MODELLING SECURE MULTIAGENT SYSTEMS

Haralambos Mouratidis[1]
University of Sheffield
Computer Science Department
211 Portobello Street
S11 4DP, Sheffield, UK
H.Mouratidis@dcs.shef.ac.uk

Paolo Giorgini
University of Trento
Department of Information and Communication Technology
Via Sommarive, 14 38050 Povo, Italy
Paolo.Giorgini@dit.unitn.it

Gordon Manson
University of Sheffield
Computer Science Department
211 Portobello Street
S11 4DP, Sheffield, UK
G.Manson@dcs.shef.ac.uk

ABSTRACT
Security plays an important role in the development of multiagent systems. However, a careful analysis of software development processes shows that the definition of security requirements is, usually, considered after the design of the system. This is, mainly, due to the fact that agent oriented software engineering methodologies have not integrated security concerns throughout their developing stages. The integration of security concerns during the whole range of the development stages could help towards the development of more secure multiagent systems. In this paper we introduce extensions to the Tropos methodology to enable it to model security concerns throughout the whole development process. A description of the new concepts is given along with an explanation of how these concepts are integrated to the current stages of Tropos. An example from the health care sector is used to illustrate the above.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements / Specification – *Methodologies*;

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent Systems*.

General Terms
Design, Security.

Keywords
Agent Oriented Software Engineering, Methodologies, Multiagent Systems, Security.

1. INTRODUCTION
Agent Oriented Software Engineering (AOSE) introduces an alternative approach in analysing and designing complex computerised systems [1,2]. According to AOSE, a complex computerised system is viewed as a multiagent system in which autonomous software agents (subsystem) interact with each other in order to satisfy their design objectives. AOSE provides designers with more flexibility in their analysis and design. The actual design of the system takes place by specifying a multiagent system as a society, similar to a human society, consisting of entities that possess characteristics similar to humans such as mobility, and intelligence with the capability of communicating.

It has been argued, *"if agents are to realise their potential as a software engineering paradigm, then it is necessary to develop software engineering techniques that are specifically tailored to them"* [3]. Thus, agent oriented software engineering have been developed to one of the most active research areas within the agent research community, and many methodologies have been developed. However, it is recognised amongst the agent research community [1,2] the need of developing a complete methodology for analysing and designing multi-agent systems. The main role of such a methodology will be to help in all the phases of the development of a system, and more importantly, to help capture and model the unique characteristics that agent-oriented systems introduce such as flexibility, autonomous problem solving, and the rich interactions between the individual agents.

Security plays an important role in the development of multiagent systems and is considered as one of the main issues to be dealt for agent technology to be widely used outside the research community. As a result, research on security for Multiagent systems is an important area within the agent research community. However, the research has been mainly focused on the solution of individual security problems of the multiagent systems, such as attacks from an agent to another agent, attacks from a platform to an agent, and attacks from an agent to a platform.

Only little work has been carried out to integrate security concerns into an agent-oriented methodology. Developers of agent oriented software engineering methodologies mainly neglect security. The common approach towards the inclusion of security within a system is to identify security requirements after the definition of a system. This approach has provoked the emergence of computer systems afflicted with security vulnerabilities [4]. From the viewpoint of the traditional security paradigm, it should be possible to eliminate such problems through better integration of security and systems engineering.

In this paper, we introduce extensions to the Tropos methodology to accommodate security concerns during the software development stages. Section 2 provides an overview of how security is usually defined and gives some real facts of security failures. Section 3 describes our approach of integrating security and systems engineering in Tropos, by describing the modelling features we introduce in our approach,

[1] Full-time student.

and how these are integrated to the current concepts and stages of the methodology. In Section 4 we illustrate our approach with the aid of an example taken from the health care sector, and in Section 5 we present a discussion of related work. Section 6 presents some concluding remarks and future work.

2. SECURITY

Security is usually defined in terms of the existence of any of the following properties:

- *Confidentiality*: The property of guaranteeing information is only accessible to authorised entities and inaccessible to others
- *Authentication*: The property of proving the identity of an entity
- *Integrity*: The property of assuring that the information remains unmodified from source entity to destination entity
- *Access Control*: The property of identifying the access rights an entity has over system resources
- *Non repudiation*: The property of confirming the involvement of an entity in certain communication
- *Availability*: The property of guaranteeing the accessibility and usability of information and resources to authorised entities

Failure of any of the above-mentioned security properties might lead to many dangers ranging from financial losses to sensitive personal information losses.

According to the Computer Crime and Security Survey, contacted by the Computer Security Institute (CSI) during the first three quarters of 2002, about ninety (90) percent of respondents, mainly large US corporations and US government agencies, detected computer security breaches and eighty (80) percent acknowledged financial losses due to those security breaches.

Security vulnerabilities have also been dramatically increased the last few years. According to the CERT Coordination Center[2] while during 1995, 171 vulnerabilities were reported, this number increased to 3,222 during the first three quarters of 2002. In addition, the last 10 years the number of incidents reported has increased from 773 (in 1992) to 73,359 (the first three quarters of 2002).

All those figures prove that security is not considered as much as it should. A reason for this is that for software developers, security interferes with features and time to market. Thus, currently the definition of security requirements is usually considered after the design of the system. This typically means that security enforcement mechanisms have to be fitted into a pre-existing design therefore leading to serious design challenges that usually translate into software vulnerabilities.

The same is true for Multiagent systems. A main reason for this situation is that developers of agent-oriented methodologies have mainly neglected security. Although, many agent oriented methodologies have been developed during the last few years [2,5], very little evidence have been reported of methodologies that they adequately integrate security and systems engineering within their development process. Agent Oriented software engineering considers security as a non-functional requirement. However, differently than other non-functional requirements, such as performance and reliability, the definition of security is usually considered after the design of the system.

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

The integration of security concerns within the context of a multiagent system will require for the subsystems (agents) of the system to consider the security requirements when specifying their goals and interactions therefore causing the propagation of security requirements to the rest of the subsystems.

3. SECURE TROPOS

Tropos [5] is a software development methodology, for building agent-oriented software systems, that uses concepts such as actors, goals, soft goals, tasks, resources (see figure 1 for graphical representation) and intentional dependencies (see figure 2a for graphical representation) throughout all the phases of the software development [6]. A key feature of Tropos is that it 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.

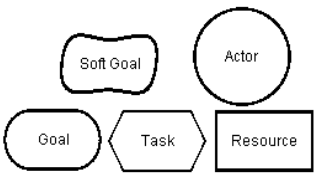

Figure 1: Graphical Representation of Tropos concepts

Tropos supports four development stages, namely *early* and *late requirements*, *architectural design*, and *detailed design*. Early and late requirements analysis represents the initial phases in the Tropos methodology and the final goal is to provide a set of functional and non-functional requirements for the system-to-be. Both phases, early and late, share the same conceptual and methodological approach. This means, that most of the techniques used during the early requirements analysis are used for the late as well. The main difference is that during the early requirements analysis, the developer models the main stakeholders of the system and their dependencies, while in the late requirements analysis the developer models the system itself by introducing it as another actor and model its dependencies

[2] http://www.cert.org/

with the other actors of the organisation. The architectural design stage defines the system's global architecture in terms of actors interconnected through data and control flows (represented as dependencies). In addition, during this stage the actors of the system are mapped into a set of software agents, each characterized by its specific capabilities. During the detailed design stage, the developer specifies, in detail, the agents' goals, beliefs, and capabilities as well as the communication between the agents. For this reason, Tropos employs a set of AUML diagrams [7].

Tropos was not conceived with security in mind and as a result it fails to adequately capture security requirements [8]. The process of integrating security and functional requirements throughout the whole range of the development stages is quite ad hoc, and in addition, the concept of soft goal that Tropos uses to capture security requirements fails to adequately capture some constraints that security requirements often represent [8].

Thus, we have extended the Tropos methodology to enable developers to adequately capture security requirements. The next section describes our extensions and how they have been integrated within the Tropos methodology process.

3.1 The "secure" concepts

Extra concepts were introduced to the methodology to enable it to model security requirements during the software development process. These are:

Security Diagram [9], The *security diagram* represents the connection between security features, threats, protection objectives, and security mechanisms that help towards the satisfaction of the objectives. Security features [9] represent security related features that the system-to-be must have. Protection objectives [9] represent a set of principles that contribute towards the achievement of the security features. Threats [9] on the other hand represent circumstances that have the potential to cause loss or problems that can put in danger the security features of the system, while security mechanisms [9] identify possible protection mechanisms of achieving protection objectives.

Security Constraint [8], which represents constraints that are related to the security of the system. Since, constraints can influence the security of the system either positively or negatively, we further define positive and negative *security constraints* respectively. An example of a positive security constraint could be *Allow Access Only to Personal Information*, while a negative *security constraint* could be *Send Information Plain Text* (not encrypted).

Secure Entities [8], which represent any secure goals/tasks/ resources of the system. Secure goals are introduced to satisfy possible security constraints that exist in the system, while security tasks represent ways of achieving the introduced security goals. A resource that is related to a *secure entity* or a *security constraint* is considered a *secure resource*.

Secure Dependencies [8], represent that a dependency between two actors involves the introduction of a security constraint that must be satisfied either by the depender, the dependee or both for the dependency to be valid. Secure dependencies are categorized into *depender secure dependency*, in which the depender introduces security constraints for the dependency and the dependee must satisfy the security constraints for the dependency to be valid, *dependee Secure Dependency*, in which the dependee introduces security constraints and the depender must satisfy them, and *double Secure Dependency***,** in which both the depender and the dependee introduce security constraints for the dependency that both must satisfy for the dependency to be valid. A graphical representation of the different types of dependencies is illustrated in figure 2.

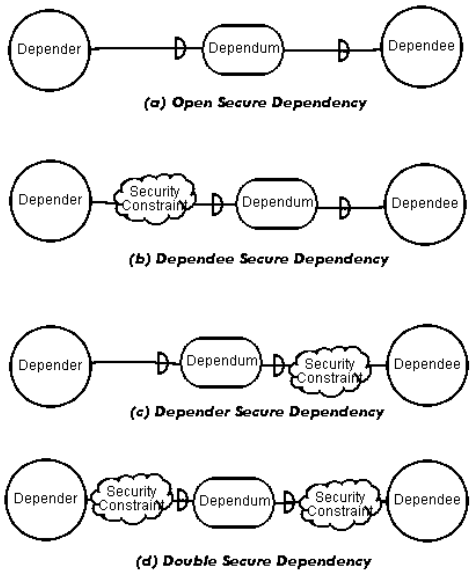

Figure 2: Different types of dependencies

Secure Capabilities, which represent capabilities that the actors (agents) of the system must have in order to help towards the satisfaction of the security requirements of the system.

3.2 Modelling Activities

Many different modeling activities contribute to the capture of security requirements and their integration within the Tropos development process. These are:

Security diagram modelling, which involves the modelling of security needs of the system-to-be, problems related to the security of the system (such as possible threats and vulnerabilities) and possible solutions to the security problems (these solutions can usually be identified in terms of a security policy that the organisation might have).

We have decided to use Tropos concepts to capture the concepts of the security diagram. This helps to better integrate the security diagram within the Tropos methodology and make its concepts easily understandable to developers familiar with Tropos. Thus, we are using the concept of a soft goal to capture security features. Soft goals in Tropos are used to model quality attributes for which there are no clear criteria for satisfaction. In the same sense, security features are not subject to any clear criteria for satisfaction. Protection objectives are represented

using the concept of a goal, because as goals define desired states of the world, protection objectives define desired security states that the system must have. To represent security mechanisms we use the concept of a task, since as a task represents a way of doing something (usually a goal), a security mechanism represents a way of achieving a security objective. The following figure shows how the above-mentioned concepts can be graphically represented.

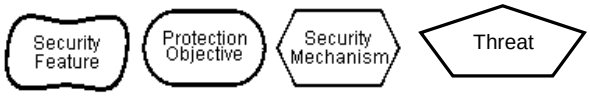

Figure 3: Security diagram concepts

Security constraint modelling involves the modeling of the security constraints imposed to the actors and the system and it allows the designer to perform an analysis by introducing relationships between the constraints or a constraint and its context [9]. Security constraints are imposed by the stakeholders (during the early requirements stage) and by the *security diagram* (during the late requirements stage) and are guaranteed by assigning capabilities (secure capabilities) to the components of the system (i.e. the actors or the agents of it). Stakeholders can impose positive and negative *security constraints*, while the constraints imposed by the *security diagram* are only positive *security constraints*. By imposing *security constraints* to different parts of the system, we are able to identify possible conflicts between security and other (functional and non functional) requirements of the system, identify (stakeholder) constraints that can put in danger the security of the system, and propose possible ways towards a design that will integrate security and systems engineering leading to the development of a more secure system. A security constraint is represented graphically as shown in figure 4.

Secure entities modelling, which is considered as complementary to the security constraints modeling. The analysis of the secure entities follows the same reasoning techniques identified by Tropos for the goal and task analysis [10]. Secure Entities are represented by introducing an S within brackets (S) before the text description as shown in figure 4.

Figure 4: Security constraint and secure entities graphical representations

Secure capability modelling, involves the identification of the capabilities of the subsystems (actors) to guarantee the satisfaction of the security constraints. In order for the system to be able to satisfy its goals and the security constraints, the system's agents has to be provided with capabilities. Secure Capabilities modelling takes place alongside with the capabilities modelling during the architectural design. Secure capabilities can be identified by considering dependencies that involve secure entities in the extended actor diagram. Later, during the detailed design, these capabilities are further specified (in terms of plans, etc) and they are coded during the implementation stage.

3.3 Integration to the current Tropos stages

A difficult but necessary task when extending a methodology is to successfully integrate the extensions to the current stages of the methodology. In our case, we have integrated the extensions as follows:

Early requirements stage: During the early requirements analysis stage the Security Diagram (SD) is constructed and security constraints are imposed to the stakeholders of the system (by other stakeholders). However, the imposed security constraints are expressed in high-level statements, so they are furthered analysed [9] and security entities are introduced to satisfy them.

Late requirements stage: During the late requirements stage security constraints are imposed to the system-to-be (by the security diagram). These constraints are further analysed according to the constraint analysis processes [9].

Architectural design stage: During the architectural design we identify the security constraints and secure entities that the new actors introduce and also during the actor decomposition we identify security sub-constraints and sub-entities. In addition secure capabilities are identified and assigned to each agent of the system.

Detailed design stage: We specify the agent capabilities and interactions taking into account the security aspects as well. In doing so we are using AUML [7] notation in which we introduce the tag of security rules. This is similar to the business rules that UML has for defining constraints on the diagrams.

4. AN EXAMPLE

In this section, we go through the development stages using a case study. This case study is part of a real-life system, called electronic Single Assessment Process (eSAP), under development at the University of Sheffield [11]. The electronic Single Assessment Process (eSAP) system is an agent-based health and social care system for the effective care of older people. To make this example simpler and more understandable, we consider a substantial part of the eSAP system.

4.1 Early Requirements

Early Requirements stage is concerned with the understanding of a problem by studying an existing organisational setting. The output of this phase is an organisational model, which includes relevant actors and their respective dependencies. In our example, we consider the following actors for the eSAP system:

- *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

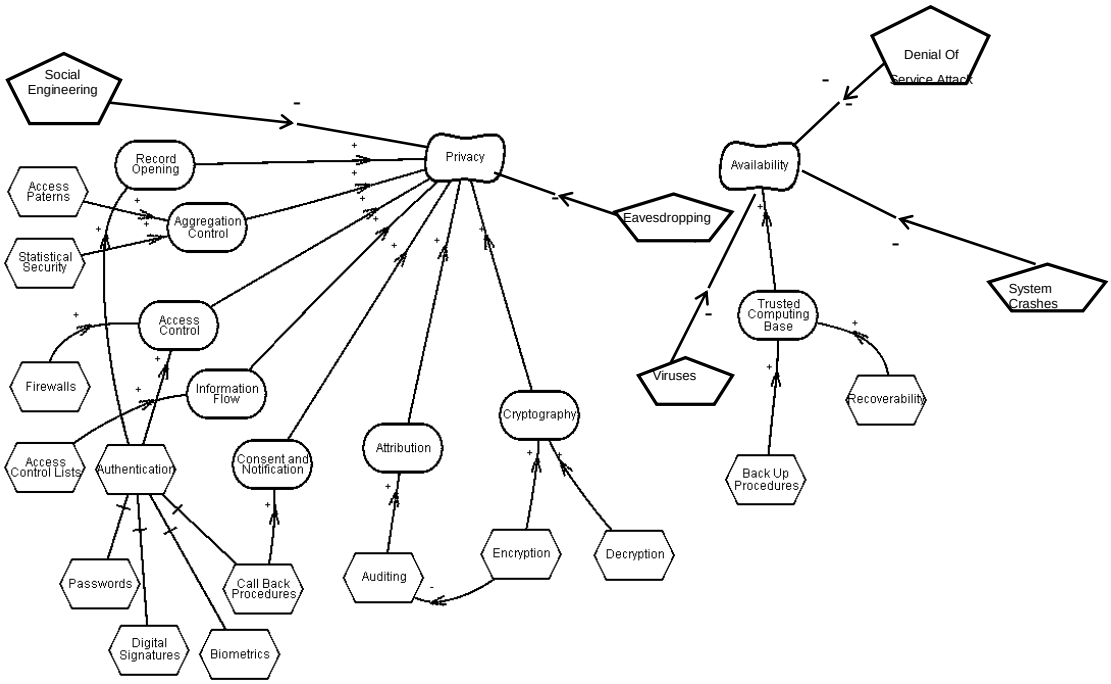

Figure 5: Part of the security diagram of the eSAP system

The first step in the early requirements analysis is the construction of the security diagram. The main security features of the security diagram for the eSAP system are privacy, integrity and availability. However, for our example we consider only two desired security feature, namely privacy and availability. A part of the security diagram, taking into account privacy and availability is shown in figure 5.

The next step involves the modelling of goals, dependencies and security constraints between the stakeholders (actors). For this purpose we are employing actors' diagram. In such a diagram each node represents an actor, and the links between the different actors indicate that one depends on the other to accomplish some goals. In addition, the imposed security constraints (by other stakeholders) indicate that the actors must satisfy them for the dependencies to be valid. For example, the *Older Person* depends on the *Benefits Agency* to *Receive Financial Support*. However, the *Older Person* worries about the privacy of their finances so they impose a constraint to the *Benefits Agency* actor, to keep their financial information private. The *Professional* depends on the *Older Person* to *Obtain (Older Person) OP Information*, however one of the most important and delicate matters for the older person (as with any patient) is the privacy of their personal medical information, and the sharing of it. Thus, most of the times, the Professional is imposed a constraint to share this information if and only if consent is achieved. One of the main goals of the *R&D Agency* is to *Obtain Clinical Information* in order to perform tests and research. To get this information the *R&D Agency* depends on the *Professional*. However, the *Professional* is imposed a constraint (by the Department of Health) to *Keep Patient Anonymity*. Figure 6 illustrates part of the actor diagram of the eSAP system taking into consideration the above-mentioned constraints that are imposed to the stakeholders of the system.

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.

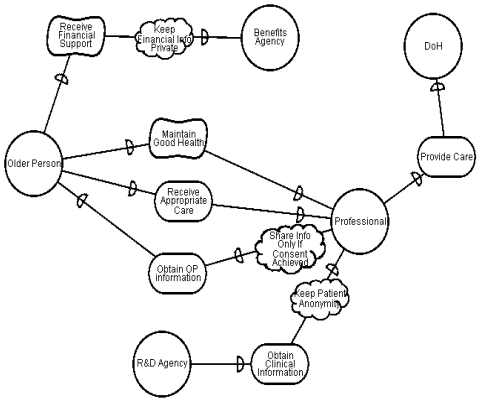

Figure 6: The stakeholders of the eSAP System

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). For example, the *Professional* actor (figure 7) has been imposed two *security constraints* (*Share Info Only If Consent Achieved* and *Keep Patient Anonymity*). During the means-end analysis of the *Professional* actor we have identified the *Share Medical Info* goal. However, this goal is restricted by the *Share Info Only If Consent Achieved* constraint imposed to the *Professional* by the *Older Person*. For the *Professional* to satisfy the constraint, a secure goal is introduced *Obtain Older Person Consent*. However this goal can be achieved with many different ways, for example a *Professional* can obtain the consent personally or can ask a nurse to obtain the consent on their behalf. Thus a sub-constraint is introduced, *Only Obtain Consent Personally*. This sub constraint introduces another *secure goal Personally Obtain*

Consent. This goal is divided into two sub-tasks *Obtain Consent by Mail* or *Obtain Consent by Phone*. The *Professional* has also a goal to *Provide Medical Information for Research*. However, the constraint *Keep Patient Anonymity* has been imposed to the *Professional*, which restricts the *Provide Medical Information for Research* goal. As a result of this constraint a *secure goal* is introduced to the *Professional*, *Provide Only anonymous Info*.

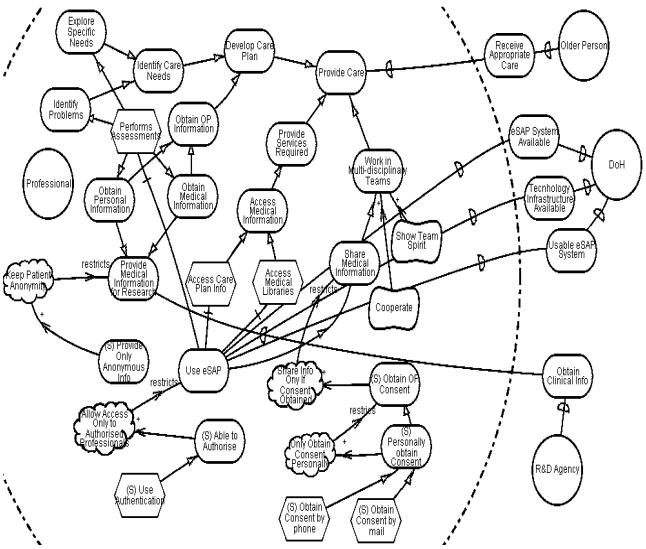

Figure 7: Professional Actor Partial Analysis

4.2 Late Requirements

When all the actors have been analysed, the next phase involves the analysis of the system-to-be. During the late requirements stage, the system-to-be is analysed within its operation environment, along with relevant functions, security concerns and qualities. The system is presented as one or more actors, who have a number of dependencies with the other actors of the organisation. These dependencies define all the functional and non-functional requirements of the system. However, in this example, we are focusing on the security modeling. From the security point of view, *security constraints* are imposed to the system-to-be (by taking into account the *security diagram*). These constraints are further analysed according to the constraint analysis processes [9].

The main aim of the eSAP system (figure 8) is to *Automate Care* in order to help professionals provide faster and more efficient care, and allow on the other hand older people get more involved in their care. Taking into consideration the *security diagram* there are two main constraints imposed (by the desired security features of the system- privacy and availability) to the eSAP's main goal - *Keep Data Private* and *Keep Data Available*. For the eSAP to satisfy these constraints two *secure goals* have been identified. *Ensure Data Privacy* and *Ensure Data Availability*.

This example focuses only on the *Keep Data Private* constraint. This constraint can be further analysed to sub-constraints *Allow Only Encrypted Transfer of Data*, *Allow Only Authorised Access*, and *Allow Access Only to Personal Care Plan*. Taking into account the *security diagram*, *secure goals* are introduced to help towards the satisfaction of the imposed *security constraints*. Thus the *secure goals Use Cryptography*, *Check Authorisation*, *Check Access Control*, and *Check Information Flow* are introduced. In addition, some of the *secure goals* are further analysed in terms of *secure tasks*.

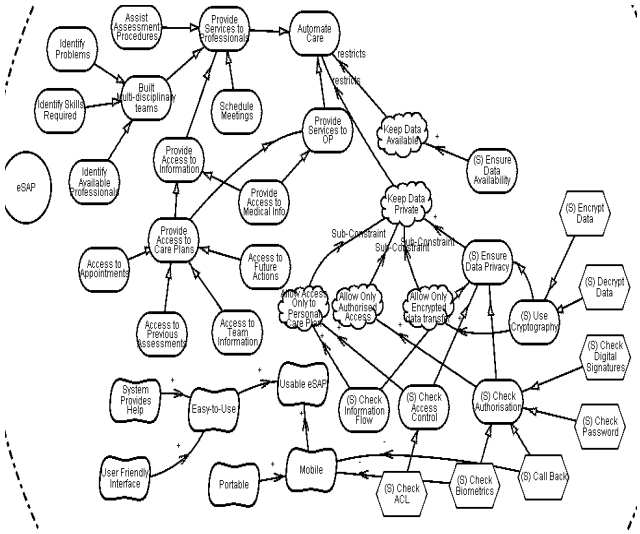

Figure 8: eSAP System Partial Analysis

Thus, the *Use Cryptography* goal is divided to two *secure tasks Encrypt Data* and *Decrypt Data*. Although these tasks could be furthered decomposed by indicating for example the type of the encryption algorithm this is not the case in this stage, since the type of the encryption algorithm depends on the implementation of the system and it will restrict the designers of the system in a particular implementation style. The *Check Authorisation* is decomposed into four *secure tasks*, *Check Password*, *Check Digital Signatures*, *Check Biometrics* and *Call Back*. However, it is indicated in the diagram that the last two tasks contribute negatively towards the mobility of the system, and this is one factor that the developers must take into consideration in the implementation of the system.

4.3 Architectural Design

The architectural design involves the addition of new actors, in which new actors are added to make the system interact with the external actors; actor decomposition, in which each actor is described in detail with respect to their goals and tasks; capabilities identification, in which capabilities needed by the actors to fulfill their goals are identified; and agent assignment, in which a set of agent types is defined and each agent is assigned one or more capabilities. From the security point of view, we identify the security constraints and secure entities that the new actors introduce and also during the actor decomposition we identify security sub-constraints and sub-entities. In addition secure capabilities are identified and assigned to each agent of the system.

As it was derived from the late requirements stage, one of the systems secure goals is to "Ensure Data Privacy". To achieve this goal the eSAP depends on the Privacy Manager (Figure 9).

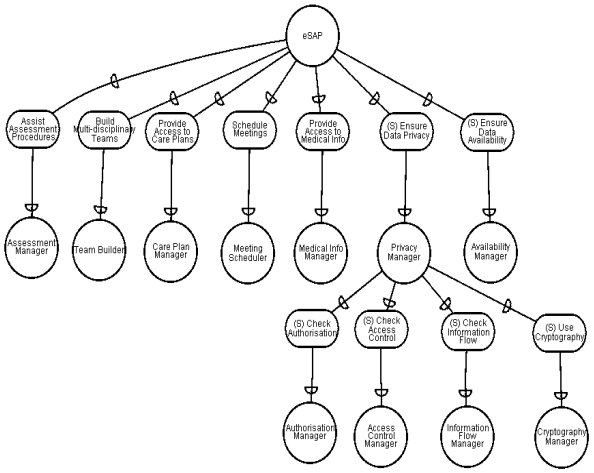

Figure 9: Actors' Decomposition Diagram

The Privacy Manager has four main secure goals, as derived from the analysis, (sub-goals to the "Ensure Data Privacy" goal) "check authorisation", "check access control", "check information flow" and "use cryptography". For achieving these goals the Privacy Manager depends on the "Authorisation Manager", "Access Control Manager", "Information Flow Manager" and "Cryptography Manager" respectively. Although the other actors of the system can be furthered decomposed, due to lack of space, this example focuses only in the privacy concerns of the system. For each new actor introduced in the system, an extended diagram is required to capture the dependencies of the new actor with the already existing actors of the system. Figure 10 shows a part (focused on the privacy) of the extended diagram for the plan "Access Care Plan Info" of the Professional. The Care Plan Manager is responsible for providing access at the Professional to "Care Plan Info". It depends on the Authorisation Manager to deal with authorisation procedures, on the Access Control Manager and the Information Flow Manager to perform access control checks and information flow checks respectively, and on the Cryptography Manager for encrypting and decrypting information.

The next step in the architectural design is to identify (secure) capabilities for each actor. Taking into consideration the extended actor diagram (figure 10), each dependency relationship can give place to one or more capabilities triggered by external events. The actors along with their capabilities with respect to the extended diagram of figure 10 are shown in Table 1. When the actors along with their capabilities have been identified the next step is the agents' assignment. A set of agent types are defined and each one of them is assigned one or more different capabilities (Table 2) with respect to the capabilities identified in the previous step (Table 1).

4.4 Detailed Design

From the security point of view, during the detailed design the developers specify the agent capabilities and interactions taking into account the security aspects derived from the previous steps of the analysis. In doing so AUML notation is employed. The only difference is the introduction of security rules. These are similar to the business rules that UML has for defining constraints on the diagrams.

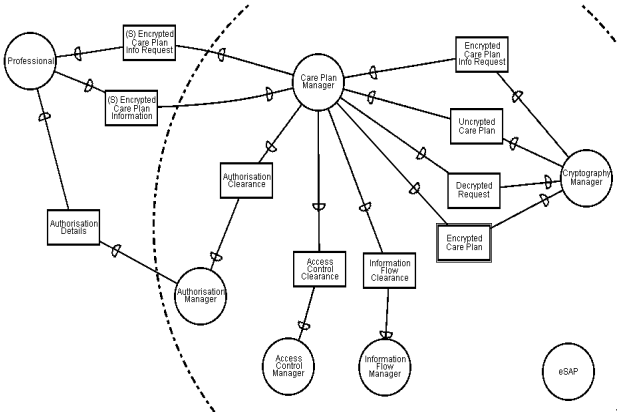

Figure 10: Extended Diagram wrt "Access Care Plan Info" task

Actors	Capability	Cap. ID
Professional	Provide Care Plan Info Request	1
	Provide Authorisation Details	2
	Obtain Care Plan Info	3
Care Plan Manager	Obtain Care Plan Info Request	4
	Provide Care Plan Info	5
	Request Encryption of Data	6
	Obtain Encrypted Data	7
	Request Decryption of Data	8
	Obtain Plain Data	9
	Obtain Authorisation Clearance	10
	Obtain Access Control Clearance	11
	Obtain Information Flow Clearance	12
Cryptography Manager	Encrypt Data	13
	Decrypt Data	14
Information Flow Manager	Provide Information Flow Clearance	15
Access Control Manager	Provide Access Control Clearance	16
Authorisation Manager	Obtain Authorisation Details	17
	Provide Authorisation Clearance	18

Table 1: Actors and their Capabilities

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. None of the existing agent oriented methodologies, to our knowledge, have been demonstrated enough evidence to support claims of adequately integrate security modeling during the whole software development stages. Only recently, some initial steps have been taken towards this direction. Eric Yu has initiated work [12] that provides ways of modeling and reasoning about non-functional requirements (with emphasis on security). Yu is using the concept of a soft goal to assess different design alternatives, and

how each of these alternatives would contribute positively or negatively in achieving the soft goal.

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

Table 2: Agents and their Capabilities

Lodderstedt et al present a modeling language, based on UML, called SecureUML [13]. Their approach is focused on modeling access control policies and how these (policies) can be integrated into a model-driven software development process. Differently than these two approaches that are focused in particular stages of the development (Yu's effort is focused only in the requirements area while Lodderstedt's work is focused in the design stage) our approach covers the whole development process. It is important to consider security using the same concepts and notations during the whole development process.

In addition, Huget [14] proposes a new methodology, called Nemo and claims that it tackles security. In his approach, security is not considered as a specific model but it is included within the other models of the methodology. Nemo is a new methodology and as such it has not been extensively presented on literature. From our point of view, the methodology tackles security quite superficial and as the developer states "*particularly, security has to be intertwined more deeply within models" [14]*. Thus, more evidence will be required to satisfy the claim of the developer that the methodology tackles security.

6. CONCLUSIONS AND FUTURE WORK

This paper presents results from our work to extend Tropos methodology to enable it to consider security requirements throughout its development stages. During the process of extending Tropos some very useful observations were obtained. First of all, the concept of constraints is a natural extension of the Tropos methodology and it allows for a systematic approach towards the modelling of security requirements. This is because, although functional and security requirements are defined alongside, a clear distinction is provided. Secondly, the security diagram allows identifying desired security requirements very early in the development stages, and helps to propagate them until the implementation stage, introducing a security-oriented paradigm to the software process. In addition, the iterative nature of the methodology, allows the re-definition of security requirements in different levels therefore providing a better integration with system functionality.

However, this in an ongoing research and more work is required to achieve our aim, which is to provide a well guided process of integrating security and functional requirements throughout the software development process of agent-based systems, using the same concepts and notations throughout the process. Currently we are working on refining the identified concepts, notations, and the process, and we are integrating our extensions to the Formal Tropos [6] specification language. This will enable us to formally evaluate our extensions, since Formal Tropos is amenable to formal analysis.

7. REFERENCES

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

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

[3] M. Wooldridge, N. R. Jennings, D. Kinny, " The GAIA Methodology for Agent-Oriented Analysis and Design", *Journal of Autonomous Agents and Multi-Agent Systems 3*, (3) pp. 285-312, 2000

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

[5] 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.

[6] A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. in *Proc. of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, 28 May - 1 June 2001.

[7] 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.

[8] 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

[9] H. Mouratidis, "Extending Tropos Methodology to accommodate Security", Computer Science Report, University of Sheffield, September 2002

[10] P. Bresciani, A. Perini, P. Giorgini, G. Giunchiglia, J. Mylopoulos, "Modelling early requirements in Tropos: a transformation based approach", Agent Oriented Software Engineering II, M. Wooldridge, G. Weiβ (eds), Lecture Notes in Computer Science, Springer-Verlag 2222, 2002

[11] 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

[12] L. Liu, E. Yu, J. Mylopoulos, "Analysing Security Requirements as Relationships Among Strategic Actors", in the Proceedings of 2nd Symposium on Requirements Engineering for Information Security, North Carolina - USA, November 2002.

[13] 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.

[14] M.P. Huget, "Nemo: An Agent-Oriented Software Engineering Methodology", in the Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle – USA, November 2002.