International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

SECURE TROPOS: A SECURITY-ORIENTED EXTENSION OF THE TROPOS METHODOLOGY

HARALAMBOS MOURATIDIS

[1] *School of Computing and Technology, University of East London, England*

h.mouratidis@uel.ac.uk

PAOLO GIORGINI

[2] *Department of Information and Communication Technology, University of Trento, Italy*

paolo.giorgini@dit.unitn.it

Although security plays an important role in the development of multiagent systems, a careful analysis of software development processes shows that the definition of security requirements is, usually, considered after the design of the system. One of the reasons is 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 can help towards the development of more secure multiagent systems. In this paper we introduce extensions to the Tropos methodology to enable it to model security concerns throughout the whole development process. A description of the new concepts and modelling activities is given along with a discussion on how these concepts and modelling activities are integrated to the current stages of Tropos. A real life case study from the health and social care sector is used to illustrate the approach.

Keywords: Agent-Oriented Software Engineering; Security Engineering; Methodologies; Tropos

1. Introduction

Work within the agent research community has led towards the development of the Agent Oriented Software Engineering (AOSE) paradigm. AOSE introduces an alternative approach in analysing and designing complex distributed computerised systems [24, 43, 21], according to which a complex computerised system is viewed as a multiagent system [43], in which a collection of autonomous software agents (subsystems) interact with each other in order to satisfy their design objectives. Therefore, developers view 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 capability of communicating [33].

Due to these characteristics, agent oriented systems are gaining popularity in different areas of our everyday life, such as military, health care, education, finance and transportation. As a result, these systems include information related to many aspects of someone’s private life, such as bank accounts, educational qualifications, and health records.

Security has been identified as an important issue [22, 33] for the widespread use of agent technology. However, the common approach towards the inclusion of security within an agent oriented system is to identify security requirements after the definition of the system [12, 30, 35]. This typically means that security enforcement mechanisms have to be fitted into a pre-existing design. This approach leads to serious design challenges that usually translate into the emergence of computer systems afflicted with security vulnerabilities [1, 39].

Research efforts, so far, have mainly focused on the solution of individual security problems, such as attacks from an agent to another agent; attacks from a platform to an agent; and attacks from an agent to a platform [23]. In general, developers of agent oriented methodologies have neglected security and although the agent oriented software engineering is progressing rapidly and many agent oriented methodologies [13, 17, 20, 21, 42, 44] have developed during the last few years; agent oriented software engineering practices and methodologies do not meet the needs for resolving the security related problems, and fail to provide evidence of successfully integrating security concerns. As a result, developers find no help when considering security during the development of multiagent systems.

In this paper we extend the Tropos methodology to enable developers to consider security issues during the software development process. Our aim is to provide an easy to use development methodology that will allow developers (1) to integrate security related analysis in order to identify desirable security aspects (2) reason about these aspects and (3) develop a system that is composed of agents able to satisfy the desirable security aspects of the system. It is not our aim to provide a methodology to analyze specific security protocols and mechanisms. Although, this is an important area, it is outside the scope of our work.

The paper, which is an extended and revised version of [33] and integrates results from [34, 35], is structured as follows. Section 2 introduces the relation between security and agent oriented software engineering and it discusses related work. Section 3 provides an overview of the Tropos methodology, whereas in Section 4 we present the security-related extensions to the Tropos methodology. Section 5 describes how the proposed extensions are integrated within the development stages of the Tropos methodology and it illustrates the extensions with the aid of an example taken from the health and social care sector. Section 6 concludes this paper and presents directions for future work.

2. Security and Agent Oriented Software Engineering

Security of computer based information systems is concerned with methods providing cost effective and operationally effective protection of information systems from undesirable events [28]. In principle, security is usually defined in terms of the existence of confidentiality, authentication, integrity, access control, non repudiation and availability. Each of these properties is considered during the development of systems for different reasons. However, most of the times, the development of a system would require the consideration of more than one of these properties.

As Anderson claims [1], "*security engineering is about building systems to remain dependable in the face of malice, error or mischance*". The process of securing an information system is usually a trade off between security requirements and other non-functional[1] and functional requirements. Security requirements are usually considered non-functional requirements [10], and they are defined as "*a manifestation of a high-level organisational policy into the detailed requirements of a specific system*" [12].

Nevertheless, differently than other non-functional requirements, such as reliability and performance, for which software engineers have recognised the need to integrate into the software development processes [11]; security still remains an afterthought and the usual approach towards the inclusion of security within a system is to identify security requirements after the definition of a system or to consider security only in certain stages of the development process. However, this approach often leads to problems [1], since security mechanisms have to be fitted into a pre-existing design, therefore leading to serious design challenges that usually translate into software vulnerabilities [39]. Literature provides many examples of security disasters that took place while trying to upgrade non-secure systems to secure systems (see for instance [4]).

To eliminate the above development mismatches, which result from the lack of security consideration during the development process, we believe security should be considered during the whole development process and it should be defined together with the requirements specification. Taking security into account along with the functional requirements throughout the development stages helps to limit the cases of conflict, between security and functional requirements, by identifying them very early in the system development, and find ways to overcome them. This argument is also supported in the literature [12, 26, 41]. However, to consider security issues throughout the development process of a software system, software engineering methodologies must provide developers with models and processes to help them model security concerns.

The agent oriented software engineering paradigm presents a feasible approach for the integration of security into software engineering. This is due to the appropriateness of agent oriented philosophy, for dealing with 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 [25] 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 subsystems. In addition, the agent oriented view is perhaps the most natural way of characterising security issues in

[1] Non-functional requirements introduce quality characteristics, but they also represent constraints under which the system must operate [37, 38].

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.

However, current agent oriented methodologies do not meet the needs for resolving the security related problems [41], and fail to provide evidence of integrating successfully security concerns throughout the whole range of the development process. In other words, they fail to adequately provide a security-oriented approach in the development of agent oriented software systems. Nevertheless, recently, work has initiated towards the solution of this problem.

Liu et al. [29] have presented work to identify security requirements during the development of multiagent systems by analysing the relationships between strategic actors, such as users and stakeholders, and potential attackers. In this work, three different types of analysis techniques are proposed: agent oriented, goal oriented and scenario based analysis. In addition, Yu and Cysneiros [45] provide an approach to model and reason about non-functional requirements (with emphasis on privacy and security). They are using the concept of soft-goal to assess different design alternatives, and they determine how each of these alternatives contributes in achieving the soft-goal.

Both of these works are mainly focused on the requirements analysis area and not on the whole development process. In addition, both Liu and Yu employ the concept of soft-goal to help them in their analysis. Although soft-goals can support the security related analysis of the system during the requirements analysis stage, they do not provide enough detail when considering security in the later stages of the development process. Therefore, as it has been argued in the literature [31], the concept of soft-goal does not adequately model security issues throughout the development process.

Moreover, Huget [19] proposes a new agent oriented methodology, called Nemo, and he 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 a result it has not presented in the literature enough to allow a thorough examination. However, 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*" [19]. Therefore, more evidence will be required to satisfy the claim of the developer that the methodology tackles security.

In addition, Giorgini at al. have introduced in [15] an enhancement of Tropos / i* that is based on the clear separation of roles in a dependency relation between those offering a service (the merchant processing a credit card number), those requesting the service (the bank debiting the payment), and those owning the very same data (the cardholder). In [16] they have proposed a PKI/trust management requirements specification and analysis framework based on the clear separation of trust and delegation relationship. However, the analysis produced by these approaches results in a high level analysis of security properties and they lack a well defined process to transform such high level analysis to operational security properties of the agents of the system.

In addition to the above approaches, a large number of related works comes from close disciplinary areas such as requirements engineering [10, 11] object oriented software engineering [26, 27, 30, 49, 52, 55] and patterns [54]. These approaches provide a first step towards the integration of security and software engineering and have been found helpful in modelling security requirements. However, they only guide the way security can be handled within a certain stage of the software development process. For example, McDermott and Fox's approach [49] is used only during the requirements analysis, whereas Jurgen's analysis [27] take place in a fairly low level and it is suited to a more operational analysis. In other words, Jurgen's approach is only applicable during the design stage.

Differently than the presented related work, this paper proposes an approach that covers the whole development process using the same concepts and notations. As argued earlier, considering security issues throughout the development process by using the same concepts and notations is very important when developing software systems with security on mind.

3. Tropos

For our work, we have decided to extend the Tropos methodology, rather than creating a new methodology from scratch. This decision took place because we are interested in enabling an agent oriented software engineering methodology to model security, rather than creating one more methodology on the (already) large amount of existing ones [13,17,20,21,42,44]. Moreover, the decision [35] to extend Tropos amongst all the other available agent oriented software engineering methodologies was based on the fact that Tropos spans in all the development stages using the same concepts; it is easily extensible and it is more security-aware than other agent oriented software engineering methodologies [35]. In addition, the Tropos methodology is well integrated with other approaches, such as the UML, in which some security work has taken place [26, 27, 30], and therefore existing work can be considered and incorporated within the proposed approach. Moreover, the modelling concepts of Tropos are well suited to model security requirements, which are usually expressed using notions such as agents and high level goals such as confidentiality and authentication [15].

Tropos[2] is a novel agent oriented software engineering methodology tailored to describe both the organisational environment of a multiagent system and the system itself. Tropos is characterised by three key aspects [9, 36, 17, 8]. Firstly, it deals with all the phases (requirements analysis, system design and implementation) of a system development, adopting a uniform and homogeneous way that it 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, emphasising the need to understand not only *what* organisational goals are

[2] The name Tropos derives from the Greek "Τρόπος" which means "way of doing things" but also has the connotation of "easily changeable, easily adaptable".

required but also *how* and *why* the intended system would meet its organisational goals. 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 [6, 7]. Such transformations allow developers to perform precise inspections of the development process by detailing the higher level notions introduced in the previous stages of the development.

Tropos adopts the *i** modelling framework [46], 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 [46] represents an entity that has intentionality and strategic goals within the multiagent system or within its organisational setting. A role represents an abstract characterisation of the behaviour of a social actor within some specialised context or domain of endeavour [46]. A position represents a set of roles, typically played by one agent. A (hard) goal [46] represents a condition in the world that an actor would like to achieve. In other words, goals represent actors' strategic interests. In Tropos, the concept of 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 [46]. For instance, an example of soft-goal is "the system should be scalable". A task (also called plan in Tropos) represents, at an abstract level, a way of doing something [17]. 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 [17] 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 [46] 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 dependency describes the nature of an agreement (called dependum) between dependee and depender. A capability [17] represents the ability of an actor of defining, choosing and executing a task for the fulfilment of a goal, given certain world conditions and in presence of a specific event. Figure 1 depicts a graphical representation of the above-mentioned concepts as used in the Tropos methodology.

Tropos methodology covers five main software development stages: *Early* and *Late Requirements* analysis, *Architectural* design, *Detailed* design, and *Implementation*. Both early and late requirements analysis share the same methodological approach. As a result, most of the ideas and concepts used during the early requirements are also used during the late requirements. The Tropos process is presented in detail in [8].

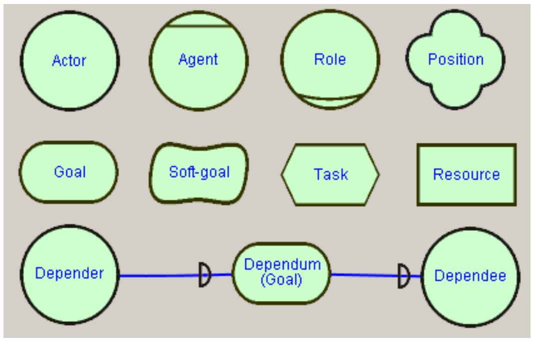

Fig. 1. The concepts of the Tropos methodology

4. Extending Tropos with security related concepts

The Tropos methodology partially [31] tackles security modelling by allowing developers to capture security requirements, as well as other non-functional requirements, as soft-goals. The usage of soft-goals to model general non-functional requirements although it allows developers to define, to some extent, together security and other functional and non-functional requirements, it does not help in providing a clear distinction between security and the other requirements of the system [35]. Such a distinction is made even harder by the lack of definition of the Tropos concepts, such as goals, tasks, and dependencies, with security in mind. Moreover, as discussed in previous work [31], although the current Tropos concepts allow clear identification of the dependencies between the actors; some possible (security) constraints that might be imposed to some of these actors are not captured. The lack of modelling such constraints results in an analysis, and eventually in a design, that lacks essential (security) information and it restricts the modelling of security properties during the system development. In addition, the methodology fails to integrate security modelling during the early requirements analysis stage, since the modelling of security requirements as soft goals is introduced during the analysis of the system-to-be (architectural design stage). However, all the actors play an important role with respect to the security of the system and all of them should be analysed with security in mind.

4.1. *THE SECURE CONCEPTS*

As discussed in the previous section, the current ontology of the Tropos methodology fails to adequately model security during the development process of a multiagent system. To enable developers to adequately capture security requirements we introduce the concept of constraint and we extend it with respect to security. In addition, the Tropos concepts of dependency, goal, task, resource, and capability are also extended with security in mind. This section aims to describe these newly introduced and extended concepts, which are defined within the Tropos project as secure concepts.

4.1.1. *Constraint and Security Constraint*

Constraints can represent a set of restrictions that do not permit specific actions to be taken or prevent certain objectives from being achieved and more often [40] are integrated in the specification of existing textual descriptions. Because of its importance in the system development, the concept of constraint has been introduced to the Tropos methodology as a separate concept [35], and the meta-model of the Tropos modelling language has been extended by introducing the construct for modelling constraints [35].

For the purposes of our work, we introduce the concept of security constraint. A security constraint is captured through a specialization of constraint and it 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*.

It is worth mentioning that in our work, security constraints do not represent specific security protocol restrictions, which should be specified during the implementation of the system, and not during the analysis and design, but they contribute to a higher level of abstraction. This higher level of abstraction allows for a generalised design free of models biased to particular implementation languages.

Graphically, security constraints are modelled as illustrated in Figure 2; as clouds within which the description of the (security) constraint is shown.

4.1.2. *Secure Dependency*

A secure dependency [35] 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). We define three different types of secure dependency. In a *dependee secure dependency* (see Figure 2-a), the depender depends on the dependee and the dependee introduces security constraint(s) for the dependency. In a *depender secure dependency* (see 2-b), the depender depends on the dependee and the depender introduces security constraint(s) for the dependency. In a *double secure dependency*, the depender depends on the dependee and both the depender and the dependee introduce security constraints for the dependency. Both must satisfy the security constraints introduced to achieve the secure dependency (see 2-c).

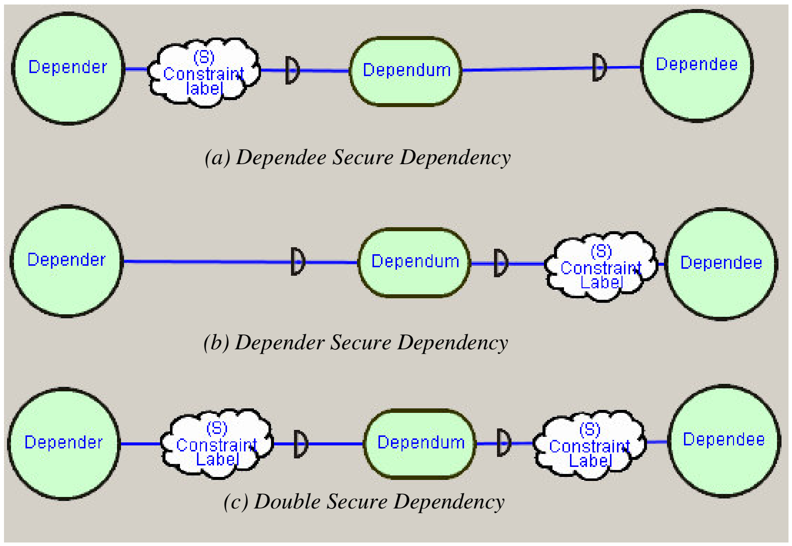

Fig. 2. Secure dependencies

4.1.3. *Secure Entities*

In our work, we use the term secure entity to represent a secure goal, a secure task or a secure resource. 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. A secure capability represents the ability of an actor/agent to achieve a secure goal, carry out a secure task and/or deliver a secure resource. The graphical representation of the Tropos entities has been extended to enable modelling of secure entities. Secure entities are indicated by the presence of an S within brackets before the description of the entity as shown in Figure 3.

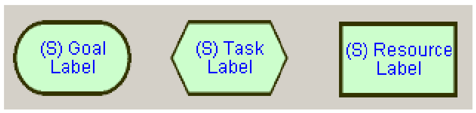

Fig. 3. Secure Entities

4.2. *SECURITY MODELLING ACTIVITIES*

The above-presented secure concepts form the basis of modelling security within the Tropos methodology. However, to make use of the above concepts different modelling activities contribute to the capturing and the analysis of the security requirements of a multiagent system. There are four main modelling activities: the security reference modelling, the security constraints modelling, the secure entities modelling and the secure capability modelling. The following sections briefly describe each one of those[3].

4.2.1. *Security Reference Modelling*

The security reference modelling involves the identification of security needs of the system-to-be, problems related to the security of the system, such as threats and vulnerabilities, and also possible solutions (usually these solutions are identified in terms of a security policy that the organisation might have) to the security problems. During the security reference modelling activity, the security reference diagram is constructed, after analysing the security requirements of the system-to-be and its environment. The main purpose of the security reference modelling is to allow flexibility during the development stages of a multiagent system and also to save time and effort. Many system developers face security issues similar to the issues faced by other developers. 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.

Although the security reference diagram is constructed during the initial stages of the system development, it is not isolated from the rest of the development, since the security reference modelling analysis 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 to identify possible means (security mechanisms) that contribute towards the satisfaction of the security constraints that are introduced to the system.

During the security reference modelling activity, developers consider the security features of the system-to-be, the protection objectives of the system, the security mechanisms, and also the threats to the system's security features. Security features represent security-related attributes that the system under development must demonstrate. Examples of security features are privacy, availability, and integrity. Protection objectives represent a set of principles or rules that contribute towards the achievement of the security features. These principles identify possible solutions to the security problems and usually they can be found in the form of the security policy of the organisation. Examples of protection objectives are authorisation, cryptography and accountability. Security mechanisms represent standard security methods for helping towards the satisfaction of the protection objectives. Some of these methods are able to prevent security attacks, whereas others are able only to detect security breaches. It must be noticed that furthered analysis of some security mechanisms is required to allow developers to identify possible security sub-mechanisms. A security sub-mechanism represents a specific way of achieving a security mechanism. For instance, authentication denotes a security mechanism for the fulfilment of a protection objective such as

[3] For a more detailed description please refer to [35]

authorisation. However, authentication can be achieved by sub-mechanisms such as passwords, digital signatures and biometrics. Threats represent circumstances that have the potential to cause loss; or problems that can put in danger the security features of the system. Examples of threats are social engineering, password sniffing and eavesdropping attacks.

A graphical representation of the above-mentioned concepts of the security reference diagram is depicted in Figure 4.

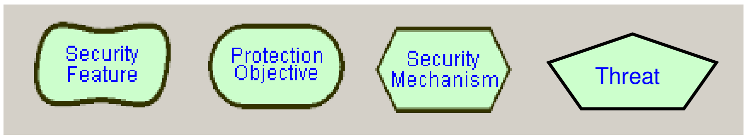

Fig. 4. Security reference diagram concepts

Two types of links are employed to connect the above concepts. A *positive contribution* link is used when one node helps in the fulfilment of another, whereas a *negative contribution* link is used when a node contributes towards the denial of another. Graphically, a positive contribution link is modelled as an arrow, which points towards the node that is satisfied, with a plus (+) whereas a negative contribution link is represented as an arrow with a minus (-) as shown in Figure 5.

Using the above concepts and notations, a developer can construct the security reference diagram, according to the security requirements of the system under development. However, this construction process can be affected by the security related experience of the developer. For more experienced developers, the construction is easier to perform and check, whether less security experienced developers might find it more difficult. To overcome this kind of mismatch, we have developed a transformation system[4] to allow developers to precisely inspect, by checking whether or not the diagram follows the construction rules, the development of the security reference diagram.

4.2.2. *Security constraint modelling*

The security constraint modelling involves the modelling of the security constraints imposed to the actors and the system, and it allows developers to perform an analysis by introducing relationships between the security constraints or a security constraint and its context. Security constraint modelling is divided into a number of smaller modelling activities such as *security constraint delegation*, *security constraint assignment*, and *security constraint analysis*. *Security constraint delegation* allows the delegation of a security constraint from one actor to another. Security constraints can be imposed to actors through a secure dependency. However, it might be the case that one actor delegates the responsibility for the satisfaction of a dependency to another actor, and as a result the security constraint imposed by the secure dependency is also delegated.

[4] Readers interested in the definition of the transformation system please check Appendix.

In cases where a security constraint is not delegated to another actor, the goals of the actor the imposed security constraint satisfies must be identified. The assignment of a security constraint to a goal is indicated with a contribution link that carries the "restricts" tag. This is known as *security constraint assignment*. When a security constraint is imposed to a goal (or task), two analysis processes are employed. *Security constraint decomposition*, which aims to further decompose the security constraint, and *secure goal introduction*, which identifies possible secure goals that the constraint might introduce to the system.

A security constraint can be decomposed into one or more security sub-constraints, which define more precisely a security constraint. The decomposed constraint is called the "root" constraint, and its satisfaction is implied if and only if all the security sub-constraints are satisfied. Furthermore, security constraints can introduce goals to an actor. This is known as secure goal introduction. The purpose of these goals is to help towards the achievement of the security constraint. In other words, during the process of secure goal introduction, the developer refines the goals of an actor to allow the satisfaction of a security constraint.

These activities should be combined with Tropos's original modelling activities when developing a system. It depends on the designer to decide which activity must be employed at which stage of the system development, since the main aim of these activities is not to restrict the designer to a step-by-step development of the system-to-be, but rather to provide a framework that allows the developer to go from a high level design to a more precise and defined version of the system.

4.2.3. *Secure entities modelling*

The Secure entities modelling involves the analysis of secure goals, tasks and resources identified in a multiagent system, and it is considered complementary to the security constraints modelling. Moreover, it follows the same reasoning techniques, such as means-end, contribution and decomposition analysis, that Tropos employs for goal and task analysis [7]. In particular, during the security entities modelling, means-end analysis aims at identifying secure tasks and resources that provide means for achieving a secure goal; contribution analysis permits developers to identify secure goals that contribute positively or negatively to the secure goal being analysed; and decomposition provides a decomposition of a secure goal and/or task into sub-goals and sub-tasks respectively.

4.2.4. *Secure capability modelling*

The modelling of secure capabilities involves the identification of the secure capabilities of the multiagent system's actors and agents to guarantee the satisfaction of the security constraints. Secure capabilities can be identified by considering dependencies that involve secure entities in the extended actor diagram. When identified, the secure capabilities are furthered specified in terms of plans of particular agents of the system.

4.3. *The security process*

There are three main aims when considering security issues throughout the development stages of a multiagent system. Firstly to identify the security requirements of the system; secondly to develop a design that meets the specified security requirements; and thirdly to validate the developed system with respect to security. Having the above in mind, the security-oriented process in Secure Tropos is one of identifying the security requirements of the multiagent system, transform these requirements to a design that satisfies them, and validate the developed system with respect to security.

The ***first step*** (takes place during the early and late requirements) in the proposed security oriented process aims to identify the security requirements of the system. Security requirements are identified by employing the modelling activities described in the previous section, such as security reference, 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, which guarantee the satisfaction of the identified security constraints, are imposed to the actors of the system.

The ***second step*** in the process (during architectural and detailed design) consists of identifying a design that satisfies the security requirements of the system, as well as its functional requirements. To achieve this, agents are identified with the aid of the Tropos modelling techniques [8] and secure capabilities that guarantee the satisfaction of the security entities identified during the previous step are given to the agents. It is worth mentioning that in this stage, different architectural styles might be used to satisfy the functional requirements of the system. However, there should be an evaluation of how each of these architectural styles satisfies the security requirements of the system. Although, this in general is left to the developers, a process [35] that is based on the measure of satisfiability [51] can be employed to determine whether for example a mobile agent or a client server architecture is more likely to satisfy the security requirements of the system under development.

The ***third step*** of the process is the validation of the developed solution. 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 [35]. It is worth mentioning that the validation rules are divided into two different categories, the inter-model rules and the outer model rules, the first allows the validation of each model individually, whereas the second allows the validation of the consistency between the different developed models. The inner model rules allow developers to validate the relationships between the components of the different security related models, such as the relationship between the security features and the threats in the security reference diagram; to validate the consistency between same components appeared in more than one models, such as a security constraint that appears in the actors' model as well as in the goal model; and validate the consistency when delegation of components between actors takes place.

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.

5. Integrating the proposed extensions to Tropos: the eSAP case study

The previous section described the proposed security extensions to the Tropos methodology. To provide a better understanding of the approach, in this section we describe5, with the aid of a real life case study, how the proposed extensions can be integrated within the development phases of the Tropos methodology and how they can be practically applied. The case study is based on the development of the electronic Single Assessment Process (eSAP) system [35], an agent-based health and social care system for the effective care of older people. To make this example simpler and more understandable, we consider a substantial part of the eSAP system, since our aim is not to describe in detail the analysis and design of the eSAP but rather to make the application of the proposed security related extensions easier to understand.

5.1. *Early Requirements*

The early requirements analysis is the first stage of the Tropos methodology and its output is an organisational model, which includes relevant actors, and their respective dependencies.

Regarding the proposed extensions, during the early requirements analysis stage the security reference diagram is constructed and security constraints are imposed to the stakeholders of the system (by other stakeholders). In addition, the imposed security constraints are expressed (initially) in high-level statements, and then they are further analysed [35] and security entities are introduced to satisfy them.

In our case study, we consider five actors. The Professional actor, who represents a health and social care professional; the Older Person, who represents a patient over 65; the DoH actor, which represents the English Department of Health, the R&D Agency actor, which represents a research and development agency interested in obtaining medical information for research purposes; and the Benefits Agency, which represents an agency that financially helps the older person.

The first step in the early requirements analysis is the construction of the security reference diagram. The main security features of the security reference diagram for the

[5] It is worth mentioning that in the presented case study we have focused our analysis in the security issues, and we are not presenting in detail the techniques of the Tropos methodology. Readers interested in this should refer to [8].

eSAP system are privacy, integrity and availability [35], and a part of it[6] is shown in figure 5.

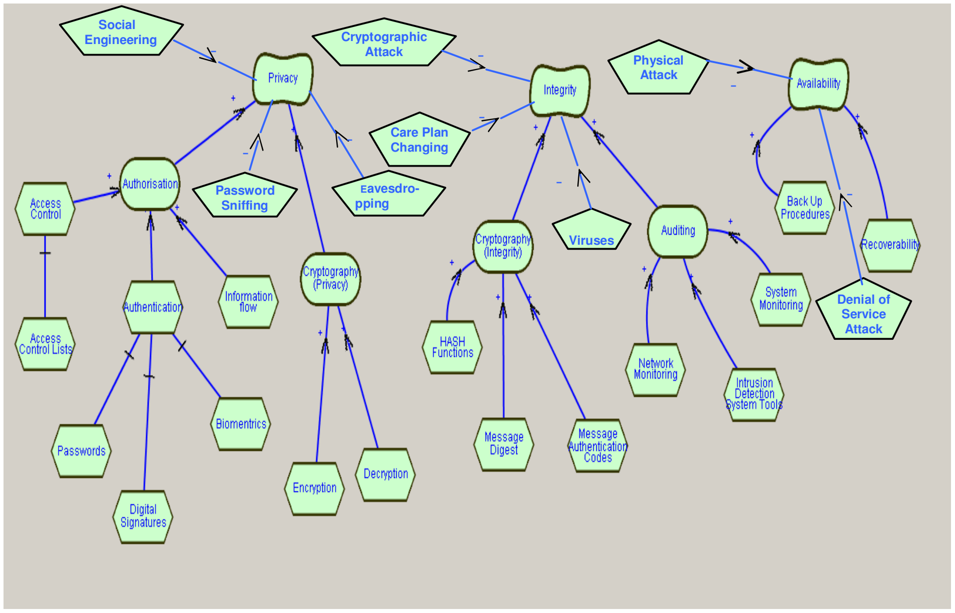

Fig. 5. Part of the eSAP security reference diagram

The next step of the process involves the modelling of the stakeholders of the system together with their goals, dependencies and security constraints. For this purpose, Tropos employs the actor diagram [8]. In such a diagram each node represents an actor, and the links between the different actors indicate that one depends on the other to accomplish some goals. In addition, the imposed security constraints (by other stakeholders) indicate that the actors must satisfy them for the dependencies to be valid. A part of the eSAP case study actor diagram is shown in Figure 6. In the eSAP case study, the Professional depends on the Older Person to Obtain (Older Person) OP Information; however one of the most important and delicate matters for the Older Person (as with any patient) is the privacy of their personal medical information and the sharing of it. Thus, most of the times, the Professional is imposed a constraint to share this information only if the older person's consent has been obtained.

Similarly, the Older Person depends on the Benefits Agency to Receive Financial Support. However, the Older Person worries about the privacy of their finances and as a result they impose a constraint to the Benefits Agency actor, to keep their financial information private. Moreover, one of the main goals of the R&D Agency

[6] The illustrated security reference diagram has been constructed after analysing all the different issues regarding the security of the eSAP as described in [35].

is to Obtain Clinical Information in order to perform tests and research. To get this information the R&D Agency depends on the Professional. However, the Professional is imposed a constraint (by the Department of Health) to Keep Patient Anonymity.

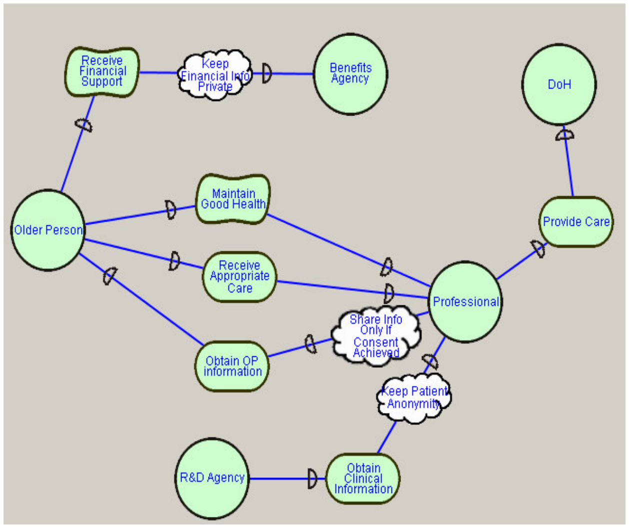

Fig. 6. Partial actor diagram for the eSAP case study

When the stakeholders, their goals, the dependencies between them, and the security constraints have been identified, the next step of this phase is to analyse more in depth each actor's goals and the security constraints imposed to them. In addition, secure entities are introduced to help towards the satisfaction of the imposed security constraints.

The security analysis starts by assigning the security constraints of the actor, to the goals of the actor they (the security constraints) restrict. As indicated in the previous sections, this assignment is indicated using a constraint link (a link that has the "restricts" tag).

Consider for instance the Professional actor (see Figure 7). According to the analysis that took place in the actor diagram (see Figure 6), the Professional actor has been imposed two *security constraints* (Share Info Only If Consent Achieved and Keep Patient Anonymity). By analysing the Professional actor's goals and tasks, we have identified the Share Medical Info goal [32]. However, this goal is restricted by the Share Info Only If Consent Achieved constraint imposed to the Professional by the Older Person. For the Professional to satisfy the constraint, a secure goal is introduced Obtain Older Person Consent. However this goal can be achieved with many different ways, for example a Professional can obtain the consent personally or can ask a nurse to obtain the consent on their behalf. Therefore, a sub-constraint is introduced, Only Obtain Consent Personally. To achieve this sub-constraint the secure goal Personally Obtain

Consent is introduced to the actor, which is divided into two sub-tasks: Obtain Consent by Mail or Obtain Consent by Phone.

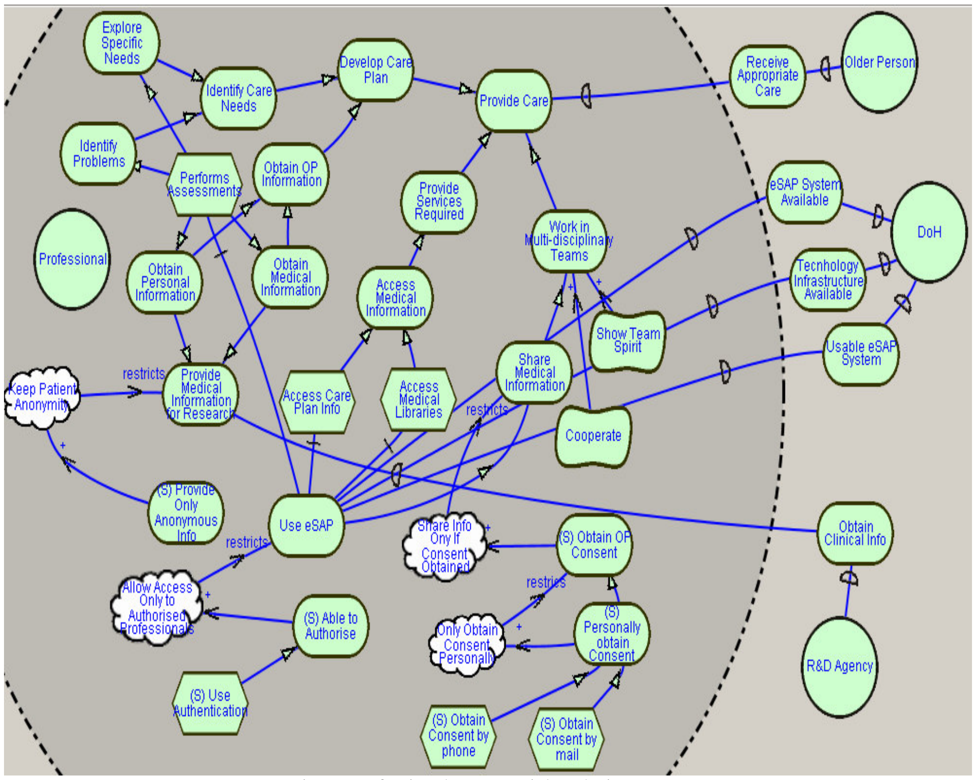
Explore Specific Needs
Identify Care Needs
Develop Care Plan
Provide Care
Receive Appropriate Care
Older Person
Identify Problems
Performs Assessments
Obtain OP Information
Provide Services Required
Work in Multi-disciplinary Teams
eSAP System Available
DoH
Professional
Obtain Personal Information
Obtain Medical Information
Access Medical Information
Tecnhology Infrastructure Available
Show Team Spirit
Usable eSAP System
Keep Patient Anonymity
restricts
Provide Medical Information for Research
Access Care Plan Info
Access Medical Libraries
Share Medical Information
Cooperate
restricts
(S) Provide Only Anonymous Info
Use eSAP
restricts
Share Info Only If Consent Obtained
(S) Obtain OP Consent
Obtain Clinical Info
Allow Access Only to Authorised Professionals
(S) Able to Authorise
restricts
Only Obtain Consent Personally
(S) Personally obtain Consent
R&D Agency
(S) Use Authentication
(S) Obtain Consent by phone
(S) Obtain Consent by mail

Fig. 7. Professional actor partial analysis

Moreover, one of the goals of the Professional actor is to Provide Medical Information for Research. However, this goal is restricted by the Keep Patient Anonymity constraint, which has been imposed to the Professional (see Figure 7). As a result, to satisfy this security constraint the secure goal Provide Only anonymous Info is introduced to the Professional.

5.2. *Late Requirements*

When all the actors have been analysed, the next phase involves the analysis of the system under development within its operational environment, and along with relevant functions, and qualities. The system is introduced as one more actors, to which existing actors delegate responsibilities for some of the goals and the dependencies that they cannot satisfy. The delegated dependencies define all the functional and non-functional requirements of the system.

Regarding the proposed extensions, security constraints are imposed to the system-to-be (by taking into account the security diagram) and these constraints are further analysed according to the security constraint analysis processes [35], and security goals and entities necessary for the system to guarantee the security constraints are identified.

The main goal of the eSAP system (see Figure 8) is to Automate Care, and therefore help professionals provide faster and more efficient care, and also allow older people get more involved in their care.

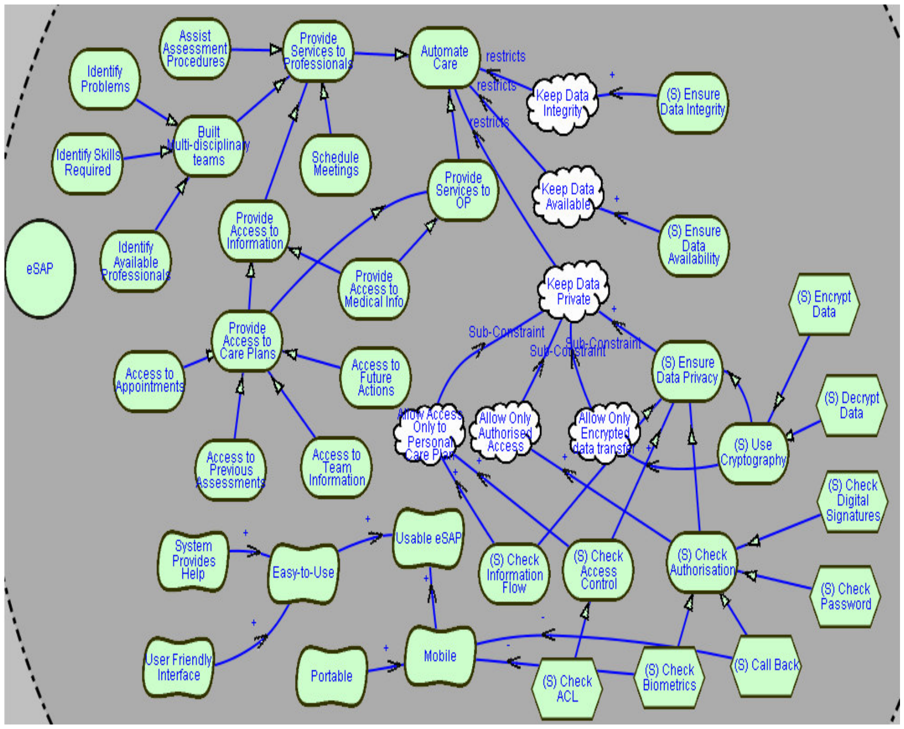

Fig. 8. eSAP System Partial Analysis

Taking into consideration the security reference diagram (see Figure 5) there are three main constraints imposed (by the desired security features of the system- privacy, integrity and availability) to the eSAP's main goal - Keep Data Integrity, Keep Data Available *and* Keep Data Private. For the eSAP to satisfy these constraints three secure goals have been identified. Ensure Data Integrity, Ensure Data Availability *and* Ensure Data Privacy.

This example focuses only on the Keep Data Private security constraint. This security constraint can be further analysed to security sub-constraints: Allow Only Encrypted Data Transfer, Allow Only Authorised Access, and Allow Access Only to Personal Care Plan. Taking into account the security reference diagram, secure goals are introduced to help towards the satisfaction of the imposed security constraints.

Thus the *secure goals* Use Cryptography, Check Authorisation, Check Access Control, and Check Information Flow are introduced. In addition, some of the secure goals are further analysed in terms of secure tasks.

For instance, the Use Cryptography goal is divided to *two secure tasks:* Encrypt Data and Decrypt Data. Although these tasks could be furthered decomposed by indicating for example the type of the encryption algorithm this is not the case in this stage, since the type of the encryption algorithm depends on the implementation of the system and it will restrict the designers of the system in a particular implementation style.

Moreover, the Check Authorisation goal is decomposed into four secure tasks: Check Password, Check Digital Signatures, Check Biometrics and Call Back. However, it is indicated in the diagram that the last two tasks contribute negatively towards the mobility of the system, and this is one factor that the developers must take into consideration in the implementation of the system.

5.3. *Architectural Design*

The architectural design involves (1) the addition of new actors, in which new actors are added to make the system interact with the external actors; (2) actor decomposition, in which each actor is described in detail with respect to their goals and tasks; (3) capabilities identification, in which capabilities needed by the actors to fulfil their goals are identified; and (4) agent assignment, in which a set of agent types is defined and each agent is assigned one or more capabilities.

From the security point of view, we identify the security constraints and secure entities that the new actors introduce and also during the actor decomposition we identify security sub-constraints and sub-entities. In addition secure capabilities are identified and assigned to each agent of the system. Moreover, during the architectural design the developers should decide for the architecture of their system. As mentioned earlier, such a decision should take into account the security requirements of the system. In other words, an architectural style that satisfies as much as possible the system's security requirements should be chosen. Although, there are different ways to determine that and in general it is left to the developers, a process that involves the evaluation of different architectural styles with respect to the security requirements of the system has been proposed [50]. The analysis that took place following this process, has identified [35] that a client server architectural style would satisfy more the security requirements of the eSAP system, than for example a mobile agent architectural style. As a result, the system has been designed with this consideration in mind.

It was derived from the late requirements stage that one of the system's secure goals is to Ensure Data Privacy. Responsibility for the achievement of this goal is delegated from the eSAP to the newly introduced sub-actor Privacy Manager.

The Privacy Manager has four main secure goals (see Figure 9), Check Authorisation, Check Access Control, Check Information Flow and Use Cryptography. Therefore, the Privacy Manager is decomposed and responsibilities for the satisfaction of these secure goals are delegated to the Authorisation Manager, Access Control Manager, Information Flow Manager and Cryptography Manager respectively.

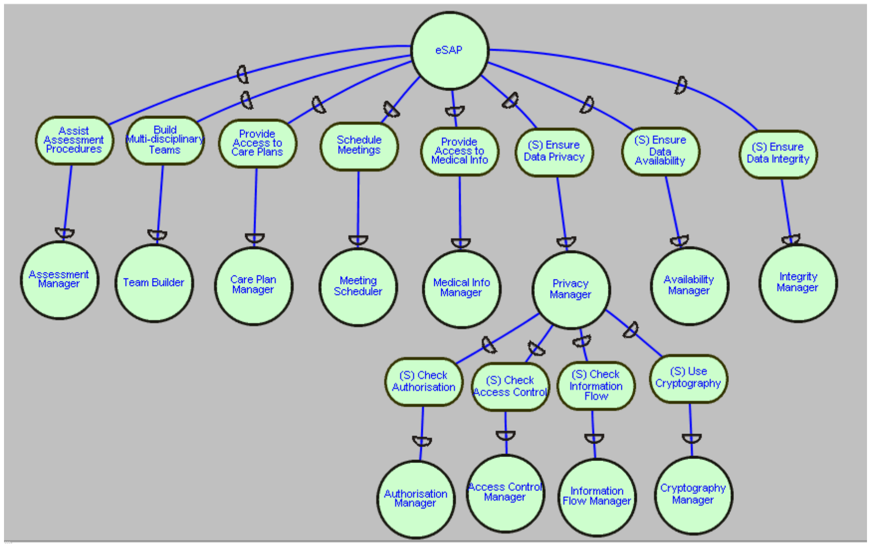

Fig. 9. Actors' decomposition diagram

For each new (sub) actor introduced in the system, an extended diagram is required to capture the dependencies of the new actor with the already existing actors of the system. Figure 10 shows a part (focused on the privacy) of the extended diagram for the task Access Care Plan Info of the Professional actor (see Figure 7). The Care Plan Manager is responsible for providing the Professional access to the Care Plan Info. It depends on the Authorisation Manager to deal with authorisation procedures, on the Access Control Manager and the Information Flow Manager to perform access control checks and information flow checks respectively, and on the Cryptography Manager for encrypting and decrypting information.

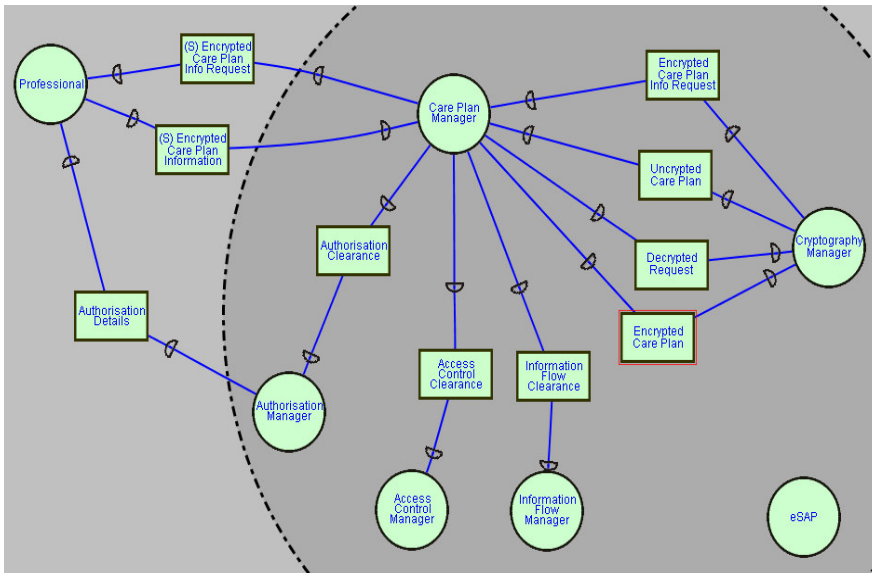

Fig. 10. Extended diagram with respect to “Access Care Plan Info” task

The next step in the architectural design is to identify (secure) capabilities for each actor. Taking into consideration the extended actor diagram (Figure 10), each dependency relationship can give place to one or more capabilities triggered by external events. The actors along with their capabilities with respect to the extended diagram of Figure 10 are shown in Table 1.

When the actors along with their capabilities have been identified the next step is the agents’ assignment. A set of agent types are defined and each one of them is assigned one or more different capabilities (Table 2) with respect to the capabilities identified in the previous step (Table 1).

Table 1. Actors and their Capabilities

Actors	**Capability**	**Cap. ID**
Professional	Provide Care Plan Info Request	1
	Provide Authorisation Details	2
	Obtain Care Plan Info	3
Care Plan Manager	Obtain Care Plan Info Request	4
	Provide Care Plan Info	5
	Request Encryption of Data	6
	Obtain Encrypted Data	7
	Request Decryption of Data	8
	Obtain Plain Data	9

	Obtain Authorisation Clearance	10
	Obtain Access Control Clearance	11
	Obtain Information Flow Clearance	12
Cryptography Manager	Encrypt Data	13
	Decrypt Data	14
Information Flow Manager	Provide Information Flow Clearance	15
Access Control Manager	Provide Access Control Clearance	16
Authorisation Manager	Obtain Authorisation Details	17
	Provide Authorisation Clearance	18

5.4. *Detailed Design*

From the security point of view, during the detailed design the developers specify the agent capabilities and interactions taking into account the security aspects derived from the previous steps of the analysis. In doing so AUML [3] notation is employed. The only difference is the introduction of security rules. These are similar to the business rules that UML has for defining constraints on the diagrams (see for instance Figure 11). In this case, security constraints can be formally expressed (and verified) with the aid of the Object Constraint Language (OCL). However, for reasons of simplicity we present here the security constraints with the aid of notes as shown in Figure 11.

Table 2. Agents and their capabilities

Agent	**Capabilities**
Professional	1,2,3
Care Plan Agent	4,5,6,7,8,9,10,11,12
Privacy Agent	13,14,15,16,17,18

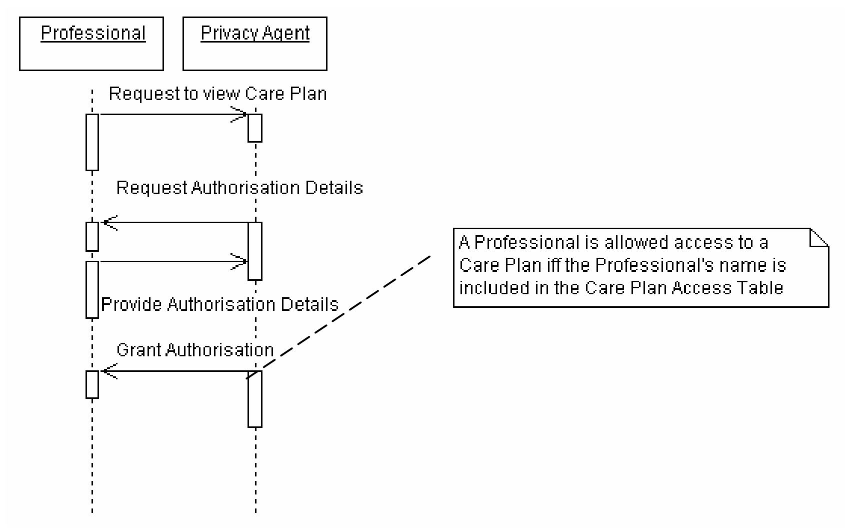
Professional
Privacy Agent
Request to view Care Plan
Request Authorisation Details
Provide Authorisation Details
Grant Authorisation
A Professional is allowed access to a Care Plan iff the Professional's name is included in the Care Plan Access Table

Fig. 11. An example of a security rule modelled in a sequence diagram

6. Conclusions and Future Work

In this paper we presented Secure Tropos, an extension to Tropos methodology, which allows developers to consider security issues throughout the development process of multiagent systems. Secure Tropos provides a well-guided process, consisting of various modelling activities, which allows developers to consider security issues during the development of multiagent systems. The key point on the security process of Secure Tropos is the usage of the same concepts and notations throughout the development stages. The usage of the same concepts together with the definition of the relationships between those concepts allows us to validate the solution in different ways. First of all, by tracking the developed solution all the way back to the security requirements and the security policy. This allows validation of the design by making sure that all the security policy rules are considered by the security solution. Although this does not guarantee a 100% secure system, such a claim cannot be made by any approach or any system, it guarantees that the developed design takes into account all the security requirements of the system, and appropriate capabilities are given to the agents of the system (and thus on the system) to satisfy these requirements. Secondly, the developers can validate each of the development steps by following a set of consistency and validation rules that have been proposed [35] for the security process.

The above validation processes are supported by automated techniques and tools to assist developers during the development. In particular, a security pattern language has been developed [35] to assist developers in transforming the analysis specification to a design by applying proven solutions in a systematic and structured way. Moreover, Security Attack Scenarios [35] have been proposed and integrated into the Secure Tropos process to validate the security solution against the system's security requirements. In addition, formal definition of the methodology's concepts is supported using the formal Tropos language and an automated tool, T-Tool [16], allows the automatic consistency validation of the specifications and it assists developers in identifying whether the proposed solution respects a number of desired security properties. T-Tool animations can also be used to give the user immediate feedback on its implications.

This is 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 development stages of multiagent systems. Currently, we are working on refining the process, to make it applicable even by developers with minimum knowledge of security.

References

1. R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, Wiley Computer Publishing, 2001.
2. M. Andries, G. Engels, A. Habel, B. Hoffmann, H-J Kreowski, S. Kuske, D. Plump, A. Schurr, G. Taentzer, Graph Transformation for Specification and Programming, Science of Computer Programming, 1999.
3. 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.), Lecture Notes in Computer Science, pp. 91-103, Springer, Berlin, 2001.
4. Paul Beynon-Davies, Information systems `failure': case of the LASCAD project, European Journal of Information Systems, 1995
5. M. Bradshaw, Software Agents, American Association Artificial Intelligence Publication, 1997.
6. P. Bresciani, P. Giorgini, The Tropos Analysis Process as Graph Transformation System, In Proceedings of the Workshop on Agent-oriented methodologies, at OOPSLA 2002, Seattle, WA, USA, Nov, 2002.
7. P. Bresciani, A. Perini, P. Giorgini, G. Giunchiglia, J. Mylopoulos, Modelling early requirements in Tropos: a transformation based approach, In Agent Oriented Software Engineering II. M. Wooldridge, and G. Weiβ (eds.). Lecture Notes in Computer Science, Springer-Verlag 2222, 2002.
8. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos A. Perini, TROPOS: An Agent Oriented Software Development Methodology. Journal of of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers Volume 8, Issue 3, Pages 203 - 236, May 2004.
9. J. Castro, M. Kolp, J. Mylopoulos, Towards Requirements-Driven Information Systems Engineering: The Tropos project, In Information Systems (27), pp 365-389, Elsevier, Amsterdam - The Netherlands, 2002.
10. L. Chung, B. Nixon, Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach, In Proceedings of the 17th International Conference on Software Engineering, Seattle- USA, 1995.

11. Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed Requirements Acquisition, Science of Computer Programming, Special issue on the 6th International workshop of Software Specification and Design, 1991.
12. P. Devanbu, and S. Stubblebine, Software Engineering for Security: a Roadmap, Proceedings of the conference of The future of Software engineering, 2000.
13. R. Evans, P. Kearney, J. Stark, G. Caire, F. J. Carijo, J. J. Gomez Sanz, J. Pavon, F. Leal, P. Chainho, and P. Massonet. MESSAGE: Methodology for Engineering Systems of Software Agents, AgentLink Publication, September 2001
14. K. Fischer, D. Hutter, M. Klusch, W. Stephan, Towards Secure Mobile Multiagent Based Electronic Marketplace Systems, Electronic Notes in Theoretical Computer Science, Vol. 63, Elsevier, 2002.
15. P. Giorgini, F. Massacci, J. Mylopoulos, Requirement Engineering meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard, In Proceedings of the 22nd International Conference on Conceptual Modeling (ER'03), Chicago, Illinois, 13-16 October, 2003.
16. P. Giorgini, F. Massacci,J. Mylopoulous, and N. Zannone. Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning. In Proceedings of the Second International Conference on Trust Management (iTrust 2004), Lecture Notes in Computer Science 2995, pages 176-190. Springer-Verlag Heidelberg, 2004.
17. F. Giunchiglia, J. Mylopoulos, A. Perini, The Tropos Software Development Methodology: Processes, Models and Diagrams, Lecture Notes in Computer Science 2585, pp 162-173, Springer 2003
18. D. Gollmann, Computer Security, John Willey and Sons, July 2001
19. M.-P. Huget, 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 (eds), Seattle, USA, November 2002.
20. C. A. Iglesias, M. Garijo, J. Gonzalez, J. R. Velasco, Analysis and design of multiagent systems using MAS-CommonKADS, Workshop on Agent Theories, Architectures and Languages, 1997.
21. C. Iglesias, M. Garijo, J. Gonzales, A survey of agent-oriented methodologies, Intelligent Agents IV, Lecture Notes in Computer Science, Springer-Verlag 1555, 1999.
22. W. Jansen, Countermeasures for Mobile Agent Security, Computer Communications, Special Issue on Advanced Security Techniques for Network Protection, Elsevier Science BV, November 2000.
23. W, Jansen, T. Karygiannis, Mobile Agent Security, National Institute of Standards and Technology, Special Publication 800-19, August 1999.
24. N. R. Jennings, An agent-based approach for building complex software systems, Communications of the ACM, Vol. 44, No 4, April 2001
25. N. R. Jennings, M. Wooldridge, 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
26. Jan Jürjens, Towards Secure Systems Development with UMLsec, Fundamental Approaches to Software Engineering (FASE/ETAPS) 2001, International Conference, Genoa 4-6 April 2001
27. J. Jürjens, UMLsec: Extending UML for Secure Systems Development, UML 2002, Lecture Notes in Computer Science 2460, pp 412-425, Springer 2002
28. V. P. Lane, Security of Computer Based Information Systems, Macmillan education ltd, 1985
29. L. Liu, E. Yu, J. Mylopoulos, Analyzing Security Requirements as Relationships Among Strategic Actors, in the Proceedings of the 2^{nd} Symposium on Requirements Engineering for Information Security (SREIS'02), Raleigh, North Carolina, October 2002.

30. 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.
31. 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.
32. H. Mouratidis, P. Giorgini, I. Philp, G. Manson, Using Tropos Methodology to Model and integrated Health Assessment System, In Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information systems (AOIS-02) at CAiSE2002, Toronto, Canada, 2002.
33. 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, pp. 859-866, ACM 2003.
34. H. Mouratidis, P. Giorgini, G. Manson, An Ontology for Modelling Security: The Tropos Approach, in the Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES 2003), Invited Session on Ontology and Multi-Agent Systems Design (OMASD'03), Oxford-England, September 2003.
35. 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, 2004.
36. A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Towards an Agent Oriented Approach to Software Engineering, Proceedings of the Workshop Dagli oggetti agli agenti: tendenze evolutive dei sistemi software, Modena - Italy, 4-5 Sept 2001.
37. G.C. Roman, A Taxonomy of Current Issues in Requirements Engineering, IEEE Computer, Vol. 18, No. 4, pp 14-23, April 1985.
38. I. Sommerville, Software Engineering – Sixth Edition, Addison-Wesley Publishing Company, 2001.
39. W. Stallings, Cryptography and Network Security: Principles and Practice, Second Edition, Prentice-Hall 1999.
40. E. Steegmans, J. Lewi, M. D'Haese, J. Dockx, D. Jehoul, B. Swennen, S. Van Baelen, P. Van Hirtum, EROOS Reference Manual Version 1.0, Department of Computer Science, K.U.Leuven, CW Report 208,176 p. Leuven, B, 1995.
41. T. Tryfonas, E. Kiountouzis, A. Poulymenakou, Embedding security practices in contemporary information systems development approaches, Information Management & Computer Security, Vol 9 Issue 4,pp 183-197, 2001.
42. Mark Wood, Scott A. DeLoach, An Overview of the Multiagent Systems Engineering Methodology, in Agent-Oriented Software Engineering, P. Ciancarini, M. Wooldridge, (Eds.), Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January 2001.
43. M. Wooldridge, P.Ciancarini, Agent-Oriented Software Engineering: The State of the Art, In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering. Springer-Verlag Lecture Notes in AI Volume 1957, January 2001
44. M. Wooldridge, N. R. Jennings, D. Kinny, A methodology for Agent-Oriented Analysis and Design, In O. Etzioni, J. P. Muller, and J. Bradshaw (eds.), Agents '99: Proceedings of the Third International Conference on Autonomous Agents, Seattle, WA, May 1998.
45. E. Yu, L. Cysneiros, Designing for Privacy and Other Competing Requirements, 2nd Symposium on Requirements Engineering for Information Security (SREIS' 02), Raleigh, North Carolina, 15-16 November, 2002.
46. E. Yu, Modelling Strategic Relationships for Process Reengineering, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.
47. B. W. Boehm, Software Engineering Economics, Prentice Hall, 1981.

48. C. Meadows, A Model of Computation for the NRL protocol analyser, Proceedings of the 1994 Computer Security Foundations Workshop, 1994.
49. J. McDermott, C. Fox, Using Abuse Care Models for Security Requirements Analysis, Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.
50. H. Mouratidis, P. Giorgini, G. Manson, 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.
51. 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.
52. S. Rohrig, Using Process Models to Analyze Health Care Security Requirements, International Conference Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet, L'Aquila, Italy, January, 2002.
53. B. Schneier, Secrets & Lies: Digital Security in a Networked World, John Wiley & Sons, 2000.
54. M. Schumacher, U. Roedig, Security Engineering with Patterns, in the Proceedings of the 8th Conference on Pattern Languages for Programs (PLoP 2001), Illinois-USA, September 2001.
55. G. Sindre, A. L. Opdahl, Eliciting Security Requirements by Misuse Cases, Proceedings of TOOLS Pacific 2000, November 2000.

Appendix

The proposed transformation system, defines the construction of the security reference diagram in terms of a graph transformation [2], in which a diagram (graph) is progressively derived through subsequent more and more precise versions of it, according to the application of a set of rules to the diagram. Such rules are called graph transformation rules and a precise definition can be found in [2]. It is worth mentioning that the security reference diagram transformation system is based on the graph transformation system introduced by Andries et al. [2], and the analysis proposed for Tropos' actor and goal diagrams by Bresciani and Giorgini [6]. Moreover, regarding the transformation rules, the notion of a graph transformation rule proposed by Bresciani and Giorgini [6] for Tropos diagrams is followed.

Formally, the security reference diagram can be represented as a special case of a labelled directed diagram $G = <\mathrm{N}, \mathrm{E}, s, t, l>$, where

- *N* is a finite set of nodes that can be connected by one or more edges of the finite set E;

s and t are two functions that assign the source and the target node to each node respectively $s, t : \mathrm{E} \to \mathrm{N}$;

l represents a label function for each of the nodes and edges. In addition, for the security reference diagram we can assume that $l : \mathrm{E} \cup \mathrm{N} \to <T, L>$, where *T* = {SecurityFeatures, SecurityThreats, Protection Objectives, SecurityMechanisms} and *L* represents a set of identifiers.

In addition, a graph transformation rule is defined as a pair $r = \langle L, R \rangle$, where L and R are graphs called the left-hand-side (*LHS*) and the right-hand-side (*RHS*) of the rule. From the analysis done by Bresciani and Giorgini [6] it derives that a graph H can be obtained from a graph G by the application of a set of transformation rules $P=\{r_1,...,r_n\}$ as $G \overset{P}{\Rightarrow} H \ \ or \ \ \overset{P}{\Rightarrow}$ in the case G is the empty graph.

However, such a derivation process is non-deterministic due to the choice of a particular rule, at each step. Additionally, the chosen rule might be applicable to several occurrences of the graph's LHS [2]. To control this kind of non-determinism during the construction of the security reference diagram, we have assigned a set of priority rules:

Rule 1: *Introduce the security features to the diagram*

$LHS :< \{\},\{\},\{\},\{\},\{\} >$

$RHS :< \{n_1\},\{\},\{\},\{\},\{n_1 \mapsto < SF,* >\} >$

The application of this rule results in the introduction of a new security feature (*SF*) in the *RHS* graph.

Rule 2: *Introduce the security threats and associate them with the security features*

$LHS :< \{n_1\},\{\},\{\},\{\},\{n_1 \mapsto < SF,* >\} >$

$RHS :< \{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_1 \mapsto < SF,* >, n_2 \mapsto < ST,* > e_1 \mapsto < NegCon, \varepsilon >\}$

The application of this rule results in the introduction of a security threat (ST) in the RHS graph and the introduction of new edge(s) associated with this node.

Rule 3: *Introduce the protection objectives and associate them with the security features*

$LHS :< \{n_1\},\{\},\{\},\{\},\{n_1 \mapsto < SF,* >\} >$

$RHS :< \{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_1 \mapsto < SF,* >, n_2 \mapsto < PO,* > e_1 \mapsto < PosCon, \varepsilon >\}$

The application of this rule results in the introduction of a protection objective (PO) in the RHS graph and the introduction of new edge(s) associated with this node.

Rule 4: *Introduce the security mechanisms and associate them with the protection objectives*

$LHS :< \{n_1\},\{\},\{\},\{\},\{n_1 \mapsto < PO,* >\} >$

$RHS :< \{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_1 \mapsto < PO,* >, n_2 \mapsto < SM,* > e_1 \mapsto < PosCon, \varepsilon >\}$

The application of this rule results in the introduction of a security mechanism (SM) in the RHS graph and the introduction of new edge(s) associated with this node.

Rule 5: *Decompose the security mechanisms to security sub-mechanisms*

$LHS :< \{n_1,n_2\},\{\},\{\},\{\},\{n_i \mapsto < SM,* >\} >$

$RHS :< \{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_i \mapsto < SM,* >, e_1 \mapsto < AND-DEC, \varepsilon >\}$

The application of this rule results in the introduction of new node(s) and edge(s) associated with the security mechanisms of the diagram.

Taking into account the above transformation system rules, we have developed an algorithm (shown in Figure 12) for the construction of the security reference diagram. The algorithm works as follows: First all the known security features are applied to the diagram. Then the threats are applied to the diagram and the contribution links between the threats and the security features are indicated. Then the protection objectives and their associations to the security features are applied. Finally, the security mechanism are introduced and associated to the protection objectives, and if necessary they are further analysed to security sub-mechanisms. This process assumes that the developers know all the elements of the diagram prior to its construction. However, most likely the developers will come across some new required security issues after the application of a specific rule. For instance, a security feature can be identified after the application of the security feature rule (rule 1). To avoid a delay in the analysis, it is convenient sometimes to allow some simple exceptions. Thus, it may be preferable to introduce the new node (by applying the corresponding rule) and then continue with the rest of the rules. For this reason, we have introduced to our algorithm an outer **REPEAT** loop to deal with situations where the application of one rule for a particular node, might require the application of a rule for another node.

```
BEGIN
Initialise Graph G (**should be empty in the initialisation process**)
  REPEAT
       REPEAT
               'choose rule 1';
               'choose an occurrence' i for the application of rule 1;
                G: = (G\I(L\R)+ I(R\L)
       UNTIL G = desired graph or no rule 1, for no occurrence i, remains;
       REPEAT
               'choose rule 2';
               'choose an occurrence' i for the application of rule 2;
              G: = (G \ I (L\R) + I (R\L)
       UNTIL G = desired Graph or no rule 2, for no occurrence i, remains;
       REPEAT
               'choose rule 3';
               'choose an occurrence' i for the application of rule 3;
              G: = (G \ I (L\R) + I (R\L)
       UNTIL G = desired Graph or no rule 3, for no occurrence i, remains;
       REPEAT
               'choose rule 4';
               'choose an occurrence' i for the application of rule 4;
              G: = (G \ I (L\R) + I (R\L)
       UNTIL G = desired Graph or no rule 4, for no occurrence i, remains;
       REPEAT
               'choose rule 5';
               'choose an occurrence' i for the application of rule 5;
              G: = (G \ I (L\R) + I (R\L)
       UNTIL G = desired Graph or no rule 5, for no occurrence i, remains;
  UNTIL all rules are satisfied for all occurrences;
END
```

Fig. 12. The algorithm for the construction of the security reference diagram