A Constraint-based approach for modeling secure multiagent systems[1]

Paolo Giorgini[1], Haralambos Mouratidis[2]

[1] DIT, University of Trento, Italy
paolo.giorgini@dit.unitn.it
[2] School of Computing and Technology, University of East London, England
h.mouratidis@uel.ac.uk

Abstract. Security plays an important role in the development of multiagent systems. However, a careful analysis of software development processes shows that the definition of security requirements is, usually, considered after the design of the system. This is, mainly, due to the fact that agent oriented software engineering methodologies have not integrated security concerns throughout their developing stages. The integration of security concerns during the whole range of the development stages could help towards the development of more secure multiagent systems. In this paper we introduce a constraint-based approach that extends the Tropos methodology in order to model security concerns throughout the whole development process. A description of the new concepts is given along with a security oriented process that integrates these concepts in Tropos.

1 Introduction

Security plays an important role in the development of multiagent systems and is considered as one of the main issues to be dealt for agent technology to be widely used outside the research community. As a result, research on security for Multiagent systems is an important area within the agent research community. However, the research has been mainly focused on the solution of individual security problems of the multiagent systems, such as attacks from an agent to another agent, attacks from a platform to an agent, and attacks from an agent to a platform.

Only very little work has taken place in considering security requirements as an integral part of the whole software development process. None of the existing agent oriented methodologies, to our knowledge, have been demonstrated enough evidence to support claims of adequately integrate security modeling during the whole software development stages. Only recently, some initial steps have been taken towards this

[1] This paper is a revised and extended version of: H. Mouratidis, P. Giorgini, G. Manson. Modelling Secure Multiagent Systems, In the Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne-Australia, ACM Press, July 2003

direction. Liu and Yu have initiated work [12] that provides ways of modeling and reasoning about non-functional requirements (with emphasis on security). Yu is using the concept of a soft goal to assess different design alternatives, and how each of these alternatives would contribute positively or negatively in achieving the soft goal.

Lodderstedt et al present a modeling language, based on UML, called SecureUML [13]. Their approach is focused on modeling access control policies and how these (policies) can be integrated into a model-driven software development process. Differently than these two approaches that are focused in particular stages of the development (Yu's effort is focused only in the requirements area while Lodderstedt's work is focused in the design stage) our approach covers the whole development process. It is important to consider security using the same concepts and notations during the whole development process.

In addition, Huget [14] proposes a new methodology, called Nemo and claims that it tackles security. In his approach, security is not considered as a specific model but it is included within the other models of the methodology. Nemo is a new methodology and as such it has not been extensively presented on literature. From our point of view, the methodology tackles security quite superficial and as the developer states "*particularly, security has to be intertwined more deeply within models"* [14]. Thus, more evidence will be required to satisfy the claim of the developer that the methodology tackles security.

Because of the lack of a structured approach to consider security issues in the development of computer systems, the common approach towards the inclusion of security within a system is to identify security requirements after the definition of a system. This approach has provoked the emergence of computer systems afflicted with security vulnerabilities [4]. From the viewpoint of the traditional security paradigm, it should be possible to eliminate such problems through better integration of security and systems engineering.

We believe that security concerns should be considered during the whole development process of a multiagent system and it should be defined together with the requirements specification. Taking security requirements into account together with the functional requirements of a Multiagent system throughout the development stages helps to limit cases of conflict between security and system requirements, by identifying them very early in the system development, and find ways to overcome them. On the other hand, adding security as an afterthought not only increases the chances of such a conflict to exist, but it requires huge amount of money and valuable time to overcome it, once they have been identified (usually a major rebuild of the system is needed).

In this paper, we introduce extensions to the Tropos methodology to accommodate security concerns during the software development stages. Section 2 provides an overview of the Tropos methodology, and Section 3 introduces the secure concepts of the secure Tropos. In Section 4 we discuss the security oriented process of the secure Tropos, whereas section 5 concludes the paper.

2 Tropos and Secure Tropos

Tropos [5] is a software development methodology, for building agent-oriented software systems, that uses concepts such as actors, goals, soft goals, tasks, resources and intentional dependencies throughout all the phases of the software development [6]. A key feature of Tropos is that it pays great deal of attention to the early requirements analysis that precedes the specification of the perspective requirements, emphasizing the need to understand the how and why the intended system would meet the organisational goals.

Tropos adopts the *i** modelling framework [15], which uses the concepts of actors, goals and social dependencies for defining the obligations of actors (dependees) to other actors (dependers). This means the multiagent system and its environment are viewed as a set of actors, who depend on other actors to help them fulfil their goals.

An actor [15] represents an entity that has intentionality and strategic goals within the multiagent system or within its organisational setting. An actor can be a (social) agent, a position, or a role. Agents can be physical agents, such as a person, or software agents. A role represents an abstract characterisation of the behaviour of a social actor within some specialised context or domain of endeavour [15]. A position represents a set of roles, typically played by one agent.

A (hard) goal [15] represents a condition in the world that an actor would like to achieve. In other words, goals represent actor's strategic interests. In Tropos, the concept of a hard-goal (simply goal hereafter) is differentiated from the concept of soft-goal. A soft-goal is used to capture non-functional requirements of the system, and unlike a (hard) goal, it does not have clear criteria for deciding whether it is satisfied or not and therefore it is subject to interpretation [15]. For instance, an example of a soft-goal is "the system should be scalable".

A task (also called plan) represents, at an abstract level, a way of doing something [Giu02]. The fulfilment of a task can be a means for satisfying a goal, or for contributing towards the satisficing of a soft-goal. In Tropos different (alternative) tasks, that actors might employ to achieve their goals, are modelled. Therefore developers can reason about the different ways that actors can achieve their goals and decide for the best possible way.

A resource [6, 10] presents a physical or informational entity that one of the actors requires. The main concern when dealing with resources is whether the resource is available and who is responsible for its delivery.

A dependency [15] between two actors represents that one actor depends on the other to attain some goal, execute a task, or deliver a resource. The depending actor is called the depender and the actor who is depended upon is called the dependee. 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. Soft-goal dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely whereas task dependencies are used in situations where the dependee is required to perform a given activity. Resource dependencies require the dependee to provide a resource to the depender. By depending on the dependee for the dependum, the depender is able to achieve goals that it is otherwise unable to achieve on their own, or not as easily or not as well [15]. On the other hand, the depender becomes vulnerable, since if the dependee fails to deliver the dependum, the depender is affected in their aim to achieve their goals.

A capability [6, 10] represents the ability of an actor of defining, choosing and executing a task for the fulfillment of a goal, given certain world conditions and in presence of a specific event.

Figure 1 **Error! Reference source not found.**depicts a graphical representation of the above-mentioned concepts as used in the Tropos methodology.

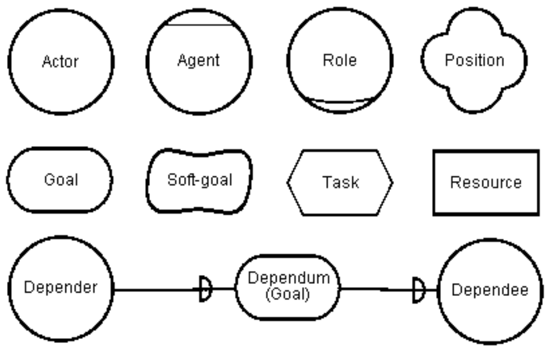

Fig. 1. The concepts in Tropos

Tropos supports four development stages, namely *early* and *late requirements*, *architectural design*, and *detailed design*. 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 AUML diagrams [7].

Tropos was not conceived with security in mind and as a result it fails to adequately capture security requirements [8, 9]. The process of integrating security and functional requirements throughout the whole range of the development stages is quite ad hoc, and in addition, the concept of soft goal that Tropos uses to capture security requirements fails to adequately capture some constraints that security requirements often represent [8, 9].

Therefore, we have extended the Tropos methodology to enable developers to adequately capture security requirements. The next section describes our extensions.

3 Secure Concepts in Secure Tropos

Extra concepts were introduced to the methodology to enable it to model security requirements during the software development process. These are:

Security Diagram [9], which represents the connection between security features, threats, protection objectives, and security mechanisms that help towards the satisfaction of the objectives. Security features [9] represent security related features that the system-to-be must have. Protection objectives [9] represent a set of principles that contribute towards the achievement of the security features. Threats [9] on the other hand represent circumstances that have the potential to cause loss or problems that can put in danger the security features of the system, while security mechanisms [9] identify possible protection mechanisms of achieving protection objectives. The main purpose of the security reference diagram is to allow flexibility during the development stages of a multiagent system and also to save time and effort. Many systems under development are similar to systems already in existence. Therefore the security reference diagram can be used as a reference point that can be modified or extended according to specific needs of particular systems.

The analysis done during the construction of the security reference diagram can be used later in the development process to identify security constraints that must be introduced to the system-to-be (by taking into account the security needs of the system) and also by identifying possible means (security mechanisms) that contribute towards the satisfaction of the security constraints that are introduced to the system.

The notation of the security reference diagram can be adapted to reflect the notation of the methodology that the diagram is integrated. This is very useful since it allows developers to work with well-known concepts and allows them to use the same concepts throughout the development process. In this work, concepts from the Tropos methodology such as soft-goals, goals and tasks are used to model security features, protection objectives and security mechanisms respectively

Security Constraint [9], which represents, generally speaking, constraints that are related to the security of the system. A security constraint is defined *as a restriction related to security issues, such as privacy, integrity and availability, which can influence the analysis and design of a multiagent system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives* [9].

Since, security constraints can influence the security of the system either positively or negatively, we further define positive and negative *security constraints* respectively. An example of a positive security constraint could be *Allow Access Only to Personal Information*, while a negative *security constraint* could be *Send Information Plain Text* (not encrypted). A graphical representation of a security constraint can be found in figure 3.

Secure Entities [9], which represent any secure goals/tasks/ resources of the system. A **secure goal** represents the strategic interests of an actor with respect to security. Secure goals are mainly introduced in order to achieve possible security constraints that are imposed to an actor or exist in the system. However, a secure goal does not particularly define how the security constraints can be achieved, since alternatives can be considered. The precise definition of how the secure goal can be achieved is given by a **secure task**. A secure task is defined as a task that represents a particular way for satisfying a secure goal. A **secure resource** can be defined as an informational entity that is related to the security of the multiagent system. Secure resources can be divided into two main categories. Those that display some security characteristics, imposed by other entities, such as security constraints, secure goals, secure tasks and secure dependencies.

In addition, the graphical representation of the Tropos entities has been extended to enable it to model the secure entities. Secure entities are indicated by the presence of an S within brackets before the description of the entity as shown in figure 2.

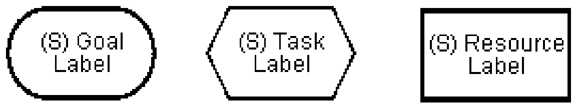

Fig. 2. Secure Entities

Secure Dependencies [9], represent that a dependency between two actors involves the introduction of a security constraint that must be satisfied either by the depender, the dependee or both for the dependency to be valid. Secure dependencies are categorized into *depender secure dependency*, in which the depender introduces security constraints for the dependency and the dependee must satisfy the security constraints for the dependency to be valid, *dependee Secure Dependency,* in which the dependee introduces security constraints and the depender must satisfy them, and *double Secure Dependency***,** in which both the depender and the dependee introduce security constraints for the dependency that both must satisfy for the dependency to be valid. A graphical representation of the different types of secure dependencies is shown in figure 3.

Secure Capabilities, which represent capabilities that the actors (agents) of the system must have in order to help towards the satisfaction of the security requirements of the system.

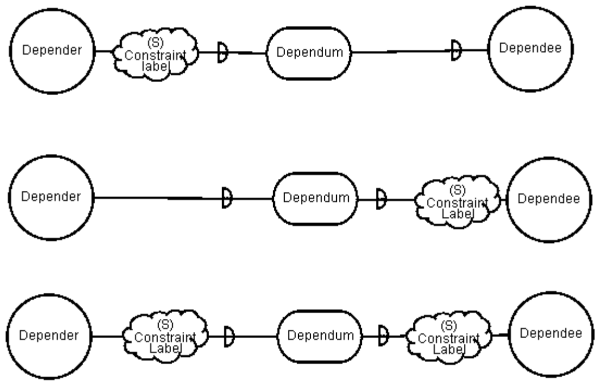

Fig. 3. Secure Dependencies

4 Modeling Security with Secure Tropos

The security-oriented process of secure Tropos is mainly divided into four sub-activities; (1) The identification of security requirements of a multiagent system; (2) the selection amongst alternative architectural styles for the system-to-be according to the identified security requirements; (3) the development of a design that satisfies the security requirements of the system; (4) and the attack testing of the multiagent system under development.

To make the process easier to understand, we use a case study from the health and social care sector, the electronic Single Assessment Process (eSAP) system [9], an agent based health and social are information system.

The process of identifying the security requirements of the system is basically one of analysing the security needs of the stakeholders and the system in terms of security constraints imposed to the system and the stakeholders, and identify secure goals and entities that guarantee the satisfaction of the security constraints.

The first step in the security process consists of the construction of the security reference diagram [9]. For instance, the security reference diagram of the eSAP system is shown in figure 4.

As shown in the figure, the main security features for the electronic single assessment process system are privacy, integrity and availability. Health and social care professionals are worried that using such a system introduces risks for the privacy of personal health and social care information. Therefore privacy of health and social care information, such as the health and social care plans used in the electronic single assessment process, is the number one security concern in such a system. According to the Good Medical Practice, patients have a right to expect that you will not pass on any personal information, which you learn in the course of your professional duties unless they agree.

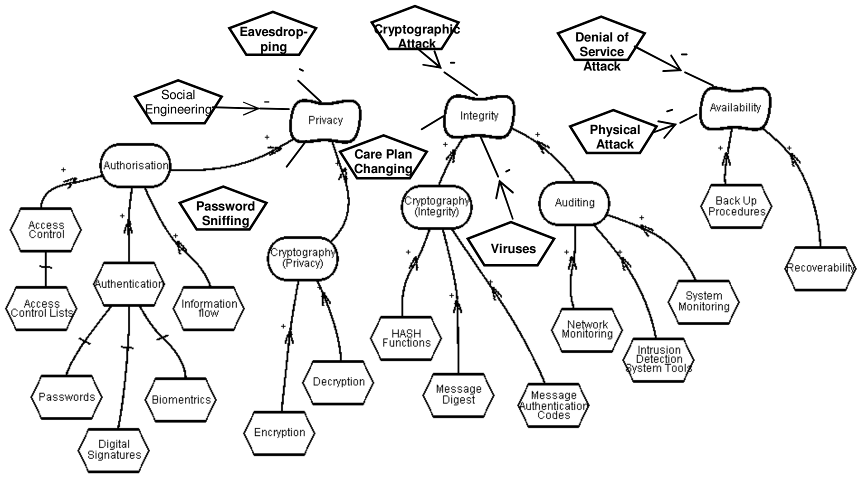

Fig. 4. eSAP security reference diagram

Other important concerns are integrity and availability. Integrity assures that information is not corrupted and availability ensures the information is always available to authorised health and social care professionals. If assessment information is corrupted or it is not available the care provided to the patients by the health and social care professionals will not be efficient or accurate. Therefore, it is necessary to find ways to help towards the privacy, the integrity and the availability of personal health and social care information.

When the security reference diagram is complete, the analysis of the actors of the multiagent system takes place and security constraints are imposed to the actors of the system. In addition, security constraints are imposed to the system-to-be, with the aid of the security reference diagram as shown in figure 5.

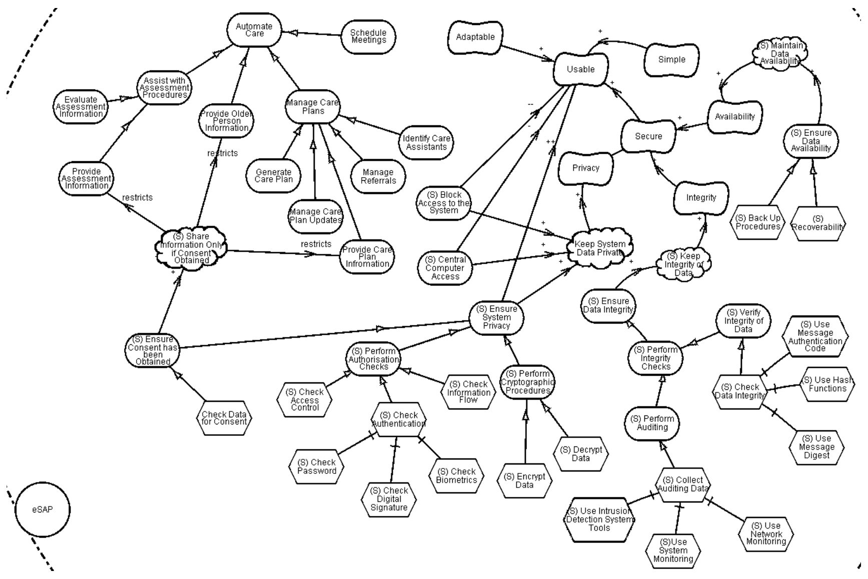
Automate Care
Schedule Meetings
Assist with Assessment Procedures
Evaluate Assessment Information
Provide Older Person Information
Manage Care Plans
Identify Care Assistants
Provide Assessment Information
restricts
restricts
Generate Care Plan
Manage Referrals
Manage Care Plan Updates
(S) Share Information Only if Consent Obtained
restricts
Provide Care Plan Infromation
(S) Ensure Consent has been Obtained
Check Data for Consent
(S) Perform Authorisation Checks
(S) Check Access Control
(S) Check Information Flow
(S) Check Authentication
(S) Check Password
(S) Check Digital Signature
(S) Check Biometrics
eSAP
Adaptable
Usable
Simple
(S) Maintain Data Availability
Availability
Secure
(S) Ensure Data Availability
Privacy
Integrity
(S) Block Access to the System
(S) Central Computer Access
Keep System Data Private
(S) Keep Integrity of Data
(S) Back Up Procedures
(S) Recoverability
(S) Ensure System Privacy
(S) Ensure Data Integrity
(S) Verify Integrity of Data
(S) Use Message Authentication Code
(S) Perform Integrity Checks
(S) Check Data Integrity
(S) Use Hash Functions
(S) Use Message Digest
(S) Perform Cryptographic Procedures
(S) Decrypt Data
(S) Encrypt Data
(S) Perform Auditing
(S) Collect Auditing Data
(S) Use Intrusion Detection System Tools
(S)Use System Monitoring
(S) Use Network Monitoring

Fig. 5. Analysis of the eSAP actor

When the security requirements of the system-to-be and the involved actors have been identified, the next step in the process consists of identifying an architectural style for the system that will satisfy the security requirements.

For this reason, we have developed an analysis technique to enable developers to select among alternative architectural styles using as criteria the non-functional requirements of the multiagent system under development. The technique is based on an independent probabilistic model, which uses the measure of *satisfiability* proposed by Giorgini et al. [16]. S*atisfiability* represents the probability that a non-functional requirement will be satisfied. Therefore, the analysis involves the identification of specific non-functional requirements and the evaluation of different architectural styles against these requirements.

The evaluation results in contribution relationships from the different architectural styles to the probability of satisfying the non-functional requirements of the system. To express the contribution of each style to the satisfiability of each non-functional requirement of the system, a weight is assigned. Weights take a value between 0 and 1. For example, 0.1 means the probability that the architectural style will satisfy the non-functional requirement is very low (the style is not suitable for satisfying the requirement). On the other hand, a weight of 0.9 means the probability that the architectural style will satisfy the non-functional requirement is very high (the style is suitable for satisfying the requirement).

The weights of the contribution links are assigned after reviewing different studies, evaluations, and comparisons involving the architectural styles under evaluation. When the contribution weights for each architectural style to the different non-functional requirements of the system have been assigned, the best-suited architectural style is decided. This decision involves the categorization of the non-functional requirements according to the importance to the system and the identification of the architectural style that best satisfies the most important non-functional requirement using a propagation algorithm, such as the one presented by Giorgini et al. [16].

In this example, we consider two architectural styles, a hierarchical style – *client/server* - and a mobile code style *-mobile agents*. As shown in figure 6, each of the two styles satisfies differently each of the non-functional requirements of the system. For instance, the mobile agents style allows more scalable applications (weight 0.8), because of the dynamic deployment of the mobile code.

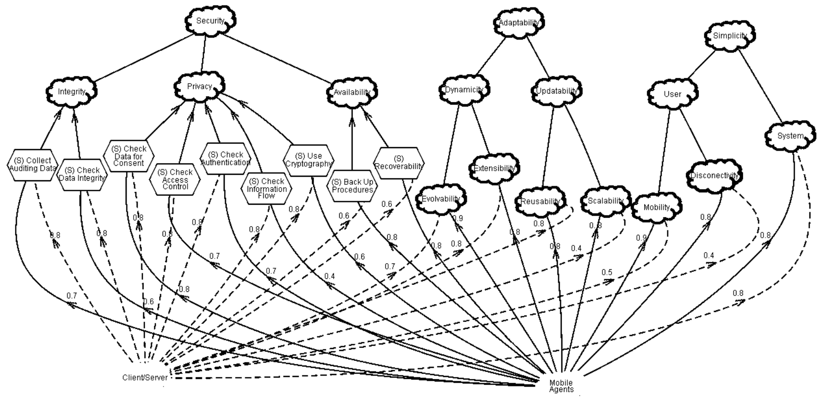

Fig. 6. client-Server versus Mobile Agents

As concluded from the analysis presented in figure 6 (and elaborated more in [9]), the client/server style satisfies more the privacy requirements of the system than the mobile agents style. This is mainly because mobility is involved in the mobile agents style. Therefore, although protection of a server from mobile agents, or generally mobile code, is an evolution of security mechanisms applied in other architectural styles, such as client/server; the mechanisms focused on the protection of the mobile agents from the server cannot, so far, prevent malicious behaviour from occurring but may be able to detect it. Consider for example, the Check Information Flow secure task of the eSAP. The information flow property is more easily damaged by employing mobile agents (weight 0.4) since possible platforms that a mobile agent could visit might expose sensitive information from the agent. In the case of the client/server style (weight 0.8) sensitive information is stored in the server and existing well-proven security measures could be taken to satisfy the information flow attribute.

The third activity of the security process involves the development of a design that satisfies the security requirements of the system. For this, a pattern language consisting of security patterns for multiagent systems is proposed and this language is integrated within the development process of the Tropos methodology. Security patterns document proven solutions to security related problems in such a way that are applicable by non-security specialists. Therefore, the application of security patterns in the development of multiagent systems can provide effective answers to the above-mentioned questions, since non-security specialists can rely on expert knowledge and apply well-proven solutions to solve security problems in a structured and systematic way. The use of security patterns enables non-security specialists to identify patterns for transforming the security requirements of their system into design, and also be aware of the consequences that each of the applied security patterns introduce to their system. Additionally, because security patterns capture well-proven solutions, it is more likely that the application of security patterns will satisfy the security requirements of the system.

Therefore, we have developed a security pattern language [9] and we have integrated it within our security-oriented process. Figure 7 describes the relationship of the patterns of the language as well as their relationship with existing patterns. Each box indicates a pattern, where a solid-line box indicates a security pattern that belongs to the language developed by this research and a dashed-line box indicates a related existing pattern. White triangles depict generalisations/ specialisation and solid lines associations of type uses/ requires.

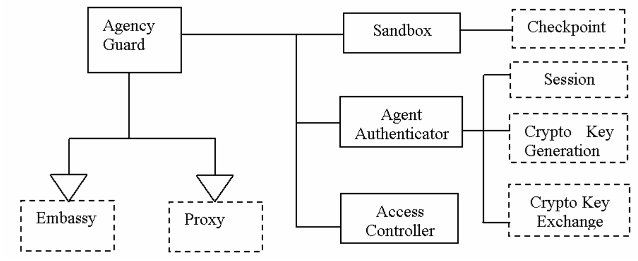

Fig. 7. The security pattern language

The AGENCY GUARD is the starting point of applying the patterns of the language and it is a variant of the *Embassy* and the *Proxy* patterns. It uses the AGENT AUTHENTICATOR pattern to ensure the identity of the agents, the SANDBOX pattern in order to restrict the actions of agents, and the ACCESS CONTROLER pattern to restrict access to the system resources.

On the other hand, the SANDBOX pattern can implement the *Checkpoint* pattern, and the AGENT AUTHENTICATOR pattern can use the *Session* pattern to store credentials of the agent. Moreover, the AGENT AUTHENTICATOR employs the *Cryptographic Key Generation* and the *Cryptographic Key Exchange* patterns for further cryptographic actions.

To understand how the patterns of the language can be applied during the development of a system, from the internal analysis of the eSAP system we have concluded that Information Flow, Authentication and Access Control checks must be performed in order for the eSAP system to satisfy the secure goal Ensure System Privacy. In the case of the Information Flow secure task, the eSAP should be able to control how information flows within the system, and between the system and other actors. For example, the system should be able to control who requires access to the system and, by considering the security policy, to grant or deny access to the system. With respect to the Authentication checks, the system should be able to authenticate any agents that send a request to access information of the system, and in the case of the Access Control, the system should be able to control access to its resources. To meet these goals, The AGENCY GUARD pattern can be used to grant/deny access to the system according to the security policy, the AGENT AUTHENTICATOR pattern can be used to provide authentication checks and the ACCESS CONTROLER pattern to perform access control checks as shown in figure 8. The use of these patterns not only satisfies the fulfillment of the secure goals of the system but also guarantees the validity of the solution.

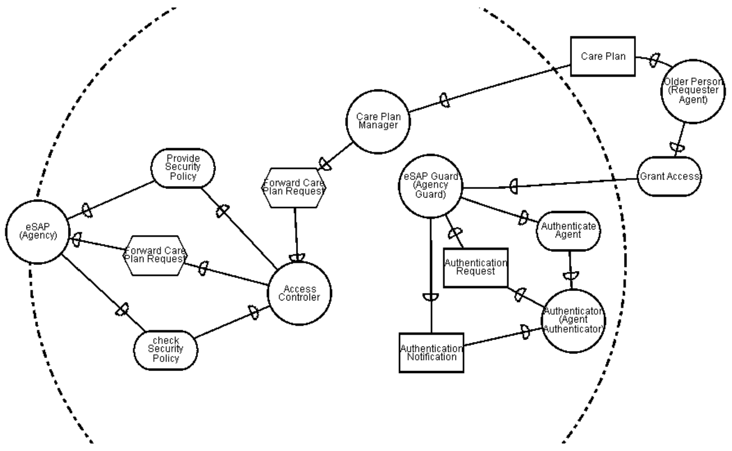
Care Plan
Older Person (Requester Agent)
Care Plan Manager
Grant Access
eSAP Guard (Agency Guard)
Provide Security Policy
Forward Care Plan Request
Authenticate Agent
eSAP (Agency)
Forward Care Plan Request
Authentication Request
Access Controler
Authenticator (Agent Authenticator)
check Security Policy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

Fig. 8. Using the pattern language

The last activity of the security process in secure Tropos involves the testing of the developed solution against potential security attacks. For this reason, we have developed a process that is based on security attack scenarios.

A Security Attack Scenario (SAS) is defined *as an attack situation describing the agents of a multiagent system and their secure capabilities as well as possible attackers and their goals, and it identifies how the secure capabilities of the system prevent (if they prevent) the satisfaction of the attackers' goals* [9].

A security attack scenario involves possible attacks to a multiagent system, a possible attacker, the resources that are attacked, and the agents of the system related to the attack. An attacker is depicted as an agent who aims to break the security of the system. The attacker intentions are modelled as goals and tasks and their analysis follows the same reasoning techniques that the Tropos methodology employs for goal and task analysis. Attacks are depicted as dash-lined links, called attack links, which contain an "attacks" tag, starting from one of the attacker's goals and ending at the attacked resource.

The process is divided into three main stages [9]: creation of the scenario, validation of the scenario, and testing and redefinition of the system according to the scenario. During the creation of a scenario, Tropos goal diagram notation is used for analysing the intentions of an attacker in terms of goals and tasks, identify a set of attacks according to the attacker's goals, and also identify the agents of the system that posses capabilities to prevent the identified attacks.

When the scenarios have been created, they must be validated. Therefore, during the scenario validation process software inspections are used. The inspection of the scenarios involves the identification of any possible violations of the Tropos syntax and of any possible inconsistencies between the scenarios and the models of the previous stages. Such an inspection involves the use of validation checklists. Such a check list has been proposed for instance in [9].

When the scenarios have been validated, the next step aims to identify test cases and test, using those test cases, the security of the system against any potential attacks. Each test case is derived from a possible attack depicted in the security attack scenarios. The test cases are applied and a decision is formed to whether the system can prevent the identified attacks or not. The decision whether an attack can be prevented (and in what degree) or not lies on the developer. However as an indication of the decision it must be taken into consideration that at least one secure capability must help an attack, in order for the developer to decide the attack can be prevented. Attacks that cannot be prevented are notated as solid attack links, as opposed to attacks that the system can prevent and which are notated as dashed attack links.

For each attack that it has been decided it cannot be prevented, extra capabilities must be assigned to the system to help towards the prevention of that attack. In general, the assignment of extra secure capabilities is not a unique process and depends on the perception of the developer regarding the attack dangers. However, a good approach could be to analyse the capabilities of the attacker used to perform the attack and assign the system with capabilities that can revoke the attacker's capabilities.

For instance, let us consider an interception attack scenario in which a possible attacker wishes to attack the privacy of the system, in other words to obtain information such as assessment information or a care plan. As identified in the analysis of the security reference diagram, social engineering, password sniffing and eavesdropping are the main threats to the privacy of the system. Therefore, the attacker's main goal can be decomposed to Read Data and Get Access to the System sub-goals as shown figure 9. The first sub-goal involves the attacker trying to read the data that it is transmitted to and from the eSAP system, whereas the second sub-goal involves the attacker trying to break into the system and gain access to it.

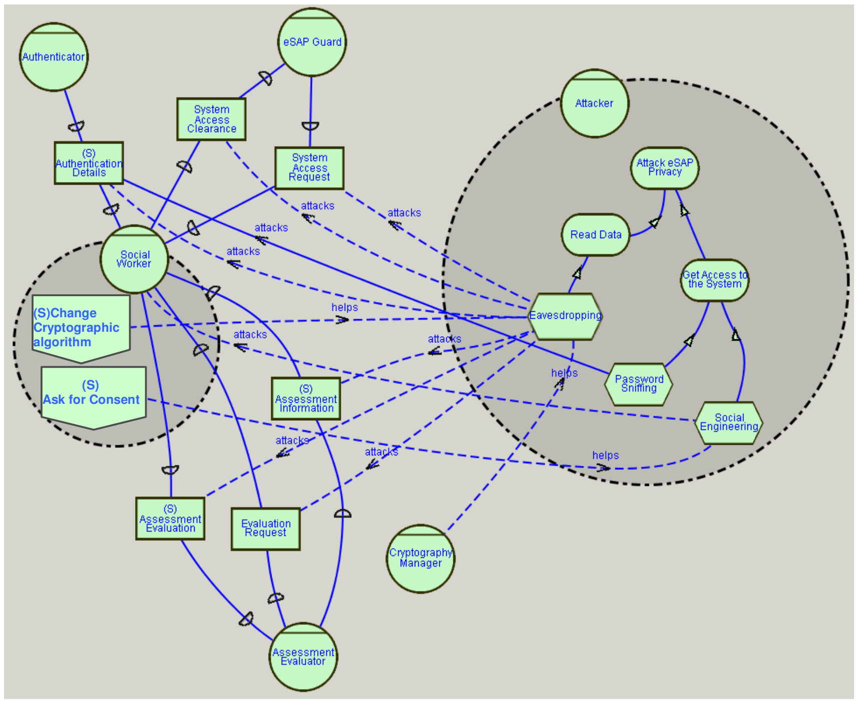

Fig. 9. An example of a security attack scenario

To accomplish the first sub-goal the Attacker should try to read the data transferred between the Social Worker and the eSAP system's actors such as the Assessment Evaluator and the Authenticator. To accomplish the second sub-goal, the Attacker might use password sniffing or social engineering. In the first case, the Attacker scans all the resources that flow in the network looking for passwords whereas in the case of social engineering, the Attacker tries to deceive the Social Worker in order to obtain valuable information, such as their authorisation details that will allow them to gain access to the system.

Therefore, for the presented attack scenario, the reaction of the system should be tested against three test cases, read data, password sniffing and social engineering. Due to lack of space we only present the read data test case as shown below.

Test Case 1: read data
Precondition: The Social Worker actor tries to obtain an assessment evaluation. The Attacker tries to read the transmitted data.
System expected security reaction: The system should prevent Attacker from

reading any important information.
Discussion: The Attacker will try to read the data from any resource transmitted between the external agents and the eSAP system. However, curerntly the system and its external agents have capabilities to encrypt and decrypt data. As a result all the important data is transmitted across the network encrypted and therefore it is difficult for the Attacker to read it. However, the Attacker might try to obtain (or sometimes even guess) the encryption key.
Test Case Result: The system is protected against read data attacks. However, a recommendation would be for the system to have capabilities to change the cryptographic algorithm often.

5 Conclusions

This paper presents results from our work to extend Tropos methodology to enable it to consider security requirements throughout its development stages. During the process of extending Tropos some very useful observations were obtained. First of all, the concept of constraints is a natural extension of the Tropos methodology and it allows for a systematic approach towards the modelling of security requirements. This is because, although functional and security requirements are defined alongside, a clear distinction is provided. Secondly, the security diagram allows identifying desired security requirements very early in the development stages, and helps to propagate them until the implementation stage, introducing a security-oriented paradigm to the software process. In addition, the iterative nature of the methodology, allows the re-definition of security requirements in different levels therefore providing a better integration with system functionality.

However, this in an ongoing research and more work is required to achieve our aim, which is to provide a well guided process of integrating security and functional requirements throughout the software development process of agent-based systems, using the same concepts and notations throughout the process. Currently we are working on refining the identified concepts, notations, and the process, and we are integrating our extensions to the Formal Tropos [6] specification language. This will enable us to formally evaluate our extensions, since Formal Tropos is amenable to formal analysis.

References

[1] N. R. Jennings, M. Wooldridge, “Agent-Oriented Software Engineering” *in Handbook of Agent Technology* (ed. J. Bradshaw) AAAI/MIT Press 2001

[2] C. Iglesias, M. Garijo, J. Gonzales, “A survey of agent-oriented methodologies”, *Intelligent Agents IV*, A. S. Rao, J. P. Muller, M. P. Singh (eds), Lecture Notes in Computer Science, Springer-Verlag, 1999

[3] M. Wooldridge, N. R. Jennings, D. Kinny, “ The GAIA Methodology for Agent-Oriented Analysis and Design”, *Journal of Autonomous Agents and Multi-Agent Systems 3*, (3) pp. 285-312, 2000

[4] W. Stallings, “Cryptography and Network Security: Principles and Practice”, Second Edition, Prentice-Hall 1999.

[5] 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.

[6] A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. in *Proc. of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, 28 May - 1 June 2001.

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

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

[9] H. Mouratidis, “A Security Oriented Approach in the Development of Multiagent Systems: Applied to the Management of the Health and Social Care Needs of Older People in England”, PhD thesis, University of Sheffield, England, 2004

[10] P. Bresciani, A. Perini, P. Giorgini, G. Giunchiglia, J. Mylopoulos, “Modelling early requirements in Tropos: a transformation based approach”, Agent Oriented Software Engineering II, M. Wooldridge, G. Weiβ (eds), Lecture Notes in Computer Science, Springer-Verlag 2222, 2002

[11] H. Mouratidis, i. Philp, G. Manson, “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, June 2002

[12] L. Liu, E. Yu, J. Mylopoulos, “Analysing Security Requirements as Relationships Among Strategic Actors”, in the Proceedings of 2nd Symposium on Requirements Engineering for Information Security, North Carolina - USA, November 2002.

[13] T. Lodderstedt, D. Basin, J. Doser, “SecureUML: A UML-Based Modelling Language for Model-Driven Security”, in the Proceedings of the 5th International Conference on the Unified Modeling Language, 2002.

[14] M.P. Huget, “Nemo: An Agent-Oriented Software Engineering Methodology”, in the Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle – USA, November 2002.

[15] E. Yu, “Modelling Strategic Relationships for Process Reengineering”, PhD thesis, Department of Computer Science, University of Toronto, Canada, 1995

[16] P. Giorgini, J. Mylopoulos, E Nicchiarelli, R, Sebastiani, “Reasoning with Goal Models”, in the Proceedings of the 21st International Conference on Conceptual Modeling (ER2002), Tampere, Finland, October 2002.