creation and management of keys and to use encryption algorithms.
- JSSE (Java Secure Socket Extension) [3]: it allows to exchange critical information through a network using a secure data transmission (SSL).

4.1 Basic Concepts

A JADE platform may be located in different hosts and have different containers. In order to introduce security in such an open and distributed environment, JADE-S structures the agent platform as a multi-user environment in which all components (agents, containers, etc.) belong to authenticated (through a login and a password) users, who are authorised by the administrator of the system to perform certain privileged critical actions. The general scheme of this environment is shown in Fig. 5. In each platform there is a permissions file that contains the set of actions that each user is authorised to perform.

Internally, an agent proves its identity by showing an Identity Certificate signed by the Certification Authority (provided in a transparent way to the agent when it registers in the system and provides the login and the password of its owner). Using these digitally signed certificates the platform may allow or deny certain actions to each agent.

4.2 Authentication

As explained above, each component of the platform belongs to an authenticated user. The user that boots the platform also owns the AMS and DF agents and the main container.

[1] More information at http://java.sun.com/products/jass/index-14.html
[2] More information at http://java.sun.com/security
[3] More information at http://java.sun.com/j2se/1.4/docs/guide/security/jsse/SEERefGuide.ht

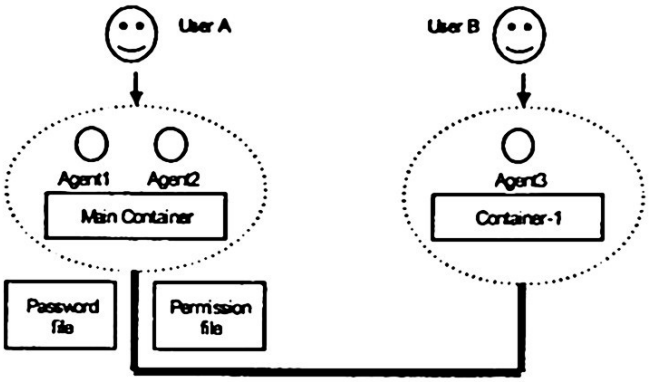

Fig. 5. General architecture of a secure agent platform

When a user wants to join the platform (through one of his agents), it has to provide his login and password. These data are checked with the passwords file contained in the platform, which is stored in a ciphered way, like Unix passwords. The passwords file is unique and is loaded with the main container.

Each agent owned by this user will have an Identity Certificate that contains its name, its owner and the signature of the Certification Authority.

4.3 Permissions and Access Restrictions

In a JADE-S platform the permissions to access resources are given to the different entities by following the mechanism defined by JAAS, the new system provided by Java, for user-based authentication.

Thus, it is possible to assign permissions to parts of the code and to its executers, restricting the access to certain methods, classes or libraries depending on who wants to use them. An entity can only perform an action (send a message, move to another container) if the Java security manager allows it. The set of permissions associated to each identity is stored in the access rights file of the platform (which is also unique and is loaded when the platform is booted).

Java provides a set of permissions (apart from those that may be defined by the user) on the basic elements of the language. Each permission has a list of related actions that may be allowed or denied (e.g. the `FilePermission` controls reading/writing/updating files).

4.4 Certification Authority and Certificates

The Certification Authority is the entity that signs the certificates of all the elements of the platform. To do that, it owns a couple of public/private keys so that, for each certificate, it creates an associated signature by ciphering it with its private key (which is secret). Then, when the identity of an entity has to be checked, the signature may be unencrypted with the public key of the Authority (which is publicly known) and we can check that the identity that the

entity wanted to prove matches the one provided by the Authority. The secure platform JADE-S provides a Certification Authority within the main container. Each signed certificate is only valid within the platform in which it has been signed.

4.5 Secure communication

In order to provide a secure communication between agents located in different hosts or containers, JADE-S uses the SSL protocol (Secure Socket Layer) that provides privacy and integrity for all the connections established in the platform. This is a way of being protected against network sniffers.

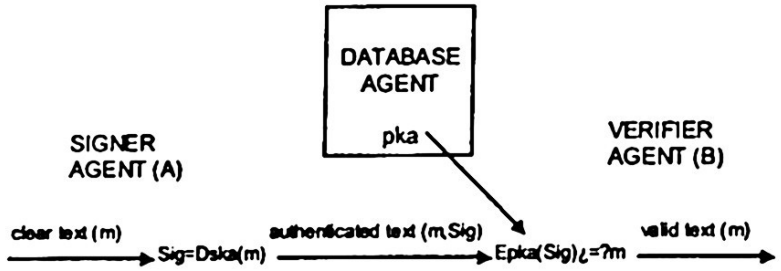

Fig. 6. Signing a message with a public key

4.6 Implementation

The main features of our security model are the following:

- All the messages exchanged among the agents are encrypted at the transport level with SSL, and the server is authenticated via an Authentication Certificate.
- We have established diverse security levels by using different user identities, and associating an owner (identified with a login and a password) to each agent (with a set of permissions for each one). Concretely, we have defined two virtual users associated to two groups of agents:
  - Internal agents: they execute inside the main container of the platform. We have linked these agents to the "root" user, which does not have any constraint.
  - External agents: they cover all the Personal agents that run from another host through an External Container. We have linked them to the "guest" user which has the minimum permission access (communication with the Broker Agent).
- It is a centralised access model through the Broker agent, that implements a software security control (See Fig. 1).

- Personal agents cannot communicate directly with any other agent (even with the DF).
- User authentication through the signature of critical messages using a public key mechanism (See Fig. 6).

5 Conclusions and future work

HECASE is part of the Agent Cities initiative ([1]) whose main aim is to provide agent-based services that improve the quality of life of the citizens and the visitors of a city. The deployed application certainly contributes towards this objective, by giving to the users the possibility of accessing medical data in an easy and efficient way.

This system is fully implemented in a prototype version, with all the agents running in standard PCs. The system has been designed to benefit two kinds of users:

a) Citizens that need medical services, who could access their medical record from anywhere, make appointments with a certain doctor based on his preferences, and obtain any kind of information related to the medical centres, departments and doctors of a given city.
b) Health care personnel, who could automatically access and update the patient's medical record during the examination, and would have their workload decreased, due to the fact that citizens could make queries or appointments on their own.

The system[4] described in this paper has been implemented using JADE [13]. The content of all messages is written in RDF ([4]). The medical records database has been implemented using MySql. Moreover, we use the plug-in JADE-S[5] and the cryptographical library Cryptonite ([5]) to provide a secure infrastructure in the platform [9].

Finally, we summarise some research lines to follow in the future:

- The test of the system with real data of medical centres, departments and doctors.
- The implementation of user agents deployed in mobile environments (such as PDAs or mobile phones). The LEAP library [13], compatible with JADE, may be used for this purpose.
- The integration of new agents that simulate other medical entities like Medical Service Agents (X-rays, clinical analysis ...).
- The use of medical guidelines to ease the doctors labour by proposing a set of actions or treatments to be followed for a medical problem.

[4] Our system is accessible live in the Agent Cities network of platforms (see access instructions at http://grusma.etse.urv.es/hecase/)

[5] JADE Security (JADE-S) is a plug-in and it can be obtained from the JADE's Website (http://sharon.cselt.it/projects/jade)

- The design of a hierarchy of Brokers that controls a subset of medical entities (e.g: medical centres located in the same city). We will study the coordination of these brokers in order to provide a more efficient solution instead of the centralised Broker approach.

Acknowledgements

This system has been developed with the support of AgentCities.NET through the deployment grant "Deployment of agent-based health care services" and a Student Mobility Grant. In relation to this grant, we acknowledge Marek Jawurek from University of Aachen (Germany) for his contributions on the system. The authors also acknowledge the support of the Spanish thematic network "Creación de un entorno innovador para la comunicación de agentes inteligentes" (MCyT, TIC2001-5108-E).

References

1. Agentcities. IST project IST-2000-28384 Agentcities, 2000. http://www.agentcities.net/.
2. Generalitat de Catalunya. Law on the information rights concerning health, the autonomy of the patient and the clinical documentation. Law 21/2000. 29th December 2000. In *Quaderns de Legislació, 31*, pages 1–35. Health and Social Security Department, Generalitat de Catalunya, 2001. ISBN 84-393-5459-2.
3. Generalitat de Catalunya. Servei Català de la Salut (ICS), 2003. http://www.gencat.net/scs.
4. W3C Consortium. Resource Description Framework (RDF) - W3C Semantic Web Activity, 2003. http://www.w3.org/RDF/.
5. Cryptonite. Package of Java logi.crypto, 2000. More information available on http://logi.org/logi.crypto.
6. J. Domingo and J. Herrera. *Criptografia per als serveis telemàtics i el comerç electrònic*. EdiUOC, 1999.
7. FIPA. FIPA Specification, 2003. http://www.fipa.org.
8. FIPA. Foundations for Intelligent Physical Agents (FIPA), 2003. http://www.fipa.org.
9. A. Moreno, D. Sánchez, and D. Isern. Security Measures in a Medical Multi-Agent System. In *Sisè Congrés Català d'Intel.ligència Artificial (CCIA'03)*, Palma de Mallorca, Illes Balears, October 23-26, 2003 (in press).
10. A. Moreno, A. Valls, D. Isern, and D. Sánchez. GRUSMA1: Experience on the deployment of agent-based health care services. In *Applications of Software Agent Technology in the Health Care Domain*. To appear in WhiteStein series in Software Agent Technologies, Zurich, Switzerland, 2003 (in press).
11. Sun Microsystems. Java Security, 2003. More information available on http://java.sun.com/security.
12. TILab S.p.A. JADE Tutorial: Security Administrator Guide v.3.01beta, March 2003. More information available on http://sharon.cselt.it/projects/jade.
13. TILab S.p.A. Java Agent Development Framework (JADE) v.3.01beta, March 2003. More information available on http://sharon.cselt.it/projects/jade.

14. P. Vincke. *Multicriteria Decision-Aid.* John Willey and Sons. 1992. ISBN: 0471931845.
15. M. Wooldridge. *An introduction to Multi-Agent Systems.* John Wiley and Sons. 2002. ISBN 047149691X.