Author(s): Mouratidis, Haralambos; Giorgini, Paolo
Article title: Enhancing secure Tropos to effectively deal with security requirements in the development of multiagent systems
Year of publication: 2009
Citation: Mouratidis, H; Giorgini, P. (2009) 'Enhancing secure Tropos to effectively deal with security requirements in the development of multiagent systems' In: Barley, M. et al (Eds) SASEMAS 2004-2006, LNAI 4342 pp 8-26
Link to published version: http://dx.doi.org/10.1007/978-3-642-04879-1_2
DOI: 10.1007/978-3-642-04879-1_2

Enhancing secure Tropos to effectively deal with security requirements in the development of multiagent systems

H. Mouratidis[1], P. Giorgini[2]

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

Abstract. The consideration of security requirements in the development of multi-agent systems is a very difficult task. However, only few approaches have been proposed that try to integrate security issues as internal part of the development process. Amongst them, secure Tropos has been proposed as a structured approach towards the consideration of security issues in the development of multiagent systems. In this paper we enhance secure Tropos by integrating to its stages: (i) a process for selecting amongst alternative architectural styles using as criteria the security requirements of the system; (ii) a pattern-based approach to transform security requirements to design, and (iii) a security attack scenarios approach to test the developed solution. The electronic single assessment process (eSAP) case study is used to illustrate our approach.

1 Introduction

Recently agent orientation is presented as the next major paradigm for the development of large complex computer systems. Although there are some convincing arguments for believing that agent orientation will be of benefit for engineering certain complex software systems [6], much work is still required so that it becomes widely accepted as a major development paradigm, and more research is required towards many areas related to it. One of these areas is security.

Security of information systems is an area that has received great attention the last few decades mainly due to the storage of sensitive information on computer systems, and the wide interconnection of these systems through networks and the Internet. As a result, security research is considered one of the most active areas of computer science and engineering research. Nevertheless the high amount of work on this area, current information systems are not considered totally secure. According to the U.S. National Research Council [17] poor design is one of the major reasons for the development of insecure systems. Current work on information systems security has been focused on the development and verification of security protocols and other low level solutions, and the consideration of security requirements as part of the whole system development has been neglected.

This is also the case for multiagent systems, since current agent oriented methodologies do not usually consider security as an integral part of their

development stages and as a result agent developers do not actually find any help when considering security during the stages of the system development. This is mainly because the consideration of security in the development phases is a demanding and difficult task, due to the following reasons:

(a) developers, who are not security specialists, usually need to develop multiagent systems that require knowledge of security;
(b) Many different concepts are used between security specialists and software engineers. As a result, there is an abstraction gap that makes the integration of security and software engineering more difficult;
(c) there is an ad hoc approach towards security analysis;
(d) It is difficult to define together security and functional components and at the same time provide a clear distinction. For instance, which components are part of the security architecture and which ones are part of the functional specification;
(e) It is difficult to move from a set of security requirements to a design that satisfies these requirements, and also understand what are the consequences of adopting specific design solutions for such requirements;
(f) It is difficult to get empirical evidence of security issues during the design stages. This makes the process of analysing security during the design stage more difficult;
(g) It is difficult to fully test the proposed security solutions at the design level.

We believe that the agent oriented software engineering paradigm presents a feasible approach for the integration of security to software engineering due to the appropriateness of agent oriented philosophy, for dealing with the security issues that exist in a computer system. Security requirements are mainly obtained by analysing the attitude of the organisation towards security and after studying the security policy of the organisation. As mentioned in [6] agents act on behalf of individuals or companies interacting according to an underlying organisation context. The integration of security within this context will require for the rest of the subsystems (agents) to consider the security requirements, when specifying their objectives and interactions therefore causing the propagation of security requirements to the rest of the subsystem. In addition, the agent oriented view is perhaps the most natural way of characterising security issues in software systems. Characteristics, such as autonomy, intentionality and sociality, provided by the use of agent orientation allow developers first to model the security requirements in high-level, and then incrementally transform these requirements to security mechanisms [12].

In previous work [13, 14, 15] we have presented models and techniques towards the solution of problems a, b, c, and d. For example, we proposed a well guided security-oriented process that considers the same concepts and notations throughout the development lifecycle and it allows the parallel definition of security and functional requirements providing at the same time a clear distinction.

In this paper we extend our previous work to deal with problems e, f and g by extending the current secure Tropos. In particular, we propose a process for selecting amongst alternative architectural styles, a pattern-based approach to transform the

analysis to design, and a security attack scenarios approach to test the developed solution.

Our work is not the only one in the agent paradigm pool that tries to integrate security issues as an internal part of the development process. Liu et al. [10] presented work in which security requirements are analysed as relationships between actors, such as users, stakeholders and potential attackers. Liu proposes three different kinds of analysis techniques: agent oriented, goal oriented and scenario based analysis. Agent oriented analysis is used to model potential threats and security measures, whereas goal oriented analysis is employed for the development of a catalogue to help towards the identification of the different security relationships on the system. Finally, the scenario based analysis is considered an elaboration of the other two kinds of analysis. Yu and Cysneiros [21] use 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. However, these approaches only guide the consideration of security issues on specific development stages, in other words both of them are focused only in the requirements engineering area. Our approach, in contrast, considers security issues throughout the development process. As indicated in [3], it is important to consider security issues throughout the development process.
Moreover, Huget [5] has proposed an agent oriented software methodology, called Nemo, which considers some security aspects. However, in his approach security is not considered as a specific concept but it is integrated within the other models of the methodology. As indicated earlier, it is important to model together security and functional requirements but at the same time provide a clear distinction. Moreover, Nemo tackles security quite superficially and as Huget states [5] “particularly, security has to be intertwined more deeply within the models”.

The rest of the paper is structured as follows. Section 2 of the paper provides an overview of secure Tropos, mainly for readers not familiar with the methodology, whereas Section 3 introduces our approach providing answers to the problems e, f, and g presented above. Section 4 concludes the paper.

2 An overview of Secure Tropos

Tropos [1] is an agent oriented software engineering methodology, in which notions such as actors (entities that have strategic goals and intentionality), goals (an actor’s strategic interests), soft-goals (goals without clear criteria whether they are satisfied or not), tasks (represent in an abstract level a way of doing something), resources (represent a physical or informational entity) and intentional dependencies (indicate that one actor depends on another in order to attain some goals, execute some tasks, or deliver a resource) are used in all the phases of the system development from the first phases of the early analysis, down to the actual implementation.

The Tropos methodology is mainly based on four phases [1]: *Early Requirements Analysis*, aimed at defining and understanding a problem by studying its existing organizational setting; *Late Requirements Analysis*, conceived to define and describe the system-to-be, in the context of its operational environment; *Architectural Design*, that deals with the definition of the system global architecture in terms of subsystems;

and the *Detailed Design* phase, aimed at specifying each architectural component in further detail, in terms of inputs, outputs, control and other relevant information.

During the phases of early and late requirements analysis Tropos employs two main types of diagrams: the actor diagram and the goal diagram. An actor diagram, describes the actors (depicted as circles), their goals (depicted as oval and bubble shapes) and the network of dependency relationships amongst the actors (two arrowed lines connected by a graphical symbol varying according to the dependum, i.e. goal, task, or resource). An example is given in Figure 2. A goal diagram represents the analysis of an actor's goals, conducted from the view point of the actor, by using three basic reasoning techniques: means-end analysis, contribution analysis and AND/OR decomposition. It is drawn as a balloon and contains graphs whose nodes are goals (ovals) and/or tasks (hexagonal shape) and whose arcs are the different relationships that can be identified among its nodes. An example is given in Figure 3.

Although, the Tropos methodology was not conceived with security in mind, we have presented in previous work a set of security related concepts [13, 14, 15] (some resulted from security related extensions of existing concepts), to enable it to model security issues throughout the development of multiagent systems. This security oriented extension, which is known as secure Tropos, includes the following security related concepts.

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 the information 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 [12]. Graphically a security constraint is depicted as a cloud and it is positioned in the side of the actor who has to satisfy it (see for instance Figure 2).

Additionally to security constraints, Secure Tropos defines secure dependencies. *A secure dependency* introduces security constraint(s) that must be fulfilled for the dependency to be satisfied. Both the depender and the dependee must agree for the fulfilment of the security constraint in order for the secure dependency to be valid. That means the depender expects from the dependee to satisfy the security constraint(s) and also that the dependee will make an effort to deliver the dependum by satisfying the security constraint(s).

Secure Tropos uses the term secure entity to describe any goals and tasks related to the security 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 [12].

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.

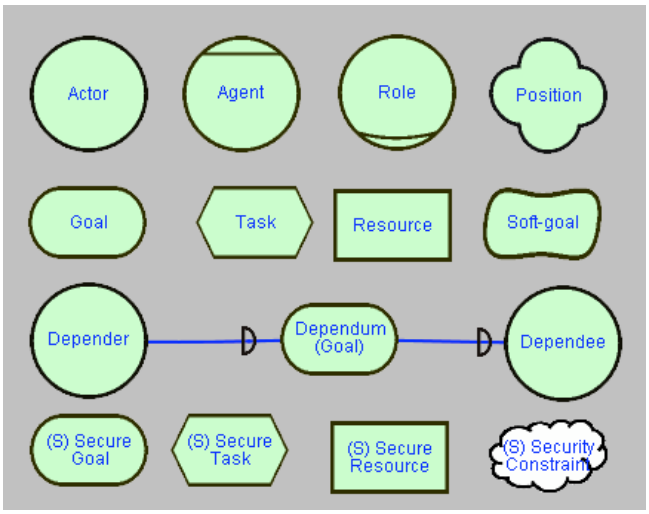

Fig. 1. Tropos and Secure Tropos notation

It is worth mentioning that the process in secure Tropos is one of analysing the security needs of the stakeholders and the system in terms of security constraints imposed to the stakeholders (early requirements) and the system (late requirements), identifying secure entities that guarantee the satisfaction of the security constraints, and assigning capabilities to the system (architectural design) to help towards the satisfaction of the secure entities. Security requirements are identified by employing the modelling activities of secure Tropos [12], such as security reference diagram construction, security constraints and secure entities modelling. In particular, the security constraints imposed to the system and the stakeholders, are identified and secure goals and entities that guarantee the satisfaction of the identified security constraints are imposed to the actors of the system.

The secure Tropos process allows for two types of validation. A model validation and design validation. The model validation involves the validation of the developed models (for example, the goal diagram or the actor diagram) with the aid of a set of validation rules [12]. The design validation aims to validate the developed solution against the security policy of the system. A key feature of the Secure Tropos that allows us to perform such a validation is the fact that the same secure concepts are used throughout the development stage. Moreover, the definition of these concepts allows us to provide a direct map between them, and therefore be able to validate whether the proposed security solution satisfies the security policy.

3 Enhancing secure Tropos

Secure Tropos provides solutions towards the first four problems identified in the Introduction. The approach described in this section enhances our work on secure Tropos in order to provide answers to problems e, f, and g. To achieve this aim, we integrate into the current secure Tropos process three sub-activities; (1) the system's architectural style selection according to its security requirements; (2) the

transformation of the security requirements to a design that satisfies these requirements; (3) and the attack testing of the multiagent system under development.

3.1 Illustrating the approach

To better illustrate the above sub-activities and their integration within the secure Tropos, we consider a complete description of the electronic single assessment process (eSAP) case study that we have introduced in previous work [16]. After analyzing the security requirements of the individual actors during the early requirements analysis [12], the electronic Single Assessment Process system (eSAP) is introduced and responsibility is delegated to it for some of the actors' goals. Since dependencies are delegated from the actors to the eSAP system, possible security constraints regarding those dependencies are also delegated as shown in Figure 2.

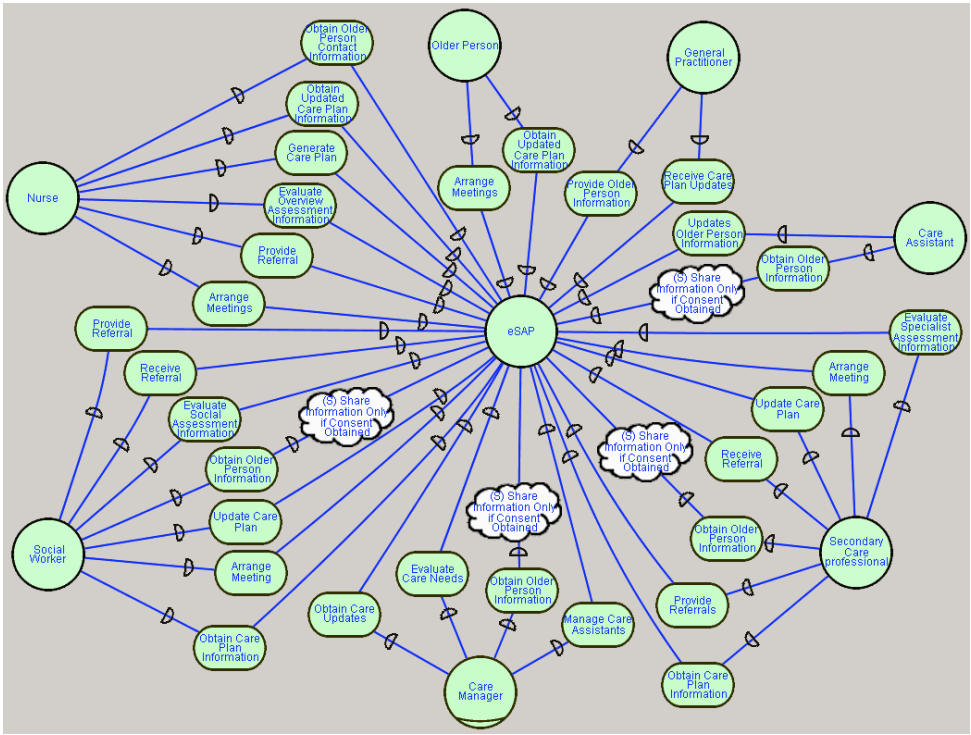

Fig. 3. Actor's diagram including the eSAP system

For example, before the introduction of the eSAP system, the Social Worker was depending on the Nurse to Obtain Older Person Information. However, this secure dependency involves the security constraint (restricting the Nurse) Share Information Only if Consent Obtained. With the introduction of the eSAP system, the Social Worker actor depends on the eSAP to Obtain Older Person Information, therefore the eSAP becomes responsible for satisfying the Share

Information Only if Consent Obtained security constraint that is delegated together with the secure dependency.

Then, the eSAP system is analysed and secure goals/tasks are imposed to the system to help towards the satisfaction of its security constraints as shown in Figure 3. For example, the Keep System Data Private security constraint can be fulfilled by blocking access to the system, by allowing access only from a central computer, or by ensuring system privacy. However, the first two contribute negatively to the usability of the system, i.e. the system will be secure but it will not be used. On the other hand, the Ensure System Privacy secure goal is considered the best solution since it provides security to the system and it doesn't affect (dramatically) its usability.

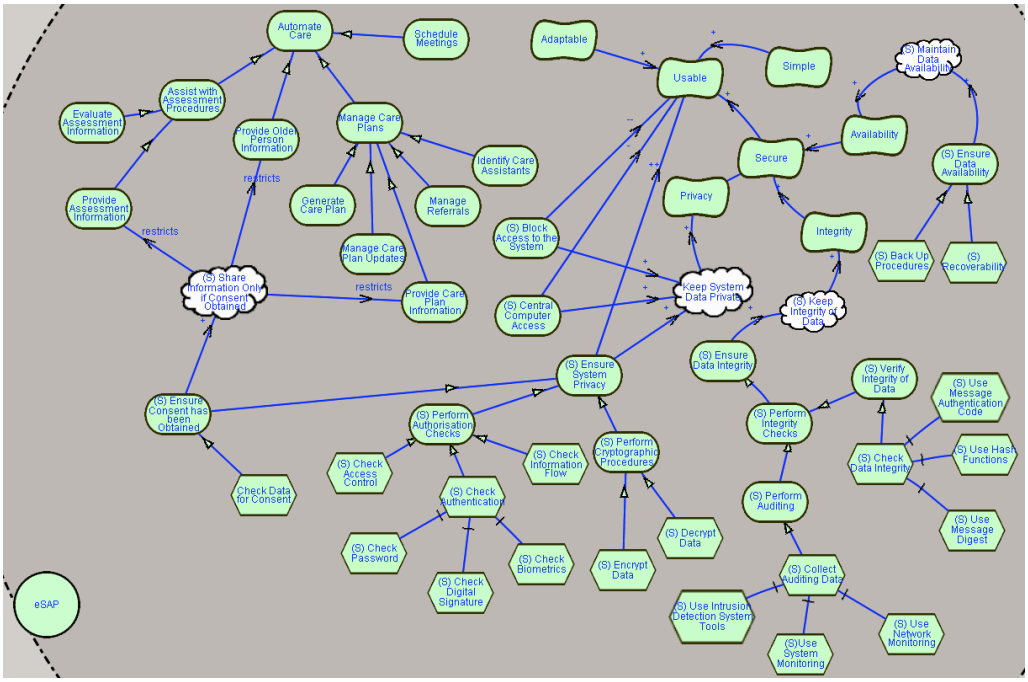

Fig. 4. Goal diagram for the eSAP system

When the requirements analysis is complete, the next stage is the architectural design. According to [2] one of the most important issues is to identify the architectural style that is more suitable for the eSAP system. For instance, we could try to identify whether a client/server or a mobile agent architectural style is more suitable for our system.

3.1.1 Selecting the system's architecture according to its security requirements

An important requirement of a security-oriented approach is to allow developers to explore different architectural designs or in other words, to allow developers to reason about alternative design solutions according to the security requirements of a multiagent system.

For this reason, this research has developed an analysis technique to enable developers to select among alternative architectural styles[1] using as criteria the non-functional requirements of the multiagent system under development. Although, this process can be used to select an architectural style according to any of the non-functional requirements of the system, as mentioned above, one of the most important issues in the eSAP is security. Consequently it is natural to decide about the architectural style of the system taking into account the issues involved, in other words the system's security requirements.

We use the measure of satisfiability [4] to compare different architectural styles with respect to non-functional requirements (such as security). S*atisfiability* represents the probability that a non-functional requirement will be satisfied. The contribution of each style to the satisfiability of non-functional requirements is expressed by a weight ranging 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. Figure 4 indicates the non-functional requirements of the eSAP system (as identified during the early and late requirements analysis –Figures 2,3) along with the satisfiability contributions from the client/server and the mobile agents architectural styles.

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. [4].

From earlier analysis (see Figure 3) it is concluded that security of the eSAP involves privacy, integrity and availability. Therefore, it is desirable to select a suitable architectural style for the system having these security requirements as criteria. According to the propagation algorithm, the satisfiability for the security is given by the following:

$$S_{(security)} = Min\,(S_{(Privacy)}, S_{(integrity)}, S_{(availability)})$$

where

$$S_{(Integrity)} = Min\,(S_{(Collect\ auditing\ data)}, S_{(check\ Data\ Integrity)})$$

$$S_{(Privacy)} = Min\,(S_{(check\ data\ for\ consent)}, S_{(Check\ access\ control)}, S_{(Check\ authentication)}, S_{(Check\ Information\ Flow)}, S_{(Check\ Cryptography)})$$

$$S_{(Availability)} = Min\,(S_{(Back\ up\ procedures)}, S_{(Recoverability)})$$

[1] To avoid confusion it must be noted that architectural styles differ from architectures in that " a style can be thought of as a set of constraints on an architecture" [Bas98].

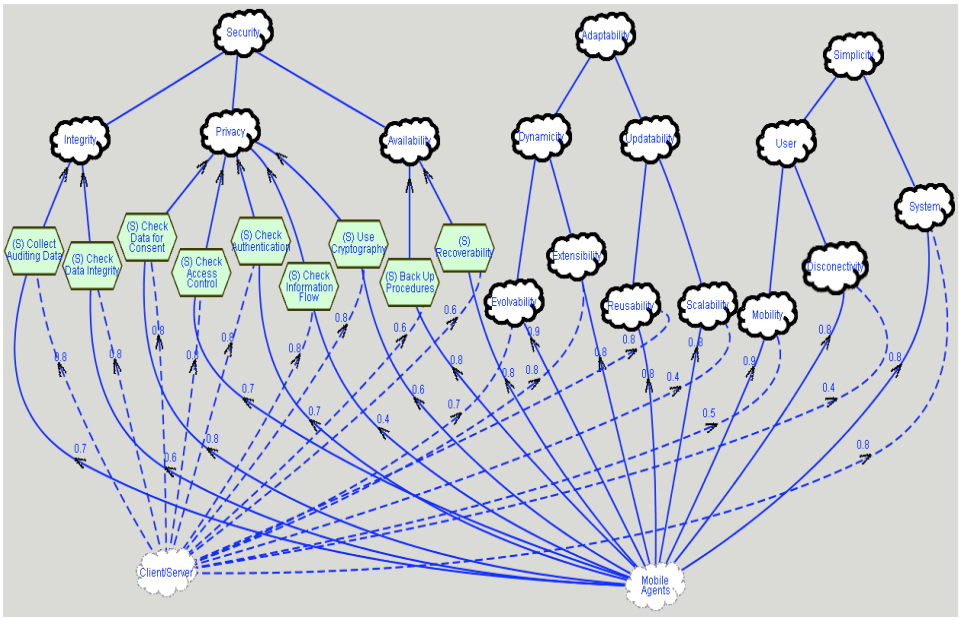

Fig. 5. Selecting between architectural styles

In the presented example, for the client/server architecture we have the following values (as derived from Figure 4): $S_{(\text{Collect auditing data})} = 0.8$, $S_{(\text{check Data Integrity})} = 0.8$, $S_{(\text{check data for consent})} = 0.8$, $S_{(\text{Check access control})} = 0.8$, $S_{(\text{Check authentication})} = 0.8$, $S_{(\text{Check Information Flow})} = 0.8$, $S_{(\text{Check Cryptography})} = 0.8$, $S_{(\text{Back up procedures})} = 0.6$, $S_{(\text{Recoverability})} = 0.6$.

Therefore, $S_{(\text{Integrity})} = 0.8$, $S_{(\text{Privacy})} = 0.8$, $S_{(\text{Availability})} = 0.6$, and as a result S (Security) = 0.6. that means the probability that the client/server architecture will satisfy the security requirements of the system is 60%.

Applying the same technique to calculate the probability that the mobile agent will satisfy the security requirements of the system we have the following:

$S_{(\text{Collect auditing data})} = 0.7$, $S_{(\text{check Data Integrity})} = 0.6$, $S_{(\text{check data for consent})} = 0.8$, $S_{(\text{Check access control})} = 0.7$, $S_{(\text{Check authentication})} = 0.7$, $S_{(\text{Check Information Flow})} = 0.4$, $S_{(\text{Check Cryptography})} = 0.6$, $S_{(\text{Back up procedures})} = 0.8$, $S_{(\text{Recoverability})} = 0.8$.

Therefore, $S_{(\text{Integrity})} = 0.6$, $S_{(\text{Privacy})} = 0.4$, $S_{(\text{Availability})} = 0.8$, and as a result S (Security) = 0.4, meaning that the probability that the mobile agents architectural style will satisfy the security requirements of the system is 40%.

As a result, for the given security requirements, the client/server style satisfies better the security of the eSAP system.

3.1.2 Transform the analysis to design

When the architectural style has been chosen, the next step of the architectural design stage aims to decompose the system in order to identify internal actors who will satisfy the system's (secure) goals. In the presented example, the eSAP actor is decomposed to internal actors and the responsibility for the fulfilment of the eSAP's goals is delegated to these actors as shown in Figure 5.

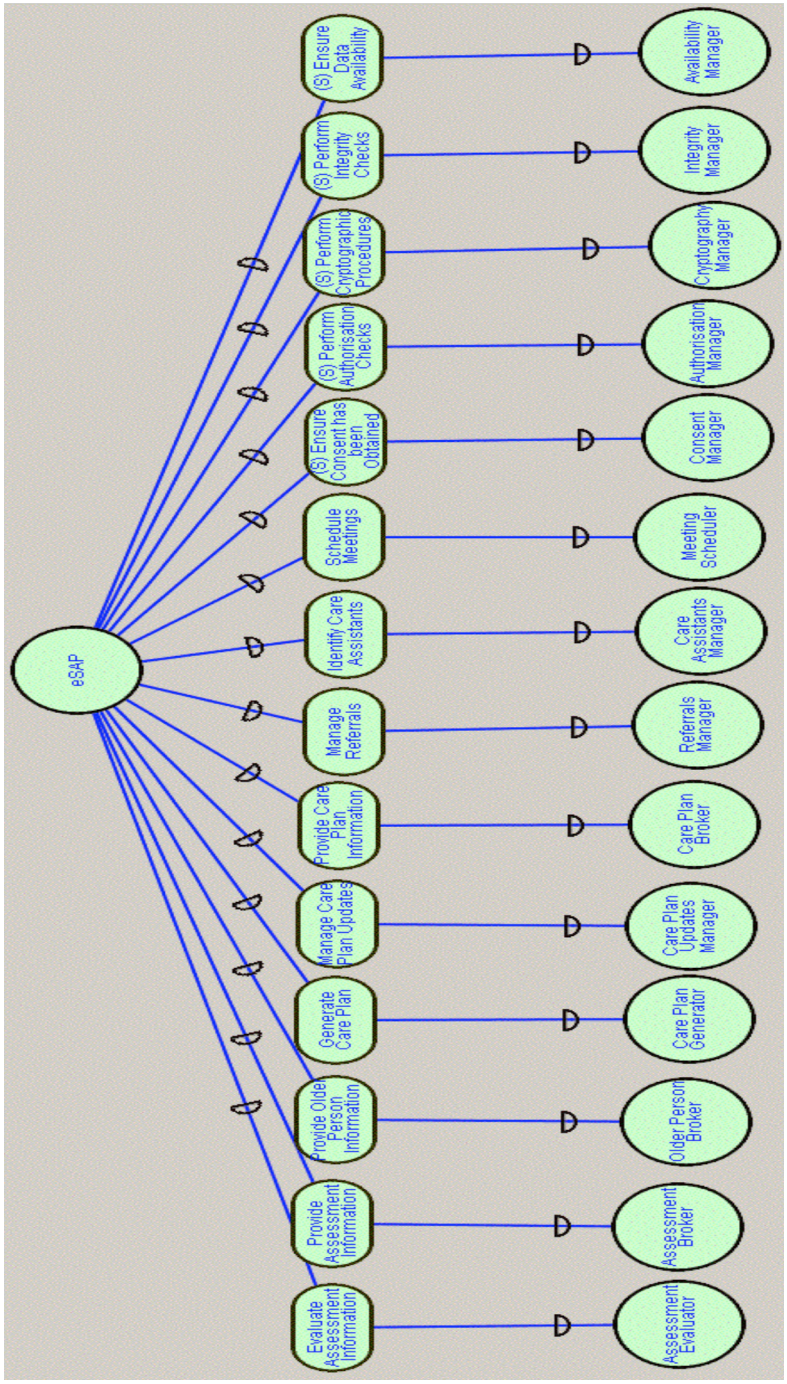

Fig. 7. eSAP actor partial decomposition

For instance, the Evaluate Assessment Information goal is delegated to the Assessment Evaluator, whereas the Provide Assessment Information goal is delegated to the Assessment Broker. In addition, the Older Person Broker and the Consent Manager actors have been introduced to the eSAP system to fulfill the responsibility of the eSAP system to satisfy the secure dependency Obtain Older Person Information together with the Share Information Only if Consent Obtained security constraint.

With respect to security the identification of some of the actors is a difficult task, especially for developers with minimum knowledge of security. To help developers, security patterns can be used. "*A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution*" [19]. In other words, 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 greatly help to identify the required actors in a structured manner that does not put in danger the security of the system by providing a solution customised to the problem. 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 [15] and we have integrated it within our security-oriented process. Figure 6 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.

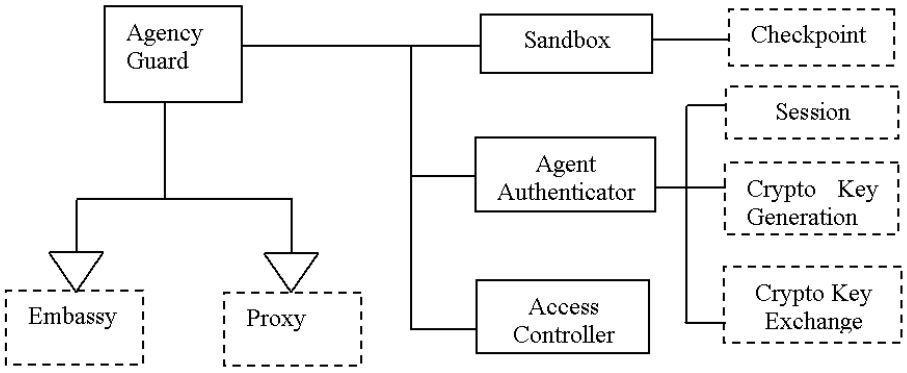

Fig. 8. The pattern language roadmap

The AGENCY GUARD is the starting point of applying the patterns of the language and it is a variant of the *Embassy* [7] and the *Proxy* [18] 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* [20] pattern, and the AGENT AUTHENTICATOR pattern can use the *Session* [20] pattern to store credentials of the agent. Moreover, the AGENT AUTHENTICATOR employs the *Cryptographic Key Generation* [9] and the *Cryptographic Key Exchange* [9] patterns for further cryptographic actions.

To understand how the patterns of the language can be applied during the development of a system, consider the internal analysis of the eSAP system (see Figure 3). It was 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 7. 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.

Moreover, developers know the consequences that each pattern introduces to the system. In the presented example, for instance, the application of the AGENCY GUARD means that only the AGENCY GUARD must be tested for correct enforcement of the agency's security policy (consequence), the application of the AGENT AUTHENTICATOR means that during implementation only the AGENT AUTHENTICATOR must be checked for assurance (consequence), whereas the application of the ACCESS CONTROLER means that different policies can be used for accessing different resources (consequence).

[2] The use of italics in this section indicates patterns not developed by this research

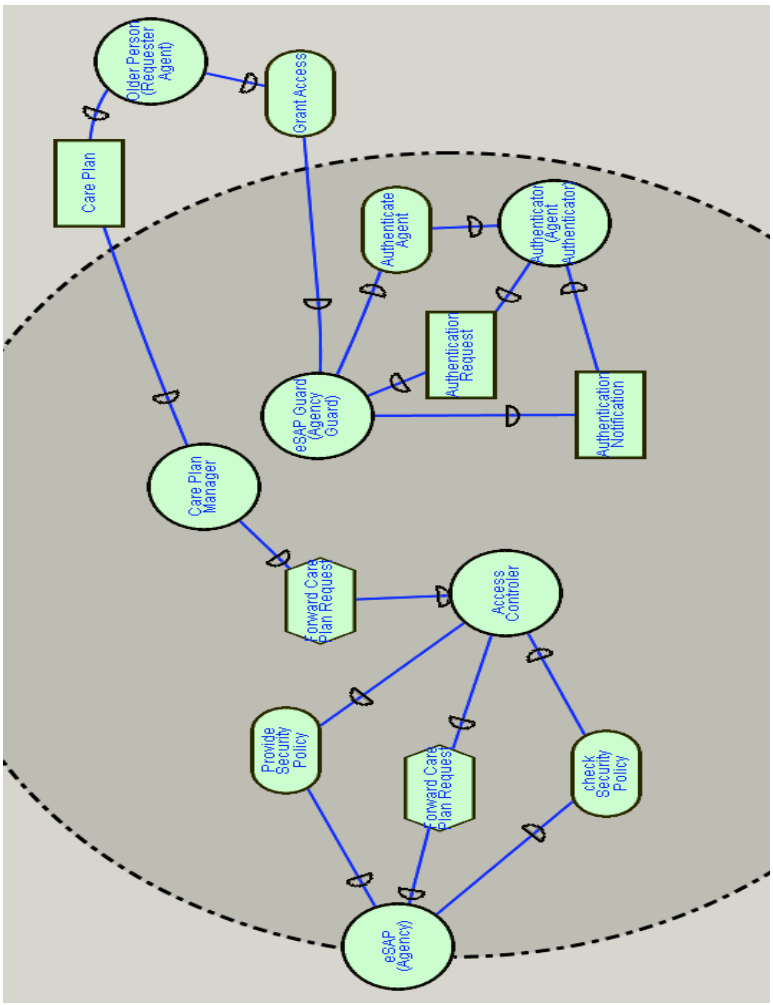

Fig. 9. Using the pattern language

3.1.3 Testing the developed solution

When the agents of the system have been identified along with their secure capabilities, it is very important for developers to test how their solution copes with potential 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.*

Security attack scenarios aim to test how the system copes with different kinds of security attacks. Therefore a scenario includes enough information about the system

and its environment to allow validation of the security requirements. 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 [12]: creation of the scenario , validation of the scenario , and testing and redefinition of the system according to the scenario. Even though the presented process is introduced as a sequence of stages, in reality is highly iterative and stages can be interchanged according to the perception of the developers. 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. Therefore, the agents of the system related to the identified attack(s) are modelled. The secure capabilities, of each agent, that help to prevent the identified attacks are identified and dashed-links (with the tag "help") are provided indicating the capability and the attack they help to prevent.

When the scenarios have been created, they must be validated. Therefore, during the scenario validation process software inspections [8] 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 [12].

Although inspections have been proposed by this research for the validation of the security attack scenarios, other techniques could also be applied depending on the developers' experience and the nature of the system. For instance, two well known validation techniques for requirements specification are walkthroughs and prototyping [8].

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. Each test case includes a precondition (the state of the system before the attack), a system expected security reaction (how the system reacts in the attack), a discussion that forms the basis for the decision regarding the test case, and a test case result that indicates the outputs of the test case.

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, consider the scenario related to the eSAP in which the Social Worker wishes to obtain an Assessment Evaluation , and an Attacker wishes to attack the integrity of the eSAP system. As identified in the analysis of the security reference diagram [12], three main threats are involved in this kind of attack, cryptographic attacks , care plan changing and viruses. Therefore, the Attacker's main goal, Attack eSAP Integrity , can be decomposed to Modify Content of Messages, Change Values in Data Files , and Alter Programs to Perform Differently as shown in Figure 8.

The first sub-goal involves the Attacker trying to modify the content of any messages transmitted over the network. To fulfill this goal, the Attacker might try to employ cryptographic attacks to any resource transmitted between any external actors and the eSAP system. The second sub-goal indicates the Attacker trying to change the values in data files of the system. The fulfilment of this goal can be satisfied by means of changing the data of resources stored in the eSAP system. The third sub-goal indicates the attempt of the Attacker to alter a program so it performs differently. Mainly this can be achieved using viruses that can alter the behaviour of specific programs (agents) in order to enable the attacker to gain access to the system or to system's information.

Three main test cases are identified for this scenario [12], cryptographic attacks , data changing attacks and viruses attacks. For each of these attacks, a test case is constructed. In this paper we only present the password sniffing and the viruses test cases as shown in Table 1 and 2. By applying different test cases many useful results can be obtained about a system. For instance, for the presented case study the test case of the modification attack scenario identified that an agent should be introduced to the system to monitor the eSAP and take effective measurements against any possible viruses.

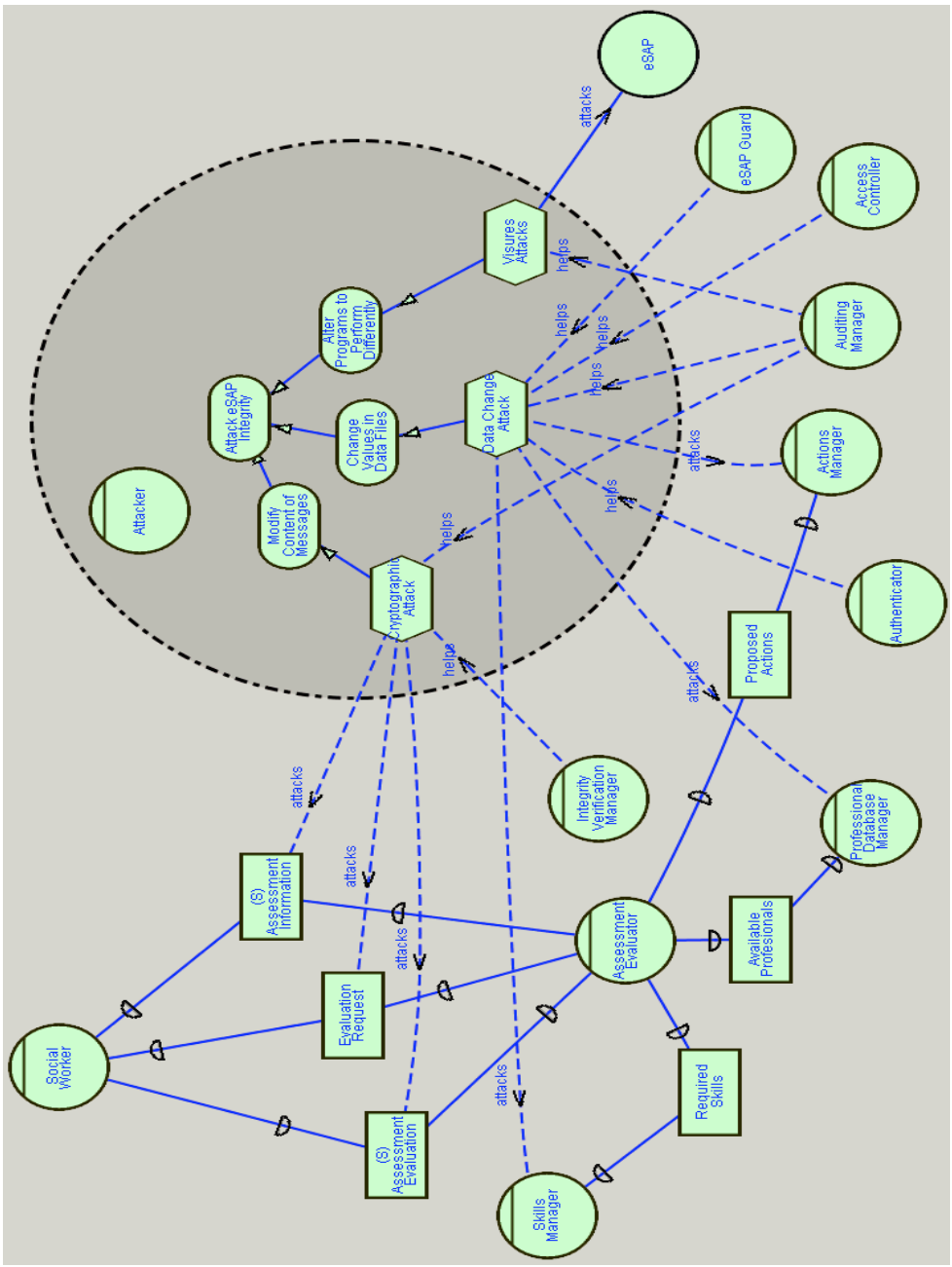

Fig. 11. An example of an attack scenario

Test Case 1: Password sniffing
Precondition: The Social Worker tries to obtain access to the eSAP system by providing their authorisation details. The Attacker tries to intercept the authorisation details.
System expected security reaction: prevent the Attacker from obtaining users' passwords
Discussion: the main target of the Attacker would be all the resource transmitions between the Social Worker and the eSAP system that contain any kind of authorisation details. Although authorisation details are enrypted, this is not enough since password sniffing takes place from a compromised computer belonging to the network. As a result, the Attacker is able to decrypt any message. A good technique to defend against password sniffing is to use one-time-passwords. A one-time-password is a password that is valid for only one use. After this use, it is not longer valid, and so even if the Attacker obtains such a password it is useless. However, the users must be able to gain access to the system more than once. This can be accomplished with what is commonly known as a password list. Each time a user tries to access the system they provide a different password from a list of passwords.
Test Case Result: Currently the system fails to adeqautely protect against password sniffing attacks. For the eSAP system to be able to react in a password sniffing attack, the external agents of the system (such as the Nurse, the Social Worker , the Older Person) must be provided with capabilities to provide passwords from a password list.

Table 1. The password sniffing

Test Case 2: Viruses
Precondition: The Attacker tries to change the system behaviour by using some kind of virus.
System expected security reaction: The system should be able to prevent viruses.

Discussion: Viruses consist one of the most sophisticated threats to computer systems. It is quite common for attackers to send viruses to computer systems they want to attack in order to exploit vulnerabilities and change the behaviour of the system. Although many effective countermeasures have been developed for existing types of viruses, many new types of viruses are also developed frequently. An ideal measurement against viruses is prevention. In other words, viruses should not get into the system. However, this is almost impossible to achieve. Therefore, the best approach is to be able to detect, identify and remove a virus. Auditing helps towards the detection of the virus. However, apart from this the eSAP system is not protected against viruses.
Test Case Results: The eSAP system needs to be integrated with an anti-virus program to enable it to effectively detect, identify and remove any possible viruses. Such a program, which could be another internal agent of the eSAP system, should be able to monitor the system and take effective measurements against any possible viruses.

Table 2. The viruses test case

By applying these test cases many useful results can be obtained for the system under development. For instance, for the eSAP system, firstly it was identified that the system provides enough protection against some of these attacks. Secondly, for the attacks that the system did not provided adequately protection, extra agents and extra secure capabilities were identified and modifications took place in the eSAP system. For example, the external agents of the system were given the capability to provide passwords from a password list, and the Authenticator was given capabilities to successfully process such passwords. The lack of such capabilities was identified by the application of the password-sniffing test case of the interception attack scenario. Moreover, an agent, called Viruses Monitor , was introduced to the system to monitor the eSAP and take effective measurements against any possible viruses. The lack of such an agent was identified by the application of the viruses test case.

4 Conclusions

In this paper, we have presented an extension to the secure Tropos methodology to deal with the problems e, f, and g identified in the Introduction. To achieve this aim, we have integrated into the secure Tropos process three sub-activities; (1) the system's architectural style selection according to its security requirements; (2) the transformation of the security requirements to a design that satisfies these requirements; (3) and the attack testing of the multiagent system under development.

Each of those activities is important for the consideration of security for different reasons. The technique for selecting amongst different architectural styles and its integration within the Tropos methodology allows the explicit definition of the technique and allows developers to evaluate and select between different designs according to the system's security requirements. This, in turn, allows developers to analyse security requirements and base design solutions on this analysis. Moreover, the integration of the pattern language within the development stages of the secure Tropos allows novice security developers to reason about the consequences a

particular design will have on their system, and therefore develop a design that will satisfy the security requirements of the system.

On the other hand, the introduction of security attack scenarios to test the system's response to potential attacks allows developers to test the developed security solution during the design stage. This, in turn, allows developers to re-consider particular system functions with respect to security until the system under development satisfies all the security requirements.

An important consideration of our approach is that it covers all the stages from the early requirements down to design, using the same concepts and notation. This is important since it allows the validation of the developed security solution by tracking the developed solution all the way back to the security requirements and the security policy. Moreover, the illustrated approach is implementation independent. Although many important issues may arise from the choice of a specific language, the consideration of such issues as part of the proposed process would restrict developers to specific solutions. We believe that a software engineering methodology should not restrict developers, and this claim is supported by various researchers within the agent oriented software engineering research area, as demonstrated by the development of a huge amount of agent oriented methodologies (for example see [1, 5, 6] that claim to be language independent and they do not consider implementation stages.

However, there are plans for future work. We first aim to integrate our approach with other related approaches, such as UMLsec, in order to extend the applicability domain. Secondly, we plan to develop tools that will allow to perform most of the, manual for the time being, checking and validations automatically. Thirdly, we plan to test the approach in a wider domain by using different case studies.

References

1. Bresciani, P., Giorgini, P, Giunchiglia, F, Mylopoulos, J., Perini, A., TROPOS: An Agent Oriented Software Development Methodology, in press, Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers.
2. Castro, J., Kolp, M., Mylopoulos, J., Towards Requirements-Driven Information Systems Engineering: The Tropos project. In Information Systems (27), pp 365-389, Elsevier, Amsterdam - The Netherlands, 2002.
3. Devanbu, P., Stubblebine, S., Software Engineering for Security: a Roadmap. Proceedings of the conference of The future of Software engineering, 2000.
4. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R., Reasoning with Goal Models, in the Proceedings of the 21st International Conference on Conceptual Modeling (ER2002), Tampere, Finland, October 2002.
5. Huget, M-P., Nemo: An Agent-Oriented Software Engineering Methodology, In Proceedings of OOPSLA Workshop on Agent-Oriented Methodologies, John Debenham, Brian Henderson-Sellers, Nicholas Jennings and James Odell, Seattle, USA, November 2002.

6. Jennings N.R., Wooldridge M., Agent–Oriented Software Engineering, in the Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99), Valencia, Spain, 1999
7. Kolp, M., Giorgini, P., Mylopoulos, J., A Goal-Based Organizational Perspective on Multi-Agent Architectures, in the Proceedings of the 8th International Workshop on Agent Theories, architectures, and languages (ATAL-2001), Seattle-USA, August 2001.
8. Kosters, G., Pagel, B.U., Winter, M., Coupling Use Cases and Class Models, Proceedings of the BCS-FACS/EROS workshop on "Making Object Oriented Methods More Rigorous", Imperial College, London-England, 1997
9. Lehtonen, S., Pärssinen, J., A Pattern Language for Cryptographic Key Management, Proceedings of the 7th European Conference on Pattern Languages of Programs (EuroPLoP), Irsee, Germany, June 2002.
10. Liu, L., Yu, E., Mylopoulos, J., Analyzing Security Requirements as Relationships Among Strategic Actors, 2nd Symposium on Requirements Engineering for Information Security (SREIS'02). Raleigh, North Carolina, 2002.
11. Mouratidis, H., Giorgini, P., Manson, G., Philp, I., A Natural Extension of Tropos Methodology for Modelling Security, in the Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle-USA, November 2002.
12. Mouratidis, H., A Secuirty 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, 2004.
13. Mouratidis, H., Giorgini, P., Manson G., Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems, in the Proceedings of the 15th Conference on Advance Information Systems (CAiSE 2003), Velden-Austria, June 2003.
14. Mouratidis, H., Giorgini, P., Manson, G., Modelling secure multiagent systems, in the proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, pages 859-866, ISBN 1-58113-683-8, ACM 2003.
15. Mouratidis, H., Giorgini, P., Schumacher, M., Manson, M., Security Patterns for Agent Systems, Proceedings of the Eight European Conference on Pattern Languages of Programs (EuroPLoP), Irsee, Germany, June 2003.
16. Mouratidis, H., Philp, I., Manson, G., A Novel Agent-Based System to Support the Single Assessment Process for Older People, in the Journal of Health Informatics (9) 3, pp. 149-163, September 2003.
17. National Research Council, Computer At Risk: Safe Computing in the Information Age, National Academy Press, Washington, D.C., USA, 1991
18. Norman L. Kert, J., Vlissides, M., Coplien, J-O, "Pattern Languages of Program Design 2", Addison Wesley Publishing, 1996

19. Schumacher, M., Roedig, R., Security Engineering with Patterns, in the proceedings of the 8th Conference on Pattern Languages for Programs (PLoP 2001), Illinois-USA, September 2001.
20. Yoder, J. Barcalow, J., Architectural Patterns for Enabling Application Security, Proceedings of the 4th Conference on Pattern Languages of Programs (PLoP 1997), Monticello, Illinois, USA, September 1997
21. Yu, E., Cysneiros, L., Designing for Privacy and Other Competing Requirements, 2nd Symposium on Requirements Engineering for Information Security (SREIS' 02), Raleigh, North Carolina, 2002.