A Natural Extension of Tropos Methodology for Modelling Security

Haralambos Mouratidis[1], Paolo Giorgini[2], Gordon Manson[1], Ian Philp[3]
[1]*Computer Science Department, University of Sheffield, England*
{h.mouratidis, g.manson} @dcs.shef.ac.uk
[2] *Department of Information and Communication Technology, University of Trento, Italy*
paolo.giorgini@dit.unitn.it
[3] *Sheffield Institute for Studies on Ageing, University of Sheffield, England*
i.philp@sheffield.ac.uk

Abstract

Although security is an important issue when developing complex computerised systems, very little work has been done in integrating security concerns in the agent-oriented methodologies. This paper introduces extensions to the Tropos methodology to accommodate security. A description of new concepts is given along with an explanation of how these concepts are integrated to the current stages of Tropos. The above is illustrated using an agent-based health and social care information system as a case study.

1. Introduction

It is recognised amongst the agent research community [1,2,3] the need for developing a complete methodology for analysing and designing multi-agent systems. The main role of such a methodology will be to help in all the phases of the development of a system, and more importantly, to help capture and model the unique characteristics that agent-oriented systems introduce such as flexibility, autonomous problem solving, and the rich interactions between the individual agents.

Security is an important issue when developing complex computerised systems. According to [4] "Security concerns must inform every phase of software development, from requirements engineering to design, implementation, testing and deployment". We believe that if agent-oriented software engineering is to become widely accepted as a basis for developing complex computerised systems, agent-oriented software engineering methodologies must unify system engineering with security engineering.

Although many agent-oriented software engineering methodologies have been developed during the last few years (see [5] for an overview on the state of the art), very little work has been done in integrating security concerns during the analysis and design of an agent-based system. The common approach towards the inclusion of security within a system is to identify security requirements after the definition of a system. This approach has provoked the emergence of computer systems afflicted with security vulnerabilities [6]. From the viewpoint of the traditional security paradigm, it should be possible to eliminate such problems through more extensive use of formal methods and better software engineering.

This paper introduces extensions to *Tropos*, an agent oriented software engineering methodology [7], to accommodate security concerns during the requirements analysis. The concept of constraints is also defined and the reason for employing constraints and how they can help in the analysis of the system's security is described.

The paper is organized as follows. Section 2 introduces security in software engineering while Section 3 presents extensions to the *Tropos* methodology necessary for capturing security aspects of the system under development. In Section 4 we describe the security modeling features that we use in our approach, and in Section 5 we present how our approach can be integrated on the current *Tropos* stages. Finally, Section 6 presents some concluding remarks and directions for future work. The eSAP System (an agent-based health and social care information system) is used throughout the paper to illustrate the proposed extensions.

2. Security in Software Engineering

Security is usually defined in terms of the existence of properties such as confidentiality, authentication, integrity, access control, non-repudiation and availability [6] and the ability to overcome possible threats.

The security requirements of the system are obtained after studying the security policy[1] of the organisation. Currently, the definition of security requirements is usually considered after the design of the system. This typically means that security enforcement mechanisms

[1] A security policy is the set of decisions that, collectively, determines an organisation's posture towards security.

have to be fitted into a pre-existing design therefore leading to serious design challenges that usually translate into software vulnerabilities. Adopting a security focus through the overall system development process represents a solution to mitigate such problems.

Software engineering considers security requirements, as well as performance and reliability requirements, as non-functional requirements. Non-functional requirements introduce quality characteristics but also they represent the constraints under which the system must operate. Software designers have already recognised the importance of integrating non-functional requirements, such as performance and reliability, into software design processes [8], however security requirements are still an afterthought.

There are at least two reasons for the lack of support for security engineering [9]. Firstly, security requirements are generally difficult to analyse and model, and secondly because of developer lack of expertise for secure software development. Furthermore, security policies are generally specified in terms of security models that are not integrated with general software engineering models.

In the current state of the art, security properties are, within the requirements engineering process, supported by a qualitative reasoning rather than a formal reasoning. Existing formal methods support the verification of a protocol, which has already been specified [10], while qualitative directions provide a process-oriented approach to represent non-functional requirements as potentially conflicting or concordant goals and using them during the development of software systems [11]. These approaches only guideline the way non-functional requirements, such as security, can be handled within a certain stage of the software development process.

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

The agent-oriented paradigm represents a feasible approach for the integration of security to software engineering. As mentioned in [1] agents act on behalf of individuals or companies interacting according to an underlying organisational context. The integration of security within the context will require for the rest of the subsystems to consider the security requirements, specified in the security policy, when specifying their objectives and interactions therefore causing the propagation of security requirements to the rest of the subsystems. However, at present, no agent-oriented software methodology considers security requirements as an integral part of the whole software development process. In the following section we extend *Tropos* methodology in order to consider security concerns in the early and late requirements analysis phases.

3. Extending Tropos to Accommodate Security

Tropos is a methodology, for building agent-oriented software systems, tailored to describe both the organisational environment of a system and the system itself. *Tropos* adopts the *i** modelling framework [12], which uses the concepts of actors, who can be (social) agents (organisational, human or software), positions or roles, goals and social dependencies (such as soft goals, tasks, and resources) for defining the obligations of actors (dependees) to other actors (dependers). The type of the dependency describes the nature of an agreement (called dependum) between dependee and depender. Goal dependencies represent delegation of responsibility for fulfilling a goal; softgoal dependencies are similar to goal dependencies, but their fulfilment cannot be defined precisely; task dependencies are used in situations where the dependee is required to perform a given activity; and resource dependencies require the dependee to provide a resource to the depender. Graphically, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are represented as ovals, clouds, hexagonal, and rectangle, respectively; and the dependencies have the form depender → dependum → dependee.

Currently in *Tropos,* the process of integrating security and functional requirements throughout the whole range of the development stages is quite ad hoc. A systematic process that will guide the developer in considering security requirements (as well as other non-functional requirements) during the whole development phases is necessary. Such a process will provide guidance, and it will use the same concepts and notations throughout the development phases.

The last few years, work on agent oriented software engineering has been focused on providing a complete methodology that will help in all the phases of the development of an agent-oriented system, and more importantly it will help to capture and model the unique characteristics that agent-oriented systems introduce such as flexibility, autonomous problem-solving, and

interactions between individual agents. However, security has been mainly ignored and very little work has been taken place in order for agent oriented software engineering methodologies to support security concerns during the development stages.

On the other hand, although work has taken place on trying to capture non-functional requirements (including security) and consider different design alternatives [13, 14], none of these approaches were developed with the agent paradigm in mind.

Trying to close this gap, Eric Yu has recently initiated work that provides ways of modelling and reasoning about non-functional requirements (with emphasis on Security). Yu is using the concept of a soft goal to assess different design alternatives, and how each of these alternatives would contribute positively or negatively in achieving the soft goal.

The concept of a soft goal is "used to model quality attributes for which there are no a priori, clear criteria for satisfaction, but are judged by actors as being sufficiently met" [15].

However, non-functional requirements may relate to system's quality attributes, or alternatively may define constraints on the system [16,17]. Qualities are properties or characteristics of the system that its stakeholders care about, while constraints are restrictions, rules or conditions imposed to the system and unlike qualities are (theoretically) non negotiable. Thus, although the concept of a soft goal captures qualities, it fails to adequately capture constraints.

Possible constraints might be imposed to the system as restrictions for satisfying the system's goals (global or for each individual component). For example security constraints might be imposed on the system representing restrictions related to its security. Constraints might affect the analysis and design of the system, by restricting some alternative design solutions, conflict with some of the requirements of the system, and also by refining some of the goals of the system or introducing new ones that help the system towards the satisfaction of the constraint.

3.1 Constraints

Constraints are limitations (restrictions) that do not permit specific actions to be taken or prevent certain objectives from being achieved. Thus, constraints can represent a set of conditions; rules and restrictions imposed on a system, and the system must be operating in such a way that none of them will be violated.

More often [19] constraints are integrated in the specification of existing concepts and are expressed in terms of informal textual descriptions. However, this approach leads many times to misunderstanding and an unclear definition of a constraint, and its role in the development of the system. This often results in errors in the very early development stages that propagate to the later stages of development causing many and serious problems when discovered, if they are discovered. In addition, integrating constraints on existing concepts might introduce problems when a system is modified.

Thus, we believe constraints must be introduced as a separate concept, next to other existing concepts (in our case actors, goals, soft goal, task and resources) during the whole range of the development process.

Defining constraints as a separate concept does not mean isolate them from the rest of the system. Constraints are closely related with the parts of the system they restrict. The part of the system that a constraint restricts is called the context of the constraint [19]. In our case, the context can include a different number of goals, soft goals, and dependencies of the system.

In addition, a constraint can contribute either positively or negatively to functional and non-functional requirements. This basically depends on the type of constraints (for example performance, reliability or security constraints) and the purpose for which they have been imposed to the system (for example to restrict access to the system).

Constraints can be human-imposed or environment-imposed. The first category includes constraints imposed by stakeholders, users, or actors, while the second category involves constraints imposed by organizations, policies, laws, rules or regulations.

3.2 The eSAP case study

To illustrate the need to extend *Tropos*, let us consider a system similar to the eSAP system first presented by Mouratidis et al [18]. In this system we have five actors:

- *Older Person*: The Older Person that wishes to receive appropriate health and social care (patient)
- *Professional*: The health and/or social care professional
- *DoH*: The English Department of Health
- *Benefits Agency*: An agency that helps the older person financially
- *R&D Agency*: A research and development agency.

To model the goals and the dependencies between the stakeholders (actors) *Tropos* introduces actor diagrams. 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. The actors diagram for the above actors is shown in figure 1.

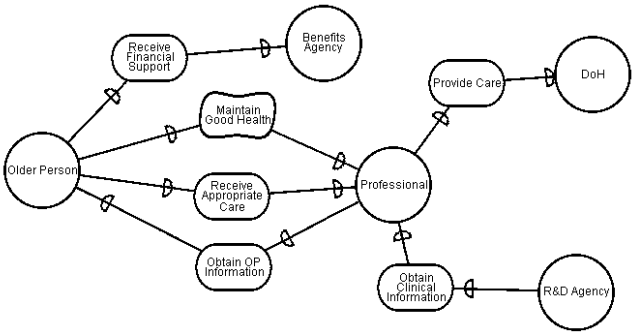

Figure 1. The stakeholders of the eSAP system

Although the dependencies between the actors are shown clearly, some possible constraints (in our case related to security) that might be imposed to some of the actors are not captured.

For example, the *Older Person* depends on the *Benefits Agency* to *Receive Financial Support* but *Older Person* might introduce a constraint to the *Benefits Agency* such as to keep their financial information private. *R&D Agency* depends on the *Professional* to *Obtain Clinical Information* but the *Professional* might be restricted (for example by the *DoH* or the *Older Person*) to provide only anonymous medical information. In addition, the *Older Person* might restrict the *Professional* by imposing a constraint to share medical information only if the *Older Person*'s consent is obtained.

Thus the *Professional* has to achieve their goals while having to satisfy different constraints imposed to them. Taking into considerations these constraints, it helps to refine existing goals and also introduce, in the later stages of the development, some extra (secure) goals to the *Professional* such as to *Obtain Older Person Consent*, to help towards the satisfaction of the constraints.

An alternative way would be to introduce goals (related to security) to the actors without first imposing any constraints. For example, in the above example a goal such as *Obtain Older Person Consent* could be introduced to the *Professional* actor without analysing any constraints that could be imposed to the system. This would be possible, but it would represent a totally ad hoc process, depending only on the experience and the capability of the designer. Using the concept of constraints helps to provide a systematic approach in refining existing goals and also identifying goals that are related to the security of the system. It also gives reasons why these goals have to be introduced to the system, and in which actors.

However there is confusion. The question *why not capturing the constraints that are imposed to a system as the system's goals?* has been asked to the authors. For example, the constraints described above could be captured as goals as shown below (the arrows indicate dependency according to the *Tropos* concepts [7]):

Older Person -> Keep Financial Info Private -> Benefit Agency

R&D Agency -> Only provide anonymous medical info -> Professional

Older Person -> Share this info only if consent is obtained -> Professional

The confusion mainly comes from the fact that constraints can be incorporated in the specification of existing concepts. However, the concept of a constraint is different from the concept of a goal. A goal represents a desired state of the world, while a constraint represents a condition, rule, or restriction towards the achievement of a goal. Although a goal can be achieved with various ways, a constraint defines a set of restrictions on how the goal will be achieved. Considering the difference between a goal and a constraint we see the statements made are not correct.

Keep Financial Info Private is not a goal of the *Older Person* (the goal is to *Receive Financial Support*) but rather a restriction imposed in achieving the goal.

R&D Agency -> Only to provide anonymous medical info -> Prof. According to this statement the *R&D Agency* has a goal to *Receive Only Anonymous Medical Information* from the *Professional*. However, this is not true because the *R&D Agency* would be happy to get named information but the *Professional* cannot provide named information because it has been restricted by the *Older Person* or by laws (Department of Health).

Older Person -> share info only if consent is obtained -> Professional. This is not a goal that the *Older Person* has and depends on the *Professional* but rather a constraint that restricts the *Professional* in achieving the goal *Provide Appropriate Care* that the *Older Person* depends on him.

4. Security Modelling Features

We introduce the concepts of *security diagram*, *security constraint*, *secure dependency*, and *secure goal*, *task*, and *resource*.

4.1. Security Diagram

A *security diagram* is constructed after analysing the security requirements of the system-to-be and its environment and it is similar to the security catalogue first introduced by Yu [15]. The process of analysing the security requirements of the system-to-be and its environment is not unique and it depends on the engineer. This process usually involves identification of the security needs of the system; problems related to the security of the system (such as possible threats and vulnerabilities) and possible solutions to the security problems (these

solutions can usually be identified in terms of a security policy that the organisation might have).

The usability of the *security diagram* is twofold. Firstly it helps a designer to identify possible constraints that must be introduced to the system-to-be (by taking into account the security needs of the system) and secondly to identify in the later stages of the design possible means (security mechanisms) that contribute to the satisfaction of the *security constraints* that are introduced to the system. In addition to that, the *security diagram* displays some general advantages:

- It provides a framework of security needs, threats and possible solutions using concepts known to the software engineer. In this work we are interested in extending *Tropos* methodology, so we have adopted the diagram using *Tropos'* concepts such as goals, tasks, and soft goals. That means the software engineer can use the same concepts throughout the whole *Tropos* development process.
- Many systems under development are similar to systems already in existence. Thus the *security diagram* can be used as a reference point that can be modified or extended according to specific needs of particular systems, saving developers time and effort.

For the construction process of the *security diagram* the engineer takes into consideration 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. The *security diagram* represents the connection between security features, threats, protection objectives, and security mechanisms that help towards the satisfaction of the objectives. Thus, each security feature identified receives positive contributions from different protection objectives and negative contributions from the threats. Positive contributions help towards the satisfaction of the security feature while negative contributions put in danger the security feature. In addition, the diagram captures possible security mechanisms that contribute positively or negatively to the protection objectives.

Security features (also protection properties) represent features associated to security that the system-to-be must have. We are using the concept of a soft goal to capture security features on the security diagram. This decision have taken place since the concept of a soft goal is "used to model quality attributes for which there are no a priori, clear criteria for satisfaction, but are judged by actors as being sufficiently met" [15]. In the same sense, security features are not subject to any clear criteria for satisfaction. Examples of security features are privacy, safety, accountability, availability, and integrity.

Threats on the other hand represent circumstances that have the potential to cause loss or problems that can put in danger the security features of the system. 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. We are representing protection objectives using the concept of a goal. This has been decided because a goal defines desired states of the world. In the same sense, a protection objective represents desired security states that the system must have.

Security mechanisms identify possible protection mechanisms of achieving protection objectives. In order to represent security mechanisms we are employing the concept of a task. A task represents a way of doing something, such as the satisfaction of a goal. However, it must be noticed that tasks (security mechanisms) can contribute positively (+) but also negatively (-) to different protection goals. The following figure shows the above-mentioned concepts and how they are graphically represented in the *security diagram*.

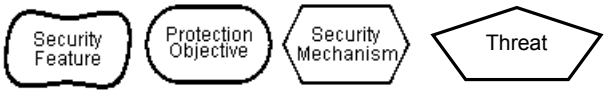

Figure 2. Graphical representation of the security concepts

A part of the *security diagram* for the eSAP system is shown on figure 3. In the presented *security diagram* we take into consideration two desired security features of the eSAP system; privacy and availability. It must be noticed that this diagram is not meant to be a precise and complete *security diagram* of the eSAP system but rather serves as an illustration diagram to help better understand the concepts and notations of a security diagram. Both privacy and availability are receiving negative contributions from different threats on the system such as Social Engineering, Eavesdropping and Cryptographic Attacks, Viruses, System Crashes and Denial of Service Attacks.

On the other hand, the privacy of the system is receiving positive contributions from different protection objectives identified during the security analysis of the system. In our case we have mainly identified the protection objectives taking into consideration the security policy proposed by Ross [20] about medical information systems.

For example one of the most important protection objectives that helps the privacy of the system is *Access*

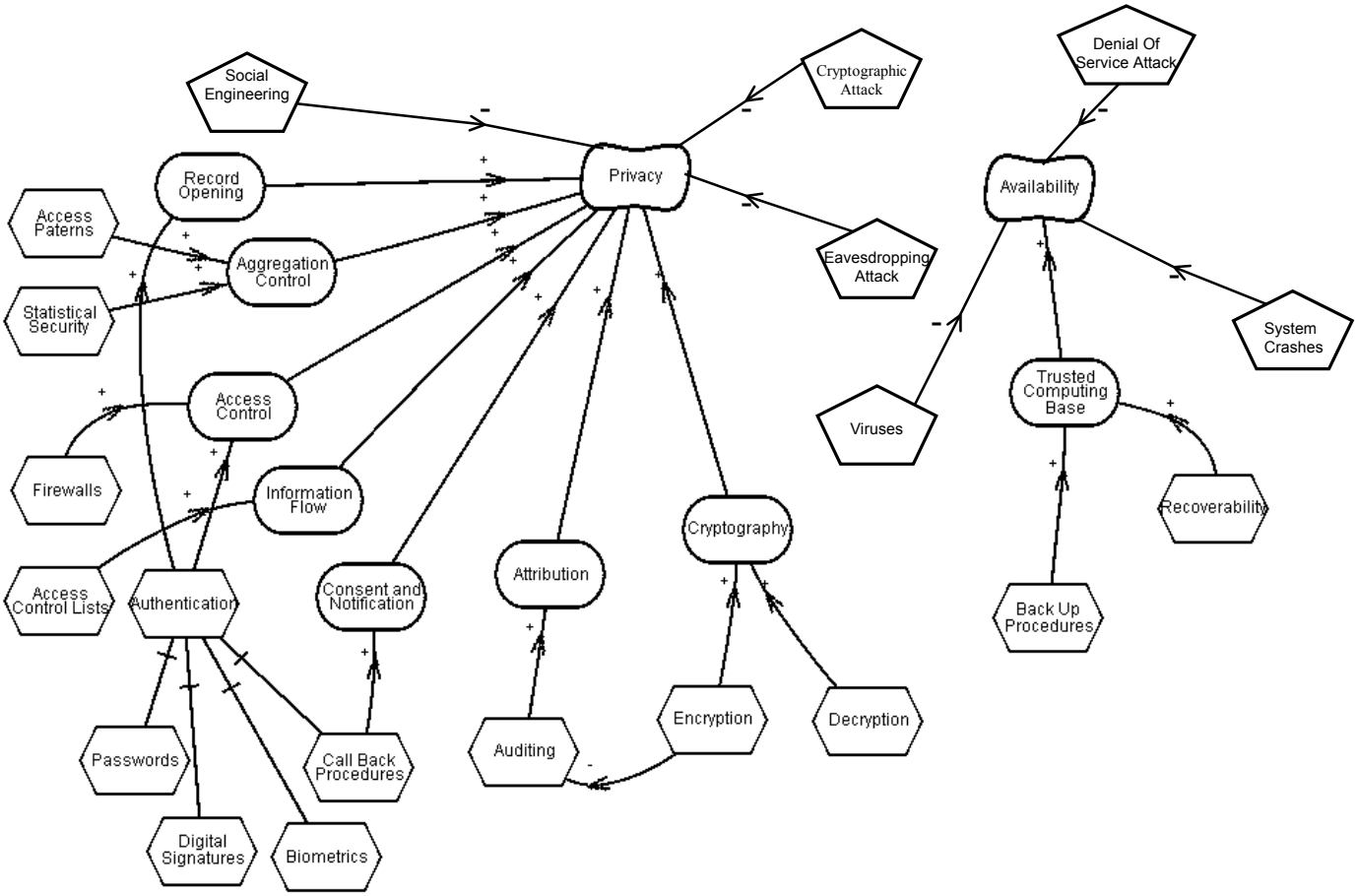

Figure 3. Part of the security diagram of the eSAP system

Control. *Access Control* is achieved basically by *Authentication* of the person that tries to access a resource (in our case a care plan). *Authentication* can be performed using different security mechanisms such as *Biometrics*, *Digital Signatures*, *Call Back Procedures* and *Passwords*. On the other hand, to help towards the availability of the system, *Back-up* and *Recoverability* procedures can be employed. It is worth mentioning that although *Encryption* helps towards the *Cryptography* protection objective, it contributes negative to the satisfaction of *Auditing*.

4.2 Security Constraints

Constraints can be categorised according to the non-functional requirement they are related to. Thus, we can have reliability, performance or security constraints just to mention few of them. In this work we are interested in imposing to a system *security constraints*, in order to help towards the security of the system. We define *security constraint* as a constraint that is related to the security of the system. Since, constraints can influence the security of the system either positively or negatively, we further define positive and negative *security constraints* respectively. An example of a positive security constraint could be *Allow Access Only to Personal Care Plan*, while a negative *security constraint* could be *Send Care Plan Plain Text* (not encrypted). In addition, *security constraints* might also contribute negatively or positively to other requirements of the system.

Security constraints are imposed by the stakeholders (during the early requirements stage) and by the *security diagram* (during the late requirements stage) and are guaranteed by assigning capabilities (security capabilities) to the components of the system (i.e. the actors or the agents of it). Stakeholders can impose positive and negative *security constraints*, while the constraints imposed by the *security diagram* are only positive *security constraints*.

As mentioned above, during the late requirements stage, the software engineer imposes security constraints to the system-to-be according to the security diagram. Most likely "root" *security constraints* will be imposed because of the security features, while sub-constraints will be imposed because of the protection objectives captured at the *security diagram*. For example, a *security diagram* that includes privacy amongst other security features could impose a security constraint *Keep Data Private* to the system. This constraint can be

furthered analysed taking into consideration different protection objectives that the *security diagram* has captured for the privacy security features, such as *Access Control or Cryptography*. In this case the "root" constraint *Keep Data Private* can be decomposed to *Allow Access Only to Personal Care Plan* and *Allow Only Encrypted Data Transfer*. However, this is not a strict process and it depends on the designers and the design decisions they might take. For example, it is possible that a protection objective will not impose a sub-constraint to a root constraint. Constraints are analysed according to the constraint analysis processes (due to lack of space these analysis processes are not described in this paper).

By imposing *security constraints* to different parts of the system, we are able to identify possible conflicts between security and other (functional and non functional) requirements of the system, identify (stakeholder) constraints that can put in danger the security of the system, and propose possible ways towards a design that will integrate security and systems engineering leading to the development of a more secure system.

It is worth mentioning that we consider a *security constraint* contributing to a higher level of abstraction, meaning that a *security constraint* does not involve the identification of particular security protocols so that it does not restrict the development of the system to a specific security solution. This means we are not taking into consideration specific security protocols that should be decided during the implementation of the system, and that most of the times restrict the design with the use of a particular implementation language. A *security constraint* is represented graphically as shown in figure 4.

Figure 4. Graphical representation of a security constrain

4.3 Secure Entities

The term *secure entities* involves any *secure goals*, *tasks* and *resources* of the system. A *secure entity* is introduced to the actor (or the system) in order to help in the achievement of a *security constraint*. For example if the professional actor has a *security constraint Share Info Only If Consent Obtained* a *secure goal* is introduced to this actor *Obtain OP Consent* in order to help in the achievement of the constraint. In a later stage, capabilities are added to the actor (according to the *security entities* added) in order to guarantee the *security constraints*.

A *secure goal* does not particularly define how the *security constraint* can be achieved, since (as in the definition of a goal) alternatives can be considered. However, this is possible through a *secure task*, since a task specifies a way of doing something. Thus, a *secure task* represents a particular way for satisfying a *secure goal*. For example, for the *secure goal Check Authorisation* we might have secure tasks such as *Check Password* or *Check Digital Signatures*.

A resource that is related to a *secure entity* or a *security constraint* is considered a *secure resource*. For example, an actor depends on another actor to receive some information. However, this dependency (resource dependency) is restricted by a constraint *Only Encrypted Info*.

Secure Entities are represented by introducing an S within brackets (S) before the text description as shown in figure 5.

Figure 5. Representation of a secure goal, task, resource.

4.4 Secure Dependencies

A *secure dependency* introduces *security constraint*(s), proposed either by the depender (most likely) or the dependee (most unlikely) in order to successfully satisfy the dependency. Both the depender and the dependee must agree in this constraint (or constraints) for the secure dependency to be valid. That means, in the depender side, the depender expects from the dependee to satisfy the *security constraints* while in the dependee side, a secure dependency means that the dependee will make an effort to deliver the dependum by satisfying the *security constraint*(s). There are two degrees of security: *Open Secure dependency* (normal dependency) and *Secure dependency*. In an *Open Secure Dependency* some security conditions might be introduced but if the dependee fail to satisfy them, the consequences will not be serious. This means that the security of the system will not be in danger if some of these conditions are not satisfied. An *Open Secure Dependency* is graphically represented (figure 6) as unmarked (as the normal dependency). On the other side, there are three different types of a secure dependency:

- *Depender Secure Dependency*, depender depends on dependee, and depender introduces security constraints for the dependency. The dependee must satisfy the security constraints introduced by the depender, otherwise the security of the dependency will be in risk. This type of secure dependency is graphically represented with a constraint at the side of the dependee (figure 6).
- *Dependee Secure Dependency,* depender depends on dependee and dependee introduces security constraints for the dependency. Depender must satisfy the security constraints introduced by the dependee in order to help in the achievement of the secure dependency. This type of secure dependency is graphically represented with a constraint at the side of the depender (figure 6).
- *Double Secure Dependency***,** depender depends on dependee and both depender and dependee introduce security constraints for the dependency. Both must satisfy the security constraints introduced to achieve the secure dependency. This type of secure dependency is represented with constraints on both sides (figure 6).

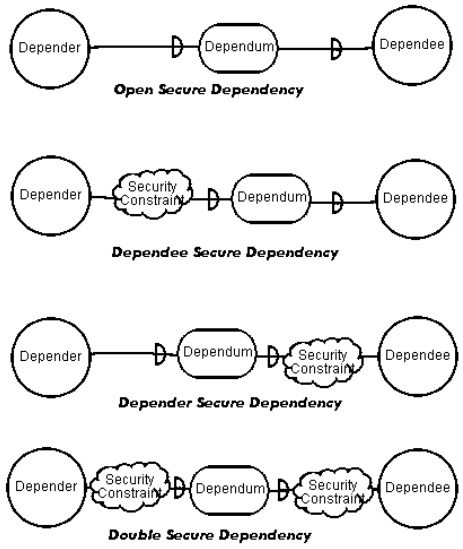

Figure 6. The different types of secure dependencies

5. Security Integration in Tropos

Tropos covers four phases of software development:

- ***Early Requirements***, concerned with the understanding of a problem by studying an existing organisational setting; the output of this phase is an organisational model, which includes relevant actors and their respective dependencies.
- ***Late requirements***, where the system-to-be is described within its operational environment, along with relevant functions and qualities. This description models the system as a (small) number of actors, which have a number of dependencies with actors in their environment; these dependencies define the system's functional and non-functional requirements.
- ***Architectural design***, where the system's global architecture is defined in terms of subsystems, interconnected through data and control flows; within the diagram, subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies.
- ***Detailed design***, where each architectural component is defined in further detail in terms of inputs, outputs, control, and other relevant information. *Tropos* uses elements of AUML [21] to complement the features of *i**. Agent capabilities and interactions are specified.

The process of security is basically one of analysing the security needs of the stakeholders and the system in terms of *security constraints* imposed to the system and the stakeholders, identify secure entities that guarantee the satisfaction of the *security constraints*, and assign capabilities to the system to help towards the satisfaction of the secure entities. So far in this work, we have focused in the integration of security during the early and late requirement stages of the *Tropos* methodology. Thus, the security process is integrated in the *Tropos* stages as follows.

5.1. Early Requirements

In the early requirements analysis the *Security Diagram* (SD) is constructed as described in a previous section. In addition, *security constraints* are imposed to the stakeholders of the system (by other stakeholders). These constraints are analysed and *security entities* are introduced.

In our example, the *Older Person* depends on the *Benefits Agency* to *Receive Financial Support*. However, the *Older Person* worries about the privacy of their finances so they impose a constraint to the *Benefits Agency* actor, to keep their financial information private. The *Professional* depends on the *Older Person* to *Obtain Information*, however one of the most important and delicate matters for a patient (in our case the older person) 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 if and only if consent is achieved. One of the main goals of the *R&D Agency* 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*.

The following figure illustrates part of the actor diagram of the eSAP system taking into consideration the above-mentioned constraints that are imposed to the stakeholders of the system.

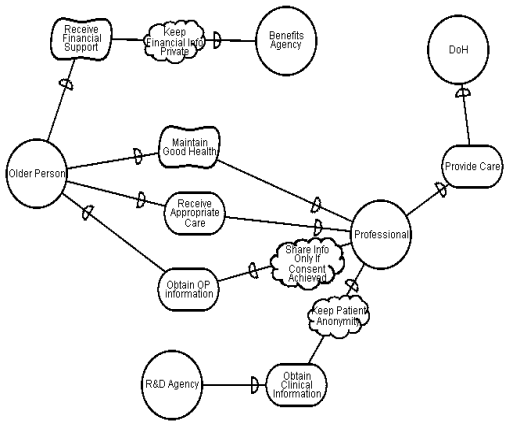

Figure 7. The actor diagram including security constraints

In addition, the security constraints imposed at each actor are further analysed by identifying which goals of the actor they restrict (Figure 8). The assignment of a *security constraint* to a goal is indicated using a constraint link (a link that has the "restricts" tag). For example, the *Professional* actor has been imposed two *security constraints* (*Share Info Only If Consent Achieved* and *Keep Patient Anonymity*). During the means-end analysis of the *Professional* actor we have identified the *Share Medical Info* goal. 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. Thus a sub-constraint is introduced, *Only Obtain Consent Personally*. This sub constraint introduces another *secure goal Personally Obtain Consent*. This goal is divided into two sub-tasks *Obtain Consent by Mail* or *Obtain Consent by Phone*.

The *Professional* has also a goal to *Provide Medical Information for Research*. However, the constraint *Keep Patient Anonymity* has been imposed to the *Professional*, which restricts the *Provide Medical Information for Research* goal. As a result of this constraint a *secure goal* is introduced to the *Professional*, *Provide Only anonymous Info*.

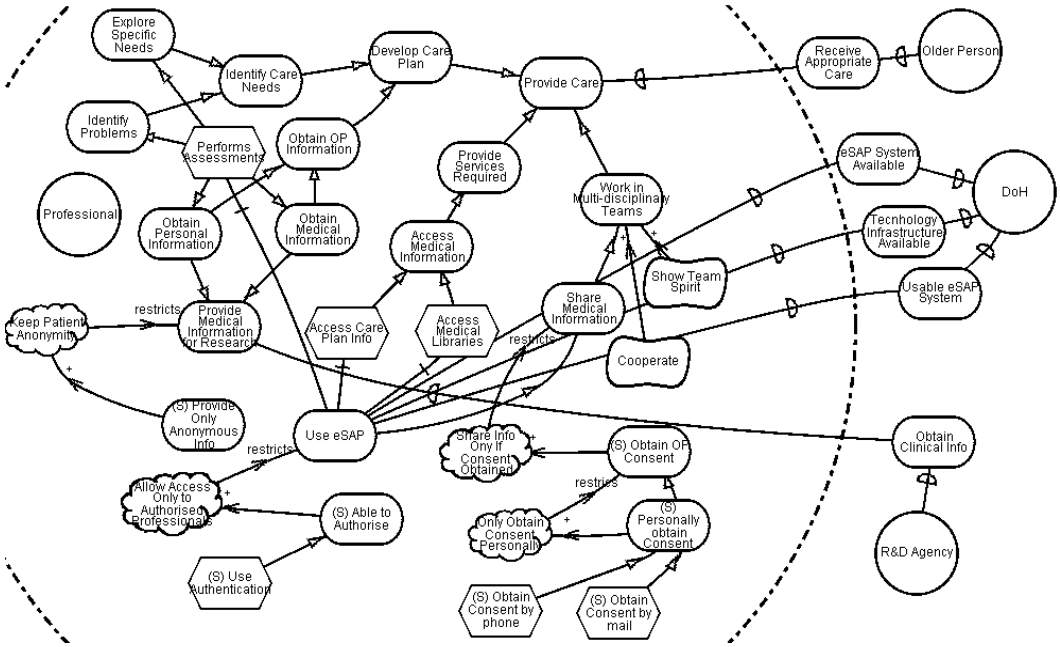

Figure 8. Means-ends analysis of the Professional Actor

5.2 Late Requirements

During the late requirements stage, *security constraints* are imposed to the system-to-be (by taking into account the *security diagram*). These constraints are further analysed according to the constraint analysis processes. The main aim of the eSAP system (Figure 9) is to *Automate Care* in order to help professionals provide faster and more efficient care, and allow on the other hand older people get more involved in their care. Taking into consideration the *security diagram*, of the previous section, we see there are two main constraints imposed (by the desired security features of the system-privacy and availability) to the eSAP's main goal - *Keep Data Private* and *Keep Data Available*. For the eSAP to satisfy these constraints two *secure goals* have been identified. *Ensure Data Privacy* and *Ensure Data Availability*. Although, these statements initially seem very superficial they can be further analysed. In our case we focus only on the *Keep Data Private* constraint. This constraint can be further analysed to sub-constraints *Allow Only Encrypted Transfer of Data*, *Allow Only Authorised Access*, and *Allow Access Only to Personal Care Plan*. Taking into consideration the *security 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*. Thus, the *Use Cryptography* goal is divided to two *secure tasks Encrypt Data* and *Decrypt Data*. It must be noticed in this point that although someone might thought of further decomposing these tasks by indicating for example the type of the encryption algorithm this is not the case, since the type of the encryption algorithm depends on the implementation of the system and it will restrict the developers of the system in a particular implementation style. The *Check Authorisation* 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.

6. Conclusions and future work

In this paper we have presented extensions to the *Tropos* methodology, so that it can deal with capturing security concerns of the system-to-be. During the process of extending *Tropos* it was concluded that *Tropos* methodology facilitates the consideration of security requirements for different reasons:

- By considering the overall software development process it is easy to identify security requirements at the early requirements stage and propagate them until the implementation stage. This introduces a security-oriented paradigm to the software engineering process.
- *Tropos* allows a hierarchical approach towards security. Security would be defined in different levels of complexity, which will allow the software engineer a better understanding while advancing through the process.
- Iteration allows the re-definition of security requirements in different levels therefore providing a better integration with system functionality.
- Consideration of the organisational environment facilitates the understanding of the security needs in terms of the security policy.
- Functional and non-functional requirements are defined together however a clear distinction is provided.

Our aim is to provide a clear well guided process of integrating security and functional requirements throughout the whole range of the development stages. Such a process must use the same concepts and notations throughout the development phases.

As mentioned, the presented extensions apply only to the first two stages (early and late requirements) of the *Tropos* methodology. So far we are able to impose *security constraints* that help towards the satisfaction of the desired security features. We analyse the system in terms of *security constraints* imposed to the system and the stakeholders and then identify *secure entities* that guarantee the satisfaction of the *security constraints*.

Future work involves the assignment of capabilities to the system to help towards the satisfaction of the *secure entities*, and verify the security of the system by analysing potential attacks and if necessary introduce extra secure capabilities. Then, the design of the system will take place by taking into consideration the security analysis performed in the previous stages.

More specifically, during the architectural design stage *security constraints*, which imposed to the new actors of the system, can be further identified. Then secure capabilities can be identified and assigned to each agent of the system. Finally, scenarios can be used to check the security of the system. In the case where security vulnerabilities are identified, extra secure capabilities can be introduced to the system to help towards the identified vulnerabilities.

During the detailed design stage, the agent capabilities and interactions can be specified taking into account the security aspects. AUML notation can be used by introducing the tag of security rules. This is

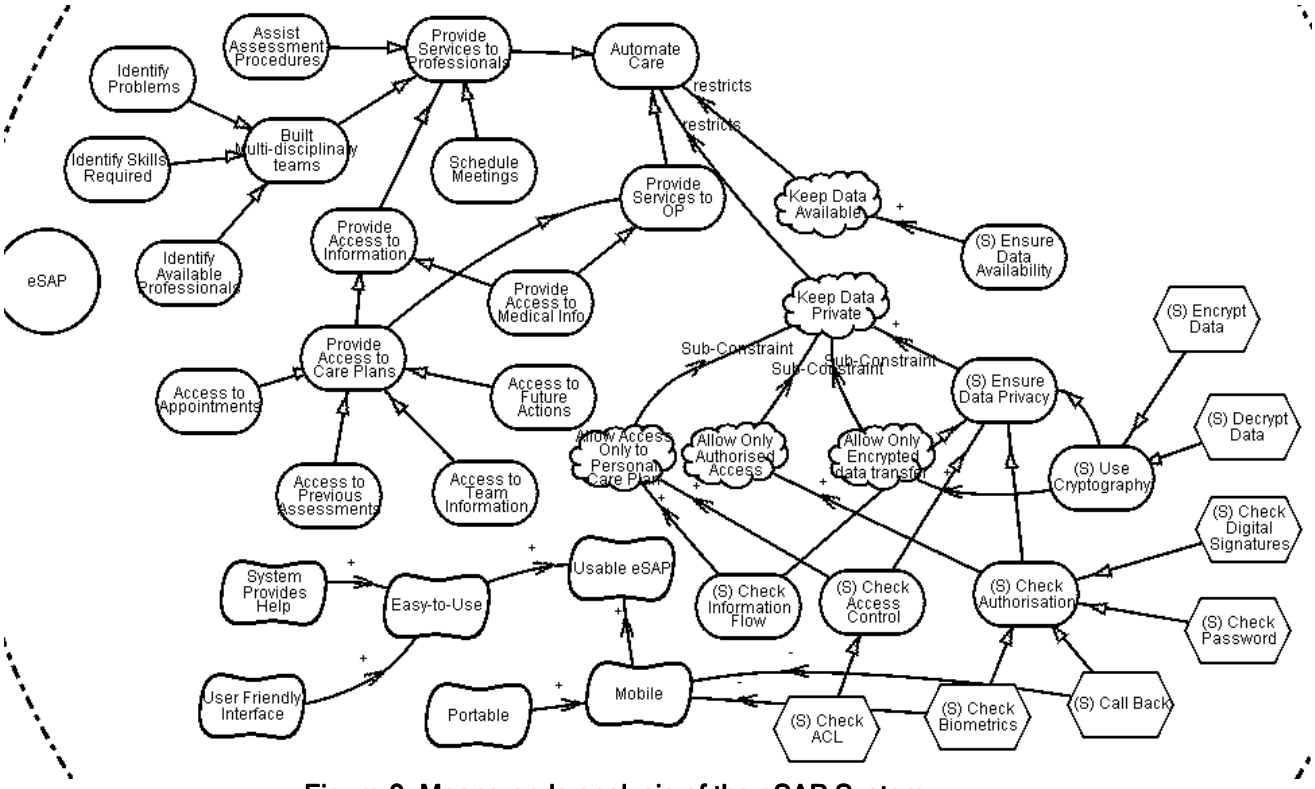

Figure 9. Means-ends analysis of the eSAP System

similar to the business rules that UML has for defining constraints on the diagrams.

In addition, we are constantly refining and checking the identified concepts, notations, and process by applying them to different real life examples in order to justify them.

Tropos graphical representation is complemented using *Formal Tropos* [22], a rich specification language inspired by *KAOS* [23]. *Formal Tropos* provides a textual notation for i* models and is amenable to formal analysis. However, *Formal Tropos* was not conceived with security in mind and it fails to adequately model many security aspects. Towards this direction, we are extending *Formal Tropos* in order to accommodate security modeling according to the above-proposed extensions.

8. Acknowledgements

The first Author would like to thank the RANK Foundation for the funding of his research project, during which this work was carried out. Also many thanks to Erika Sanchez for her comments and assistance with security matters.

9. References

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

[2] 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

[3] N. R. Jennings, "An agent-based approach for building complex software systems", Communications of the ACM, Vol. 44, No 4, April 2001

[4] P. Devanbu, S. Stubblebine, "Software Engineering for Security: a Roadmap", *Proceedings of the conference of The future of Software engineering*, 2000.

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

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

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

[8] Dardenne, A. van Lamsweerde, S. Fickas, "Goal-directed Requirements Acquisition. Science of Computer Programming", *Special issue on 6th Int. Workshop of Software Specification and Design*, 1991.

[9] B. Lampson, "Computer Security in the real world", *Annual Computer Security Applications Conference* 2000.

[10] C. Meadows, "A Model of Computation for the NRL protocol analyser", *Proceedings of the 1994 Computer Security Foundations Workshop*, 1994.

[11] L. Chung, "Dealing with Security Requirements During the Development of Information Systems", *5th International Conference on Advanced Information Systems Engineering*, 1993.

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

[13] L.K. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, "Non-Functional Requirements in Software Engineering", Kluwer Publishing, 2000

[14] L.M. Cysneiros, J.C.S.P. Leite, "A Framework for Integrating Non-Functional Requirements into Conceptual Models", Requirements Engineering Journal, Vol. 6, Issue 2, pp 97-115, April 2001

[15] E. Yu, L. Cysneiros, "Designing for Privacy and Other Competing Requirements", (to appear) 2nd Symposium on Requirements Engineering for Information Security (SREIS' 02), Raleigh, North Carolina, 15-16 November, 2002

[16] I. Sommerville, "Software Engineering", sixth edition, Addison-Wesley, 2001

[17] G.C. Roman, " A Taxonomy of Current Issues in Requirements Engineering", IEEE Computer, Vol. 18, No. 4, pp 14-23, April 1985

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

[19] Steegmans, E., Lewi, J., D'Haese, M., Dockx, J., Jehoul, D., Swennen, B., Van Baelen, S., and Van Hirtum, P., "*EROOS Reference Manual Version 1.0"*, Department of Computer Science, K.U.Leuven, CW Report 208, Leuven, B, 1995, 176 p

[20]R. Anderson, "Security Engineering", Wiley Computer Publishing, 2001

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

[22] A. Fuxman, M. Pistore, J. Mylopoulos, P. Traverso, "Model Checking Early Requirements Specification in Tropos", Proceedings of the 5th Int. Symposium on Requirements Engineering, RE' 01, Toronto, Canada, August 2001

[23] A. Dardenne, A. van Lamsweerde, S. Fickas, "Goal-directed Requirements Acquisition", Science of Computer Programming, 20, pp 3-50, 1993