Integrating Patterns and Agent-Oriented Methodologies to Provide Better Solutions for the Development of Secure Agent-Based Systems

Haralambos Mouratidis, Paolo Giorgini, Michael Weiss

Abstract
Although pattern languages have already been proposed for security modelling, such languages mostly employ concepts and notations related to object-oriented systems, and have mainly neglected the agent-oriented paradigm. In this position paper we argue about the need to define a security pattern language applicable to agent-based systems that employs concepts based in the agent-oriented paradigm. In addition, we motivate the need to integrate such a language within the development stages of an agent-oriented methodology, and we briefly discuss what such a language should contain,

Keywords: documenting pattern languages, linking requirements to patterns, patterns for secure agent-based systems, integrating patterns with a development methodology, agent-oriented software engineering methodologies

1. Introduction

Patterns capture existing well-proven experience in software development and help to promote good design practice (Coplien, 1996). According to (Buschmann, 2001), a pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly recurring structure of communicating components that solves a general problem within a particular context.

Another paradigm that has gained attention the last few years is based around the concept of an agent (Franklin, 1996), an autonomous computational entity that interacts with its environment and other agents in order to achieve its own goals. Over the last two decades agent technology has been employed in different domains, from personal assistants (Maes, 1994) to auctions (Byde, 2002) and mission-critical military tasks (Tidhar, 1999), and is considered one of the most active research areas in computing. As a result, Agent-Oriented Software Engineering (AOSE) has been proposed (Jennings, 2000) as an alternative paradigm for analysing and designing complex distributed computing systems. According to this paradigm, a complex distributed computing system can be viewed as a multi-agent system. The concept of a software agent as an autonomous system (subsystem), capable of interacting with other software agents within an environment (system) in order to satisfy its design objectives provides designers with more flexibility in their analysis and design of a complex distributed system. Developers can conceptualise the system as a society, similar to a human society, consisting of entities that possess characteristics similar to humans such as mobility, intelligence and the ability to communicate and cooperate (Jennings, 2000).

Security plays an important role in the development of computer systems. However, it is usually considered as an afterthought (Mouratidis, 2002). The common approach towards the inclusion of security within a system is to identify security requirements after its definition. This approach has lead to computer systems afflicted with security vulnerabilities (Stallings, 1999). From the viewpoint of the traditional security paradigm, it should be possible to eliminate such problems through better software engineering techniques (Anderson, 2001).

It has been argued that both patterns and the agent-oriented paradigm represent feasible approaches for the integration of security to software engineering. Security patterns capture design experience and proven solutions to security related problems in such a way that non-security experts can apply them as well (Fernandez, 2001). In addition, they prevent an ad-hoc solution as they help find proven solutions in a systematic and structured way (Buschmann, 2001).

On the other hand, it has also been argued that the agent-oriented paradigm represents a feasible approach for the integration of security with software engineering (Mouratidis, 2002). As emphasized by (Jennings, 2000), agents act on behalf of individuals or companies interacting according to an underlying organisational context. The integration of security within this organizational context will require for the rest of the subsystems to consider the security requirements when specifying their objectives and the interactions between them, therefore causing the propagation of security requirements to the rest of the subsystems. To this end, some agent-oriented software engineering methodologies (Mouratidis, 2003) have started to consider security issues as an integral part of their development stages and processes.

We believe the integration of patterns and agent-oriented methodologies will lead to a better solution for the development of secure agent-based systems. It has been argued that *"if agents are to realise their potential as a software engineering paradigm, then it is necessary to develop software engineering techniques that are specifically tailored to them"* (Wooldridge, 2000). Thus, we feel it is necessary for a pattern language related to the security of agent-based systems to employ agent-oriented concepts.

This paper is structured as follows. Section 2 motivates why a pattern language is needed for the development of secure agent-based systems, and Section 3 discusses the need to ground such a language within an agent-oriented development methodology. Section 4 briefly describes what a pattern language for secure agent-based systems should contain, and Section 5 provides some concluding remarks.

2. Need for a pattern language for secure agent-based systems

Research on security for multiagent systems is an important area within the agent research community. Research is focused in three main categories (Jansen, 1999): *protect agencies from malicious agents*; *protect agents from other agents*; and *protect agents from malicious agencies*. This research is broader than the research focused on security of mobile agent systems, which is really only a special instance of this classification.

However, one of the problems of developing secure systems is that often non-security specialists have to develop them. Thus, even though agent-oriented methodologies that integrate security and systems engineering have been started to emerge (Mouratidis, 2003), a critical question remains: *How it can be assured that non-security specialists will have the knowledge to transform the requirements to design?* In projects that are stressed on time and budget developers must "acquire" security knowledge within a short timeframe and make sure that the system developed will work according to the requirements. A developer should know which designs are suitable for their problem, and any consequences an existing design will impose on their systems.

Object-oriented developers have realised the use of security patterns can be a valuable solution, since it helps to easily identify in which situations a pattern of the language can be applied and also identify the consequences of applying a specific pattern to the security of the system. However, this is not the case for the developers of agent-based systems who have, mainly, neglected the integration of security patterns during the development stages of secure agent-based systems.

One of the reasons is the fact that not many security patterns for agent-based systems have been documented. Nevertheless, many techniques used for securing agent-based systems have been successfully employed in many examples and are well understood. Thus, we believe it is time to start documenting them as patterns. However, as stated by (Deugo, 1999), documenting some techniques as patterns, does not mean to document the problem and the solution, since such documentation can be found in many papers describing the techniques, but rather to provide a deeper understanding of the forces and the context of the problems that give rise to the proposed solutions.

One might argue that many object-oriented patterns, such as Authenticator (Lee Brown, 1999) or already proposed pattern languages for security modelling (Fernandez, 2001) can be applied to the development of secure agent-based systems. However, as mentioned in the introduction, for the agents to realise their potential, software engineering techniques specifically tailored to them must be developed (Wooldridge, 2000). The main role of such techniques (methodologies) will be to help in all the development stages of agent-oriented systems and in particular capture and model the unique characteristics that agent-oriented systems introduce such as flexibility, autonomous problem-solving, and the richness of interactions between the individual agents. Also, differences in the way agents and objects communicate and interact with their environment, the level of autonomy agents possess, and the fact that agents are often highly mobile motivate the separate notion of an agent pattern (Deugo, 1999). In addition, the social nature of agent-based systems and the different security requirements introduces a void that existing patterns – agent, object, or otherwise – have not filled.

Thus, security patterns have to be documented in a manner specific to agent-based systems, using agent-oriented concepts. In addition, it is important to develop a pattern language consisting of those patterns. Developing such a pattern language will provide a mutual ontology that will enables the pattern language to be integrated within the development stages of current agent- oriented methodologies. Having a mutual ontology between a pattern language and agent-oriented methodologies will help the language to be understandable by people working in the development of agent-based systems. The modelling concepts, the notation and the development stages of an agent-oriented methodology can provide such a mutual ontology.

3. Need to ground the pattern language within a development methodology

As mentioned above, we believe that the integration of patterns and agent-oriented methodologies will provide a complete[1] and mature solution for the development of secure agent-based systems. We say complete since we believe those two paradigms

[1] We must note, the word *complete* does not refer to the pattern language (since by definition a language evolves and grows over time with the extensions of existing patterns, or introduction of new ones) but rather to the integration of the pattern language and the agent-oriented methodology.

complement each other, and we say mature since such integration will provide novice developers with the capability to implement secure agent systems.

However, two critical and important questions arise at this point (that have been also considered as important questions from the organisers of this workshop): *How can a developer identify a starting point in using a pattern language?* And *how can a developer select which patterns of the language to apply next according to the security requirements of the system?*

We believe answers to those questions can be given when a pattern language is fully integrated with an agent-oriented development methodology. The modelling language (ontology) of the methodology will provide a framework that will form the base for the development and application of the pattern language. The patterns of the language (and the relationships between them) would be expressed and described by employing concepts from the methodology ontology. In addition, the guidelines and the structure processes of the methodology will allow the explicit definition of the applicability of the pattern language within the methodology stages and will allow developers to evaluate and select from the different patterns of the language according to the system security requirements.

We briefly present Tropos, as an example of a candidate, for the integration of patterns with an agent-oriented methodology. *Tropos*[2] is a methodology for building agent-oriented software systems, tailored to describing both the organisational environment of a system and the system itself. Tropos is characterised by three key aspects (Perini, 2001). Firstly, it deals with all the phases of system requirements analysis and system design and implementation adopting a uniform and homogeneous way that is based on the notion of agents and all the related mentalistic notions, such as actors, goals, tasks, resources, and intentional dependencies. Secondly, Tropos pays a great deal of attention to the early requirements, emphasizing the need to understand not only *what* organisational goals are required, but also *how* and *why* the intended system would meet the organisational goals. This allows for a more refined analysis of the system dependencies, leading to a better treatment not only of the system's functional requirements but also of its non-functional requirements, such as security, reliability, and performance (Perini, 2001). Thirdly, Tropos is based on the idea of building a model of the system that is incrementally refined and extended from a conceptual level to executable artefacts, by means of a sequence of transformational steps (Bresciani, 2002).

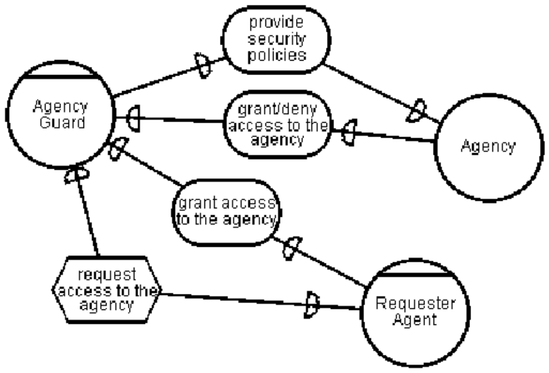
Tropos adopts the *i** modelling framework (Yu, 1995), which uses the concepts of actors, who can be (social) agents (organisational, human or software), positions or roles, goals and social dependencies (such as soft goals, tasks, and resources) for defining the obligations of actors (dependees) to other actors (dependers). This means the system (as well as its environment) can been seen as a set of actors, who depend on other actors to help them fulfil their goals. 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; softgoal dependencies are similar to goal dependencies, but their fulfilment cannot be defined precisely; 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. In addition, Tropos ontology has been extended (Mouratidis, 2003) to offers

[2] The name Tropos derives from the Greek and means "easily changeable", "easily adaptable".

concepts such as security constraint and secure entities and dependencies, to enable it to model security issues during the development stages of an agent-based system.

Tropos supports five development stages, namely *early* and *late requirements*, *architectural* and *detailed design* and *implementation*. Early and late requirements analysis represents the initial phases in the Tropos methodology and the final goal is to provide a set of functional and non-functional requirements for the system-to-be. Both phases, early and late, share the same conceptual and methodological approach. This means, that most of the techniques used during the early requirements analysis are used for the late as well. The main difference is that during the early requirements analysis, the developer models the main stakeholders of the system and their dependencies, while in the late requirements analysis the developer models the system itself by introducing it as another actor and model its dependencies with the other actors of the organisation. The architectural design stage defines the system's global architecture in terms of actors interconnected through data and control flows (represented as dependencies). In addition, during this stage the actors of the system are mapped into a set of software agents, each characterized by its specific capabilities. During the detailed design stage, the developer specifies, in detail, the agents' goals, beliefs, and capabilities as well as the communication between the agents. For this reason, Tropos employs a set of Agent UML diagrams (Bauer, 2001). During the implementation stage, the design of the system produced in the previous stage is converted into code. In Tropos the implementation stage follows step by step the design produced in the previous stages.

Taking as an example the eSAP System (Mouratidis, 2002b), an agent-based health and social care information system for the effective care of older people. Such a system has been partially analysed as presented in (Mouratidis, 2002 – Mouratidis, 2000b-Mouratidis, 2003). After analysing in detail the goals, tasks and the security constraints of the system, it is concluded (Mouratidis, 2002- Mouratidis, 2003) (during the architectural design) that one of its secure goals is to *Ensure Data Privacy*. To solve this, a combination of patterns can be applied. For example, an agent guard can be used to protect access to the agency and thus provide a solution for the *Access Control* goal of the eSAP System. Figure 1 shows how this pattern can be represented employing Tropos concepts. At this stage this pattern would be more properly considered a proto-pattern.


provide security policies
Agency Guard
grant/deny access to the agency
Agency
grant access to the agency
request access to the agency
Requester Agent
Figure 1: The Agency Guard (proto-) pattern

The idea behind the above example is that requirements can be identified by the methodology, during the early and late requirements stages, and patterns can be employed during the architectural and detailed design stages to transform the requirements to design. This helps to link the patterns used back to system requirements. In addition, the choice of a particular pattern can be justified since Tropos allows developers to perform different kinds of analysis, such as NFR and goal analysis, to represent justification for the choice of different alternatives. Similarly, security requirements can be identified and security patterns can be applied in the development of agent-based systems, providing justification, which can be traced back to early requirements analysis, for the reasons behind the choice of particular patterns.

4. What should a good pattern language for security of agent-based systems contain

A good pattern language for security of agent-based systems should contain patterns belonging to each of the areas of research (Section 2) related to the security of agent-based systems. Each of those patterns should be explicitly defined and also the relation between them must be precisely identified. A categorisation of the patterns belonging to the language could take place according to which area of the security of agent-based systems they document. Thus, we can have patterns for the protection of agencies from malicious agents, patterns for the protection of agents from another agents, and patterns for the protection of agents from malicious agencies.

An example of the first area could be a pattern that all the requests for accessing resources of the agency, or even move to this agency, are forwarded through an agent that is responsible to grant or deny access to the requests. Another example of a pattern belonging to this category could be a pattern that provides a solution in running a malicious code from tampering with the agency. This can be achieved by restricting the code in a protected environment.

Patterns that protect agents from another agents should be able to document the communications between the agents but also proofs that such communication took place. For example, a pattern could be documented to provide a non-repudiation service for agents. Repudiation occurs when an agent communicating with another agent, later claims the communication never took place. Although, an agent cannot prevent another agent from repudiating a communication, it can make sure that it has enough evidence to support their defence.

Ideally patterns that protect agents from malicious agencies must provide solutions to problems such as masquerading, denial of service, eavesdropping, and alteration. Unfortunately, a proven solution against all those problems simply does not exist yet. However, techniques that help towards the detection or the protection against some of those problems have been successfully employed. For example, the supervisor-worker pattern proposes in (Fischmeister, 1999) could provide a solution to the eavesdropping and alternation problems but is prone to masquerading or denial of service attacks.

Conclusions

In this paper we have argued for the need to develop a pattern language tailored to the development of secure agent-based systems, which will employ concepts from the agent-oriented paradigm. In addition, we have motivated reasons for integrating such a language with an agent-oriented software engineering methodology in order to provide better solutions for the development of secure agent-based systems. At the ChiliPloP 2003 workshop, we would like to discuss with attendees how methodologies (such as

Tropos) can provide a mutual ontology for representing the patterns in a pattern language, and a context for applying patterns within different development stages.

References

Anderson, R. (2001), “Security Engineering”, Wiley Computer Publishing

Bauer, B., Müller, J., Odell, J., (2001), “Agent UML: A Formalism for Specifying Multiagent Interaction”. In *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge (eds), Springer, Berlin, pp. 91-103

Bresciani P. and Giorgini. P. (2002), “The Tropos Analysis Process as Graph Transformation System”, In Proceedings of the Workshop on Agent-oriented methodologies, at OOPSLA 2002, Seattle, WA, USA

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

Coplien, J. (1996), Software Patterns, SIGS

Deugo, D., and Weiss, M. (1999), “A Case for Mobile Agent Patterns” Workshop on Mobile Agents in the Context of Competition and Cooperation, at Autonomous Agents, 19-22, 1999

Fernandez, E., and Pan, R. (2001), “A Pattern Language for Security Models”, Conference on Pattern Languages of Programming, (PLoP)

Fischmeister, S., and Lugmayr, W., (1999), “The Supervisor-Worker Pattern”, Conference on Pattern Languages of Programming (PLoP)

Franklin, S. and Graesser, A. (1996), “Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents”, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag

Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1995), “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley

Jansen, W., Karygiannis, T. (1999), “Mobile Agent Security”, National Institute of Standards and Technology, Special Publication 800-19

Jennings, N. R. (2000), “On Agent-Based Software Engineering”, Artificial Intelligence, 117(2), 277-296

Lee Brown, F., DiVietri, J., Diaz de Villegas, G., Fernandez, E. B., (1999), “The Authenticator Pattern”, Conference on Pattern Languages of Programming (PLoP)

Maes, P. (1994), “Agents that Reduce Work and Information Overload”, Communications of the ACM, 37(7), 30-40

Mouratidis, H., Giorgini, P., Manson, G. And Philp, I. (2002), "A Natural Extension of Tropos Methodology for Modelling Security", In the Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle-USA

Mouratidis, H., Philp, I., Manson, G. (2002b), "Analysis and Design of eSAP: An Integrated Health and Social Care Information System", in the Proceedings of the 7th International Symposium on Health Information Managements Research (ISHIMR2002), Sheffield

Mouratidis, H., Giorgini, P., Manson, G. (2003), "Modelling Secure Multiagent Systems", (to appear) in the proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia.

Perini, A., Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., (2001), "Towards an Agent Oriented Approach to Software Engineering. In A. Omicini and M.Viroli, editors, WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software, Modena-Italy

Stallings, W. (1999), "Cryptography and Network Security: Principles and Practice", Second Edition, Prentice-Hall

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

Wooldridge, M., Jennings, N.R., Kinny, D. (2000), " The GAIA Methodology for Agent-Oriented Analysis and Design", Autonomous Agents and Multi-Agent Systems, 3(3), 285-312

Yu, E. (1995), "Modelling Strategic Relationships for Process Reengineering", PhD thesis, Department of Computer Science, University of Toronto, Canada