A Set of Patterns for Secure Agent Systems

Haralambos Mouratidis[1], Paolo Giorgini[2], Markus Schumacher[3]

[1] Department of Computer Science, University of Sheffield, England
h.mouratidis@dcs.shef.ac.uk

[2] Department of Information and Communication Technology,
University of Trento, Italy
paolo.giorgini@dit.unit.it

[3] IT Transfer Office (ITO), Department of Computer Science,
Darmstadt University of Technology
ms@ito-tu-darmstadt.de

Abstract. Security patterns capture the experiences of experts, allowing novices to rely on expert knowledge and solve security problems in a more systematic and structured way. So far, literature provides many examples of security patterns for object-oriented systems, but no attempt has been made to document security patterns for multiagent systems. In this paper we present a set of patterns for secure agent systems that, currently, consists of four patterns.

1 Introduction

Over the last two decades multiagent systems are used in different domains of the human society from auctions [Byd02] to military systems [Tid99] and are considered one of the most active research areas in Computing. As a result security plays an important role in the development of such systems, since a security failures might lead to many dangers ranging from financial to sensitive military information losses.

Although it has been argued [Dev00] that security concerns should inform every stage of the development process, security is usually considered after the definition of a multiagent system [Mou03], leading to the development of systems afflicted with security vulnerabilities [Sta99].

One of the main reasons for this situation is that non-security experts are involved in the development of systems that require knowledge of security. The application of patterns within the security domain can provide a promising solution to this problem and it could be an easy and effective way to improve the understanding of security issues. A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution. Moreover, the advantages of security patterns are that novices can rely on expert knowledge and solve problems in a more systematic and structured way.

The overall goal of this paper is to present a set of patterns that can be applied in the development of secure agent-based systems. The rest of the paper is structured as follows. Section 2 discusses the motivation behind the development of the proposed set of patterns and also provides an overview of the set. Section 3 provides a roadmap of the proposed set and Section 4 describes the patterns. In Section 5 we describe with the aid of an example how the proposed patterns can be applied in the development of agent-based systems, and in Section 6 we present some concluding remarks and direction for future work.

2 Motivation

The idea of developing a set of patterns or a pattern language for capturing proven security solutions is not new. Essmayr et al. [Ess97] introduced object-oriented access controls (OOAC) as a result of consequently applying the object-oriented paradigm for providing access controls in object and interoperable databases. Fernandez et al. [Fer93] proposed an authorisation model for object-oriented databases based on methods that correspond to access types in the access rule. Yoder and Barcalow [Yod97] proposed a set of patterns that can be applied when developing security for an application and F. Lee Brown et al. [Bro99] proposed a pattern, called Authenticator, which performs authentication of a requesting process before deciding access to distributed objects. In addition, Fernandez and Pan proposed a pattern language for security models [Fer01]. Although this review is by no means complete[1], most of the proposed security-related patterns and languages have been developed having object orientation in mind. As stated by Fernandez and Pan [Fer01] " Our intent is to specify the accepted models as object-oriented patterns".

However, we believe that a different direction, one that will have agent-orientation in mind, should be explored. This is necessary since if patterns and pattern languages are to realise their potential in the development of agent-based systems, then it is required to develop patterns and pattern languages that are specifically tailored to the development of agent-based systems, and use agent-oriented concepts.

It is worth mentioning that many of the above-mentioned object-oriented security patterns display similarities with possible agent-oriented security patterns. These can be exploited in such a way that object-oriented patterns can be turned into agent patterns by identifying agent-specific additions (such as mobility, trust, or cooperation) that can put the security of an agent-based system at risk. Although, examples of such exploitation cannot be found, the literature provides examples of social Object-Oriented patterns turned into Agent-Oriented patterns. For example, the Master-Slave pattern is listed as a pattern in both the POSA1 book [Bus01], as well as in Lange's book [Lan98]. One distinction between these patterns is that the agent version of the pattern explicitly accounts for mobility [Deu01].

In this paper a set of four patterns is proposed that documents how an agency can be protected from malicious agents/agencies. The patterns are categorised into two main categories: Patterns that deal with agency's access issues, such as authentication,

[1] See http://www.securitypatterns.org for a complete review of security related patterns

authorisation, and access control, and patterns that deal with communication issues of the agency, such as secure communication with other agencies, and repudiation. To model our patterns we employ agent-oriented concepts used in the development of agent based systems. We feel this is necessary in order to make the patterns applicable to agent developers.

Agent orientation is based around the concept of an agent. According to Yu [Yu95] an agent as a modelling construct demonstrates the following characteristics:

- **Intentionality**. An agent can be modelled in terms of its intentional properties, such as goals, tasks, resources, beliefs and capabilities, without having to know its specific actions in terms of processes and steps. Although such a high level abstraction does not provide a complete specification for the implementation of the system, it provides developers the ability to model the functional and non-functional requirements of the system and distinguish between different alternatives at an initial stage of the development.
- **Autonomy**. Agents are autonomous and can act independently. Because of the autonomy and independence agents are free to choose from a variety of different actions to perform. Using the concept of a goal helps to model this kind of behaviour since a goal implies that they might be many ways of achieving it.
- **Sociality**. An agent most likely participates in relationships with other agents. In many traditional software engineering techniques, relationships are focused only on the exchange of data and intended functions. However, agent relationships are similar to human relationships and thus much more complex. Agent relationships involve conflicts amongst the relationships, multi-lateral relationships, and delegation of relationships.
- **Identity and boundary**. Agent orientation does not necessarily bounds the modelling concept of an (abstract) agent to that of a physical agent. Agents can be described across a range of physicality and abstractness. For example, social agents can often create new abstractions such as roles, and positions to help to define each others responsibilities and functionality.

We believe it is necessary to employ the above concepts when describing patterns for agents based systems. In doing so, we feel it is essential to describe the structure of a pattern not only in terms of the collaborations and message exchange between the participated agents but also in terms of the social dependencies and intentional attributes, such as goals and tasks, of the agents involved in the pattern. This way we can achieve a complete understanding of the pattern's social and intentional dimensions, two factors very important on agent-based systems.

To describe our patterns we employ the Tropos [Cas01] methodology. Tropos is characterized by three key aspects [Cas01]. Firstly it deals with all the pahses of system requirements analysis and system design and implementation adopting a uniform and homogeneous way. Secondly, Tropos pays great deal of attention to the early requirements analysis that precedes the specification of the requirements, emphasizing the need to understand the how and why the intended system would meet the organizational goals. This allows for a more refined analysis of the system dependencies, leading to a better treatment not only of the system functional requirements but also of its non-functional requirements, such as security, reliability and performance. Thirdly, Tropos is based on the idea of building a model of he system that is incrementally refined and extended from a conceptual level to executable artifacts, by means of a sequence of transformational steps.

Tropos adopts concepts from the *i** modelling framework [Yu95]. The main modelling concept is that of an actor. An actor has intentional properties (Intentionality) and is autonomous (Autonomy). Thus, actors can be (social) agents (organisational, human or software), positions or roles (Identity and Boundary) that have social dependencies (Sociality) for defining the obligations of some actors (*dependees*) to other actors (*dependers*). The type of the dependency describes the nature of an agreement (called *dependum*) between dependee and depender. Goal dependencies represent delegation of responsibility for fulfilling a goal; task dependencies are used in situations where the dependee is required to perform a given activity; and resource dependencies require the dependee to provide a resource to the depender. By depending on others, actors are able to achieve goals that will be very difficult or impossible to achieve on their own. To make the above-mentioned concepts more clear, we consider a small example and we consider three actors:

- *Agency.* An agency represents the environment in which an agent runs. At least one agency must be active on each host computer to enable it to execute agents.
- *Security Manager***.** This actor represents an agent that is responsible for the security of the agency.
- *External Agent*. It is an agent that does not belong to the Agency that wishes to access some information of the agency.

Figure 1 represents the relationships between those actors in terms of their social dependencies. The main goal of the external agent is to access agency information. However, the agency will allow only authorised agents to access information. To fulfil the secure agency access goal the agency depends on the Security Manager. The security manager on the other hand, depends on the External Agents in order to obtain his access detail and be able to decide to allow or deny access to the agency information.

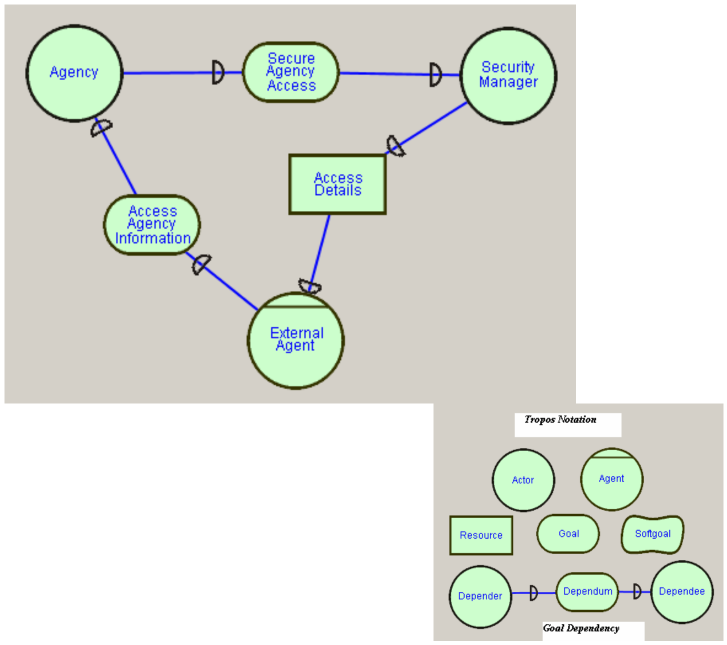


Figure 1: An example of representing social dependencies between different actors of the system

3. Pattern Roadmap

In this paper we present only a subset of patterns in the context of secure agent environments. Figure 2 illustrates the relations between these patterns as well as existing patterns. The diagram is a slight variant of a UML class diagram (the analogy to UML breaks down sooner or later. For example, the pattern name often echoes the solution and can be about dynamic actions, while a class name tends to be a “thing”, not an action). Each box indicates a pattern where a solid-line box indicates a pattern discussed in this document and a dashed-line box indicates a related, existing pattern.

The arrows between the boxes have the following meaning: the dashed lines refer to a “specialize/generalize” relation and the solid arrows refer to a “uses/requires” relation between the patterns.

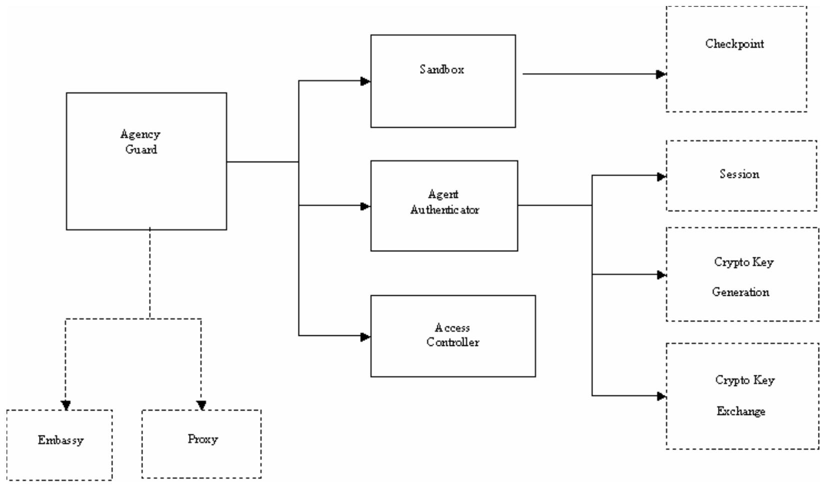


Figure 2: Patterns Roadmap

That way a hierarchy or a sequence of the patterns is build, respectively. The AGENCY GUARD is a variant of the EMBASSY and the PROXY patterns. Besides, the AGENCY GUARD is the starting point of applying the patterns described in this paper. It uses the SANDBOX pattern in order to restrict the actions of agents. Furthermore, the AGENT AUTHENTICATOR is required in order to ensure the authenticity of the agents. Moreover, the ACCESS CONTROLER is used in order to restrict the access to the system resources. The SANDBOX pattern can implement the CHECKPOINT pattern. The AGENT AUTHENTICATOR can use the SESSION pattern in order to store credentials of the agent. Besides, CRYPTOGRAPHIC KEY GENERATION and CRYPTOGRAPHIC KEY EXCHANGE is needed as a basis for further cryptographic actions.

4. A Set of Patterns

Name: AGENCY GUARD (AG)

Intent: Provide a single, non-bypassable, point of access to the agency. The AGENCY GUARD defines a structure that makes unauthorized access to the agency difficult to gain.

Context: A number of agencies exist in a network. Agents from different agencies must communicate or exchange information. This involves the movement of some agents from one agency to another or requests from agents belonging to an agency for resources belonging to another agency.

Problem: Many malicious agents will try to gain unauthorized access to agencies. If a malicious agent gains such an access, it can disclose, alter or generally destroy the data resided in the agency. Additionally, depending on the level of access the malicious agent gains, it might be able to completely shut off the agency or exhaust the agency's computational resources resulting the denial of services to authorised agents of the agency. The problem becomes greater if many "back-doors" are available in an agency enabling malicious agents to attack the agency from many places. On the other hand, not all agents trying to gain access to the agency must be treated as malicious, but access should be granted based on the security policy of the agency.

Solution: There must be a single point of access to the agency. When an agent (Requester Agent) wishes to access resources of an Agency or even move to this agency, its request are forwarded through the AGENCY GUARD that is responsible to grant or deny the access requests according to the security policy of the agency. The AGENCY GUARD is the only point of access in an agency and it is always non-bypassable, meaning all the access requests are going through it.

Social Dependencies: A graphical representation involving the actors of the pattern and their social dependencies is shown in Figure 3. The Agency depends on the AGENCY GUARD to grant/deny access to the agency. The AGENT GUARD grants / denies access according to the security policy. To obtain the security policy the AGENCY GUARD depends on the Agency. The Requester agent depends on the AGENCY GUARD to obtain access to the agency. For the AGENCY GUARD to provide access to the agency, a request must be sent from the Requester agent.

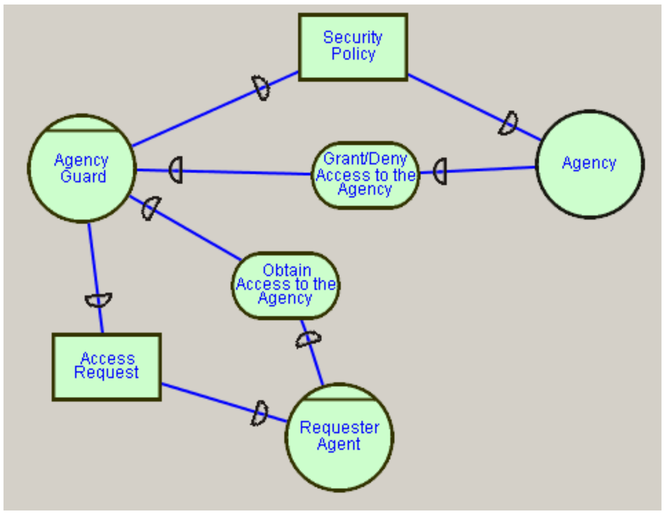


Figure 3: The AGENCY GUARD Dependencies

Consequences:

+ Only the guard should be aware of the security policy of the agency, and it is the only entity that must be notified if security policy changes (Not all the agents in the agency)
+ Only the guard must be tested for correct enforcement of the agency's security policy.
+ Only one point of access to the agency, not many backdoors.
− Only one point of access to the agency can degrade performance of the agency
– Only point of security, if it fails the security of the whole agency is in danger.

Related Patterns: The AGENCY GUARD has concepts of both the PROXY [Nor96] and the EMBASSY patterns [Kol01]. In addition, the AGENCY GUARD depends on the AGENT AUTHENTICATION pattern, in order to authenticate (verify the owner's identity) the agent requesting access. On the other hand, even if the agent is not authenticated the agency might decide to allow it to move to the agency but restrict its actions. For this reason the SANBOXING pattern can be used. In traditional terms the concept of an AGENCY GUARD is related to the SINGLE POINT OF ACCESS [Yod97] and it is referred to as the REFERENCE MONITOR [Amo94, Fer02]

Name: AGENT AUTHENTICATOR (AA)

Intent: Provide authentication services to the agency.

Context: Agents send requests to gain access to an agency or to the resources of an agency; different than the one they belong. To allow access they must be authenticated, i.e. they must provide information about the identity of their owners.

Problem: Many malicious agents will try to masquerade their identity when requesting access to an agency. If such an agent is granted access to the agency, it might try to breach the agency's security. In addition, even if the malicious agent fails to cause problems in the security of the agency, the agency will loose trust of the agent/agency the malicious agent masqueraded the identity.

Solution: Agents have to be authenticated by the agency. By authenticating the agent; the AGENCY GUARD makes sure it comes from an owner that is trustworthy for the agency. Each agent's owner and each agency have a public/private key pair. The AGENT AUTHENTICATOR can authenticate the agent on two cases: Firstly, when the agent is digitally singed with the owner's public key and secondly when the agent is digitally signed with the key of the agency that the agent resides. In the second case, the agent's agency would have authenticated the agent either if the owner signed the agent or if the agent was signed by the sending agency. In order for the second case to work, mutual trust must be involved between the sending and receiving agencies (each agency can be set up so it has a list of "trusted" agencies). In case that the AGENT AUTHENTICATOR does not trust the agency from which the agent comes from, it can reject the agent, or accept it with minimal privileges

Social Dependencies: The graphical representation of the pattern dependencies is shown in Figure 4. The Requester Agent depends on the AGENCY GUARD to obtain access to the agency. However, the AGENCY GUARD cannot authenticate the requester agent by itself, so the it depends on the AGENT AUTHENTICATOR to authenticate the agent so the AGENT AUTHENTICATOR receives a request for authentication from the AGENCY GUARD when needed. In order for the AGENT AUTHENTICATOR to authenticate the Requester Agent, the requester agent should provide evidence of its digital signature. The AGENT AUTHENTICATOR has to send the notification to the AGENCY GUARD when the agent is authenticated.

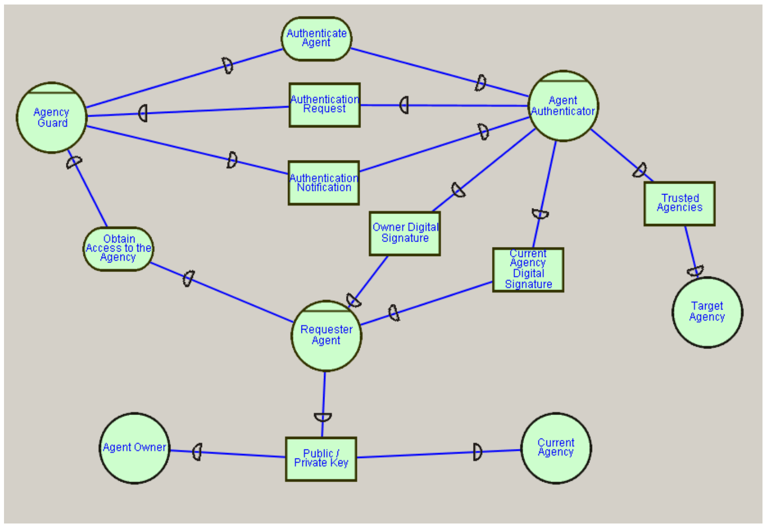


Figure 4: The AGENT AUTHENTICATOR Dependencies

Consequences:

+ Authentication concerns are only dealt once. It is not necessary to make the agents of the system more complex by providing each one of those with an authentication mechanism
+ Ensures that an agent is authenticated before actually request a resource of the agency
+ During the implementation of the system, only the AGENT AUTHENTICATOR must be checked for assurance.
− A single point of failure. If the AGENT AUTHENTICATOR fails, the security of the whole agency is in danger

Related Patterns: This pattern has some relations to patterns of the pattern language for cryptographic key generation [Leh02]. For example, a CRYPTOGRAPHIC KEY GENERATION is required. It is also important to have an appropriate CRYPTOGRAPHIC KEY EXCHANGE. Furthermore, a SESSION can be used to store the credentials of an agent for subsequent requests [Yod97]. Moreover, applying the SANDBOX pattern can be used to restrict the set of resources available to the agent.

Name: SANDBOX

Intent: Allow the agency to execute non-authorised agents in a secure manner.

Context: An agent requests to move to an agency but it is unable to provide authentication certificates. This can be the case, when the agent either is not authenticated, or it has been authenticated by an agency not trusted by the receiving agency.

Problem: An agency is more likely exposed to a huge number of malicious agents that will try to gain unauthorised access. Although the agency will try to prevent access to those agents, it is possible that some of them might be able to gain access. Thus it is necessary for the agency to operate in a manner that will minimise the damage that can be caused by an unauthorised agent that gains access. In addition, some unauthorized agents might be allowed access by the agency in order to provide services the agency's agents cannot provide. Thus, the agency must be cautious to accept such unauthorised agents without put in danger its security.

Solution: Execute the agent in an isolated environment that has full control over the agent's ingoing and outgoing messages. Implementing such a sandboxing principle prevents any malicious code from doing something is not authorised to do. The code is allowed to destroy anything within the restricted environment but it cannot touch anything outside. The concept is similar to the Java programming language's use of a virtual machine environment and the chroot environment in UNIX. Malicious code cannot do anything without first interacting with the operating system. Thus, SANBOX observes all system calls made by the code and compare them to the agency-defined policy. If any violations occur, the agency can shut down the suspicious agent.

Social Dependencies: The graphical representation of the pattern dependencies is shown in Figure 5. The agency depends on the SANDBOX agent for observing and controlling the agent's activities, and the SANDBOX depends on the Agency to know adopted policies.

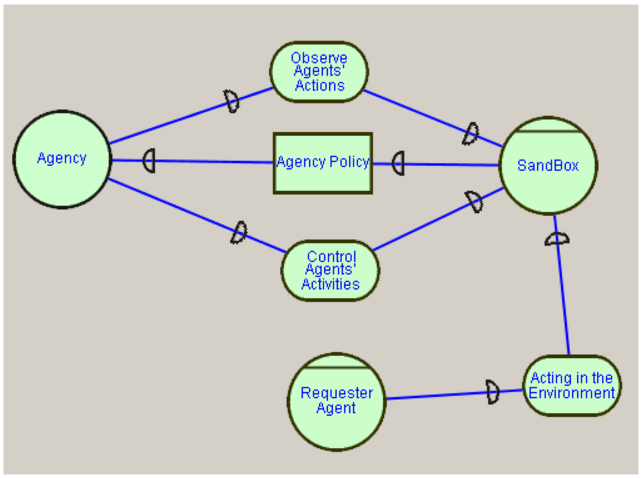


Figure 5: The SANDBOX Dependencies

Consequences:

+ Agents not authorised but valuable for the agency can be executed without compromising the security of the agency
+ Agency can identify possible attacks (by observing the actions of the agents in the SANDBOX)
− Some computational resources of the agency might be taken for non-useful actions (when non-useful agents are sandboxed)
− Introduce an extra layer of complexity on the agency

Related Patterns: A CHECKPOINT should be implemented within the SANDBOX in order to keep track of the exceptional actions and to decide what actions have to be taken based on the severity of the violation of the security policy (which defines what is allowed and what isn't). The SANBOX pattern is related to a similarly-named Java pattern [Jaw00].

Name: ACCESS CONTROLER (AC)

Intent: Allow the agency to provide access to its resources according to its security policy.

Context: Many different agents exist in an agency. Those agents most likely will require access to some of the agency's resources in order to achieve their operational goals. However, different agents might have different access permissions and are allowed access only to specific resources of the agency.

Problem: Agents belonging to an agency might try to access resources that are not allowed. Allowing this to happen might lead to serious problems such as disclosure of private information or alteration of sensitive data. In addition, more likely different security privileges will be applied to different agents on the agency. The agency should take into account its security policy and consider each access request individually. How can the agency make sure that agents access resources that are allowed to access?

Solution: An ACCESS CONTROLER exists in the agency. The ACCESS CONTROLER controls access to each resource. Thus, when an agent requests access to a resource, the request is forwarded to the ACCESS CONTROLER. The ACCESS CONTROLER checks the security policy and determines whether the access request should be approved or rejected. If the access request is approved the ACCESS CONTROLER forwards the request to the RESOURCE MANAGER.

Social dependencies: The graphical representation of the pattern dependencies is shown in Figure 6. The Requester Agent depends on the Resource Manager for the resource, and the Agency depends on the ACCESS CONTROLER for checking the request. ACCESS CONTROLER depends on the Agency for receiving the security policies and for forwarding the request, which is forwarded to the Resource Manager in case it is approved.

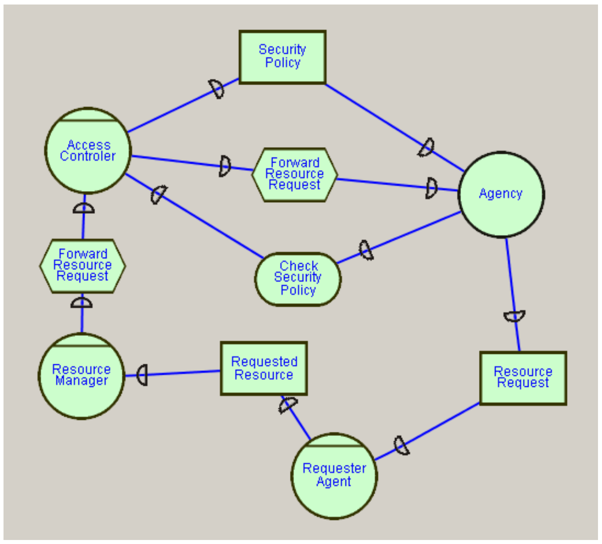


Figure 6: The ACCESS CONTROLER Dependencies

Consequences:

+ Agency's resources are used only by agents allowed to access them
+ Different policies can be used for accessing different resources
− One point of attack, if this fails the system access control system fails

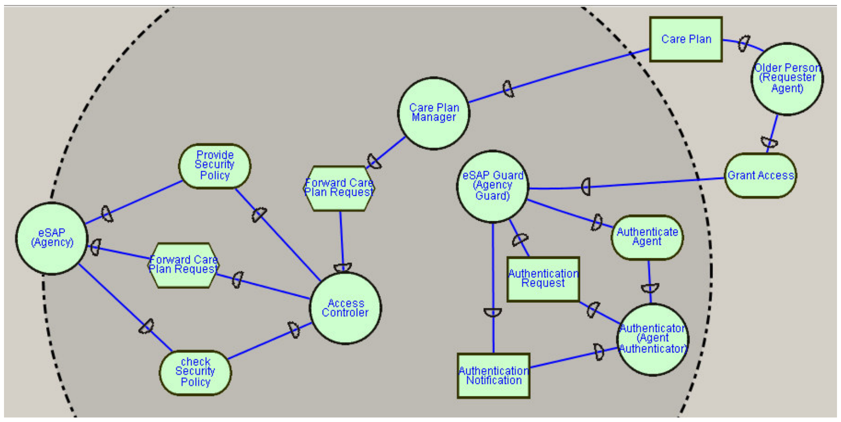
Related Patterns The ACCESS CONTROLER pattern has been inspired by the ROLE-BASED ACCESS CONTROL pattern presented by Fernandez [Fer01]. It is very similar (it can be thought of as a specialisation) to the AGENCY GUARD, but it focuses on access at resources within the agency rather than access to the agency.

5. Applying the Patterns

To illustrate the use of the proposed set of patterns we are employing the electronic Single Assessment Process (eSAP) case study [Mou03b], an agent-based health and social care information system to deliver an integrated assessment of health and social care needs of older people[2]. In previous work [Mou03] we have analysed the security issues of the system using the Tropos methodology and we have identified the need for the system to perform authentication and access control checks. Figure 7 illustrates how the AUTHENTICATOR, the ACCESS CONTROLER and the

[2] The eSAP has been extensively described in the literature [Mou02, Mou02b, Mou03, Mou03b]

AGENCY GUARD patterns can be used to satisfy those security goals of the eSAP system. Let us consider the scenario that an *Older Person* agent (requester agent) wishes to obtain information about their *Care Plan* (resource).



Care Plan
Older Person (Requester Agent)
Care Plan Manager
Provide Security Policy
Forward Care Plan Request
eSAP Guard (Agency Guard)
Grant Access
eSAP (Agency)
Authenticate Agent
Forward Care Plan Request
Authentication Request
Access Controler
Authenticator (Agent Authenticator)
check Security Policy
Authentication Notification

Figure 7: An example of using the patterns

To obtain such information the *Older Person* depends on the *eSAP Guard* agent to grant them access to the system. After the *eSAP Guard* grants access to the *Older Person* (by obtaining an authentication clearance from the *Authenticator* agent) the *Older Person* depends on the *Care Plan Manager* agent to receive information about their *Care Plan*. The *Care Plan Manager* forwards the care plan request to the *Access Controler* agent, which depends on the *eSAP System* agent to obtain the security policy.

By using the above-mentioned patterns the developer is able to identify fast, and efficiently the agents needed to satisfy the system's security goals and delegate the responsibility of the system's security goals to the actors defined by those patterns. For example, the *eSAP Guard* agent checks each agent that tries to access the system, the *Authenticator* agent satisfies the security goal by authenticating each agent that tries to access the system, and the *Access Controler* controls access to the resources (care plans) of the system.

6. Conclusions

As mentioned in the Introduction, security is usually considered after the definition of a multi-agent system leading to the development of systems afflicted with security vulnerabilities. A promising solution to this problem is to apply the pattern approach within the security domain.
In this paper we have point out need to develop security patterns for the development of secure agent-based system and we have argued that such patterns must be based on

agent-oriented concepts, such as goals and social dependencies, which are common in agent-based systems. Towards this direction, we have proposed a set of patterns based on such concepts. Our patterns could be well integrated in the existing landscape of security patterns.
Currently, we are working to expand the proposed set of patterns into a pattern language by providing more patterns related to secure agencies.

References

[Amo94] Edward Amoroso, “Fundamentals of Computer Security Technology”, Prentice Hall, 1994.

[Bro99] F. Lee Brown, James DiVietri, Graziella Diaz de Villegas, Eduardo B. Fernandez, “The Authenticator Pattern”, PLoP, 1999.

[Bus01] F. Buschmann, R. Meunier, H. Rohnert, P. Soomerlad, M. Stal, “Pattern Oriented Software Architecture: A System of Patterns”, Willey, 2001.

[Byd02] A. Byde, C. Priest, N. R. Jennings, “Decision procedures for multiple auctions”, Proceedings of 1st Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 613-620, 2002.

[Cas01] J. Castro, M. Kolp and J. Mylopoulos. “A Requirements-Driven Development Methodology,” In Proc. of the 13th Int. Conf. On Advanced Information Systems Engineering (CAiSE’01), Interlaken, Switzerland, June 2001.

[Dev00] P. Devanbu, S. Stubblebine, “Software Engineering for Security: a Roadmap”, Proceedings of the conference of The future of Software engineering, 2000.

[Ess97] W. Essmayr, G. Pernul, A.M. Tjoa, "Access controls by object-oriented concepts", Proceedings of 11th IFIP WG 11.3 Working Conference on Database Security, August 1997.

[Fer01] Eduardo B. Fernandez and Rouyi Pan. A Pattern Language for Security Models. PLoP, 2001.

[Fer02] E.B.Fernandez, "Patterns for operating systems access control", Proceedings of PLoP 2002

[Fer93] E.B.Fernandez, M.M.Larrondo-Petrie, E.Gudes, "A method-based authorization model for object-oriented databases", Proceedings of the OOPSLA 1993 Workshop on Security in Object-oriented Systems, pp 70-79.

[Jaw00] J. Jaworski and P.J. Perrone, Java security handbook, SAMS, Indianapolis, IN, 2000.

[Kol01] M. Kolp, P. Giorgini, J. Mylopoulos. A Goal-Based Organizational Perspective on Multi-Agent Architectures. Eighth International Workshop on AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL-2001) Seattle, USA, August 1-3, 2001.

[Lan98] D. Lange, M. Oshima, “Programming and Deploying Java Mobile Agents with Aglets”, Addison Wesley, 1998.

[Leh02] Sami Lehtonen and Juha Pärssinen. A Pattern Language for Cryptographic Key Management. EuroPLoP, 2002.

[Mou02] H. Mouratidis, G. Manson, P. Giorgini, I. Philp. Modelling an agent-based integrated health and social care information system for older people. Proceedings of the International Workshop on Agents applied in Health Care (at the 15th European Conference on Artificial Intelligence), Lyon-France, July 2002

[Mou02b] H. Mouratidis, P. Giorgini, G. Manson, I. Philp. Using Tropos methodology to Model an Integrated Health Assessment Systems. Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2002), Toronto-Ontario, May 2002.

[Mou03] H. Mouratidis, P. Giogini, G. Manson, "Modelling Secure Multiagent Systems", (to appear) in the Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, July 2003.

[Mou03b] H. Mouratidis, I. Philp, G. Manson. Analysis and Design of eSAP: An Integrated Health and Social Care Information System. Journal of Health Informatics 9 (2), pp 93-96, 2003

[Nor96] Norman L. Kert, John M. Vlissides, James O. Coplien, Pattern Languages of Program Design 2, Addison Wesley Publishing, 1996

[Sta99] W. Stallings, "Cryptography and Network Security: Principles and Practice", Second Edition, Prentice-Hall 1999.

[Tid99] G. Tidhar, C. Heinze, S. Goss, G. Murray, D. Appla, I. Lloyd. "Using Intelligent Agents in Military Simulation or Using Agents Intelligently". In Proceedings of Eleventh Innovative Applications of Artificial Intelligence Conference, Orlando, Florida, 1999.

[Yod97] Joseph Yoder and Jeffrey Barcalow. Architectural Patterns for Enabling Application Security. PLoP, 1997.

[Yu95] E. Yu. Modelling "Strategic Relationships for Process Reengineering", Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.