Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Information Systems 30 (2005) 609–629

www.elsevier.com/locate/infosys

When security meets software engineering: a case of modelling secure information systems☆

Haralambos Mouratidis[a,*], Paolo Giorgini[b], Gordon Manson[a]

[a]Department of Computer Science, University of Sheffield, UK
[b]Department of Information and Communication Technology, University of Trento, Italy

Received 27 February 2004; accepted 7 June 2004

Abstract

Although security is a crucial issue for information systems, traditionally, it is considered after the definition of the system. This approach often leads to problems, which most of the times 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 software engineering. This paper firstly argues for the need to develop a methodology that considers security as an integral part of the whole system development process, and secondly it 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 real-life case study; a health and social care information system.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Information system design; Requirements analysis; Security engineering

1. Introduction

As information systems (IS) become more and more critical in every aspect of the human society, from the health sector to military, so does the demand to secure these systems. This is mainly because private information is stored in computer systems and without security, organisations (and individuals) are not willing to share information or even use the technology.

Consider, for example, a health and social care information system containing health data of

☆This is an extended and revised version of the "Integrating Security and Systems Engineering: towards the Modeling of Secure Information Systems" paper presented at the 15th International Conference on Advanced Information Systems Engineering (CaiSE 2003) and published in *Advanced Information Systems Engineering*, J. Eder and M. Missikiff (Eds.), Springer LNCS 2681, 2003.

*Corresponding author. School of Computing and Technology, University of East London, Barking Campus, Longbridge Road, Dagenham RM8 2AS, UK. Tel.: +44-20-8223-3315; fax: +44-20-8223-2963

E-mail addresses: haris@uel.ac.uk (H. Mouratidis), paolo.giorgini@dit.unit.it (P. Giorgini), g.manson@dcs.shef.ac.uk (G. Manson).

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 Engineers consider security as a non-functional requirement, but unlike other non-functional requirements, such as reliability and performance, security has not been fully integrated within the development lifecycle and it is still mainly considered after the design of the system. However, security introduces not only quality characteristics but also constraints under which the system must operate. Ignoring such constraints during the development process could lead to serious problems [1], since security mechanisms would have to be fitted into a pre-existing design, therefore leading to design challenges that usually translate into software vulnerabilities [2].

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 [3], and fail to provide evidence of integrating successfully security concerns throughout the whole range of the development process.

There are at least two reasons for the lack of support for security engineering [4]:

1. 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.
2. 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 [5].

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 [6,7] 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. Section 3 describes the security extensions to the Tropos methodology to enable it to model security issues, whereas Section 4 describes a health and social care information system that is used as a case study throughout the paper. Section 5 illustrates how our approach integrates security and systems engineering within the Tropos development process and Section 6 relates our work to the literature by providing an overview of related work. Finally, Section 7 provides directions for future work and it 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 [6,8]. 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, emphasising 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 [8]. 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 [10].

Tropos adopts the *i** modelling framework [11], 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 [11]. 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. Fig. 1a shows the graphical representation of the above-mentioned concepts.

Because of these concepts, in Tropos, the system (as well as its environment) is seen as a set of actors, who depend on other actors to help them fulfil their goals. 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. Fig. 1b illustrates a graphical representation of a goal dependency.

[1]In this paper we do not consider the implementation stage. Readers interested in this stage can refer to [9].

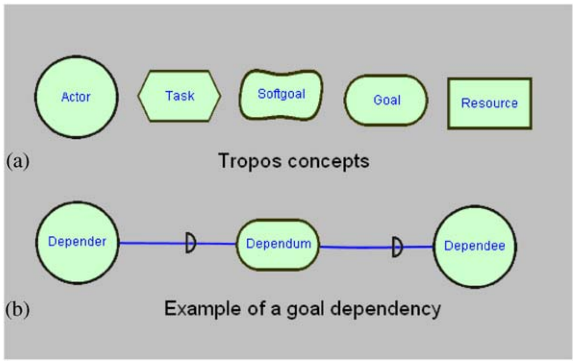

Fig. 1. Graphical representation of Tropos concepts.

Tropos covers four main software development phases:

`Early requirements analysis`, 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 analysis`, 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 further defined in terms of inputs, outputs, control, and the security aspects analysed in the previous stages. For this stage, Tropos is using elements of agent unified modelling language (AUML) [12] to complement the features of *i**.

3. The security extensions

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 [13] to enable it to consider security aspects throughout the whole development process.

To enable developers to adequately capture security requirements the concept of constraint [13] is introduced and it is extended it with respect to security. In addition, the Tropos concepts of dependency, goal, task, resource, and capability are also extended with security in mind. All these concepts are defined within the Tropos project as secure entities.

In the context of our work, a security constraint is defined as *a restriction related to security issues, such as privacy, integrity and availability, of an information system that it can influence the analysis and design of the system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives*.

A security constraint contributes to a higher level of abstraction, meaning that security constraints do not represent specific security protocol restrictions,[2] which restrict the design with the use of a particular implementation language. This higher level of abstraction allows for a generalised design free of models biased to particular implementation languages.

[2]Such security restrictions should be specified during the implementation of the system.

Moreover, the term secure entity is used within the context of our work to describe goals, tasks, and resources related to the security of the system. In other words, a secure entity represents a secure goal, a secure task or a secure resource.

`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 [11]) alternatives can be considered. However, this is possible through a secure task, since a task specifies a way of doing something [11]. 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 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`. All these security-related concepts are graphically represented as illustrated in Fig. 2.

The only difference in the representation of non-secure Tropos concepts and secure concepts is an S (Security) within brackets that appears in the beginning of the security concept description to indicate that the concept is related to the security of the information system.

A `secure dependency` [13] introduces security constraint (s), proposed either by the depender or the dependee in order to successfully satisfy the dependency. Both the depender and the dependee

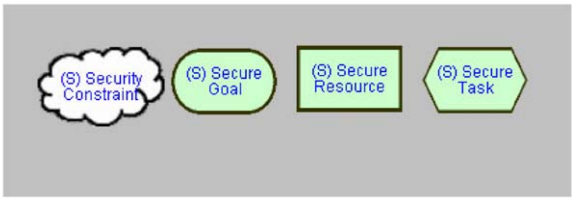

Fig. 2. Graphical representation of security-related concepts.

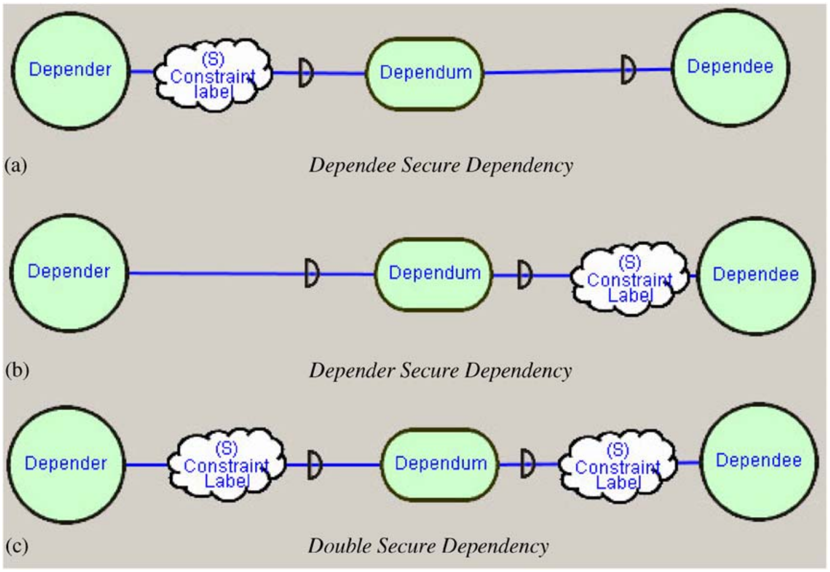

Fig. 3. Secure dependencies.

must agree for the fulfilment of the security constraint in order for the secure dependency to be valid. That means the depender expects from the dependee to satisfy the security constraint (s) and also that the dependee will make an effort to deliver the dependum by satisfying the security constraint (s).

Secure Tropos differentiates three different types of secure dependency:

- A `Dependee Secure Dependency`, in which the depender depends on the dependee and the dependee introduces security constraint (s) for the dependency.
- A `Depender Secure Dependency`, in which the depender depends on the dependee and the depender introduces security constraint (s) for the dependency.
- A `Double Secure Dependency`, in which the depender depends on the dependee and both the depender and the dependee introduce security constraints for the dependency.

To better understand the concept of secure dependency, consider a dependee secure dependency where a `doctor` (depender) depends on a `patient` (dependee) to obtain `health information` (dependum), and that the `patient` imposes to the `doctor` the security constraint 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, the `doctor` must satisfy the `share health information only if consent is obtained` security constraint introduced by the `patient` in order to help in the achievement of the `Obtain Health information` secure dependency. The different types of secure dependency are graphically illustrated in Fig. 3.

4. 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 [14], an integrated health and social care information system for the effective care of older people. Security is a very important factor in the development of the electronic single assessment process, since security of personal health information is considered a priority by many health care unions in different countries of the world including England. This is due to the fact that in cases where patients (in the case of the eSAP older people) do not trust the security of the system, they will refuse to provide complete information about their health and social care needs, and this could lead to many problems such as wrong assessment of needs, which could lead to wrong care plans.

Therefore privacy of health and social care information, such as the health and social care plans used in the electronic single assessment process, is the number one security concern in such a system. According to Good Medical Practice, patients have a right to expect that you will not pass on any personal information, which you learn in the course of your professional duties unless they agree. In addition to that, the English government and health and social care unions have agreed that electronic health care records should be at least as well protected as the paper ones.

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

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 [1] 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; and (6) availability, the information must be available whenever a person included in the access control list requires any information.

5. The development process

5.1. Early requirements

During the early requirements stage, the goals, dependencies and the security constraints between the stakeholders (actors) are modelled with the aid of an actors' diagram [8].

In such a diagram, `actors` (graphically represented as circles) are modelled together with their `goals` (represented as ovals), `soft-goals` (represented as bubbles), their `dependencies` (represented as links between the actors indicating the dependum[3]) and their `security constraints` (modelled as clouds).

In the actor's diagram, imposed security constraints are expressed in high-level statements.

For the `eSAP` case study, we consider the following actors (see Fig. 4)

- `Professional`: the health and/or social care professional;
- `Older Person`: the older person (patient) that wishes to receive appropriate health and social care;

[3]For a reminder of the graphical representation of the Tropos concepts please refer to Fig. 1.

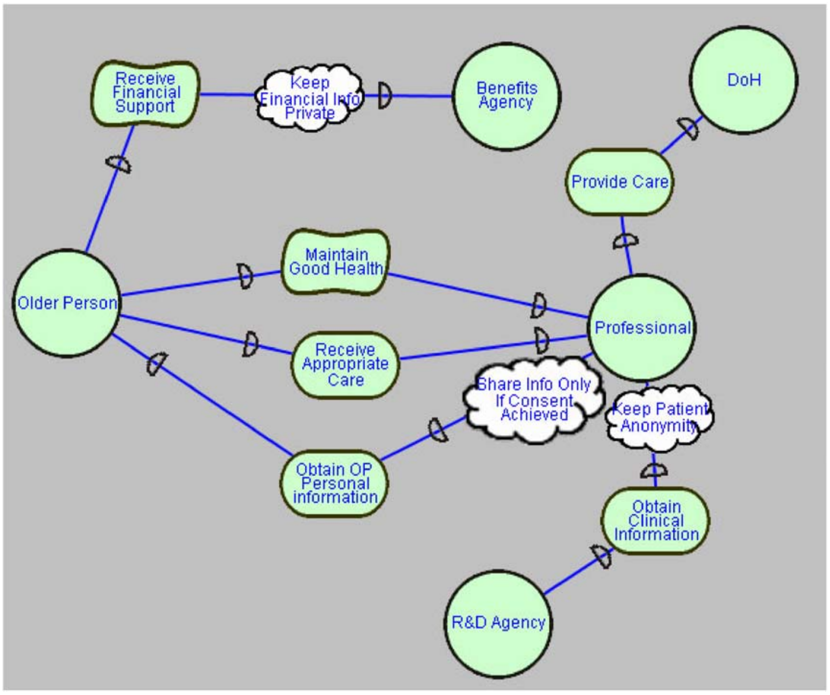
Receive Financial Support
Keep Financial Info Private
Benefits Agency
DoH
Provide Care
Maintain Good Health
Older Person
Professional
Receive Appropriate Care
Share Info Only If Consent Achieved
Keep Patient Anonymity
Obtain OP Personal information
Obtain Clinical Information
R&D Agency

Fig. 4. Actors diagram.

- 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.

The main goal for the Older Person Actor is to Maintain Good Health[4] 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 OP (Older Person) 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. Therefore, 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, since the paper is focused on the

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

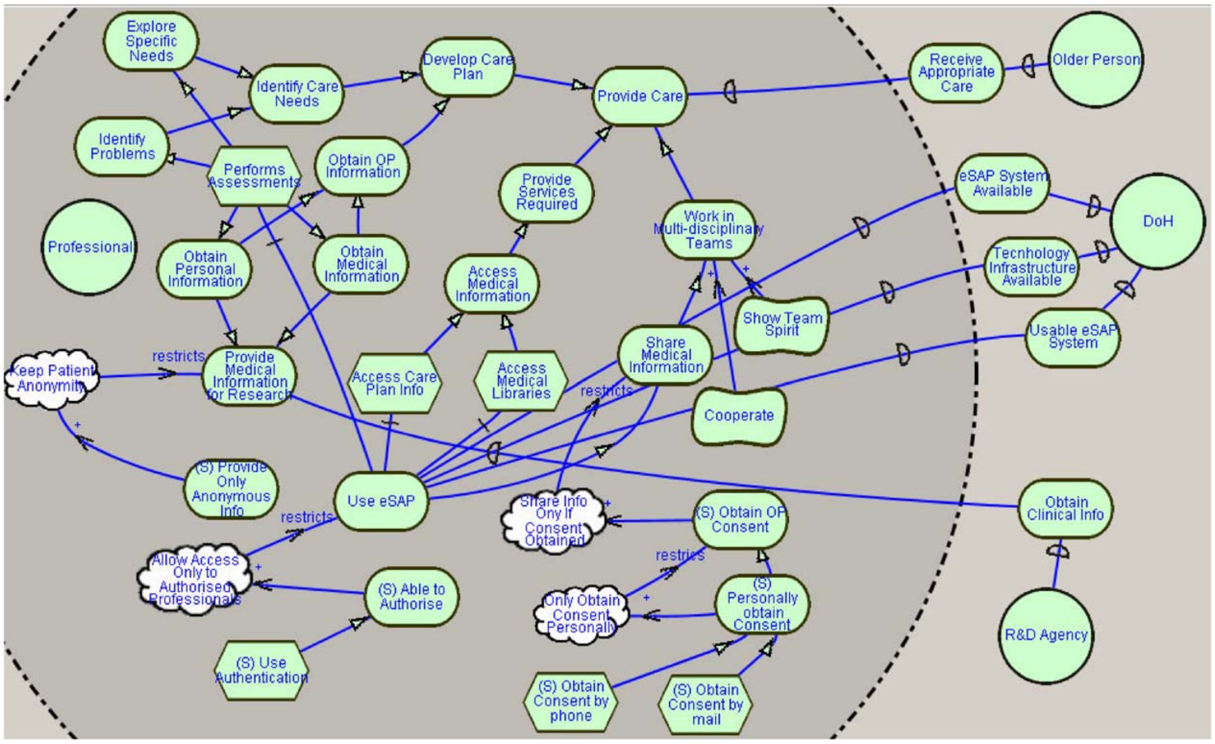
Explore Specific Needs
Identify Care Needs
Develop Care Plan
Provide Care
Receive Appropriate Care
Older Person
Identify Problems
Performs Assessments
Obtain OP Information
Provide Services Required
Work in Multi-disciplinary Teams
eSAP System Available
DoH
Professional
Obtain Personal Information
Obtain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

Fig. 5. Partial analysis of the professional actor.

secure Tropos and not in Tropos in general, we focus only in the analysis of the security constraints, and not in the goal or task analysis of each individual actor.

To model this, goal diagrams are used. In a goal diagram, each actor is represented as a dashed-line balloon within which the actor's goals and dependencies are analysed. The nodes of the diagram represent goals, soft-goals, and/or tasks whereas the links identify the different kinds of relationships between those nodes. Moreover, these links can be connected with external dependencies (identified in the actor diagram) when the reasoning of the analysis goes beyond the actor's boundary [11].

The analysis of the security constraints starts by identifying which goals of the actor they restrict. This case is known as security constraint assignment. The assignment of a constraint to a goal is indicated using a constraint link (a link that has the "restricts" tag). In addition, different alternatives can be considered for achieving the goals and the security goals of the stakeholders. For example, during the early requirements analysis (shown in Fig. 4), the Professional actor has been imposed two security constraints (Share Info Only If Consent Achieved and Keep Patient Anonymity). By analysing the Professional actor (as shown in Fig. 5) 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.

5.2. 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.

More specifically, from the security point of view, during the late requirements analysis stage, security constraints are imposed to the system-to-be. These constraints are further analysed and security goals and entities necessary for the system to guarantee the security constraints are identified.

In the presented 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.

Therefore, the eSAP system has been introduced as another actor that receives the responsibility for the fulfilment of some of the goals identified during the early requirements analysis for the actors of the system. In other words, some goals that the actors of the system cannot fulfil or are better fulfilled by the eSAP system are delegated to the eSAP System.

To satisfy all the delegated dependencies, the main goal of the eSAP system has been identified as to Automate Care. By performing a means-end analysis, presented in Fig. 6, it was identified that for the eSAP System to fulfil the Automate Care goal, the following sub-goals must be accomplished: Assist with Assessment Procedures, Provide Older Person Information, Manage Care Plans and Schedule Meetings.

Each of those sub-goals can be furthered analysed employing means-end analysis. For example, the Manage Care Plans goal can be accomplished with the fulfilment of the Generate Care Plan, Manage Care Plan Updates, Provide Care Plan Information, Manage Referrals and Identify Care Assistants sub-goals.

From the security point of view, and taking into consideration the security policy (presented in the previous section) there are three main security constraints imposed, by the desired security features of the system Privacy, Integrity and Availability, to the eSAP's main goal.

These are Keep System Data Private, Keep Integrity of the Data and Maintain Data Availability. In addition, the eSAP system must satisfy the Share Information Only if Consent Obtained security constraint imposed to the eSAP by the secure dependencies delegated by the other actors.

Each of these secure constraints can be satisfied with the aid of one or more secure goals. For example, the Keep System Data Private security constraint can be fulfilled by blocking access to the system, by allowing access only from a central computer, or by ensuring system privacy. However, the two first contribute negatively to the usability of the system, i.e. the system will be secure but it will not be used. On the other hand, the Ensure System Privacy secure goal is considered the best solution since it provides security to the system and it doesn't affect (dramatically) its usability.

Thus, for the eSAP to satisfy its security constraints the following secure goals have been identified as shown in Fig. 6 Ensure System Privacy, Ensure Data Integrity, Ensure Data Availability and Ensure Consent has been Obtained. These can be furthered analysed. For example, the Ensure System Privacy goal is further analysed into

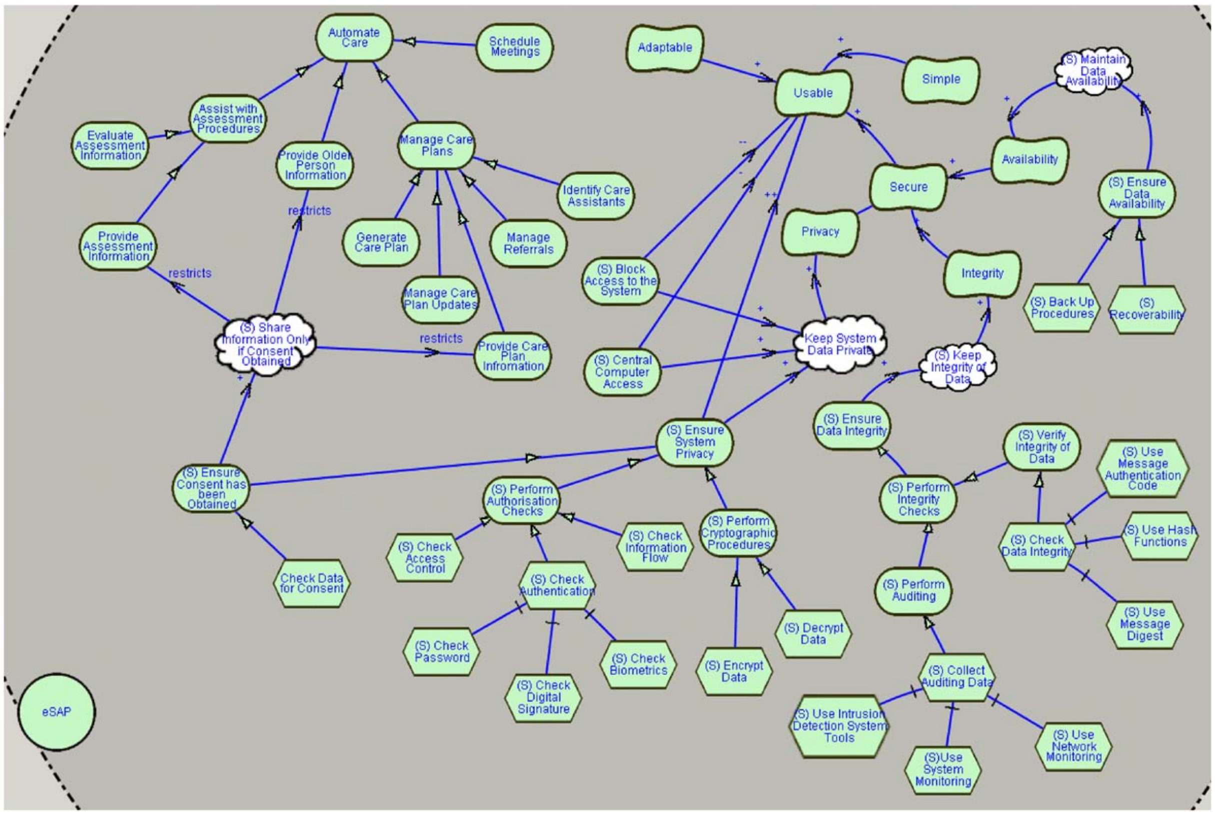
Automate Care
Schedule Meetings
Adaptable
Usable
Simple
(S) Maintain Data Availability
Evaluate Assessment Information
Assist with Assessment Procedures
Provide Older Person Information
Manage Care Plans
Identify Care Assistants
Availability
Secure
(S) Ensure Data Availability
restricts
Provide Assessment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

Fig. 6. eSAP analysis.

the Perform Authorisation Checks and Perform Cryptographic Procedures secure goals. Both of those goals must be fulfilled for the Ensure System Privacy goal to be satisfied.

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 [15]. 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.

5.3. 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[5] 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. Nevertheless, a technique is required to allow developers to reason about alternative

[5]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" [16, p. 13].

design solutions according to the security requirements of their system.

For this reason, we have developed an analysis technique to enable developers to select among alternative architectural styles using as criteria the non-functional requirements of the multiagent system under development.

Our analysis process is based on an independent probabilistic model, which uses the measure of `satisfiability` proposed by Giorgini et al. [17]. 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.

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

The 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.

In the `eSAP` system, the security of the system is one of the most important factors and it is the criterion that will guide the selection process, in this example, for the appropriate architectural style. As derived from the analysis of the eSAP, security is decomposed to privacy, integrity and availability.

We consider two architectural styles for our analysis, 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 network-based applications [18], 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 [18]. Client/server architectural style is also referred to by the mechanisms used for the connector implementation such as remote procedure call (RPC) [18]. 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 [18].

As shown in Fig. 7, 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.

As concluded from our analysis (illustrated in Fig. 7), the client/server style satisfies more the

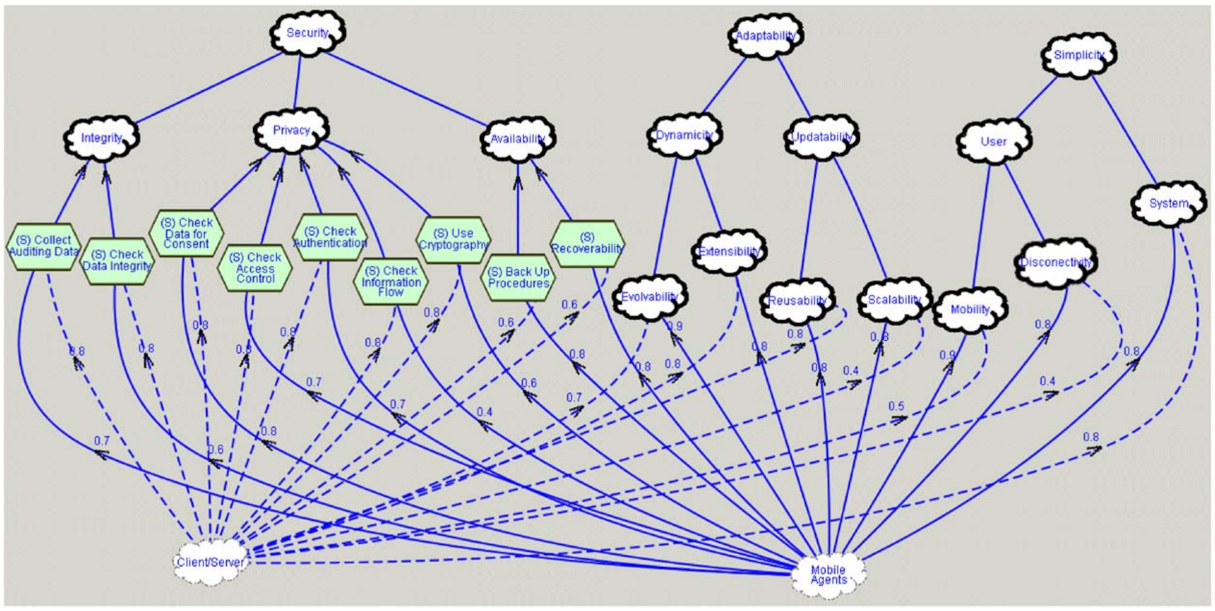

Fig. 7. Selecting amongst different architectural styles.

privacy requirements of the system than the mobile agents style. This is mainly because mobility is involved in the mobile agents style. Therefore, although protection of a server from mobile agents, or generally mobile code, is an evolution of security mechanisms applied in other architectural styles, such as client/server; the mechanisms focused on the protection of the mobile agents from the server cannot, so far, prevent malicious behaviour from occurring but may be able to detect it [19]. Consider for example, the `Check Information Flow` secure task of the `eSAP`. The information flow property is more easily damaged by employing mobile agents (weight 0.4) since possible platforms that a mobile agent could visit might expose sensitive information from the agent [19,20]. In the case of the client/server style (weight 0.8) sensitive information is stored in the server and existing well-proven security measures could be taken to satisfy the information flow attribute.

On the other hand, the mobile agents style satisfies more, than the client/server style, the availability requirements of the system. Consider for example the `recoverability` secure task of the `eSAP`. The mobile agents style contributes with a weight of 0.8. This is due to the fact that mobile agents adapt dynamically. Mobile agents can react to changes in their environment and maintain an optimal configuration for solving a particular problem [21].

From the integrity point of view, the client/server style contributes better than the mobile agents style. In the mobile agents style mobility is involved and therefore checking the integrity of the data becomes a more difficult task. This is because mobile agents cannot prevent a malicious agent platform from tampering with their code, state or data, but they can only take measures to detect this tampering [19,20]. Moreover, in the mobile agent style, the integrity of both the local and remote agent platforms must be checked.

From the above, it can be concluded that the client/server styles contributes more towards the privacy and integrity of the `eSAP`, whereas the mobile agents style contributes more towards the availability.

It is worth mentioning that the weights of the contribution links reported in Fig. 7, of each architectural style to the different non-functional requirements of the system, have been assigned after reviewing different studies [16,22], evaluations [18], and comparisons [23] involving the architectural styles. We must also note that Fig. 7 represents a partial illustration of the comparison process. For example, 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.

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. [17].

In our example, `privacy` and `integrity` are more important (in the case of the eSAP) than `availability` (most of the times, not real-time information is needed). As a result, the client/server style has been chosen as the architectural style of the system.

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. [7], an interesting decision that comes up during the architectural design is whether fulfilment 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.

In the presented example, the `eSAP` actor is decomposed, as shown in Fig. 8, to internal actors and the responsibility for the fulfilment of the eSAP's goals is delegated to these actors.

For instance, the `Evaluate Assessment Information` goal is delegated to the `Assessment Evaluator`, whereas the `Provide Assessment Information` goal is delegated to the `Assessment Broker`. In addition, the `Older Person Broker` and the `Consent Manager` actors have been introduced to the `eSAP` system to fulfil the responsibility (identified during the late requirements analysis—see Fig. 6) of the `eSAP` system to satisfy the secure dependency *Obtain* `Older Person Information` together with the `Share Information Only if Consent Obtained` security constraint.

Moreover, the `eSAP` delegates responsibility (see Fig. 8) for the fulfilment of the `Perform Authorisation Checks` security goal to three new actors, the `eSAP Guard` (delegated the `Check Information Flow` secure task), the `Authenticator` (delegated the `Check Authentication` secure task), and the `Access Controller` (delegated the `Check Access Control` secure task) as shown in Fig. 8.

In addition, the Tropos methodology introduces extended actor diagrams, in which the new actors and their dependencies with the other actors are presented. Consider for instance, the extended diagram with respect to the `Assessment Evaluator` actor, as depicted in Fig. 9. The `Assessment Evaluator` has been delegated the responsibility to satisfy the goal `Evaluate Assessment Information`. To fulfil this goal, the `Assessment Evaluator` depends on two internal actors, the `Assessment Analyser` and the `Evaluation Synthesiser`. The first is responsible for obtaining the `Assessment Information` secure resource, identify the problems of the `Older Person` according to the `Assessment Information` and provide the `Problems to the Evaluation Synthesiser`. The latter is responsible for obtaining the `Evaluation Request`, and the `Problems` and providing the `Assessment Evaluation` secure resource to the actor requesting the information (in the presented analysis to the `Social Worker`) after considering the `Problems`, the `Available`

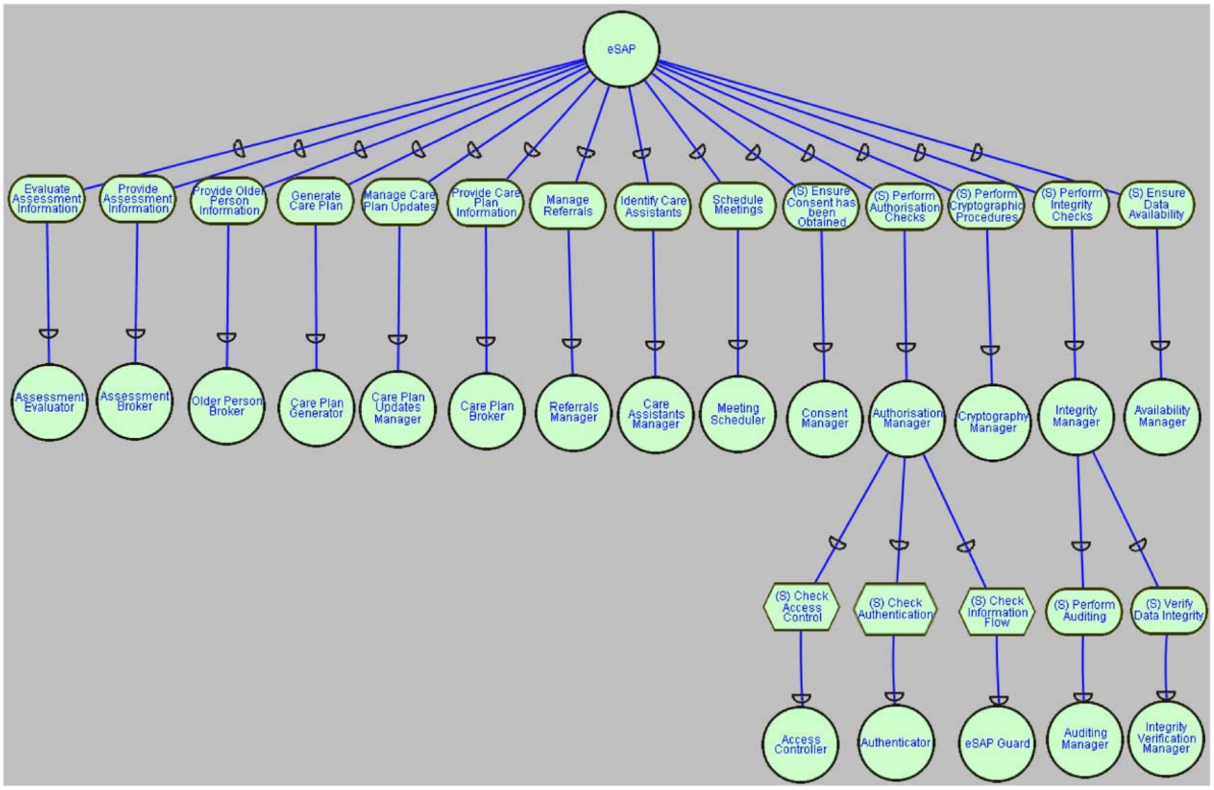

Fig. 8. eSAP decomposition.

Professionals, the Required Skills and the Proposed Actions resources.

Each of those actors can be furthered decomposed to model more precisely the actor's goals and how the actor will achieve these goals. For instance, the Assessment Synthesiser subactor of the Assessment Evaluator, can be furthered analysed as shown in Fig. 10. The main goal of the Assessment Synthesiser is to Provide an Assessment Evaluation.

To satisfy this goal, the Assessment Evaluator has to synthesise any information related to the assessment. To achieve this, the Assessment Evaluator must obtain the problems associated with the assessment, the proposed actions required to deal with the identified problems, the skills required to carry out these actions, and the available professionals that demonstrate the required skills. However, the Assessment Evaluator cannot achieve these tasks without help. Therefore, the Assessment Evaluator depends on the Assessment Analyser, the Actions Manager, the Skills Manager and the Professional Database Manager respectively to satisfy the above mentioned tasks.

The last step of the architectural design aims 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 fulfilment of a goal, given certain world conditions and in presence of a specific event [8]. For example, the Assessment Evaluator should have capabilities such as get assessment information, get proposed actions, and Provide assessment evaluation. However, the process of identifying capabilities for each actor has been extensively described in the literature [6,8,25] and thus it is not described here.

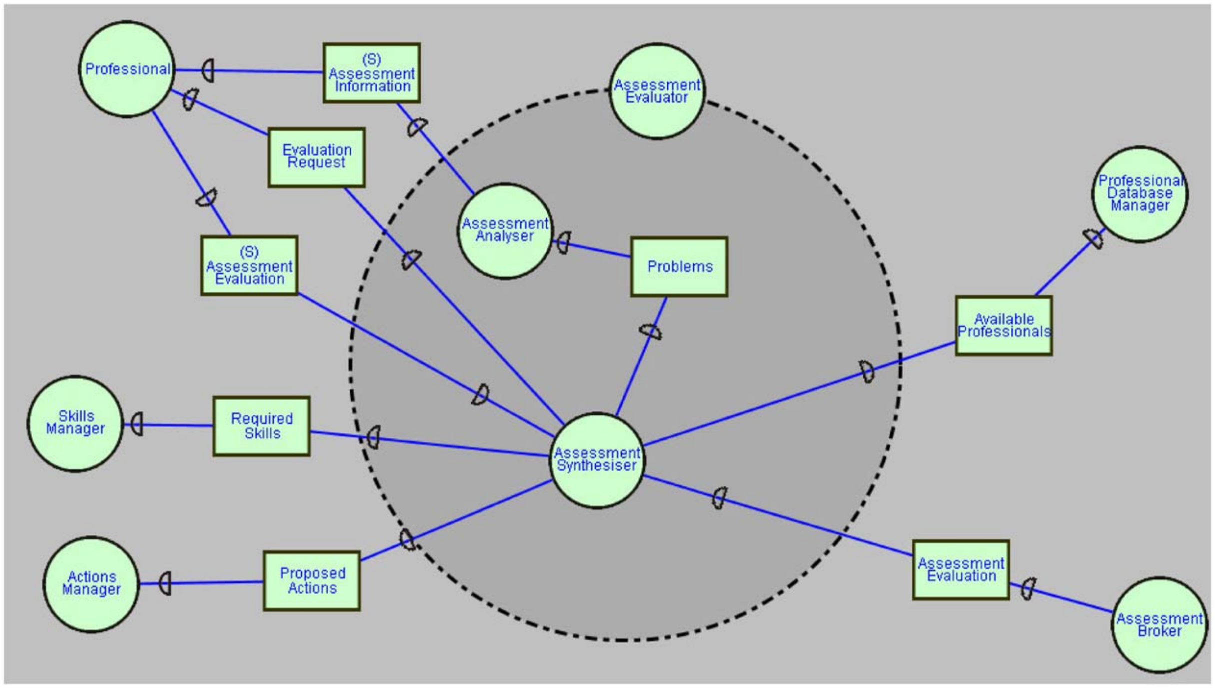

Fig. 9. An example of an extended diagram.

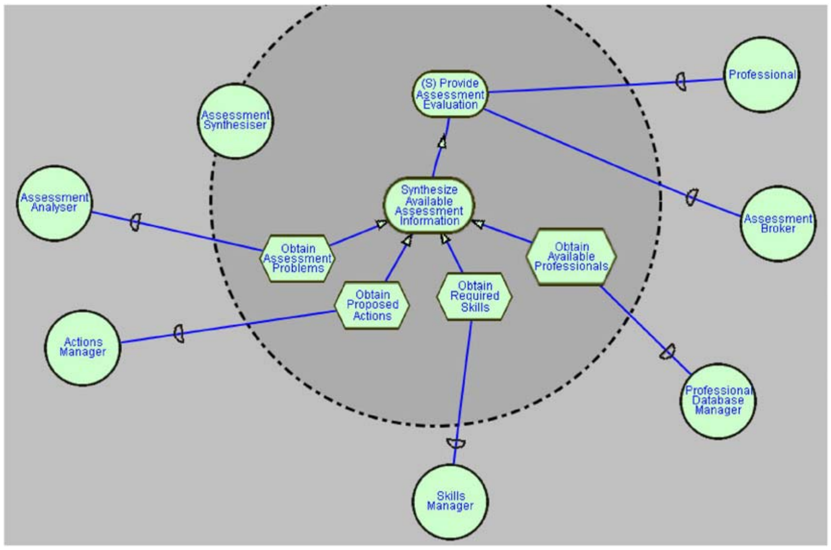
Assessment Synthesiser
(S) Provide Assessment Evaluation
Professional
Assessment Analyser
Synthesize Available Assessment Information
Assessment Broker
Obtain Assessment Problems
Obtain Available Professionals
Obtain Proposed Actions
Obtain Required Skills
Actions Manager
Professional Database Manager
Skills Manager

Fig. 10. Analysis of the assessment evaluator.

From the security point of view, secure capabilities are introduced to the actors to guarantee the satisfaction of the security constraints. A secure capability represents the ability of an actor/agent to achieve a secure goal, carry out a secure task and/or deliver a secure resource.

For example, as identified in the early requirements analysis, for the eSAP system to satisfy the

Ensure System Privacy secure goal, only encrypted data transfers across the network should be allowed. Therefore, the Assessment Information resource sent from the Professional to the Assessment Analyser (as illustrated in Fig. 9) must be encrypted. Because of this, the Professional actor should be provided with capabilities to encrypt and decrypt data. Later in the detailed design, each agent's capabilities are further specified and then coded during the implementation phase.

5.4. 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. In particular, 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 adapts a subset of the AUML diagrams proposed in [12]. These are:

- Capability Diagrams. We use AUML activity diagrams to model a (secure) capability or a set of capabilities for a specific actor. In each capability diagram, the starting state is represented by external events, activity nodes model plans, transition arcs model events, and beliefs are modelled as objects. For instance, in our case study, the Receive Assessment Evaluation capability (see Fig. 9) of the Professional actor is illustrated in Fig. 11. The Professional receives (external event – EE) the Assessment Evaluation from the eSAP System. She/he then evaluates the Assessment Evaluation and either accepts it or rejects it.
- Plan Diagrams. Plan Diagrams are used to further specify each plan node of a capability diagram. Fig. 12 illustrates the plan diagram for the Evaluate Assessment Evaluation plan belonging to the capability depicted in the diagram of Fig. 11. The plan is activated with the receipt of the Assessment Evaluation from the Professional and it ends by

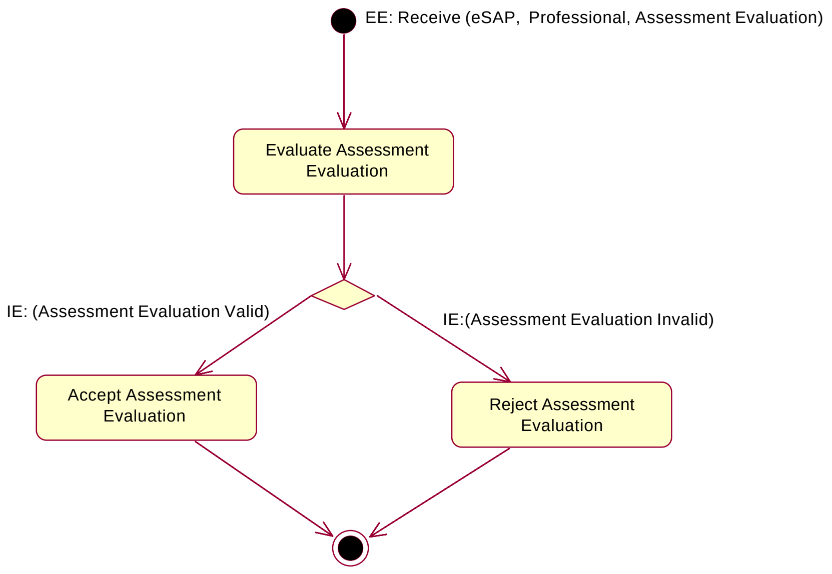
EE: Receive (eSAP, Professional, Assessment Evaluation)
Evaluate Assessment Evaluation
IE: (Assessment Evaluation Valid)
IE:(Assessment Evaluation Invalid)
Accept Assessment Evaluation
Reject Assessment Evaluation

Fig. 11. Capability diagram for the receive assessment evaluation capability.

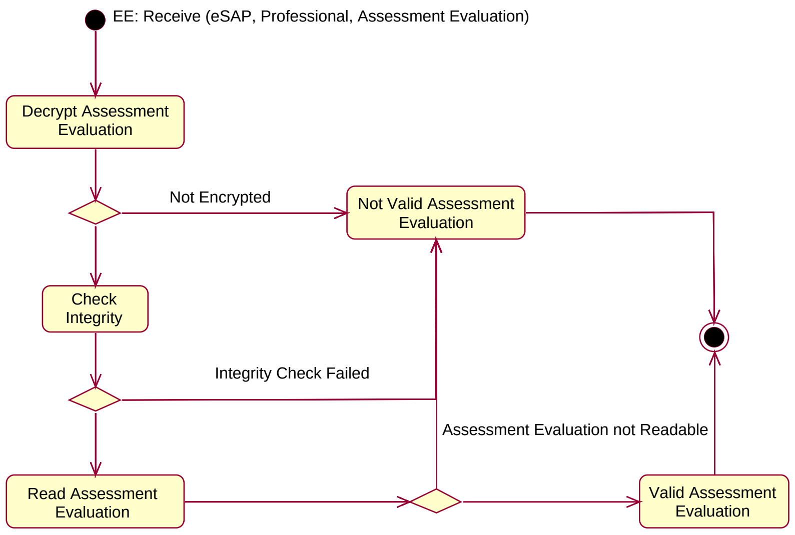
EE: Receive (eSAP, Professional, Assessment Evaluation)
Decrypt Assessment Evaluation
Not Encrypted
Not Valid Assessment Evaluation
Check Integrity
Integrity Check Failed
Assessment Evaluation not Readable
Read Assessment Evaluation
Valid Assessment Evaluation

Fig. 12. Plan diagram example.

deciding if the Assessment Evaluation is valid or invalid (In addition the plan can be terminated if Assessment Evaluation is not readable). The integrity of the Assessment Evaluation is checked. If the check is successful the Assessment Evaluation is received as Valid, else the Assessment Evaluation is considered Invalid from the Professional.

- Agent Interaction Diagrams. We apply in our case sequence diagrams modelling agent Interaction Protocols as proposed by [16]. An example of an Agent Interaction Diagram is shown in Fig. 13. This diagram illustrates interactions (illustrated as arrow-lines) between the Professional, the eSAP Guard, the Cryptography Manager and the Authenticator agents (graphically illustrated as rectangles at the top of the diagram). The Professional sends a system access request to the eSAP Guard. The eSAP Guard decrypts the incoming request (with the aid of the Cryptography Manager) and checks if the Professional has authentication clearance (with the aid of the Authenticator).

However, it would be useful to denote under what constraints authentication clearance and system access are granted. For our example, the Authenticator provides authentication clearance if the details of the Professional are valid, and the eSAP Guard provides access to the eSAP system if authentication clearance is provided. To indicate these constraints, 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 Fig. 13.

6. Related work

Literature provides only few approaches in considering security requirements as an integral part of the whole software development process.

Chung applies a process-oriented approach [26] to represent security requirements as potentially conflicting or harmonious goals and using them during the development of software systems. The proposed framework, which is called the NFR (Non-Functional Requirements) framework, represents and uses security requirements as a class of non-functional requirements and it allows developers to consider design decisions and relate

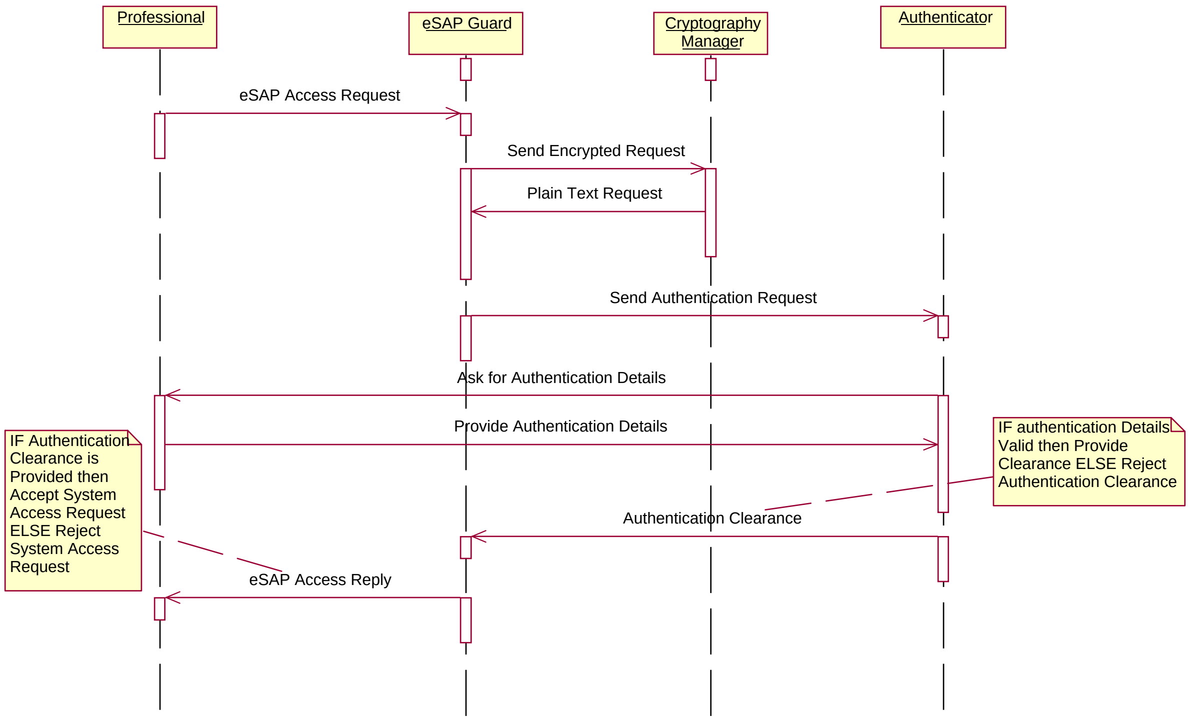
Professional
eSAP Guard
Cryptography Manager
Authenticator
eSAP Access Request
Send Encrypted Request
Plain Text Request
Send Authentication Request
Ask for Authentication Details
Provide Authentication Details
IF authentication Details Valid then Provide Clearance ELSE Reject Authentication Clearance
IF Authentication Clearance is Provided then Accept System Access Request ELSE Reject System Access Request
Authentication Clearance
eSAP Access Reply

Fig. 13. An example of an agent interaction diagram including security rule notation.

these decisions to the represented non-functional requirements.

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. The proposed approach consists of four main steps. During the first step, the general security objectives of the business process are defined, whereas during the second step the security objectives of all the constructs, such as actors and artefacts, are examined. The third step examines whether these specifications are consistent and during the fourth step a list of necessary security measures for each process component is generated.

In addition, Jurgens proposes UMLsec [28], an extension of the Unified Modelling Language (UML), to include modelling of security related features, such as confidentiality and access control. In his work, Jurgens uses four different UML diagrams; class diagrams to ensure that exchange of data obeys security levels, state-chart diagrams to prevent indirect information flow from high to low values within an object, interaction diagrams to ensure correctness of security critical interactions between objects and deployment diagrams to ensure that security requirements on communication are met by the physical layer.

Lodderstedt et al. [29] also extend UML to model security. In their work, they present a security modelling language called SecureUML [29]. They describe how UML can be used to specify information related to access control in the overall design of an application and how this information can be used to automatically generate complete access control infrastructures.

McDermott and Fox adapt use cases [5] to capture and analyse security requirements, and they call the adaption an abuse case model [5]. 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 [5].

Sindre and Opdahl [30] define the concept of a misuse case, the inverse of a use case, which describes a function that the system should not allow. They also define the concept of a mis-actor as someone who intentionally or accidentally initiates a misuse case and whom the system should not support in doing so. In their approach security is considered by analysing security related misuse cases.

The concept of obstacle is used in the KAOS framework [31] to capture undesired properties of the system, and define and relate security requirements to other system requirements. In this work, two set of techniques, based on a temporal logic formalisation, are employed to reason about obstacles to the satisfaction of goals, requirements, and assumptions elaborated in the requirements engineering process.

These (above-mentioned) approaches provide a first step towards the integration of security and software engineering and have been found helpful in modelling security requirements. However, they only guide the way security can be handled within a certain stage of the software development process. For example, McDermott and Fox's approach is used only during the requirements analysis, whereas Jurgen's analysis take place in a fairly low level and it is suited to a more operational analysis. In other words, Jurgen's approach is only applicable during the design stage.

Differently than them, this paper proposes an approach that covers the whole development process using the same concepts and notations. As mentioned in the Introduction, considering security issues throughout the development process helps to limit the cases of conflict, by identifying them very early in the system development, and find ways to overcome them.

Moreover, some of the above mentioned approaches only deal with specific security issues. For example, UMLSec is focused more in access control policies and how these policies can be integrated into a model-driven software development process. Although such an analysis is important, it is very specific and it is applicable only on the design stage of the modelling process. In contrast, the approach presented in this paper considers the whole range of security issues, from access control to authentication and integrity.

In addition to the above approaches, existing formal methods [9,32] support the verification of a security protocol, which has already been specified [33]. However, such approaches are only applicable by security specialists and cannot be easily applied by software developers. On the other hand, the approach presented in this paper uses concepts and notations derived mainly from the requirements engineering area and as a result can be applied by software developers with minimum knowledge of security engineering.

It is worth mentioning that the technique presented for the evaluation of the different architectural styles, is similar to the evaluation process for organisational styles proposed by Kolp et al. [34]. The main difference is that Kolp's process is based on a qualitative reasoning, while the technique proposed by this research is based on an independent probabilistic model, which uses the measure of *satisfiability* proposed by Giorgini et al. [17].

7. Conclusions and future work

Although Security is an important issue in the development of information 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, in Tropos, security process 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. The presented approach is characterised 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.

However, this work is by no means complete. Future work includes providing a process to verify the security of the developed information systems during the design stage and also applying our process to different case studies to refine it. We also aim to integrate our extensions to the Formal Tropos [35] 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] R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, Wiley Computer Publishing, New York, 2001.

[2] W. Stallings, Cryptography and Network Security: Principles and Practice, second Ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.

[3] T. Tryfonas, E. Kiountouzis, A. Poulymenakou, Embedding security practices in contemporary information systems development approaches, Inform. Manage. Comput. Secur. 9 (4) (2001) 183–197.

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

[5] J. McDermott, C. Fox, Using abuse care models for security requirements analysis, Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.

[6] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, TROPOS: an agent-oriented software development methodology, J. Autonomous Agents Multi-Agent Systems 8 (3) (2004) 203–236.

[7] J. Castro, M. Kolp, J. Mylopoulos, A Requirements-driven development methodology, Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), Interlaken, Switzerland, June 2001.

[8] A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Towards an agent oriented approach to software engineering. in: A. Omicini, M. Viroli (Eds.), WOA 2001—Dagli oggetti agli agenti: tendenze evolutive dei sistemi software, Modena, Italy, September 2001.

[9] M. Burrows, M. Abadi, R. Needham, A logic of authentication, Proc. Roy. Soc. Lond. A 426 (1989) 233–271.

[10] P. Bresciani, P. Giorgini, The Tropos analysis process as graph transformation system, Proceedings of the Workshop on Agent-oriented methodologies, at OOPSLA 2002, Seattle, WA, USA, Nov, 2002.

[11] E. Yu, Modelling Strategic Relationships for Process Reengineering, Ph.D. Thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[12] B. Bauer, J. Müller, J. Odell, Agent UML: a formalism for specifying multiagent interaction. in: P. Ciancarini, M. Wooldridge (Eds.), Agent-Oriented Software Engineering, Springer, Berlin, 2001, pp. 91–103.

[13] H. Mouratidis, P. Giorgini, G. Manson, I. Philp, A natural extension of Tropos methodology for modelling security, Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle-USA, November 2002.

[14] H. Mouratidis, I. Philp, G. Manson, Analysis and design of eSAP: an integrated health and social care information system, Proceedings of the Seventh International Symposium on Health Information Managements Research (ISHIMR2002), Sheffield, June 2002.

[15] M. Garzetti, P. Giorgini, J. Mylopoulos, F. Sannicolo, Applying tropos methodology to a real case study: complexity and criticality analysis, Proceedings of the Second Italian workshop on "WOA 2002 dagli oggetti agli agenti dall'informazione alla conoscenza", Milano, 18–19 November 2002.

[16] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, SEI Series in Software Engineering, Addison-Wesley, Reading, MA, 1998.

[17] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani, Reasoning with Goal Models, Proceedings of the 21st

International Conference on Conceptual Modeling (ER2002), Tampere, Finland, October 2002.

[18] R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Doctoral Dissertation, University of California, Irvine, 2000.

[19] W. Jansen, Countermeasures for Mobile Agent Security, Computer Communications, Special Issue on Advanced Security Techniques for Network Protection, Elsevier Science BV, November 2000.

[20] W. Jansen, T. Karygiannis, Mobile Agent Security, National Institute of Standards and Technology, Special Publication 800-19, August 1999.

[21] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Commun. ACM 42(3) (March 1999).

[22] J. Bosch, Design and Use of Software Architectures: adopting and Evolving a Product-line Approach, ACM Press, Addison-Wesley, Reading, MA, 2000.

[23] 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 Concurr. Comput.: Pract. Exper. 13 (2001) 71–94.

[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 Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto-Ontario, May 2002.

[26] 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.

[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, L'Aquila, Italy, January 2002.

[28] J. Jürjens, Towards Secure Systems Development with UMLsec, Fundamental Approaches to Software Engineering (FASE/ETAPS) 2001, International Conference, Genoa, 4–6 April 2001.

[29] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-Based modelling language for model-driven security, Proceedings of the Fifth International Conference on the Unified Modeling Language, 2002.

[30] G. Sindre, A.L. Opdahl, Eliciting Security Requirements by Misuse Cases, Proceedings of TOOLS Pacific 2000, November 2000.

[31] A. Dardenne, A. Van Lamsweerde, S. Fickas, Goal-directed requirements acquisition. Science of computer programming, Special issue on Sixth International Workshop of Software Specification and Design, 1991.

[32] P. Ryan, S. Schneider, Analysis and Design of Security Protocols, Pearson Professional Education, 2000.

[33] C. Meadows, A model of computation for the NRL protocol analyser, Proceedings of the 1994 Computer Security Foundations Workshop, 1994.

[34] M. Kolp, P. Giorgini, J. Mylopoulos, A goal-based organizational perspective on multi-agent architectures, Proceedings of the Eight International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), Seattle-USA, August 2001.

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