Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems

Haralambos Mouratidis[1], Paolo Giorgini[2], and Gordon Manson[1]

[1]Department of Computer Science, 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.unit.it

Abstract. Security is a crucial issue for information systems. Traditionally, security is considered after the definition of the system. However, this approach often leads to problems, which translate into security vulnerabilities. From the viewpoint of the traditional security paradigm, it should be possible to eliminate such problems through better integration of security and systems engineering. This paper argues for the need to develop a methodology that considers security as an integral part of the whole system development process. The paper contributes to the current state of the art by proposing an approach that considers security concerns as an integral part of the entire system development process and by relating this approach with existing work. The different stages of the approach are described with the aid of a case study; a health and social care information system.

1 Introduction

Information systems (IS) become more and more critical in every aspect of human society from the health sector to military. As the use of Information Systems arises, the demand to secure those systems also arises. This is true since many information systems contain private data that must be available only to authorised viewers. Take as an example a health and social care information system containing health data of different individuals. Security in such a system, as in any health and social care information system, is very important since security breaches might result in medical history to be revealed, and revealing a medical history could have serious consequences for particular individuals.

Software Engineering considers security as a non-functional requirement [1]. Non-functional requirements introduce quality characteristics, but they also represent constraints under which the system must operate [2,3]. Although software designers have been recognized the need to integrate most of the non-functional requirements, such as reliability and performance, into the software development processes [4] security still remains an afterthought.

Thus the usual approach towards the inclusion of security within a system is to identify security requirements after the definition of a system. However, considering security as an afterthought often leads to problems [7], since security mechanisms

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 63–78, 2003.

have to be fitted into a pre-existing design, therefore leading to serious design challenges that usually translate into software vulnerabilities [8].

There are at least two reasons for the lack of support for security engineering [5]. Firstly security requirements are generally difficult to analyse and model. A major problem in analysing non-functional requirements is that there is a need to separate functional and non-functional requirements yet, at the same time, individual non-functional requirements may relate to one or more functional requirements. If the non-functional requirements are stated separately from the functional requirements, it is sometimes difficult to see the correspondence between them. If stated with the functional requirements, it may be difficult to separate functional and non-functional considerations. Secondly developers lack expertise for secure software development. Many developers, who are not security specialists, must develop systems that require security features. Without an appropriate methodology to guide those developers on the development processes, it is likely that they will fail to produce effective solutions [6].

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).

However, current methodologies for IS development do not meet the needs for resolving the security related IS problems [9], and fail to provide evidence of integrating successfully security concerns throughout the whole range of the development process.

In this paper we present an approach that integrates security and systems engineering, using the same concepts and notations, throughout the entire system development process. This work falls within the context of the Tropos methodology [10] in which security requirements are considered as an integral part of the whole development process.

The paper is structured as follows. Section 2 provides an introduction to the Tropos methodology describing briefly the methodology stages and its concepts, while Section 3 describes a health and social care information system that is used as a case study throughout the paper. Section 4 illustrates how our approach integrates security and systems engineering within the Tropos development process and Section 5 relates our work to the literature by providing an overview of related work. Finally, Section 6 concludes the paper.

2 Tropos Methodology

Tropos is a development methodology tailored to describe both the organisational environment of a system and the system itself. Tropos is characterised by three key

aspects [11]. Firstly, it deals with all the phases of system requirements analysis and system design and implementation[1] adopting a uniform and homogeneous way. Secondly, Tropos pays great deal of attention to the early requirements analysis that precedes the specification of the perspective requirements, emphasizing the need to understand the how and why the intended system would meet the organisational goals. This allows for a more refined analysis of the system dependencies, leading to a better treatment not only of the system functional requirements but also of its non-functional requirements, such as security, reliability, and performance [11]. 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 [12].

Tropos adopts the *i** modelling framework [13], which uses the concepts of actors, goals, tasks, resources and social dependencies for defining the obligations of actors (dependees) to other actors (dependers). Actors have strategic goals and intentions within the system or the organisation and represent (social) agents (organisational, human or software), roles or positions (represent a set of roles). A goal represents the strategic interests of an actor. In Tropos we differentiate between hard (only goals hereafter) and soft goals. The latter having no clear definition or criteria for deciding whether they are satisfied or not [13]. A task represents a way of doing something. Thus, for example a task can be executed in order to satisfy a goal. A resource represents a physical or an informational entity while a dependency between two actors indicates that one actor depends on another to accomplish a goal, execute a task, or deliver a resource. Figure 1 shows the graphical representation of the above-mentioned concepts.

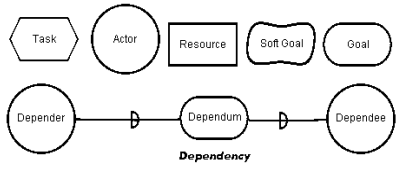

Fig. 1. Graphical Representation of Tropos Concepts

Although Tropos was not conceived with security on mind, a set of security concepts, such as *security constraint*, *secure entities* and *secure dependencies* have been proposed [14] to enable it to consider security aspects throughout the whole development process. A s*ecurity constraint* is defined as a constraint that is related to the security of the system, while *secure entities* represent any secure goals/tasks/resources of the system. *Secure goals* are introduced to the system to help in the achievement of a *security constraint*. A *secure goal* does not particularly define how the *security constraint* can be achieved, since (as in the definition of goal, see [13]) alternatives can be considered. However, this is possible through a *secure task*, since a task specifies a way of doing something [13]. Thus, a *secure task* represents a particular way for satisfying a *secure goal*. For example, for the secure goal *Authorise Access,* we might have secure tasks such as *Check Password* or *Check Digital*

[1] In this paper we do not consider the implementation stage. Readers interested in this stage can refer to [10].

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 and this dependency (resource dependency) is restricted by a constraint *Only Encrypted Info*.

A *secure dependency [14]* introduces *security constraint*(s), proposed either by the depender or the dependee in order to successfully satisfy the dependency. For example a *Doctor* (depender) depends on a *Patient* (dependee) to obtain *Health Information* (dependum). However, the *Patient* imposes a *security constraint* to the *Doctor* to share *health information only if consent is obtained*. 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). The above-mentioned security concepts are illustrated in Figure 2.

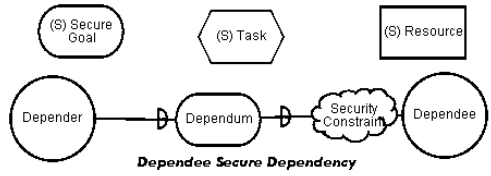

Fig. 2. Graphical Representation of the Security Concepts

Tropos covers four main software development phases:

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, their respective dependencies and the security constraints imposed to those actors.

Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and security requirements; this description models the system as a (small) number of actors, which have a number of dependencies and security constraints. These dependencies define the system's functional requirements, while the security constraints define the system's security requirements.

Architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data and control flows. Within the framework, subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies. In addition, during this stage, different architectural styles are analysed taking into account security and other non-functional requirements of the system and secure capabilities are identified and assigned to the different actors of the system to satisfy the secure entities.

Detailed design, where each architectural component is defined in further detail in terms of inputs, outputs, control, and the security aspects analysed in the previous stages. For this stage, *Tropos* is using elements of UML [15] to complement the features of *i**.

3 Case Study

This section introduces the case study that will be used in the rest of this paper to describe the security analysis process throughout the different stages of the Tropos methodology.

We consider the electronic Single Assessment Process (eSAP) system [16], an integrated health and social care information system for the effective care of older people. Security in the eSAP is an important concern, since security breaches of such system might result in personal and health information to be revealed and this could lead to serious consequences.

It must be noticed that, in our example, many functionalities of the system are omitted, since our aim is not to explore the complexity of the system, but rather to demonstrate how the Tropos methodology integrates security and systems engineering.

Throughout our case study, the security policy principles identified in [7] are used. In addition, some more principles are added: (1) System Authorisation, only authorised professionals and patients can access the system; (2) Access Control, each Care Plan shall be marked with an access control list naming the people or groups who may read it and append data to it. The system should prevent anyone not on the list from accessing the record in any way; (3) Care Plan Opening, a professional may open a care plan with themselves and the older person on the access control list. When an older person has been referred, the professional might open a record with themselves, the older person, and the referring professional on the access control list; (4) Control, only one of the professionals (most likely the professional responsible for the older person) may alter the control list, and add other professionals; (5) Information Flow, information derived from care plan A may be appended to care plan B if and only if B's Access control list is contained in A's; (6) Availability, the information must be available whenever a person included in the access control list requires any information.

4 The Development Process

4.1 Early Requirements

During the early requirements stage, the goals, dependencies and the security constraints between the stakeholders (actors) are modeled with the aid of an actors' diagram [11]. Such a diagram involves different actors, represented as nodes, and dependencies, represented as links, between the different actors that indicate that one depends on the other to accomplish some goals and also that some security constraints must be satisfied for the dependencies to be valid.

For the eSAP case study, we consider the following actors (Figure 3)

- *Professional*: the health and/or social care professional;
- *Older Person*: the Older Person (patient) that wishes to receive appropriate health and social care;
- *DoH*: the English Department of Health;

- *R&D Agency*: a Research and Development Agency interested in obtaining medical information;
- *Benefits Agency*: an agency that helps the older person financially.

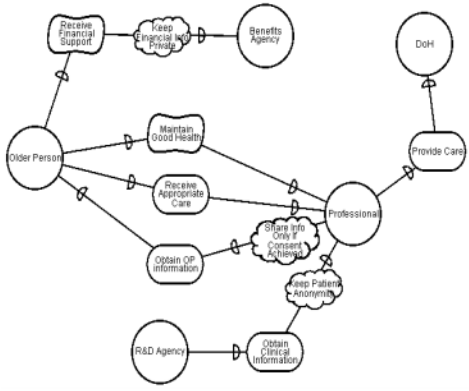

Fig. 3. Actors Diagram

The main goal for the *Older Person* actor is to *Maintain Good Health*[2] and a secondary goal is to *Receive Appropriate Care*. Since the *Older Person* cannot guarantee either of those goals alone, they depend on the *Professional* to help them satisfy them. In addition, 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 (Older Person) OP Information*. However one of the most important and delicate matters for 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 obtained*. On the other hand, 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*.

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 in more 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*. In this example, due to lack of space, we focus only in the analysis of the *security constraints*, and not in the goal or task analysis of each individual actor.

The analysis of the *security constraints* starts by identifying which goals of the actor they restrict. The assignment of a *security constraint* to a goal is indicated using a *constraint link* (a link that has the "restricts" tag). In addition, different alternatives

[2] It is captured as a soft goal since we cannot precisely define what "good health" means for different individuals.

can be considered for achieving the goals and the *security goals* of the stakeholders. For example, during the previous step (Figure 3), the *Professional* actor has been imposed two *security constraints* (*Share Info Only If Consent Achieved* and *Keep Patient Anonymity*). By analyzing the *Professional* actor (Figure 4) we have identified the *Share Medical Info* goal. However, this goal is restricted by the *Share Info Only If Consent Obtained* 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*.

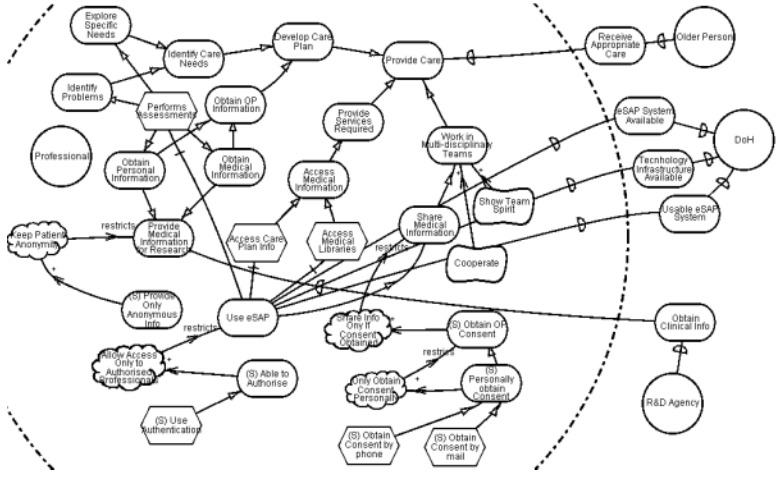

Fig. 4. Partial Analysis of the Professional Actor

4.2 Late Requirements

In the late requirements stage, the functional, security, and other non-functional requirements for the system-to-be are described. The system-to-be is introduced as one or more actors who have a number of dependencies with the other actors of the organization (defined during the early requirements stage) and it (the system) contributes to the goals of the stakeholders.

In our case study, one of the main aims of the *Department of Health* is to allow older people to get more involved in their care and also help professionals provide

more efficient care. For this reason, the *Department of Health* depends on the *electronic Single Assessment Process* (eSAP) system to automate care. Thus, the eSAP system is introduced as another actor and it is analysed using the same concepts used for the analysis of the other actors.

Figure 5 shows a partial analysis of the *eSAP* System. To automate care the *eSAP* system must provide services. This goal can be decomposed into two different goals, *Provide Services to Professionals* and *Provide Services to Older People*. Both those sub-goals must be achieved for the top goal to be achieved. In addition, each of those goals can be decomposed to a number of alternative tasks. For example, the *Provide Services to Professionals* goal can be alternatively fulfilled by tasks *Assist Schedule Meetings*, *Assist with Assessment Procedures*, or *Assist with Care Plan Management*.

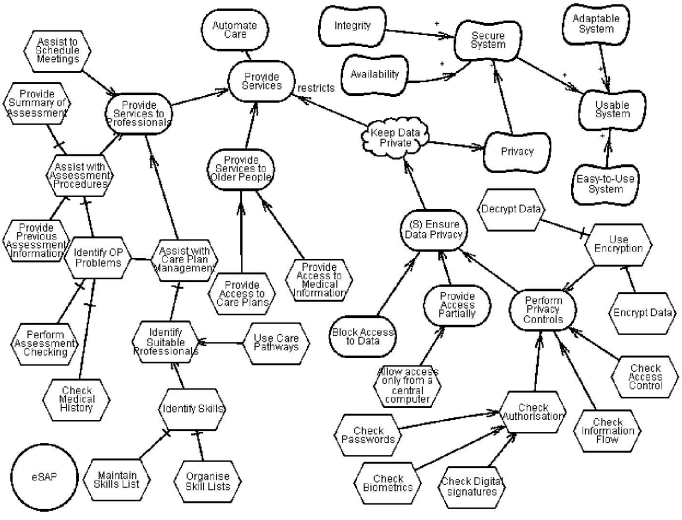

Fig. 5. Partial Analysis of the eSAP

In addition, to satisfy the security of the system, different *security constraints* are imposed to the system (according to the security policy defined in the previous section). In our case study, due to lack of space, we only consider the *Keep Data Private* security constraint that is imposed in order to contribute towards the privacy of the system. To satisfy this constraint, a secure goal *Ensure Data Privacy* is introduced to the system. This goal can be achieved either through the sub-goal *Block Access to Data* that blocks any access to the system (obviously not desirable); or provide access partially, which allows access only from a central computer; or perform privacy controls. The latter sub goal is fulfilled by the tasks *Use Encryption*, *Check Access Control*, *Check Information Flow*, and *Check Authorisation*. Each of those tasks can be achieved by considering different alternatives. For example, in order to *check authorisation* different alternatives can be considered such as *check passwords*, *check biometrics* or *check digital signatures*. An approach to evaluate the different alternatives could be to use the measures of *complexity* and *criticality* [17].

Complexity represents the effort required from an actor for achieving a (security) task, while *criticality* represents how the (security) goals of the actor will be affected if a (security) task is not achieved. Thus, by knowing how complex and how critical the different alternatives are, we can decide which alternative is the best solution.

4.3 Architectural Design

The architectural design phase defines the system's global architecture. During architectural design the first step is to identify the overall architectural organization by selecting among alternative architectural styles[3] using as criteria the non-functional requirements of the system identified in the previous stage.

However, quality characteristics (non-functional requirements) are difficult to measure since it is difficult to get empirical evidence during the design stages. For this reason, we employ an analysis process based on an independent probabilistic model, which uses the measure of *satisfiability* proposed by Giorgini et al [18]. In our example, *satisfiability* represents the probability that the non-functional requirement will be satisfied. Thus, the evaluation results in contribution relationships from the architectural styles to the probability of satisfying the non- functional requirements of the system identified in the late requirements stage. We use a weight to express the contribution of each style to the *satisfiability* of the non-functional requirements. Weights take a value between 0 and 1. For example, 0.1 means the probability that the architectural style will satisfy the requirement is very low (the style is not suitable for satisfying the requirement), while 0.9 means the probability that the architectural style will satisfy the requirement is very high (the style is suitable for satisfying the requirement).

The analysis involves the identification of more specific non-functional requirements, by refining the ones identified during the late requirements stage, and the evaluation of different architectural styles against those requirements. It must be noticed that the refinement of the security requirements took place during the late requirements analysis with the identification of secure tasks, so from the security point of view, the alternative architectural styles are evaluated against those tasks.

The weights of the contribution links reported in Figure 6, of each architectural style to the different non-functional requirements of the system, have been assigned after reviewing different studies [19,20], evaluations [21], and comparisons [22] involving the architectural styles. We must note that Figure 6 represents a partial illustration of the comparison process. For example, from the security point of view, we only consider privacy as in the previous stages of this example development. However, when the approach is employed in a complete comparison, all the security issues must be taken into account. In addition, we have omitted, in order to keep the figure simple and easy to understand, the contributions and the conflicts amongst the different non-functional requirements. For example, although privacy contributes negative to the mobility requirement, this is not shown in the figure.

We consider two architectural styles, a hierarchical style – *client/server* – and a mobile code style -*mobile agents*. We decided to consider those two since *client/server* is the most frequently encountered of the architectural styles for

[3] To avoid confusion we must note that architectural styles differ from architectures in that " a style can be thought of as a set of constraints on an architecture" [19, p. 25]

network-based applications [21], while *mobile agents* form a growing and quite different architectural style. In *client/server* style, a node is acting as a server that represents a process that provides services to other nodes, which act as clients. The server listens for requests upon the offered services. The basic form of *client/server* does not constrain how application state is partitioned between client and server components [21]. *Client/server* architectural style is also referred to by the mechanisms used for the connector implementation such as Remote Procedure Call (RPC) [21]. RPC is appropriate for *client/server* architectural styles since the client can issue a request and wait for the server's response before continuing its own processing. On the other side, in *mobile agents* style, mobility is used in order to dynamically change the distance between the processing and source of data or destination of results. The computational component is moved to the remote site, along with its state, the code it needs and possibly some data required to perform the task [21].

As shown in Figure 6, each of the two styles satisfies differently each of the non-functional requirements of the system. For instance, the *mobile agents* style allows more scalable applications (weight 0.8), because of the dynamic deployment of the mobile code. For example, a doctor wishes to access a large number of medical information, filtered according to the content. In the (pure) *client/server* architectural style (weight 0.4), the doctor would access the server data (medical information) and all the retrieved information would be transferred to the client. Then the filtering would be performed at the doctor site. In the *mobile agents* architectural style, such a filtering can be performed in the server site, where redundant information can be identified early and thus does not have to be transferred to the client. The latter approach is more scalable since the required filtering is distributed and can be performed close to the information sources.

On the other side, *mobile agents* style offers a greater opportunity for abuse and misuse, broadening the scale of security issues significantly [23]. This is due to the fact that mobility is involved. Thus, although protection of a server from mobile agents, or generally mobile code, is an evolution of security mechanisms applied in other architectural styles, such as *client/server*; the mechanisms focused on the protection of the *mobile agents* from the server cannot, so far, prevent malicious behaviour from occurring but may be able to detect it [23]. For example, the information flow property is easier to be damaged by employing mobile agents (weight 0.4) since possible platforms that a mobile agent could visit might expose sensitive information from the agent [23]. In the case of the *client/server* style (weight 0.8) sensitive information is stored in the server and existing security measures could be taken to satisfy the information flow attribute.

When the contribution weights for each architectural style to the different non-functional requirements of the system have been assigned, the best-suited architectural style is decided. This decision involves the categorization of the non-functional requirements according to the importance to the system and the identification of the architectural style that best satisfies the most important non-functional requirement using a propagation algorithm, such as the one presented by Giorgini et al [18]. In our example, security is the number one concern for the eSAP system and thus the architectural style that satisfies most the privacy (since we only consider privacy in this example) requirements of the system is the *client/server* style

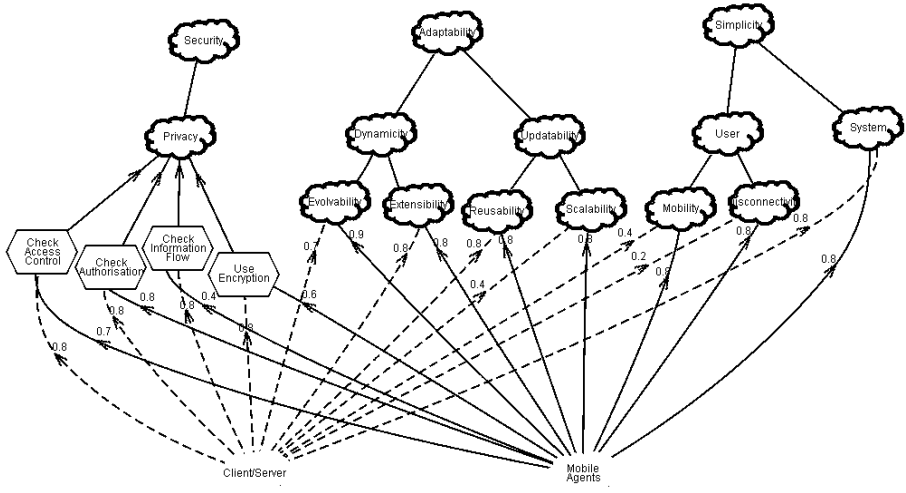

Fig. 6. Deciding for the system's architecture

(figure 6). In the case that two or more non-functional requirements are of the same importance, the presented approach can be integrated with other analysis techniques, such as the SAAM [24], to indicate which architectural style is best suited for the system-to-be.

As mentioned by Castro et al [10], an interesting decision that comes up during the architectural design is whether fulfillment of an actor's obligations will be accomplished through assistance from other actors, through delegation, or through decomposition of the actor into component actors. Thus, when various architectural styles have been evaluated, and one has been chosen, the next step of the architectural design stage involves the introduction of new actors and their dependencies, as well as the decomposition of existing actors into sub-actors and the delegation of some (security) responsibilities from the existing actors to the introduced sub-actors.

Figure 7 shows a partial decomposition of the *eSAP* system. In this example, focused on privacy, the *eSAP* system delegates responsibility for the *Ensure Data Privacy* secure goal to the *Privacy Manager*. The *Privacy Manager*, in order to efficiently fulfill the *Ensure Data Privacy* secure goal, delegates responsibility to the *Authorisation Manager* (to fulfill the *check authorisation* secure sub-goal), the *Access Control Manager* (to fulfill the *check access control* secure sub-goal), the *Information Flow Manager* (to fulfill the *check information flow* secure sub-goal) and the *Cryptography Manager* (to fulfill the *use cryptography* secure goal).

The last step of the architectural design is to identify capabilities for each of the actors by taking into account dependency relationships of the actors. A capability represents the ability of an actor of defining, choosing and executing a plan for the fulfillment of a goal, given certain world conditions and in presence of a specific event [11]. For example, the *Authorisation Manager* should have capabilities such as obtain authorisation details and provide authorisation clearance. However, the process of identifying capabilities for each actor has been extensively described in the literature [11,25] and thus it is not described here.

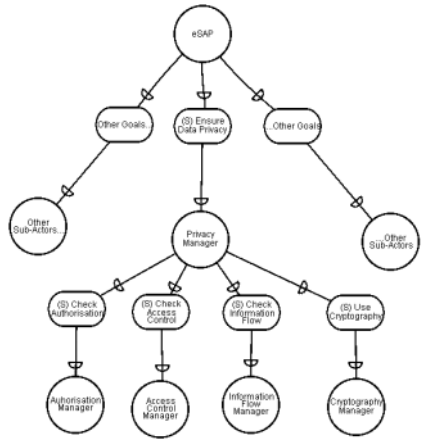

Fig. 7. eSAP system decomposition to ensure data privacy

4.4 Detailed Design

During the detailed design each component of the system, identified in the previous stages, is further specified. In Tropos the detailed design stage is based on the specifications resulted from the architectural design stage, and the reasons for a given component can be traced back to the early requirements analysis.

From the security point of view, during the detailed design the developers specify in detail the actors' capabilities and interactions taking into account the security aspects derived from the previous steps of the analysis.

For the detailed design stage, Tropos adopt a subset of UML [15] diagrams. In our example, a professional might request to view a care plan. The sequence diagram (simplified) for this request is shown in figure 8. However, it would be useful to denote under what constraints the authorisation is granted. For our example, the professional is allowed to access a care plan if and only if the professional's name is included in the care plan access table. For this purpose, we introduce security rules. These are similar to the business rules that UML has for defining constraints on the diagrams. Graphically, security rules are placed on Notes and attached to the related structure as shown in figure 8.

5 Related Work

As stated in the introduction, very little work has taken place in considering security requirements as an integral part of the whole software development process. However, literature provides some approaches towards this direction. 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 [26], while qualitative directions provide a process-oriented approach [1].

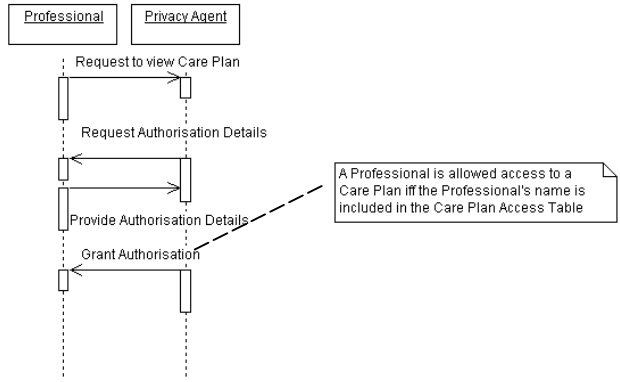

Fig. 8. Simplified sequence diagram including security rule notation

Chung applies a process-oriented approach [1] to represent security requirements as potentially conflicting or harmonious goals and using them during the development of software systems [1]. Rohrig [27], proposes an approach to re-use existing business process descriptions for the analysis of security requirements and the derivation of necessary security measures, while Liu et al [28] explore the explicit modeling of relationships among strategic actors in order to elicit, identify and analyse security requirements. The concept of obstacle is used in KAOS framework [29] to capture undesired properties of the system, and define and relate security requirements to other system requirements.

In addition, Jurgens proposes UMLsec [30], an extension of the Unified Modelling Language (UML), to include modeling of security related features, such as confidentiality and access control. Lodderstedt et al present a modeling language, based on UML, called SecureUML [31]. Their approach is focused on modeling access control policies and how these (policies) can be integrated into a model-driven software development process.

McDermott and Fox adapt use cases [6] to capture and analyse security requirements, and they call the adaption an abuse case model [6]. An abuse case is defined as a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders of the system [6]. Also, Guttorm and Opdahl [32] define the concept of a misuse case, the inverse of a use case, which describes a function that the system should not allow. In their approach security is considered by analysing security related misuse cases.

These above-mentioned approaches only guideline the way security can be handled within a certain stage of the software development process. Differently than them, our approach covers the whole development process. As mentioned in the introduction, it is important to consider security using the same concepts and notations during the whole development process.

6 Conclusions

Although Security is an important issue in the development of computerised systems, currently the common approach towards the inclusion of security within a system is to identify security requirements after the definition of a system. However, as pointed earlier, this approach leads many times to problems and systems full of security vulnerabilities. It should be possible to eliminate such problems through the integration of security concerns at every phase of the system development. To achieve this goal, methodologies must provide developers (even those not expert on security) guidance through a systematic process, which will integrate security and systems engineering at every phase of the system development cycle.

The main contribution of this paper is the introduction of a process that integrates security and systems engineering, using the same concepts and notations, in the entire system development process. The integrated security process in Tropos is 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. This process is characterized by five key ideas. Firstly 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. Secondly, *Tropos* allows a hierarchical approach towards security. Security is defined in different levels of complexity, which allows the software engineer a better understanding while advancing through the process. Thirdly, iteration allows the re-definition of security requirements in different levels therefore providing a better integration with system functionality. Fourthly, consideration of the organisational environment facilitates the understanding of the security needs in terms of the security policy. In addition, functional and non-functional requirements are defined together however a clear distinction is provided.

Future work includes applying our process to different case studies to refine it and also integrate our extensions to the Formal Tropos [33] specification language to enable us to formally evaluate it. The formal part of the work will also allow us to prove and check the properties of the system.

References

1. L. Chung, B. Nixon, “Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach”, Proceedings of the 17th International Conference on Software Engineering, Seattle- USA, 1995
2. I. Sommerville, “Software Engineering”, sixth edition, Addison-Wesley, 2001
3. 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
4. A. 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.

5. B. Lampson, "Computer Security in the real world", *Annual Computer Security Applications Conference* 2000.
6. J. McDermott, C. Fox, "Using Abuse Care Models for Security Requirements Analysis", Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.
7. R. Anderson, "Security Engineering: A Guide to Building Dependable Distributed Systems", Wiley Computer Publishing, 2001
8. W. Stallings, "Cryptography and Network Security: Principles and Practice", Second Edition, Prentice-Hall 1999.
9. T. Tryfonas, E. Kiountouzis, A. Poulymenakou. "Embedding security practices in contemporary information systems development approaches", Information Management & Computer Security, Vol 9 Issue 4, 2001, pp 183–197
10. 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.
11. A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos. "Towards an Agent Oriented Approach to Software Engineering. In A. Omicini and M.Viroli, editors, WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software, Modena-Italy, September 2001.
12. P. Bresciani and 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.
13. E. Yu, "Modelling Strategic Relationships for Process Reengineering", PhD thesis, Department of Computer Science, University of Toronto, Canada, 1995.
14. 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.
15. 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.
16. H. Mouratidis, i. Philp, G. Manson, "Analysis and Design of eSAP: An Integrated Health and Social Care Information System", in the Proceedings of the 7th International Symposium on Health Information Managements Research (ISHIMR2002), Sheffield, June 2002
17. M. Garzetti, P. Giorgini, J. Mylopoulos, F. Sannicolo, "Applying Tropos Methodology to a real case study: Complexity and Criticality Analysis", in the Proceedings of the Second Italian workshop on "WOA 2002 dagli oggetti agli agenti dall'informazione alla conoscenza", Milano, 18–19 November 2002
18. 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.
19. L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", SEI Series in Software Engineering, Addison – Wesley, 1998.
20. J. Bosch, "Design and Use of Software Architectures: adopting and evolving a product-line approach", ACM Press, Addison – Wesley, 2000.
21. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Doctoral dissertation, University of California, Irvine, 2000
22. A. Puliafito, S. Riccobene, M. Scarpa, ``Which paradigm should I use?: An analytical comparison of the client-server, remote evaluation and mobile agents paradigms'', *IEEE Concurrency and Computation: Practice & Experience*, vol. 13, pp. 71–94, 2001.
23. Kotz, D.; Mattern, F. (Eds.): Agent Systems, Mobile Agents, and Applications. Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000, pp. 57–72. LNCS 1882, Springer-Verlag, 2000

24. R. Kazman, G. Abowd, L. Bass, M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures", Proceedings of ICSE-16, Sorrento – Italy, May, 1994.
25. 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
26. C. Meadows, "A Model of Computation for the NRL protocol analyser", *Proceedings of the 1994 Computer Security Foundations Workshop*, 1994.
27. 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, January 2002, L'Aquila, Italy
28. L. Liu, E. Yu, J. Mylopoulos, "Analysing Security Requirements as Relationships Among Strategic Actors", 2nd Symposium on Requirements Engineering for Information Security (SREIS'02). Raleigh, North Carolina, October 16, 2002.
29. 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.
30. Jan Jürjens, "Towards Secure Systems Development with UMLsec", Fundamental Approaches to Software Engineering (FASE/ETAPS) 2001, International Conference, Genoa 4–6 April 2001
31. 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.
32. S. Guttorm, A. L. Opdahl, "Eliciting Security Requirements by Misuse Cases", Proceedings of TOOLS Pacific 2000, November 2000.
33. 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