Adding X–Security to Carrel: Security for Agent–Based HealthCare Applications

David Cabanillas, Steven Willmott, Ulises Cortés
Software Department, Technical University of Catalonia,
Jordi Girona 1 & 3,
08034 Barcelona, Spain
{dconrado, steve, ia}@lsi.upc.es

May 5, 2003

Abstract

The high growth of Multi–Agent Systems (MAS) in Open Networks with initiatives such as Agentcities[1] requires development in many different areas such as scalable and secure agent platforms, location services, directory services, and systems management. In our case we have focused our effort on security for agent systems.

The driving force of this paper is provide a practical vision of how security mechanisms could be introduced for multi–agent applications. Our case study for this experiment is *Carrel* [9]: an Agent–based application in the Organ and Tissue transplant domain. The selection of this application is due to its characteristics as a real scenario and use of high–risk data for example, a study of the 21 most visited health-related web sites on the Internet[2] discovered that personal information provided at many of the sites was being inadvertently leaked for unauthorized persons. These factors indicate to us that *Carrel* would be a suitable environment in order to test existing security safeguards. Furthermore, we believe that the experience gathered will be useful for other MAS.

In order to achieve our purpose we describe the design, architecture and implementation of security elements on MAS for the *Carrel* System.

1 Introduction

Nowadays, it is unsurprising that in different technologies such as Operating Systems [16], Web Services [12] and P2P [19] and different areas such as Government, Electronic Commerce [2], Health systems [5] and Industrial applications

[1] http://www.agentcities.org/
[2] prepared for the California HealthCare Foundation http://www.chcf.org

require security mechanisms. In previous mentioned technologies and applications have been made studies / approaches / ratings to include security elements, this fact reveal the importance of the concept security. The foundations of application security [21] are Cryptographic[3] and network security techniques [22](computer security). The experience obtained in these areas are / could be useful in other environments and therefore we have applied this experience to MAS. We can observe also the importance of security aspects in MAS in the last years with different studies [17], applications and practical X–Security [18] and Jade–S [23] elements.

Our objective in this paper is to show that is possible to work with a sure agent–based application. To achieve this objective we have applied X–Security package to the *Carrel* Application. X–Security implements authentication and secure communication among agents. How final outcomes we dispose of one agent-based application, multi–platform where the message are encrypted and each one agent has its identification. With this work / fact / effort we expect that the users / developers / programmers realize that elements of security in MAS exist and they could be useful.

The work we have carried out is documented in two technical reports. In the first paper [7], we described the *Carrel* system and identified security issues and counter measures. In this second paper we will exploit this analysis and address some of the security threats described in this scenario and we refine the cryptography and network security principles. The main idea is to relate security foundations with security elements on agent platforms using a concrete security tool for MAS.

2 Design

This section describes the security design system. The two possible options were X–Security [4] and Jade–S [23]. The first is a stand–alone agent that offers agent identity and ensures secure communication. The second tool approach as security by building features into an existing platform. In the process to choose the more adequate tool our principal requirement was that the work realised were useful for other applications with the mini possible change and with the maxim independence for the application. On the other hand the secure system had to be tensile to include new security elements into the future. With this requirements we decided to choose X–Security package. In the remainder of this paper we will describe how X–Security was used.

In the rest of the section, we describe the elements involved in our design as well as relevant theories in security area. These offer us the foundations, principles and practices useful in our approach. We will also explain how the

[3] http://www.ssh.fi/support/cryptography/
[4] http://agents.felk.cvut.cz/main/index.php

agent paradigm could provide new point of view in security elements and finally we define the security elements and design realized for the *Carrel* Application.

2.1 Approximation to theoretical security aspects

There are a number of well known security theories / models. The first of these theories is the OSI[5] security architecture (ISO 7498-2) which provides an useful approach in the design phase analysis since it defines a set of security services, mechanisms and description of attacks. Focusing on security *services* also makes relevant the following definition from RFC 2828[6].

"A processing or communication service that is provided by a system to give a specific kind of protection to systems resources; security services implement security policies and are implemented by security mechanisms."

Types of security services include:

- Authentication: The assurance that the communicating entity is the one that it claims to be.
- Authorization: This service provides protection against unauthorized use of resources.
- Confidentiality: The protection of data from unauthorized disclosure.
- Integrity: The assurance that the data received are exactly as sent without modifications.
- Nonrepudiation: The protection that an author of a message cannot deny sending this.

The relation between these and as X–Security confronts with these is depicted in Table 1.

The second foundation to consider is that the X–Security package could be seen as a Public–Key Infrastructure (PKI)([1], [13]). This second foundation take an agent approach because it interrelates agents and PKI. The PKI definition is:

"PKI is a security architecture that has been introduced to provide an increased level of confidence for exchanging information over an increasingly insecure network."

PKI refers to the combination of software, encryption technology and services that enable to organizations to protect the security of the information they exchange. This work provides us with experience / mechanisms such as

[5] http://www.iso.org
[6] http://www.armware.dk/RFC/rfc/rfc2828.html

Security service	How X–Security confronts it
Authentication	By means of creating / checking message signatures.
Authorization	The X–Security certificate include information related with the level of authorization.
Confidentiality	Using encrypted messages among agents.
Integrity	By using a message digest. This means that the system generates as seemingly random pattern of bits for a given input and if somebody modifies this message it is not possible to obtain the original message.
Nonrepudiation	Including message signatures the message.

Table 1: Relation between OSI security services and X–Security

Key / Certificate Life Cycle Management, Key and Certificate Management, Hierarchy and Distribution of Certification Authorities and so on. It has been used by us to design proposal solutions and describe problems in the process to include X–Security package over MAS.

A third relevant approach is named "Role–Based Access Control (RBAC) to information". [20] One of the most challenging problems in managing large networked systems is the complexity of security administration. Today, security administration is costly and prone to error because administrators usually specify access control lists for each user on the system individually. RBAC[7] is a technology that aims to reduce the complexity and cost of security administration in large networked applications.

With RBAC, security is managed at a level that corresponds closely to the organization's structure. Each user is assigned one or more roles, and each role is assigned one or more privileges that are permitted to users in that role. Security administration with RBAC consists of determining the operations that must be executed by persons in particular jobs, and assigning employees to the proper roles. Complexities introduced by mutually exclusive roles or role hierarchies are handled by the RBAC software, making security administration easier. This mechanism provide near elements to agents theory such as role and organization [10] what it makes interesting for its study in MAS environments.

2.2 Use of agents in security

Agent–based approaches is an useful technology for solving complex problems [14] and could provide a model from which a comprehensive and complete set of security services may be developed by means of agent paradigm [6]. Also it

[7] http://csrc.nist.gov/rbac/

observable the fact that the security could follow the three patterns of interactions in MAS such as cooperation, coordination and negotiation in agent theory.

But, what are the new contributions or advantages of this new paradigm respect to computer network security classic programs? We consider that the advantages offered by this new vision are implicitly inside of agents' characteristics [15]. With this vision, an agent is defined as a logical component of the security system, designed to implement a particular function or it groups to of functions. The functional modularisation of the system thus makes possible obtain a flexible security architecture. For example, we could develop agents such as Recovery Agent, Monitoring Agent, Security Services Agent, Security Certification Agent to achieve more security in our non–agent platforms and agent platforms by means of the agent paradigm.

2.3 Identification of the security elements

So far, we have explained how theoretical security foundations help design secure MAS and identified the list of elements that X–Security offers. They are:

1. Certificate Management: Manages the certificates for the agents. In the X–Security case it is named Security Certification Authority (SCA).
2. Message Encryption(8): Secures the communication between agents. Algorithms used are:
   - Message signing – SHA with DSA [8]
   - Message encrypting – RSA [12]

Certificate Management is used for securing information concerning the messages, communicated among two or more entities, such as their integrity, origin, time, and destination, can be protected by the provision of non-repudiation mechanisms. The potential conflicts about these issues may be resolved by a third–party, the notary, which must be trusted by both communicating entities. In order to perform its function, the notary must hold the necessary information to provide the required assurance in an identifiable manner.

Message Encryption: The previous algorithms (SHA, DSA and RSA) are asymmetric key algorithms family. In this type of algorithms two keys are used. The encryption key is named the *public key* and the decryption key is named the *private key*. With this system people / agent which wish to receive encrypted messages should publish their public key then the sender encrypt the message with his public key and only the person that have the private key will can decrypt the message. The main problem with approach is key distribution (transporting valid public–private key paris to users). To solve this problem we could opt for one of two approaches:

[8] http://home.pacbell.net/tpanero/crypto/dsa.html

1. Two entities involved in a communication already share a key, which somehow has been distributed to them, and

2. The use of a key distribution center.

X–Security uses the second option.

2.4 Design description

With this background, we are can go on to describe the first general design, depicted in Figure 1. The main concerns influencing this design are that *Carrel* must function open environment in which we have different MAS.

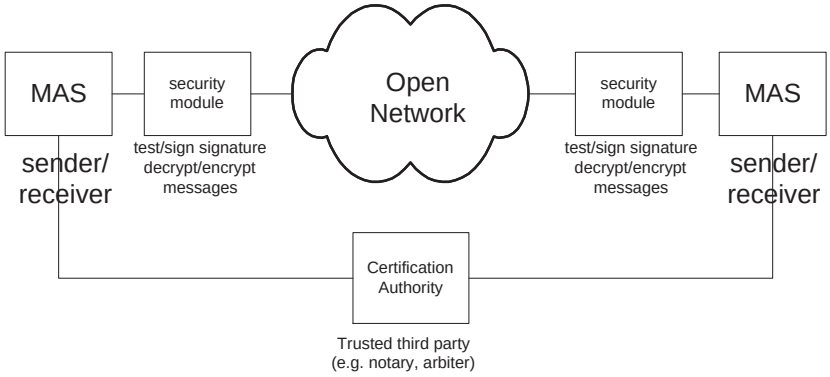

Figure 1: Security design on *Carrel*

3 Architecture

The environment and elements involved on the basis of the previous design we can now make concrete. This architecture is depicted in Figure 2 and the elements described are:

- Platforms: We have 3 platforms, Platform–1 and Platform–2 represent different UCTx applications (one per hospital) and Platform–3 represents *Carrel* application.

- SCA: Stores the certificates of different agents. The SCA acts like third-party and is on separate platform (Platform–3) because we want that the certification authority to be independent from the rest of the platforms which depend from the hospitals to which they are associated.

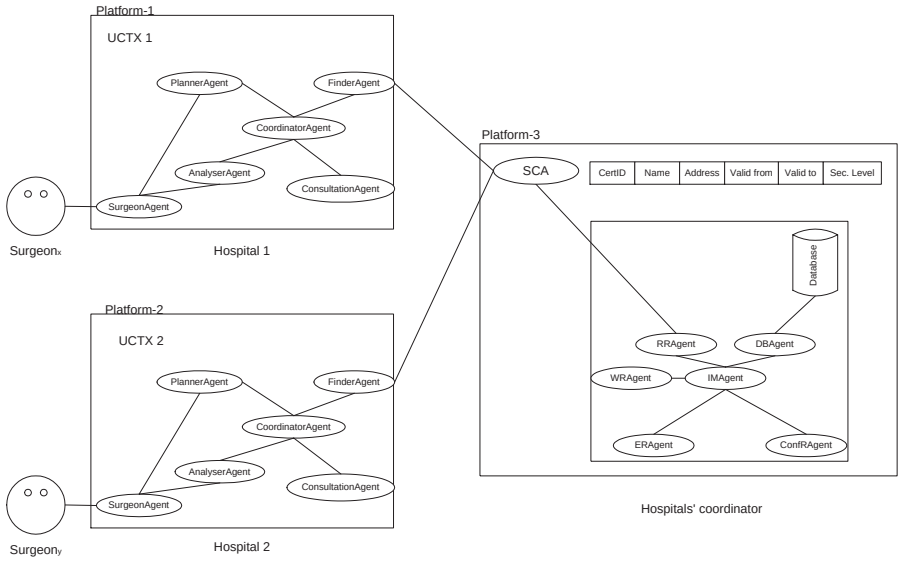

Figure 2: Security architecture on *Carrel*

3.1 Distributed Trust Architecture

An obvious starting point would be works with all the applications on the same platform (this is depicted 3). However an important issue is the need to assure security among different platforms.

Our first step was to include X–Security in our application with a simple design and we believed that the easiest way to make this was using one unique platform for all the agents in the system (we depicted this first approach in Figure 3). However this did not reflect application constraints. Our system (as was explained in [9]) is set up for different organizations where each one is separated geographically and is evident that each one of them is the owner of its platform / users / actions. For these reasons, is not possible to continue with this design.

Also X–Security does not work with multiple SCA on different platforms because there is no mechanism for certificate sharing among SCA's. On the other hand the certificates are necessary because these includes the keys for signing and encrypting messages. In order to solve this problem, X–Security' authors have extend the X–Security slot with a new element SCA AID containing AID (name and address) of SCA at which is the used certificate registered. But the whole problem is more complicated. We can not just only extend this slot, we should have to define some trust delegation between SCA's. This is necessary

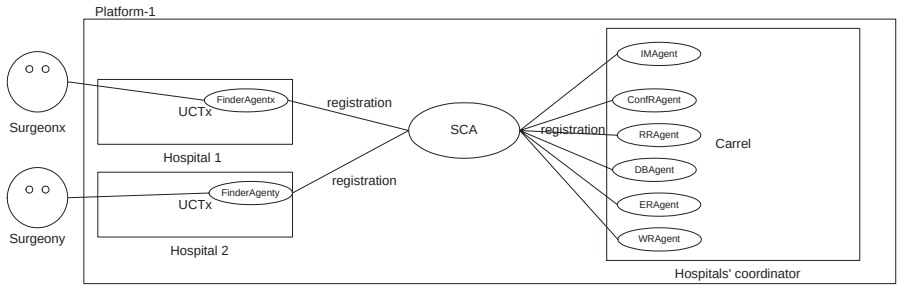

Figure 3: All of applications in the same platform makes security provision easier but is not realistic for deployment.

because otherwise anybody could start its own SCA and send an encrypted / signed message to your / other agent – and this agent will trust it, because it is signed. For this reason, we should design some SCA–trust–delegation mechanism to solve it.

There are a number of options for providing SCA services across multiple platforms:

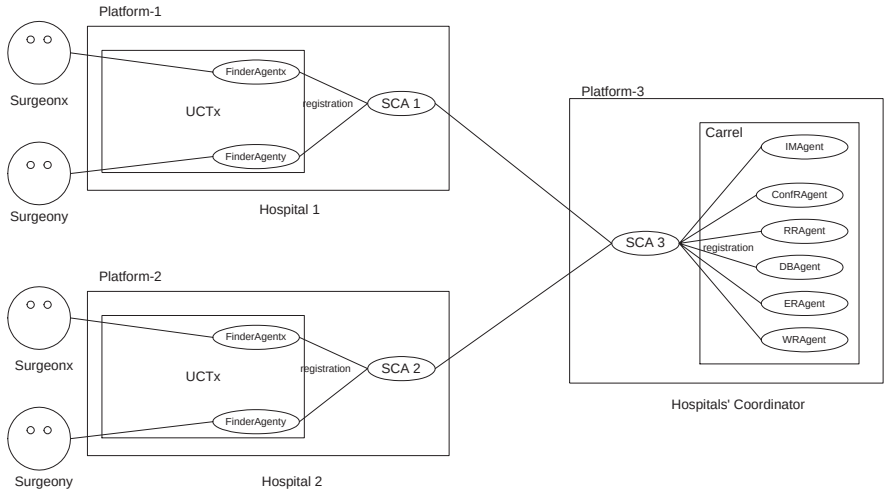

Figure 4: Second approach, distributed SCAs

Option 1 The idea, in this case, is to include the different certificates where they are needed; thus, if SCA_x is on $platform_x$ and it needs a certificate of

one agent in $platform_y$ with SCA_y is necessary to achieve this certificate and save it in SCA_x.

Option 2 To modify SCA so that it can work with different SCAs and in case that it does not find a certificate that it consulted in related SCAs.

Option 3 Create one unique authority that offers certifications. Using only one SCA for all agents in the same or in the different platforms. Imagine you have $platform_x$ and $platform_y$, we could start a new $platform_z$ with an own SCA and register all agents from both platforms x and y at this SCA in $platform_z$. At this moment agents in both platforms could send messages between themselves.

Of these options we have chosen option 3 because this offers a more accurate solution and behaviour between technical barrier and real application. X–Security does not provide certificate distributed (technical barrier) as it tries to make options 1 and 2, furthermore these options has another problem (real application), the fact that nobody will give its confidence on unguaranteed MAS.

4 Implementation

In this section we describe the changes necessary to use X–Security in Carrel and by extension we believe that the description is sufficient general for others MAS over Jade which would like to use X–Security. Annex D also describes the concrete modifications that were necessary for our application.

The SCA is a stand–alone agent that does not affect agents interaction, however it needs to start first. This is because the SCA is required like a security service, it provides by security module that is placed between the agent's core and communication layer.

The whole *Carrel* system is based on JADE 2.61 and is implemented in Java 1.4. Communication security is assured using X–Security package.

The Security class is the main communication security class in the X–Security package. This is the only class from the security package, ordinary user should deal with. If user wants to use communication security inside its agent, he / she should to create a new Security object and then start security either as Security Certification Authority using startSecCertAuth method or as an arbitrary agent using startAnyAgent method.

If the agent wants to sign or encrypt content of an ACLMessage, it can use setSignature or enCrypt methods. Messages are processed automatically, the user has to combine its the MessageTemplate with the MessageTemplate returned by the getWaitTemplate method and then pass the received message to the anyProcess method. This method returns the decrypted or tested method

or throws SecException if these tests fail. The security object can be shut down using endSecCertAuth or endAnyAgent methods.

The modifications to include X–Security are:

1 In the different agents from our application we must include the security package in order to start to work with X–Security. When is started it should register in SCA. For this reason we call the startAnyAgent which starts create security object. It is described in section 9.1.

2 Once the security package has been prepared we could use its different features. In section 9.2 we described for example how given of encrypt a message.

3 Finally, once an Agent sends signed / encrypted messages the last step is to treat these in the reception. For example, in the piece of code in section 9.3 we show how to receive a message. We should follow the same process in the rest of incoming message such as getMessage, extractContent.

5 Problems arising

In previous sections, we have related security theory and agents. Afterwards we have been described a tool that joins the two worlds. In this section we will describe concrete issues related to the use of this tool in MAS.

5.1 Communication

The idea of using asymmetric algorithms (public–private key) is that one element sends an encrypted / signed message and other element receives the message and decrypts / tests it. Therefore the basic model used is one–to–one. Nevertheless, in many MAS we have different scenarios with more complex patterns than one–to–one communication after , for example we have communications patterns and we would like send a message to group of agents or to have a protocol with different interactions among agents.

In the case that the message is addressed to multiple agents we will not known which public key should be applied over the message because we have more than one receiver and consequently more than one possible public key. We do not have the problem if it signed the message. To solve this problem (depicted in Figure 5) we have considered two options:

1. Create a group key. A key that is shared by a group of agents and is /should be used when someone would like send a message to this group (once to each receiver).

2. Send the same message n times. If the Agent would like to send a message to n receivers, we could do the same, sending the same message n times to the n receiver once.

We have chosen the second option, the main reason is that the first option is more complicated to manage. Is necessary to create, for example, a hierarchy of group keys and to include a timestamp to the keys in a group. The fact to use second option do not provoke much changes in our application.

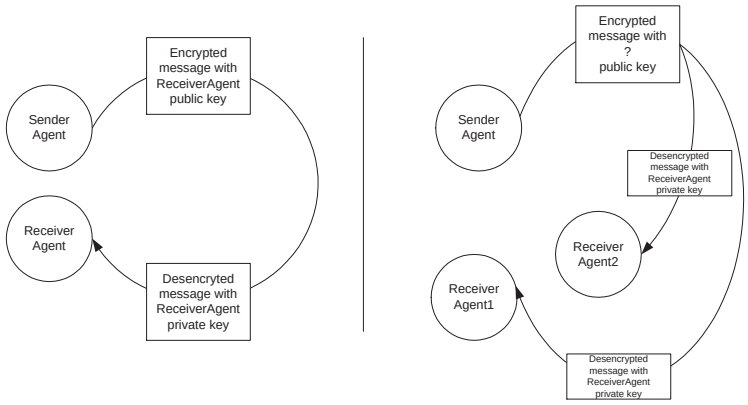

Figure 5: Communication 1 to 1 vs. Communication 1 to n

5.2 Fipa behaviours

The ACLMessage follows the FIPA 2000 "FIPA ACL Message Structure Specification" [9] specifications. In this case we would like to focus on two concrete behaviour implemented in Jade: FipaRequestResponderBehaviour and FipaQueryResponderBehaviour (or in new the Jade version which is AchieveREResponder). The first of these behaviours works as a dispatcher, reading the :content slot of received request messages and spawning different behaviours according to the action requested. To be able to handle requests, user defined behaviours must extend Action inner class; when a request message arrive, an implementation of Factory inner interface is retrieved from the registered action factories, using the action name as key. The problem observed is that if the message is encrypted and the agent uses these behaviours, when it tries to find out what type of action is inside of the content it produces an error (the same problem occurs if you do not use FIPA–SL[10] code in your message). To solve the problem we have observed three possibilities:

1. Modify the behaviour: Modifying the code of classes affected (class FipaRequestResponderBehaviour and FipaQueryResponderBehaviour) doing that before the reading the :content slot be decrypted the message (if it is necessary).

[9] www.fipa.org/specs/fipa00061/XC00061E.html
[10] http://www.fipa.org/specs/fipa00008/SC00008I.html

2. Wrapping the behaviour: Creating a level between the behaviour and the incoming messages systems which extends the behaviour to work with encrypted messages.

3. Making the platform itself responsible for encrypted / decrypted: Making that the platform encrypt / decrypt the messages between agent by means of Agent Communication Channel.

We have chosen the first option. We should modify the Jade core and it makes version dependent. The necessary modifications are included in section 10.

6 Deployment

The agents in the *Carrel* application should register their certificates (agent gains them at the creation time). The certificates are registered with SCA trusted by all community members.

During the registration phase, the agent sends its username and password to SCA. According to this data SCA sets up a security level. When one agent asks other agent to perform an action, the second agent checks competence of the first one by its certificate. Once layer has been inserted are able to assure that the rest is transparent for the user / agent implemented Agents only need to call the methods to encrypt / sign the messages that they send and call the methods decrypted / tested signature when they receive a message using security mechanisms.

In order to verify that X–Security actually addresses some of the threats listed in [7] our MAS over the threats, we have used different performances:

6.1 Correct performance

This performance is based on the normal expected behaviour of the *Carrel* system. In this scenario the agents communicate with SCA when they would like to obtain certificates of other agents or when they like to publish its certificates. In our case we have encrypted all. Inter–platform and intra–platform messages and deployed one unique SCA plug in where is *Carrel* is who manage the certificates among the different agents. We have tested that the communication among agents is encrypted. We have followed the whole different messages among the whole different agents checking that the system follow the functionalities and protocols.

We have tested different scenarios to assure that the behaviour of *Carrel* with security as good behaviour as in the previous version.

The first of these tests is when we have one agent that tries to obtain a piece. This is depicted in the first diagram of Figure 6 and we have observed that the behaviour is the expected one.

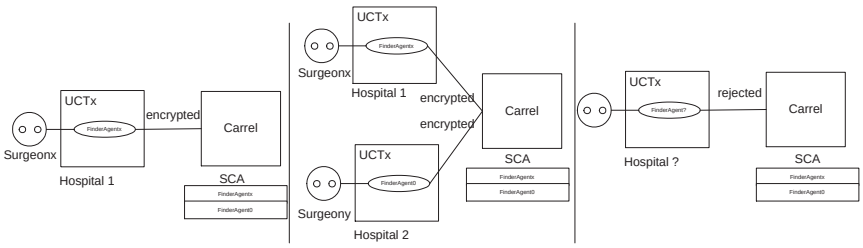

Figure 6: Different tests to assure that *Carrel* works correctly

The second test is over the same configuration but in this case we use two agents that try to obtain the same piece but the different between this two agents is one of them has a higher priority (urgency–0) the result is that this agent is selected as the one to obtain the piece (even if the agent with major priority arrives later in the auction of the piece). This is depicted in the second diagram of Figure 6.

6.2 Incorrect performance

In this performance we also checked failure cases. In particular those where we have some kind of problem such as incorrect certificate, expired certificate or different Security Level. All of these possible irregularities are related with the certificate and therefore we could observe the importance of security certification authority in all process related with the security in our MAS.

One test in this part is to verity that one agent that is not present in SCA is prevented from accessing the system. This is depicted in the third diagram of Figure 6.

7 Summary and Future Work

This paper presents a security architecture based an agent application. Our system specifically set two goals:

1. The protection of agents against the eavesdropping, and
2. The protection of the application against impersonation, unauthorised access.

Our future work reflect over the problems arisen in design section. In the next subsections we describe some of these possibilities.

7.1 Role Reputation

The structure organization and of institutions in *Carrel* mean that it could be very useful to devise a system based on of role reputation over the basic security presented here. For example, in our application we have different profiles / type of users such as coordinator, surgeons, guests and each one of them use / could use different functions.

7.2 Key Certificate Management

Due the importance of the keys in X–Security it would be useful to improve the certification management, including, for example automatic mechanisms for detection of expiration (or when this data is early to arrive). And to include new elements to sure that the owner of the key is available.

7.3 Intrusion Detection Systems (IDS)

Intrusion detection [13] is defined as "the problem of identifying individuals who are using a system without authorization and those who have legitime access to the system but are abusing their privileges".

In this area different groups: Software Technology Center[11], Autonomous Agents for Intrusion Detection[12] have applied (as is commented in [11]) agent–based approach and Intrusion Detection to detect anomalous or malicious behaviours by agents that performs a certain security monitoring function at a host.

Following the terminology used in [3] in the Figure 7 we have describe how Intrusion Detection could be treated in *Carrel*.

In the diagram 7 transceivers are entities that oversee the operation of all the agents running in their host on the other hand Monitors oversees the operation of several transceivers.

Protection, detection and response are mechanisms to ensure our system. The detection mechanism have been developed in initial state over *Carrel* we should to include new elements such as intelligent alert, audit information and so on to improve this mechanism.

[11] http://www.ipa.go.jp/STC/IDA/index.html
[12] http://www.cerias.purdue.edu/about/projects/aafid/

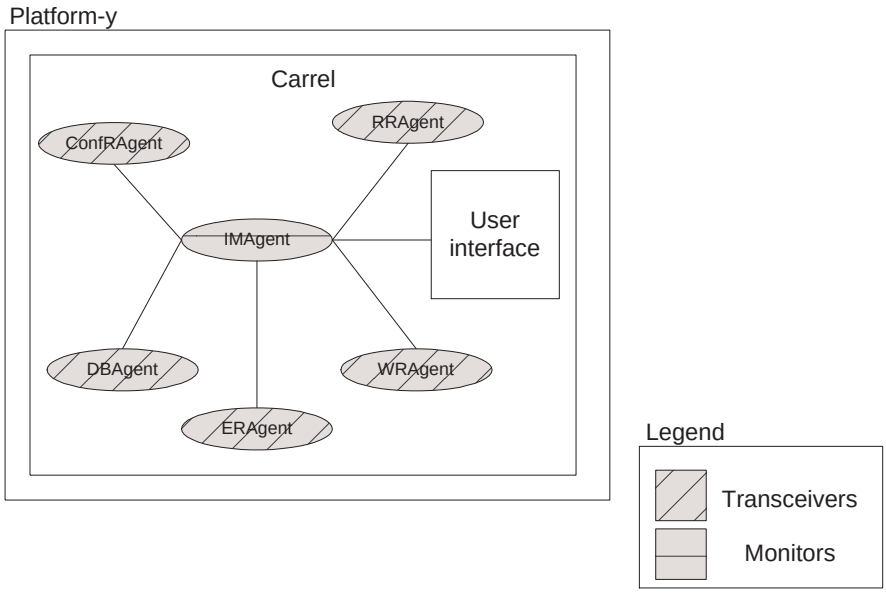

Figure 7: Intrusion Detection on *Carrel*

8 Annex A: A cryptographic example for an ACL message

In this short example we point out the difference between a simple text plain message and encrypted message. Should observe that only the content of the message is encrypted.

Text plain message:

```
(INFORM
 :sender  ( agent-identifier  :name RRAgent@visit18:1090/JADE
 :addresses (sequence http://visit18.lsi.upc.es:7778/acc ))
 :receiver  (set ( agent-identifier
 :name IMAgent@visit18:1090/JADE
 :addresses (sequence http://visit18.lsi.upc.es:7778/acc )) )
 :content  "((In_Room (agent-identifier
 :name Finder@visit18:1090/JADE
 :addresses (sequence http://visit18.lsi.upc.es:7778/acc )
 :resolvers (sequence ) ) 1 ) )"
 :language  FIPA-SL0
 :ontology  Institution-Management-Ontology
)
```

```
)
```

Encrypted message:

```
(INFORM
 :sender  ( agent-identifier  :name RRAgent@visit18:1090/JADE
 :addresses (sequence http://visit18.lsi.upc.es:7778/acc ))
 :receiver  (set ( agent-identifier
 :name IMAgent@visit18:1090/JADE
 :addresses
 (sequence http://visit18.lsi.upc.es:7778/acc )) )
 :content  "681F8EAF4D13C34447F07E341EE7A88006E98C9E1903C0E
E573805F26F117F3A7C555863164F43A10916BB9D47371FD33385F409FC
BCE04B6CDD93338786DD35A17DE1AAFF5DEB523664DCF3B3EA4BC988DB8
293849A6D73A31A16C0C5369319AFFE8BF40BE25F02C0A9AC95FCFF0E52
96E601A6CB7C5066C744B43A2E471EECCED32BFF414F5731614852EE2F6
3DE63098D44CFE58B18AE7AEFB216750E74CABE66F66F1CD491D0898C2F
6CA2BC9B625538F3CC648AD8483AB896E77032138B590515E43BFC987E0
E9A0C12D867BD82C3175E3C2A2BFD82E7A63181E63DE3C653F92682E250
03A01589F2379BC3CB1ABC108B4971E393D96F4FE4703D23424F"
 :language  FIPA-SL0
 :ontology  Institution-Management-Ontology
 :X-Security "( :type CRYPT
 :cert-ident SCA_CERTIFICATE_1 :key-ident CRYPT_1 )"
)
```

Also X–Security also adds a new slot it can be seen clearly named "X–Security" that specifies the type of encryption treatment which has been carried out on the message as well as the identification certificate and key.

9 Annex B: Including X–Security over MAS

In this section we point out the necessary modifications to make in order to use X–Security package in our application. Mainly, we should modify our application in three points described in the next subsections.

9.1 First point: Starting the agent

```
// create security for this agent
security = new Security(this);
// adding behaviour for this agent
addBehaviour(finder_behaviour);
SecConstants.setSCATimeout(1000000);
// waiting 20 second after starting agent
//(waiting for correct start of SCA)
System.out.println(""+this.getAID().getName()+"go to sleep...");
try {
```

```
   Thread.sleep(20000); } catch(Exception e) { /* exception */ }
   System.out.println(""+this.getAID().getName()+") ... wake up.");
   // starting this agent
   System.out.println("Starting NormalAgent ...");
   // starting 'Security' of this agent
   try {
      registered = security.startAnyAgent(sec_path, this.getAID(),
      sca_name, agent_user_name, agent_password);
      if(registered==false) {
         System.out.println("Registration with SCA - FAILURE");
         this.doDelete();
      }
   } catch (SecException e) { /* exception */ }
   // this agent is started
   System.out.println("NormalAgent started.");
```

9.2 Second point: Sending messages

```
try {// encrypt content
   security.enCrypt(reply);
} catch (SecException e1) {
   System.out.println(" Send crypted message - ERROR ");
   return;}
agent.send(reply);// send 'ACLMessage'
```

9.3 Third point: Receiving messages.

```
System.out.println("AnyAgentBehav:"+myAgent.getAID().getName());
System.out.println("--- Message received --- ");
// test for using security in this message
try {
   ((FinderAgent)myAgent).getSecurity().anyProcess(msg,this);
}
catch(SecException se) { // exception in security proces
   System.out.println(se.toString());
   switch(se.error) {
      // certificate not found, message mut be put back
      //to incoming queue and securty must wait for certificate
      case SecConstants.PROCESS_STOP: {
         System.out.println("AnyAgentBehav");
         System.out.println(myAgent.getAID().getName());
         System.out.println("--- Message putted back --- ");
         block();
         return;
      }
      // error in signature in message
      case SecConstants.SIGNATURE_ERROR: { break; }
      // error in encrypted message
      case SecConstants.DECRYPT_ERROR: { break; }
```

```
      // signature doesn't match sender
      case SecConstants.SIGNATURE_NOT_MATCH_SENDER:
      { System.out.println(" ### SIGNATURE is not from this SENDER ###");
      /*block(); return;*/ break;}
   }
}
 // security in message is correct
```

10 Annex C: Modification of the Jade core

We have focused over FipaRequestResponderBehaviour, but the changes are similar in other protocols which work as dispatchers requiring access to the content slot.

1. To include the security classes.

```
import ctu.mas.security.*;
import ctu.mas.security.onto.*;
```

2. In the action function, first of all, we should decrypt the message if it is necessary.

```
final public void action() {
msg = myAgent.receive(template);
if(msg!=null){
 ACLMessage message = (ACLMessage)msg.clone();
 String secslot = msg.getUserDefinedParameter("Security");
 if(secslot!=null){
 try{
  Security result=null;
  try {
   Method concatMethod;
   concatMethod=(myAgent.getClass()).getMethod("getSecurity",null);
   result = (Security) concatMethod.invoke(null, null);
  } catch (Exception e) {
   System.out.println(e);
  }
 result.anyProcess(message,this);
 }catch(SecException se)
 { // exception in security proces}
```

3. Our agents should implement a static method that returns the security variable.

```
public static Security getSecurity() {
 return security;
}
```

11 Annex D: Development details for Carrel

This annex describes concrete aspects related with to developed of *Carrel* focusing in two important aspects:

1. Upgrade to Jade
2. Functionalities offered by *Carrel*

11.1 Upgrade to Jade

For compliance with FIPA 2000 specifications it was necessary the team Jade to make some modifications to its API. This fact also forced the move from the Jade 1.X to 2.X version. Also the new components / 3rd party elements work with 2.X. version such as plug–in the MTP (Message Transport Protocol) implementation of HTTP [8] and so on.

During the realization of our design we thought about the necessity to upgrade the version of Jade over that works *Carrel*. Initially, *Carrel* was developed for the Jade platform 1.X but the fact that the new components does not work with this version they made us reconsidered which were the possibility about realize an upgrade of platform in *Carrel* application. First at all, we depicted the different options in the next list and drawn in Figure 8:

1. Maintain Carrel using Jade platform 1.X version: In this case, the idea is to deploy in different platforms, *Carrel* on one platform 1.X and the rest of elements on platform 2.X running Jade. With this solution we avoid need to update Carrel application but not the fact of modify it. This idea tries to disjoin our application and security systems running on different platforms but is necessary to communicate them. For this reason, it would be necessary to design the different protocols to offer the Carrels' services.
2. Update Carrel on Jade platform 2.X version: This option is more challenging but also the most durable. And our hope is that with this work the next *Carrel* generation would be in a better position for future evolution. We believe also that this step, was a forced step, because the update of the versions is inherent in applications such *Carrel* that are in developed phase.

The second option must chose the most important reason being the increased utility for future development is more durable into the future. Once decided, we describe the most important changes related to this work to update our application here (we also used [4] to guide the changes necessaries):

In version 1.X it was possible to call the function ReceiverBehaviour with msg parameter.

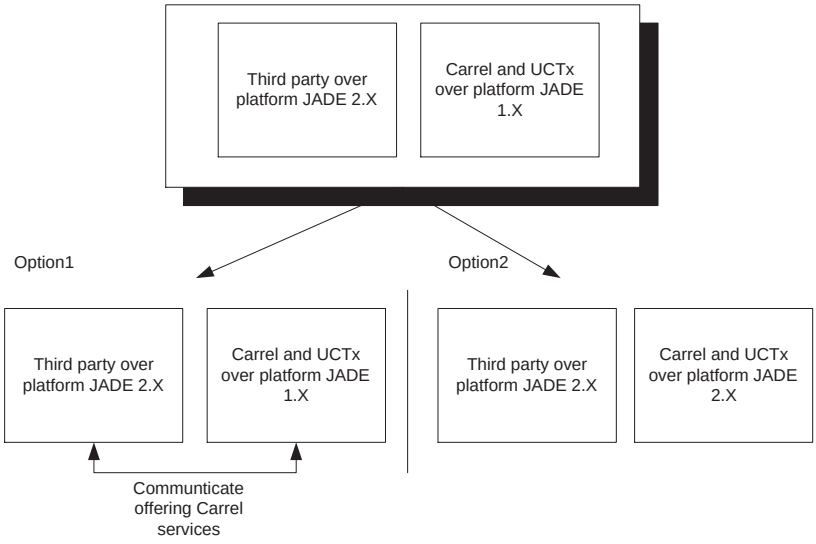

Figure 8: Elections about Carrel upgrading

```
public void action() {
 ReceiverBehaviour b = null;
 if (bReceiveNewMessage) {
  b = new ReceiverBehaviour(myAgent, msg, mt);
   myAgent.addBehaviour(b);
   bReceiveNewMessage = false;
```

In version 2.X, ReceiverBehaviour object that ends as soon as an ACL message matching a given MessageTemplate arrives or timeout expires if you do the same that in version 1.X you should to say in ReceiverBehaviour function that an infinite timeout it is be expressed by a value less than 0.

```
public void action() {
 try {
 if (bReceiveNewMessage) {
  b = new ReceiverBehaviour(myAgent, -1, mt);
  myAgent.addBehaviour(b);
  bReceiveNewMessage = false;
 } else {
  msg = b.getMessage();
```

In version 1.X we have the notion of ComplexBehaviour.

```
ComplexBehaviour finder_behaviour = new SequentialBehaviour(this);
```

In version 2.X we do not have ComplexBehaviours for this reason is necessary to change to:

```
SequentialBehaviour finder_behaviour = new SequentialBehaviour(this);
```

In version 1.X we can use addSubBehaviour.

```
parent.addSubBehaviour(new CA_RequestInitiatorBehaviour(myAgent, request, mt));
```

But in version 2.X we should change the last sentence for:

```
SequentialBehaviour s = new SequentialBehaviour();
s.addSubBehaviour(new CA_RequestInitiatorBehaviour(myAgent, request, mt));
myAgent.addBehaviour(s);
```

Despite of the different changes we should used -deprecation option when we compile our files because we have some deprecated methods. Nevertheless the application works well.

11.2 Functionalities offered by *Carrel*

The most important for us is to have a stable version of *Carrel* even if not all in the functionalities offered in his first version are present. At a first glance, we could see the stable version such a subgroup the all functionalities that work adequately and it serves to us such foundations.

Although the transformation of *Carrel* between Jade version has been successful a number of changes still remain to be done. The most important of these is restablishing *Carrel's* use of the databases storing patient data, piece data and so forth. The connection was not made because other work is currently being carried out by Damien Bousissou to change the database schemas used in the Oracle 8i database.

Therefore the upgraded version of *Carrel* only functions with dummy data rather than a connection to a complete database. The connection should be relatively easy to establish once the database schema work is complete (a matter of replacing a number of function calls).

References

[1] C. Adams and S. Lloyd. *Understanding Public–Key Infrastructure: Concepts, Standards, and Deployment Considerations*. New Riders Publishing, November 1999.

[2] S. A. Baker and P. R. Hurst. *The limits of trust: cryptography, governments, and electronic commerce*. Kluwer Law International, Boston, 1998.

[3] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. H. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC*, pages 13–24, 1998.

[4] F. Bellifemine and T. Trucco. How to upgrade user code to jade 2.0, January 2000.

[5] B. Blobel. Onconet: A secure infrastructure to improve cancer patients care. *European Journal of Medical Research*, 5(8):360–368, August 2000.

[6] W. Buchanan, A. Scott, M. Mannion, M. Naylor, and J. Pikoulas. *Agents, Network Security and Network Management, Distributed Systems and Networks*. Addison-Wesley, 2000.

[7] D. Cabanillas, S. Willmott, and U. Cortés. Threats and security safeguards in a multi-agent system medical applications. Technical Report LSI-02-76-R, Software Departament. Technical University of Catalonia. Barcelona Spain, http:/www-lsi.upc.es/dept/techreps/ps/R02-76.ps.gz, 2002.

[8] I. Constantinescu. How to use the http mtp with jade, May 2001.

[9] U. Cortés, A. López-Navidad, J. Vázquez-Salceda, A. Vázquez, D. Busquets, M. Nicolás, S. Lopes, F. Vázquez, and F. Caballero. Carrel: An agent mediated institution for the exchange of human tissues among hospitals for transplantation. In *3er Congrés Català d'Intel.ligencia Artificial*, pages 15–22. ACIA, 2000.

[10] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., April 1999.

[11] L. N. Foner. A security architecture for multi-agent matchmaking. In *Second International Conference on Multi-Agent Systems*, pages 80–87. ICMAS, AAAI press, 1996.

[12] S. Garfinkel and G. Spafford. *Web Security, Privacy & Commerce*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, second edition, 2002.

[13] Q. He, K. P. Sycara, and T. W. Finin. Personal security agent: KQML-Based PKI. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 377–384, New York, 9–13, 1998. ACM Press.

[14] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[15] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[16] D. Marti. Book reviews: *Real World Linux Security: Intrusion Prevention, Detection, and Recovery. Linux Journal*, 86:68–69, June 2001.

[17] S. Muftic, editor. *Security Architecture for Open Distributed Systems*. John Wiley and Sons, Inc., New York, NY, USA, 1993.

[18] P. Novák, M. Rollo, J. Hodík, T. Vlcek, and M. Pechoucek. X-security architecture in agentcities, 2003.

[19] A. Oram. *Peer to Peer*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, March 2001.

[20] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[21] B. Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley and Sons, Inc., New York, NY, USA, 2000.

[22] W. Stallings. *Cryptography and network security: principles and practice*. Prentice-Hall, Inc., Upper Saddle River, NJ 07458, USA, second edition, 1999.

[23] G. Vitaglione. Jade tutorial–security administrator guide, September 2002.