A Practical Security Infrastructure for Distributed Agent Applications

Lars Braubach, Kai Jander, and Alexander Pokahr

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{braubach,jander,pokahr}@informatik.uni-hamburg.de

Abstract. Security is a vital feature for most real-world distributed applications. For applications using an agent-based middleware, comprehensive support for many security-related aspects can and should already be provided by the platform in a holistic manner. Moreover, security in practice does not only concern the enforcement of well-known security objectives like authenticity and confidentiality but also requires a simple yet effective usage concept that renders it easy for developers to define their security requirements. In this paper a security concept and implementation for multi-agent systems is introduced, which focuses on external, i.e. inter-platform, security aspects. The solution encompasses a usage concept distinguishing security intents from realization details and allows service providers as well as service clients to impose security constraints on communication relationships. The security concept requirements have been elicited from and the results have been evaluated with a real-world company project in the area of distributed workflow management in business intelligence.

1 Introduction

Distributed systems allow applications to be used in a widely distributed manner, which confers unique advantages over more centralized or server-based approaches such as increased performance and fault-tolerance. These systems offer application platforms, which are capable of executing parts of the software and enabling the communication between them and other platforms in a convenient manner for the developer. However, a large number of real-world applications especially in sectors such as banking, health and communication are mandated, sometimes by law, to fulfill certain requirements regarding the security of the system. Such requirements are frequently relegated to an afterthought, for example by encrypting traffic using HTTPS, which often fails to address application requirements such as fine-grained access control. In some cases, security requirements are ignored by the software platform and thus deferred to the application level, which means that every application is forced to reimplement its own security measures. Given that initial implementations often have flaws, this amplifies the problem by given each application its own chance to include the same flaws and requiring separate modification to address the problem.

M. Klusch, M. Thimm, and M. Paprzycki (Eds.): MATES 2013, LNAI 8076, pp. 29–43, 2013.

As a result, it would be beneficial for the platform itself to offer a useful set of security features that can be employed by all applications being executed on the platform. While some features are too specialized to be provided in a generalized manner, a number of features can be offered which are often not included in such platforms. The first step towards implementing security features in a platform is to ensure that basic security is maintained even in an only partially controlled environment that is used by multiple stakeholders. Specifically, this means that platforms should *restrict access* to functionality by default and only allow it when specified. This allows the platform to prevent unauthorized access and achieve specific security goals[11]. In addition to the access restriction, some applications also require that *non-repudiation* can be established. This means that actions taken on a platform by an entity can be accounted for and it can be proven that specific service requests were placed by specific issuers.

This first step allows the inclusion of security features from the perspective of a user of the MAS infrastructure, such as an application developer. The first of these user security objectives is the *integrity* of communication between platforms. This means ensuring that an attacker will be unable to tamper with messages without the target platform noticing the change. If the communication is performed using a public network, the issue of eavesdropping needs to be accounted for by establishing *confidentiality* of the data exchanged between platforms. Since platforms restrict access by default and allow only some entities access to its functionality, the *authenticity* of the entity needs to be established. After authentication has been established, the *authorization* of the entity to perform a function needs to be verified.

In addition to the user security objectives, the usability of the security model provided by the platform must be part of the overall security concept of the system and balanced against the security requirements[12]. Here, two groups are particularly important. First, application developers should be offered easy-to-use APIs and configuration options to integrate the offered features in their application. Second, administrators of nodes that run the platform must be offered management tools, for example to generate and distribute certificates.

This paper proposes a platform-based security model for agent applications that is motivated by a real-world usage scenario which will be introduced in the next section. In Section 3, related approaches will be presented. The security concept for the platform-based approach will be introduced in Section 4, followed by an elaboration of the implementation of the approach in the Jadex agent platform in Section 5. The approach is then evaluated in Section 6 based on the real-world scenario, followed by a conclusion in Section 7.

2 Real World Scenario

In a commercial project called DiMaProFi (Distributed Management of Processes and Files) carried out with the company Uniique AG[1] a new business

[1] `http://www.uniique.de/`

intelligence tool is currently under development. The tool will target the modeling and automated execution of distributed ETL (extract, transform, load) workflows. These kinds of workflows serve the purpose of collecting and preprocessing data and files and finally move them into a data warehouse, which is subsequently used by domain experts for sales and other business evaluation tasks. The targeted scenario is naturally distributed, spanning often more than one network as the relevant source data is generated at different customer sites.

In such a setting many security related problems arise:

- The DiMaProFi application is distributed and needs to work on different nodes, on the one hand to bring about its functionality and on the other hand to save time by distributing the system load. This means that the nodes of the system have to form an overlay network which permits access to the members but prohibits access from other nodes and protects the integrity and confidentiality of the exchanged messages.
- Another important aspect is that the system needs to have access to password protected resources like databases and also the target data warehouse. To avoid the distribution of passwords among all participating nodes, a mechanism for acquiring authentication at runtime needs to be provided. In DiMaProFi, a specific password safe is used which can be interrogated by authenticated parties to obtain credentials for access restricted resources. For this approach it is important to support origin authentication of the different parties at the password safe and also provide confidential messaging to protect credentials from being eavesdropped.
- Regarding the communication with the customers it is important that in error cases the activities of the system have been exactly monitored and recorded. On the one hand this facilitates the failure recovery and on the other hand it might be important with respect to legal responsibilities concerning the service level agreements with the customer. For this purpose the correctness of the monitoring data is significant and should be safeguarded by a non-repudiation mechanism.

From this description it becomes obvious that in distributed applications like DiMaProFi security objectives become vital and mechanisms for access restriction, authentication, integrity and non-repudiation need to be established. These objectives are also considered in literature to be of primary importance [7] for distributed system security.[2] Moreover, these security solutions should be practical in the sense that they are easy to install, configure and maintain, i.e. security should not become a complicated overhead possibly fostering the bypass of important mechanisms.

3 Related Work

While security features are a frequent requirement for applications, security aspects are often either overlooked or only superficially considered when designing

[2] We decided to exclude availability here as it is more related to system design as a whole in contrast to basic design entities like agents.

multi-agent systems [8]. Most platform-based systems rely on the use of standardized or established algorithms such as AES [10] and RSA [16] and protocols like SSL/TLS [3] and Kerberos [9] to meet the requirements of the applications. Since designing both security algorithms and protocols always carries the risk of flaws and established approaches have been used and vetted for many years, using established solutions is generally a good idea. However, the approaches often lack good integration with the platform, resulting in a system that is difficult to configure and use.

Regarding distributed systems, one can distinguish two classes of security: Internal security, which attempts to enforce restrictions between components such as agents executing on the same platform, and external security, which centers on the communication between different platforms and the enforcement of restrictions between platforms. Generally speaking, internal security is fairly hard to enforce since code executed within the same process instance is usually able to circumvent security measures, either through approaches like reflection APIs or by simply reading or manipulating the process memory. A notable exception are sandboxing approaches [14], which attempt to encase certain parts of locally executed code by restricting its API access and memory reading capabilities in order to contain the sandboxed part of the application within a known scope of permissions.

A prominent example of this approach is the Java Security Manager [5], which allows the application of method-based restrictions on parts of the executing code. However, since this approach relies on complex code analysis and interpreter assistance, it is prone to include subtle flaws [4]. As a result, internal security remains difficult to maintain. Furthermore, basing security policies on the method level is often too fine-grained to be easy to configure and often allows attacks if not done with extreme caution. This complexity can be considerably reduced when considering external security.

Nevertheless, some agent-based approaches attempt to enforce internal security in addition to external security. For example, the JADE-S add-on [17] for the JADE agent platform allows the user to restrict access to platform services using access control lists (ACLs) and user authentication. While this addresses one kind of authentication with regard to administrative aspects of platform management, there is no mechanism to assist authentication for agent interactions and services. JADE-S also offers the option to encrypt and sign messages between agents, potentially opening the possibility of agent authentication and message confidentiality. However, this is supported by key pairs held by each agent. This means that enforcing inter-agent secrecy on the same platform requires the enforcement of internal security with the aforementioned complexity and there appears to be no readily available function for key distribution, which greatly increases administrative overhead and reduces usability. This problem is magnified by the lack of scalability because each requires a key pair.

Other agent systems attempt to address the problem of keeping confidentiality between platform and agent or between multiple agents. The concerns here are less about communication but rather the danger of compromising the

execution space of either the agent or the platform. For example, in [6] a method is proposed which lets mobile agents perform computations on encrypted data, preventing the platform from gaining access. In the opposite direction, sandboxing approaches are once again chosen to maintain internal security of the platform with regard to the mobile agent. Similarly, the use of controlled query evaluation (CQE) techniques is proposed in [1] in order to prevent agents from revealing confidential information during interactions. These approaches focus primarily on maintaining the confidentiality either within the same MAS application or between executing platform and agent while the approach presented in this paper focuses on maintaining the confidentiality of inter-agent communication against an outside attacker.

Due to the difficulties associated with the enforcement of internal security through sandboxing, many approaches forego this approach, instead focusing on the easier-enforced external security similar to the approach presented in this paper, which relies either on well-tested operation system process security, virtualization or physical separation of machines to aid in access restriction. Since additional platforms can always be added automatically as additional processess, this reduces the implementation difficulties of sandboxing by offloading the issue to the operation system process security, which receives a higher degree of scrutiny. For example, WS-Security provides features for encryption and signatures which can be used to secure web services [13]. This approach provides support for external security for web service calls. However, while aspects such as certificate formats and the use of security tokens is part of this approach, it does not offer an easy key distribution mechanism. Furthermore, authentication relies, in large part, on the application layer with support mostly provided for basic aspects such as confirmation of signatures.

4 Security Concept

In this section, the security concept is introduced. First, the scope of the security concept is explained, i.e., the setting in which security measures are applied and which security objectives are supported. Afterwards details of the security model are illustrated to show how the security objectives can be achieved.

4.1 Scope of the Security Model

Multi-agent systems can be deployed in various settings ranging from simple closed networks to internet scale distributed systems. The security model presented here aims at open distributed systems, in which agents are used to realize some of the systems functionality. This functionality is exposed to the outside as services, which e.g. may be accessed using message-based interaction protocols. It is assumed that agent platforms represent the nodes in the network and each agent platform is under some administrative control.

The focus is on security issues due to agent services being accessed from the outside (cf. Fig. 1). With regard to the usage of these services four common security objectives can be identified.

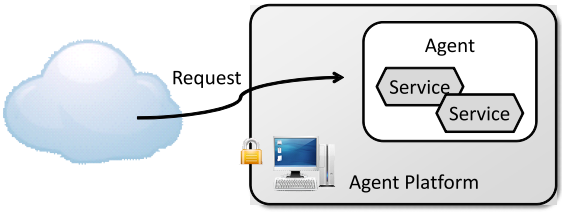

Fig. 1. Basic security model

Access Restriction: The functionality must only be provided to those who are allowed to use it.

Authentication: The agent should be able to uniquely identify the user of its functionality, e.g. for accounting purposes.

Confidentiality The usage of the functionality as well as private data should not be visible to others.

Non-Repudiation Users of the functionality should not be able to deny that a certain request was used.

Due to the focus on outside requests, certain security issues are intentionally not covered in the model. First, issues of mobile code are excluded, because mobile agents are considered unnecessary for many real world applications. Moreover, security issues due to an attacker having direct access to the computer running the agent platform are not considered. Such an attacker might be able to install malicious code or sniff sensitive data from local memory. Yet, these kinds of attacks do not require agent-specific solutions, but can be already covered with user-based access control, anti-malware-checkers, etc.

4.2 Security Model Details

Security objectives are related to individual services. Each service may require a different combination of security objectives. E.g. instantiating a new data management process in DiMaProFi only requires access control but no confidentiality, because no sensitive data is transferred. In a later step of the process, confidentiality might become necessary, e.g. when the process accesses a sensitive customer file.

Therefore, security policies are introduced to allow fine-grained, yet easy to use configuration of security objectives. These security policies may be specified both at the sender as well as the receiver side (cf. Fig. 2). Typically, the receiver side providing the service will already specify the general security objectives that apply to any usage of the service. Therefore, the receiver can choose to demand that all access to the service must be authorized, that data must always be transmitted confidentially etc. Furthermore, the sender side can supply a custom policy for each request to demand further security measures. For example, if a service does not demand confidentiality in general, the sender can still request that communication be encrypted for a specific interaction.

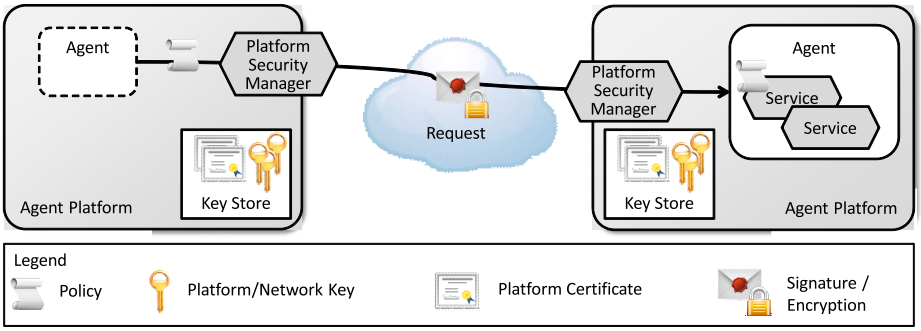

Fig. 2. Security model details

To achieve the desired security objectives, a security manager is introduced that operates as part of each agent platform. This approach allows all security-related functionality to be handled by the agent platform itself and relieves agent programmers from tedious and time-consuming implementations of security details. The platform security managers are responsible for processing the requests before they are sent and after they are received. Each request between two agents is thus routed through two platform security managers, one at the sender and one at the receiver side (cf. Fig. 2). In the following, for each security objective it will be described, how it can be realized inside the platform security managers using further security concepts, such as keys and certificates.

Access Restriction. By specifying an according security policy for an agent service, a developer can control, if agents from other platforms may access the service. For simplicity, the model allows two modes of access restriction. One that requires authentication and one that does not. The simple mode without authentication relies on the concept of trusted platforms and trusted networks as described below. The idea is that an agent platform should allow access to all services of its agents when requested by remote agents from trusted platforms or from platforms in trusted networks. All other agents are only allowed to access services, which have been explicitly marked as public by the developer. The validation of trusted platforms is done by using platform and network keys, which are kept in a key store on each platform. The difference between platform and network keys is that a platform key allows only access to a single platform, while a network key allows access to a logical network of platforms. A platform only has one own platform key but it may participate in any number of trusted networks and thus may have multiple network keys. In addition to its own platform key, the key store also contains the known keys of remote platforms.

An example is shown in Fig. 3. Platform A has a local platform key as well as network keys for networks *N1* and *N2*. As it shares network and key for *N1* with platform B, platform A will consider B as trusted. Similarly, platform E is trusted by A as both share the key of network *N2*. Furthermore, platform G

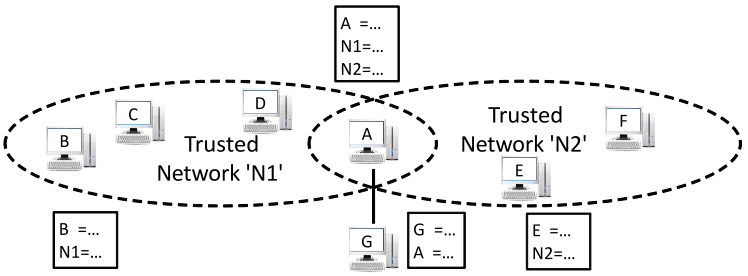

Fig. 3. Trusted platforms and trusted networks

is trusted by A, because G is in possession of A's platform key. Due to their disjoint network memberships, platforms B and E would not trust each other.

Whenever a request is issued to a remote platform, the security manager on the sending side will intercept the request and enhance it with digests of the relevant platform and network keys. The security manager at the receiver side also produces digests of its local keys and checks if one of them matches the sent digests. If a match is found, the platform can be considered as trusted as it shares a common secret with the sender. Due to the use of digests, no keys need to be transferred and therefore cannot be sniffed by eavesdroppers. Furthermore, using timestamps as part of the digested text provides protection against replay-attacks.

The second access restriction mode with authentication allows more fine-grained control over which services may be accessed by which remote platform. This can be specified in two ways. First, the names of allowed platforms can be annotated in the service policy. Second, virtual group names can be specified that are later mapped to concrete platform names. The first way is simpler, but may introduce maintainability issues as changes in platform names may require security policies to be updated. The second way allows for role-based access control as the virtual names can be used to represent roles. Therefore, developers can specify which roles are allowed to access which services and administrators later configure, which platforms are allowed to play which roles. The details of the authentication mechanism are described next.

Authenticity and Integrity. Authenticity is about establishing the identity of a communication partner and integrity means that no tampering with message contents is possible. They come hand in hand as requirements for other security objectives such as access control and non-repudiation. In the presented security model, authenticity and integrity are established by the use of digital signatures and platform certificates. When authentication is required, the security manager of the sending platform will add a digital signature to the message, which is produced by the private key of the platform. To validate the authenticity and integrity using the digital signature, the security manager of the receiving platform uses the platform certificate, which contains the public key of that platform.

The means that only when the receiving platform already has the certificate of the remote platform, the digital signature can be validated. Therefore an important aspect of authentication is the distribution of platform certificates.

A common and safe but tedious way is to manually install the certificates. In a network of n platforms, $n * (n - 1)$ installation operations would be required to be performed by platform administrators. Therefore alternative solutions are provided to simplify the process. These solutions allow obtaining certificates during the process of validation a signature and thus are called certificate acquisition protocols. The first protocol uses a certificate server as a trusted third party. Due to the central server only $n*2$ operations would be required to install each platform certificate at the server and also install the server certificate at each platform. An additional advantage of the approach is that when adding a platform, only the newly added platform and the server need to be considered as no additional certificates need to be installed at previously existing platforms.

The second protocol is based on trust relationships to other platforms. It realizes a consensus mechanism that asks a set of available platforms for a specific and currently unknown platform certificate and compares the received results. If a configurable threshold of platforms has delivered the same certificate it is accepted and added to the internal key store. The number of manual installation operations of this protocol directly depends of the chosen threshold for accepting a certificate. In the best case (threshold=1) 0 manual installations are necessary, while in general with (threshold=x) $(x - 1) * n$ operations are required.

In order to reduce the efforts of manual installation to a minimum a two-staged process can be used. In the first stage the network of platforms, e.g. running in a company intranet, has to be cut-off from the internet so that so no malicious platforms can participate. In this initialization phase the certificate acquisition has to be enabled at the security manager of each platform and e.g. the consensus protocol can be used with threshold=1. After having set up the network a complete certificate exchange is started in which each platform requests certificates of all other visible platforms. After the exchange has finished the initialization phase is completed, the certificate acquisition can be turned off and the internet connection can be reestablished.

Confidentiality. Confidentiality can be specified at the receiver side for all requests to a service as well as at the sender side for a specific request. In both cases, confidentiality means that all communications pertaining to the service usage (i.e. requests as well as replies) should be encrypted to prohibit outsiders to gain access to the exchanged information. When performing confidential communication, the platform security managers are responsible for ensuring that the corresponding messages are sent through secure channels. For confidential communication between different platforms, the use of secure message transport protocols is obligatory. If no such transport is available, the sending agent is notified about the request failure and the message is discarded.

Non-repudiation. Non-repudiation means that it can be proven that a certain request was issued by a certain party. To achieve this it is required that

authenticity and integrity of requests can be established. To prove the issuance of a request at a later point means further that all relevant requests should be logged. This is achieved using a mechanism for monitoring the service invocations. For each agent it can be specified, which services are of interest and thus which requests need to be written to the log.

4.3 Usability Concerns

One main focus of the presented security model is ease of use at two levels - for the agent programmer and for the platform administrator. The presented model is easy to use for agent programmers, because security issues are specified as non-functional properties. As a result, the agent implementations do not need to deal with security issues leading to an advantageous separation of concerns. Furthermore, using the non-functional properties, security configurations can easily be added to existing agent implementations. Usability is further facilitated by the flexibility and simplicity of the model. For example, for access control, the developer can choose from three access modes: public, trusted, and authenticated access. The public access mode allows easily enabling global access to non-critical public services without compromising security of critical services. The trusted access mode is enabled by default thus automatically protecting any service implementation without further effort from unauthorized access. Therefore, agent platforms and applications can safely be hosted in open networks without having to consider security at first. Finally, authenticated access allows fine-grained role-based access control, if required.

For administrators, usability issues arise as the platforms need to be configured with two conflicting goals in mind: 1) keep the platform as closed as possible to prevent any malicious activities and 2) keep the platform as open as necessary for the distributed application to operate correctly. The concept of trusted networks provides a simple solution for this conflict that is applicable to many real world situations. By establishing a trusted network, an administrator can easily allow platforms from heterogeneous company networks to interoperate without exposing their functionality to the outside. More control about which platforms are allowed to access which services can be exercised by using platform certificates, which come at the cost of some administrative overhead for managing local and remote certificates on each platform. To ease the administrative burden of certificate management, the model proposes protocols for (semi-)automatic certificate exchange.

5 Implementation Aspects

The proposed security concept has been implemented in the Jadex platform [15]. The platform uses an extended agent concept called active components, which in essence allows agents to expose and use explicit services with asynchronous object-oriented interfaces. Regarding the usage of security aspects, a distinction is made between the security objectives and their enforcement, which is the task

```
@SecureTransmission
@Authenticated({"DatabaseUser", "Admin"})
public interface IPasswordSafeService {
  public IFuture<Credentials> fetchCredentials(String ressourceid);
  ...
}
```

Fig. 4. Provider side security usage example

```
IFileRegistrationService service = searchService();
ServiceCall call = ServiceCall.getInvocation();
call.setProperty(SecureTransmission.SECURE_TRANSMISSION, true);
service.registerFile(file);
```

Fig. 5. Client side security usage example

of the platform's security manager realized as security service. The specification of security objectives differs for service provider and client side.

5.1 Provider Side Usage

The specification of aspects at the provider side is handled with different Java annotations that can be attached to a service method or the service interface itself (which means that the objective is applicable for all methods of the interface). In Fig. 4 it is highlighted with an example how security annotations can be used. In this case a cutout of the DiMaProFi password safe service is shown, which is besides other things in charge of providing credentials for agents that need to access password restricted resources like databases. In order to make all methods confidential and authenticated the corresponding annotations (`@SecureTransmission` and `@Authenticated`) are attached to the interface `IPasswordSafeService`. Additionally, the authentication is parameterized with two roles ("DatabaseUser" and "Admin"), which are allowed to access the password safe. These role names are mapped to concrete platform names in the security manager of the platform.

5.2 Client Side Usage

At the client side security objectives can also be requested dynamically.[3] This is achieved by a concept similar to a thread local[4] but not based on a thread but on a service invocation, i.e. a specific service call object allows for equipping an invocation with additional meta data that is automatically preserved between caller and callee during the complete call, regardless if the call is performed locally or remotely. The service call object can be fetched and attributed with security objectives by the caller before a service call is issued. An example usage, again taken from DiMaProFi, is shown in Fig. 5. It can be seen, how a service call can be made confidential at runtime. For this purpose the service call object is

[3] Please note that not all security objectives can be set up without the receiver side, e.g. authentication requires the receiver side to declare which identities are allowed to use a service.

[4] A thread local is an object that belongs to a thread and which can be used to store and retrieve data without having to pass it explicitly as method parameter value.

fetched via the static method getInvocation() of the ServiceCall class. Afterwards, the secure transmission property is set to true in the call object and the service call (here a file registration operation) is performed as usual.

5.3 Administration Tools

To facilitate the administration of security aspects of the platform tool support is provided. In Fig. 6 a screenshot of the platform security manager interface is depicted. The tool supports four different use cases via a tabbed view. First, the local password settings can be configured, i.e. the platform password can be changed, en- or disabled. Second, certificate and key pair administration can be accessed via the key store tab (active in Fig. 6). This view displays the content of the currently used key store, i.e. the contained key pairs and certificates and on the other hand it allows for manually adding or removing keys and certificates of trusted platforms. Importing a certificate can be either done by using a certificate file or by directly requesting it from the corresponding target platform (if that platform is currently online). Moreover, the view allows for selecting and configuring the used certificate acquisition mechanism (lower right of Fig. 6). It is also possible to completely disable automatic certificate acquisition. Third, the remote passwords tab can be used to view, add or remove credentials of known remote platforms. Alternatively, in the fourth tab network names and passwords can be managed to set up virtual platform networks without having to specify individual platform passwords.

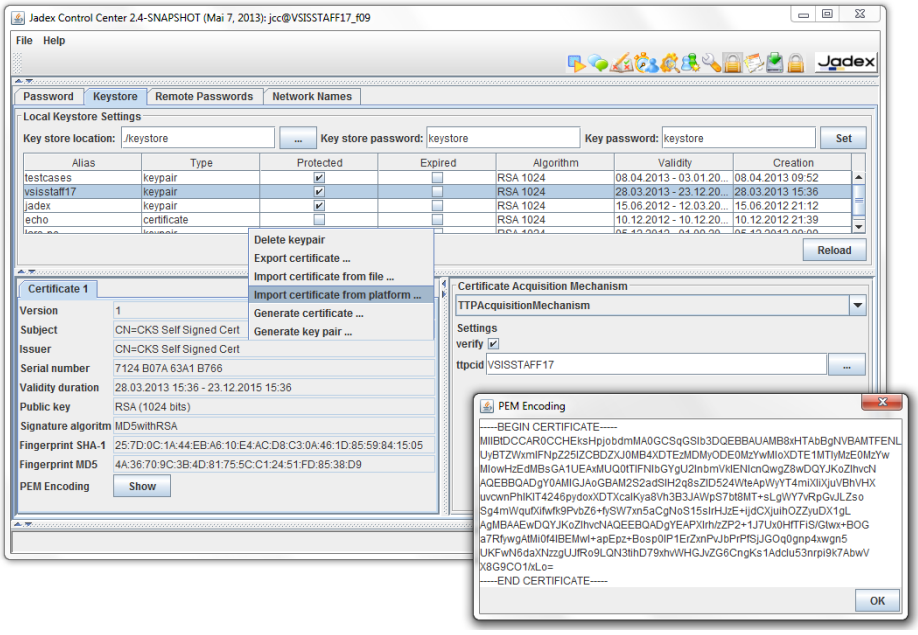

Fig. 6. Platform security manager user interface

6 Evaluation in Practice

The evaluation is based on the real world application scenario presented in Section 2. The DiMaProFi system consists of loosely coupled nodes, each hosting a Jadex platform and application components. The application components use local and remote services of each other in a transparent way. Service discovery is performed dynamically at runtime and can take into account non-functional properties. The overlay network of agent platforms is created by an awareness mechanism, which automatically discovers other nodes [2].

- To prevent unauthorized platforms from using services of DiMaProFi the virtual network access restriction mechanism is employed. The usage is very simple as it is sufficient to start each platform with an application specific virtual network name and password. In this way these platforms share a common secret and can communicate seamlessly. To further avoid denial of service attacks the visibility of the platforms is also hidden. This means that the platforms use a private relay server for managing awareness between different physical networks.
- In order to solve the problem of access to protected resources the password safe concept has been realized using a corresponding service that allows for requesting credentials for a protected action. The service itself exposes its methods only for authenticated users by declaring a set of allowed user names. Within the platform security manager these user names are mapped to trusted platform certificates. Incoming requests need to provide a signed digest that is verified before the call is processed.
- The monitoring infrastructure of DiMaProFi is based on the service and component monitoring of the Jadex platform. This infrastructure creates events for service calls in the system and automatically sends them to a local monitoring service. This service saves the events and offers a subscription based interface for fetching possibly filtered events. For DiMaProFi a custom component has been realized that uses the monitoring service and employs workflow specific rules to detect errors as early as possible. Given that authentication is used in service calls the monitoring logs can be used for proving also non-repudiation of the corresponding invocations.

Besides achieving the overall security objectives within the project it was of crucial importance to tailor the solutions in a way that they become easy to administer and use. In this respect especially the virtual security network and the certificate distribution concepts were considered very helpful by our practice partners as those render it possible to change the underlying nodes infrastructure without huge configuration efforts.

7 Conclusion

In this paper an approach for achieving external security within multi-agent systems operating in open networks has been presented that is directly motivated by

real world requirements of a company project dealing with business intelligence workflows. The security concept supports the achievement of the security objectives integrity, confidentiality, authentication and non-repudiation for service calls. In contrast to most other works, usability was a key factor of the proposed solution. Hence, the usage as well as the administration have been designed to be as simple as possible with usage based on security annotations at the service provider side and dynamically added security meta information at the client side. Moreover, tedious aspects of administration have been resolved at the platform level. Using virtual networks makes it easy to set up large sets of communicating nodes without configuration efforts. Furthermore, also automatic platform certificate acquisition protocols have been included, which largely relieve an administrator from installing platform certificates manually. As part of future work it is planned to include security aspects also in the service search mechanism. This will allow searching for services satisfying specific security features.

References

1. Biskup, J., Kern-Isberner, G., Thimm, M.: Towards enforcement of confidentiality in agent interactions. In: Proceedings of the 12th International Workshop on Non-Monotonic Reasoning (NMR 2008), University of New South Wales, Technical Report No. UNSW-CSE-TR-0819, pp. 104–112 (September 2008)
2. Braubach, L., Pokahr, A.: Developing Distributed Systems with Active Components and Jadex. Scalable Computing: Practice and Experience 13(2), 3–24 (2012)
3. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. Internet Engineering Task Force (August 2008)
4. Goichon, F., Salagnac, G., Frénot, S.: Exploiting Java Code Interactions. Technical Report RT-0419, INRIA (December 2011)
5. Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A.: The Java Language Specification, 7th edn. Addison-Wesley Professional, California (2012)
6. Lee, H., Alves-Foss, J., Harrison, S.: The use of encrypted functions for mobile agent security. In: HICSS, pages 10. IEEE, New York (2004)
7. Moffett, J.D.: Distributed Systems Security. A. Kent, J.G. Williams 15 (1995)
8. Nagaraj, S.V.: Securing multi-agent systems: A survey. In: Meghanathan, N., Nagamalai, D., Chaki, N. (eds.) Advances in Computing & Inform. Technology. AISC, vol. 176, pp. 23–30. Springer, Heidelberg (2012)
9. Neuman, B.C., Ts'o, T.: Kerberos: An authentication service for computer networks. Comm. Mag. 32(9), 33–38 (1994)
10. NIST. Advanced Encryption Standard (AES) (FIPS PUB 197). National Institute of Standards and Technology (November 2001)
11. NIST. Underlying Technical Models for Information Technology Security. National Institute of Standards and Technology (Decmeber 2001)
12. Norman, D.A.: The way i see it: When security gets in the way. Interactions 16(6), 60–63 (2009)
13. OASIS. Web Services Security: SOAP Message Security 1.1 (February 2006)
14. Oey, M., Warnier, M., Brazier, F.: Security in Large-Scale Open Distributed Multi-Agent Systems. In: Autonomous Agents, Rijeka, Croatia, pp. 1–27. IN-TECH (2010)

15. Pokahr, A., Braubach, L.: The active components approach for distributed systems development. International Journal of Parallel, Emergent and Distributed Systems 28(4), 321–369 (2013)
16. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 120–126 (1978)
17. Vila, X., Schuster, A., Riera, A.: Security for a Multi-Agent System based on JADE. Computers & Security 26(5), 391–400 (2007)