From Goal-Driven Security Requirements Engineering to Secure Design

Haralambos Mouratidis, Jan Jurjens[1]

***Abstract*— Security of intelligent software systems is an important area of research. Although security is traditionally considered a technical issue; security is in fact a two dimensional problem, which involves technical as well as social challenges. Goal-Driven Requirements Engineering (GDRE) has been proposed in the literature as a suitable paradigm for the analysis of security issues and elicitation of security requirements at both the social and technical level. Nevertheless, there is lack of approaches, which would support the successful transformation of the elicited, using GDRE approaches, security requirements to design. This paper presents work that fills this gap. The presented approach, which is based on the integration of a Goal-Driven Security Requirements Engineering (GDSRE) methodology and a Model-Based Security Engineering (MBSE) method, has some important features: (1) It provides a structured process to translate the results of the GDSRE method to a design, which satisfies these requirements; (2) it allows the simultaneous elicitation and analysis of the security requirements and the functional requirements of the system; (3) it allows consideration of both the social and the technical dimensions of the system's security; (4) it guides software engineers towards a design that is amenable to formal verification with the aid of automated tools. We demonstrate the applicability of the proposed approach at the hand of an application to the electronic purse standard Common Electronic Purse Specifications (released by Visa International and others).**

1 Introduction

Security is traditionally approached as a technical issue that requires a technical solution. This treatment of security has led to the development of a number of security mechanisms and protocols that on one hand are successfully used in modern software systems but on the other hand, they have failed to ensure an acceptable degree of security. Recent research [1] has argued that one of the reasons for this situation is the wide-spread of software systems and their usage in almost every part of the human society. As a result, security of such systems has been transformed from a mono-dimensional technical issue to a two-dimensional issue that includes a technical dimension (related to challenges and problems associated to the available technology and the infrastructure of software systems) and a social dimension (which includes issues and problems related to the involvement of humans in securing software systems). To effectively consider both dimensions, the research literature [1,6,9,10,11] argues that it is essential for security to be considered from the early stages and throughout the software development life-cycle and a sound software engineering methodology needs to be developed that supports the simultaneous analysis of both dimensions of security.

However, it has remained true over the last 30 years (since the seminal paper [2]) that no coherent and complete methodology to ensure security in the construction of large general-purpose systems exists yet, in spite of very active research and many useful results addressing particular sub-goals, as well as a large body of security engineering knowledge [3].

In this paper[2], we integrate a Goal-Driven Security Requirements Engineering (GDSRE) methodology (called Secure Tropos) with a Model-Based Security Engineering (MBSE) approach (called UMLsec) to create a structured methodology for secure software systems development. In particular, the presented approach takes the high-level concepts and modelling activities of the Secure Tropos methodology [5] and enriches them with a low level security-

[1] The first author is with the School of Computing, Information Technology and Engineering, University of East London, Docklands Campus, 4-6 University Way, E16 2RD, London, England, email: haris@uel.ac.uk

The Second Author is with the Computing Department, Open University, Walton Hall, Milton Keynes, Buckinnghamshire, MK7 6AA, England, email: j.jurjens@open.ac.uk

[2] Which is an extended and revised version of [4]

engineering ontology and models derived from the UMLsec approach [6]. Moreover, an enhanced goal-driven development process is defined that considers security from the early stages of the development process. By integrating the two approaches, we manage to take advantage of the strong parts of each approach and at the same time minimize their limitations. This achieves several goals. First of all, it allows software engineers to consider both dimensions of security in a structured and well defined way. Initially, by employing a goal oriented requirements engineering analysis, the social dimension of security is considered by analysing the environment of the system-to-be and by facilitating understanding of the security needs of the stakeholders and the security issues imposed to the system by its environment. Then, the technical dimension of security is considered, by supporting the transformation of the security requirements to a design that is amenable to formal verification with the aid of automatic tools [7]. Secondly, the approach allows the definition of security requirements early in the process and in different levels and as a result it provides a security–aware environment where security and functional requirements are analysed simultaneously.

This paper is structured as follows. Section 2 provides some background information about security aware software engineering and Section 3 introduces Secure Tropos and UMLsec. Section 4 discusses the suitability of the two approaches for the integration and Section 5 presents the integration of the two approaches. Section 6 demonstrates the integrated approach at the hand of an application to the electronic purse standard Common Electronic Purse Specifications (CEPS) and Section 7 discusses related work. Finally Section 8 concludes the paper and presents directions for future work.

2 Security-Aware Software Engineering

It is well known that perfect security is seldom possible to achieve and thus the goal is to provide an acceptable security level, usually by trading security requirements with other functional and non-functional requirements of the system-to-be [8]. To effectively make such trade-offs, software system developers need to have a clear understanding of both the technical issues surrounding the security of a software system as well as the social issues influencing the software system's security. In other words, it is not enough just to consider security mechanisms and protocols, but an understanding of the human factor and the environment of the software system is also required. The human factor has a significant impact on security [8] and the effectiveness of a system depends on the forces of its environment [9].

As such, it has been widely argued in the literature that in order to obtain such understanding, software engineers need to consider security as part of the software system's development process. Devanbu and Stubblebine [10] argued that "security concerns should inform every phase of software development, from requirements engineering to design, implementation, testing and deployment". In addition, Mouratidis [5] indicated that "From the viewpoint of the traditional security paradigm, it should be possible to eliminate some [security related] problems through better integration of security and software engineering". Such arguments are consistent with previous research outputs that indicated that security mechanisms cannot be "blindly" inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way [2].

A large number of researchers [1,11,12,13,14] also argue for the need not only to consider security as part of the development process but in fact to consider it as early as possible. There is general agreement that integration starting from the earliest stages is essential. It is well known that mistakes early in the software process can have far reaching consequences in subsequent

stages that are difficult and costly to remedy.

However, this is far from what happens in current practice, where in some cases security requirements are not elicited and reasoned by the software developers and the design does not provide a security infrastructure according to the needed security requirements. In contrast, security requirements are mostly copied from a generic set of security requirements and specific system security design is replaced by a set of standard security mechanisms which are introduced in the system. This is problematic, since very little consideration is given on the implications of inserting such mechanisms into the system's architecture. As a result, security may conflict with the system's requirements, which most of the times translate into security vulnerabilities [3,15]. In other cases where security requirements are considered, they are often developed independently of the rest of the requirements engineering activity and hence are not integrated into the mainstream of the requirements activities. As a result, security requirements that are specific to the system and that provide for protection of essential services and assets are often neglected [16].

In this paper, we present a methodology based on a goal-driven requirements engineering process. In particular, our work provides support to:

1. elicit security requirements, based on a goal-driven security requirements analysis;
2. develop a design, based on the identified security requirements;
3. validate the security properties of the design according to the security requirements of the system.

To achieve the above, we employ the Secure Tropos and the UMLsec approaches. The next section provides an overview of these two approaches.

3 Secure Tropos and Umlsec

3.1 Secure Tropos: A Goal-Driven Security Requirements Engineering Methodology

Secure Tropos [5,17] is a security-oriented extension of the widely known requirements engineering methodology Tropos. The Tropos (and as a result the Secure Tropos) methodology is mainly based on four phases [18]:

- *Early Requirements Analysis*, aimed at defining and understanding a problem by studying its existing organizational setting;
- *Late Requirements Analysis*, conceived to define and describe the system-to-be, in the context of its operational environment;
- *Architectural Design*, that deals with the definition of the system global architecture in terms of subsystems; and the
- *Detailed Design* phase, aimed at specifying each architectural component in further detail, in terms of inputs, outputs, control and other relevant information.

The main differences of Secure Tropos in relation to other security oriented approaches are:

1. social issues of security are analysed during the early requirements stage;
2. security is considered simultaneously with the other requirements of the system-to-be;
3. the methodology supports not only requirements stages but also design stages.

Secure Tropos introduces a number of security related concepts to the Tropos methodology. A *security constraint* is defined as a restriction related to security issues, such as privacy, integrity, and availability, which can influence the analysis and design of the information 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 [5]. Graphically, a security constraint is depicted as a hexagon and it is positioned in the side of the actor who has to satisfy it (see for instance Figure 6). Additionally, Secure Tropos defines secure dependencies. *A secure dependency* introduces security constraint(s) that must be fulfilled for the dependency to be satisfied [5]. Secure Tropos uses the term secure entity to describe any goals and tasks related to the security of the system. A *secure goal* represents the strategic interests of an actor with respect to security. Secure goals are mainly introduced in order to achieve possible security constraints that are imposed to an actor or exist in the system. However, a secure goal does not particularly define how the security constraints can be achieved, since alternatives can be considered. The precise definition of how the secure goal can be achieved is given by a secure task. A *secure task* is defined as a task that represents a particular way for satisfying a secure goal.

The security oriented process of Secure Tropos is mainly divided into four sub-activities: (1) The identification of security requirements of an information system, in which the security needs of the stakeholders and the system are analysed in terms of security constraints imposed to the system and the stakeholders, and secure goals and entities are identified that guarantee the satisfaction of the security constraints; (2) the selection amongst alternative architectural styles for the system-to-be according to the identified security requirements, which allows developers to reason about alternative design solutions according to the security requirements of a system; (3) the development of a design that satisfies the security requirements of the system, which allows developers to employ a security pattern language to satisfy the security requirements of the system; (4) the attack testing of the system under development, which allows developers to test how the system under development copes with any possible attacks.

A detailed description of Secure Tropos concepts and process can be found in [5].

3.2 Model-based Security Engineering using UMLsec

In Model-based Security Engineering [6,19,20], recurring security requirements (such as secrecy, integrity, authenticity and others) and security assumptions on the system environment, can be specified either within a UML specification, or within the source code (Java or C) as annotations.

One can use MBSE within model-based development (see Figure 1). Here one first constructs a model of the system. Then, the implementation is derived from the model: either automatically using code generation, or manually, in which case one can generate test sequences from the model to establish conformance of the code regarding the model. The goal is to increase the quality of the software while keeping the implementation cost and the time-to-market bounded.

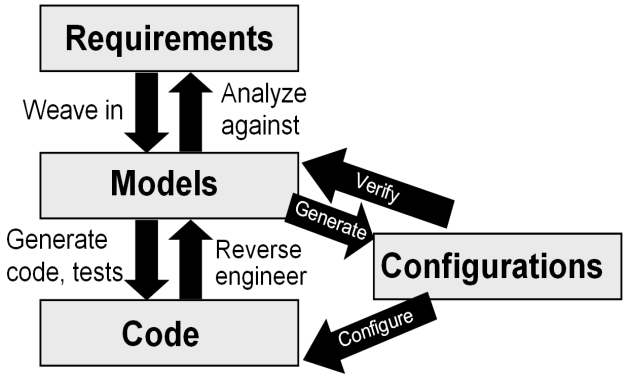

Figure 1: Model Based Security Engineering

In UMLsec, recurring security requirements, such as secrecy, integrity, and authenticity are offered as specification elements by the UMLsec extension. These properties and its associated semantics are used to evaluate UML diagrams of various kinds and indicate possible security vulnerabilities. One can thus verify that the desired security requirements, if fulfilled, enforce a given security policy. One can also ensure that the requirements are actually met by the given UML specification of the system. UMLsec encapsulates knowledge on prudent security engineering and thereby makes it available to developers who may not be experts in security. The extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate security requirements and assumptions on the system environment. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics mentioned below.

The tags defined in UMLsec represent a set of desired properties. For instance, "freshness" of a value means that an attacker can not guess what its value was. Moreover, to represent a profile of rules that formalise the security requirements, the following are some of the stereotypes that are used: «critical», «high», «integrity», «internet», «encrypted», «LAN», «secrecy», and «secure links». If relevant, their profile also contains the possible attackers associated to them as shown in Table 1.

Stereotype	**Threats** *default*()	**Threats** *insider*()
Internet	{delete, read, insert}	{delete, read, insert}
Encrypted	{delete}	{delete, read, insert}
LAN	Ø	{delete, read, insert}

Table 1: *Attackers and threats per stereotype in the UMLsec*

The definition of the stereotypes allows for model checking and tool support. As an example consider «secure links». This stereotype is used to ensure that security requirements on the communication are met by the physical layer. More precisely, when attached to a UML subsystem, the constraint enforces that for each dependency d with stereotype $s \in (\{<< \text{secrecy} >>, << \text{integrity} >>, << \text{high} >>\})$ between subsystems or objects on different nodes, according to each of the above stereotypes, there shall be no possibilities of an attacker reading, or having any kind of access to the communication, respectively. A detailed explanation of the tags and stereotypes defined in UMLsec can be found in [6].

4 Integration Suitability of Secure Tropos and UMLsec

There are various reasons for selecting secure Tropos and UMLsec from the large number of different existing methodologies and modelling languages. Secure Tropos considers the social dimension of security as well as the high-level technical dimension of it. Firstly, an analysis regarding social aspects of security takes place in which the security requirements of the stakeholders, users and the environment of the system are analysed, with the aid of goal analysis techniques, and identified. Then, the methodology continues with a more technical dimension by considering the system and identifying its secure requirements, and allowing developers to identify the architecture of their systems with respect to the identified requirements. On the other hand, UMLsec covers also the later stages of the development, such as the detailed design stage

(detailed definition of security components), the implementation stage (model-based testing and code verification against UMLsec specification) and the deployment phase (through analysis of user configurations against UMLsec models).

We have identified the relative strengths of these two approaches, which indicate what makes each of these approaches suitable for our purpose, as well as combinational strengths, which indicate why these two approaches are suitable for integration. ***Individually***, Secure Tropos considers security issues throughout the development stage, from the early requirements analysis down to implementation. Moreover, it allows developers not only to identify security issues but also to reason about them, and it provides a security pattern language to assist developers without much security knowledge to specify the architecture of the system according to its security requirements. On the other hand, UMLsec encapsulates established rules of prudent security engineering in the context of widely known notations, and thus makes them available to developers without extensive training in security. In addition, UMLsec supports automated verification of design models against security properties. ***Combinational***, both of the approaches are extensions of well-known approaches (Tropos and especially UML) and this makes the approach easily accessible to a large number of researchers and practitioners. Also, the strength of Secure Tropos (requirements analysis) is complementary to the strengths of UMLsec (security design analysis) and vice versa, therefore providing a complete solution. In addition, the use of UML models during the design stage of the Tropos methodology makes the integration of Secure Tropos and UMLsec more natural.

However, and despite the above advantages the two approaches demonstrate some limitations. The existing Secure Tropos language is mainly focused on the early and late requirements stages and fails to provide constructs and notation to define in detail the security components of the system. In particular, for the *Detailed Designed* stage the methodology is mainly based on UML diagrams with minor extensions to indicate some security issues [5]. These models are limited with respect to the security specification of each component of the system. On the other hand, the UMLsec approach does not consider the social dimension, since the only analysis that it offers at the early stages of the development (stages at which the social issues are introduced) is use case diagrams, which do not consider the social security requirements of the system's stakeholders. Moreover, it provides constructs mainly for the definition and validation of the security components of a system. Therefore, the UMLsec approach lacks a process to identify the security requirements of the system. We believe that integrating these two approaches will lead us to a complementary approach for secure software systems development. In fact, the integration of UMLsec to the secure Tropos methodology provides a framework of particular strength throughout all the development stages as shown in Figure 2 .

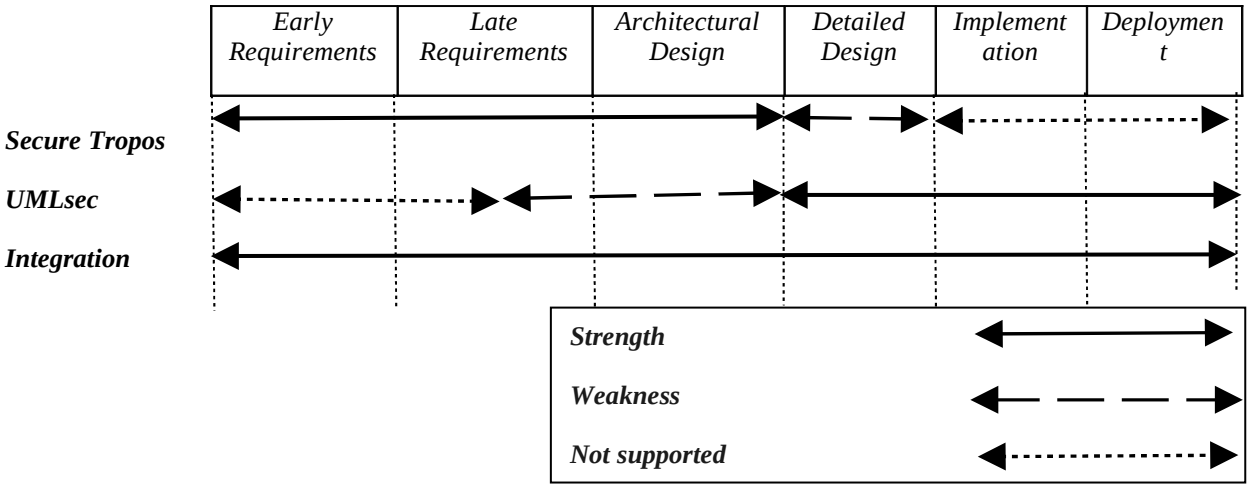

Figure 2: *The complementary nature of the two approaches*

5 Integration Through Translation

In this project we integrate the two approaches using functional integration [21]. In this type of integration given models of each approach stay intact and guidelines to translate the models from one approach to another and indicate the inputs and the outputs of these models are defined. In translating between Secure Tropos and UMLsec two main challenges must be addressed: firstly, how to define a uniform and easy to follow development process building on the strengths of the two approaches and minimizing their weaknesses and secondly, how to represent the concepts and models of Secure Tropos to the concepts and models of UMLsec.

To overcome the first challenge we have defined a new goal-driven security-aware process that is based on the Secure Tropos process but it enhances it by adding support for the development of the UMLsec models. To overcome the second challenge we have developed a well defined set of mapping guidelines and steps to assist developers in moving from the Secure Tropos models to the UMLsec models.

5.1 The Security-Aware Process

The new security aware process includes four main stages[3]:

- *Security Analysis of System Environment*,
- *Security Analysis of System*,
- *Secure System Design*, and
- *Secure Components Definition*.

In each of these stages, a number of activities have been identified and each of the activities results in a number of different analysis and/or design models as shown in Figure 3. Although for reasons of simplicity we describe the stages in a sequential order, it is worth pointing out that we expect developers to follow an iterative approach.

5.1.1 Security Analysis of System Environment

The main aim of this stage is to understand the social dimension of security by considering the social issues, of the system environment, which might affect its security. In doing so, the environment in which the system will be operational is analysed with respect to security. In particular, in line with the Secure Tropos methodology, the stakeholders of the system along with their strategic goals are analysed in terms of actors (*Stakeholders Analysis Activity*) who have strategic goals and dependencies for achieving some of those goals. Goal analysis techniques [18] such as means-end analysis and decomposition are widely used during this activity. Then the security needs of those actors are analysed (*Security Constraints Analysis Activity*) in terms of security-related constraints that are imposed to those actors. Moreover, security goals and entities are identified (*Secure Entities Analysis Activity*), for each of the participating actors, to satisfy the imposed security constraints. In particular, developers examine the security constraints imposed on individual actors, and documented in the security-enhanced goal diagram, and identify any related secure goals that assist in satisfying those security

[3] In this paper we focus on the integration of the two approaches. As such, we do not cover the deployment and implementation stages of the development process for which our integration does not provide anything new, i.e. these two stages receive inputs in UMLsec concepts from the secure components definition stage.

constraints. The process of identifying secure goals is similar to the process used in goal-oriented approaches and involves techniques such as means-end analysis 5.1.4. However, such techniques are combined with a number of security-related techniques such as attack trees 5.1.4 and security reference diagrams 5.1.4. The Secure Goal Introduction 5.1.4 analysis enables developers to refine the goals of an actor to allow the satisfaction of a security constraint. In some cases it is necessary to decompose security constraints into more detailed security constraints. In doing so, the AND decomposition technique is employed. The decomposed constraint is called the "root" constraint, and its satisfaction is implied if and only if all the security sub-constraints are satisfied. Identified secure goals are documented in a security-enhanced goal diagram. The above analysis activities are modelled in terms of different diagrammatic notations as shown in Figure 3 . With respect to security, a security-enhanced actor diagram is used to analyse the actors of the environment of the system along with their secure dependencies and security constraints. On the other hand, a security-enhanced goal diagram allows a deeper understanding of how the actors, modelled in the security-enhanced actor diagram, reason about goals to be fulfilled, plans to be performed and availability of resources 5.1.4. The security-enhanced goal diagram complements the security-enhanced actor diagram with the reasoning that each actor requires about its internal security goals, secure plans and secure resources. In other words, the security-enhanced goal diagram presents a more focus analysis on each one of the actors identified during the security-enhanced actor diagram.

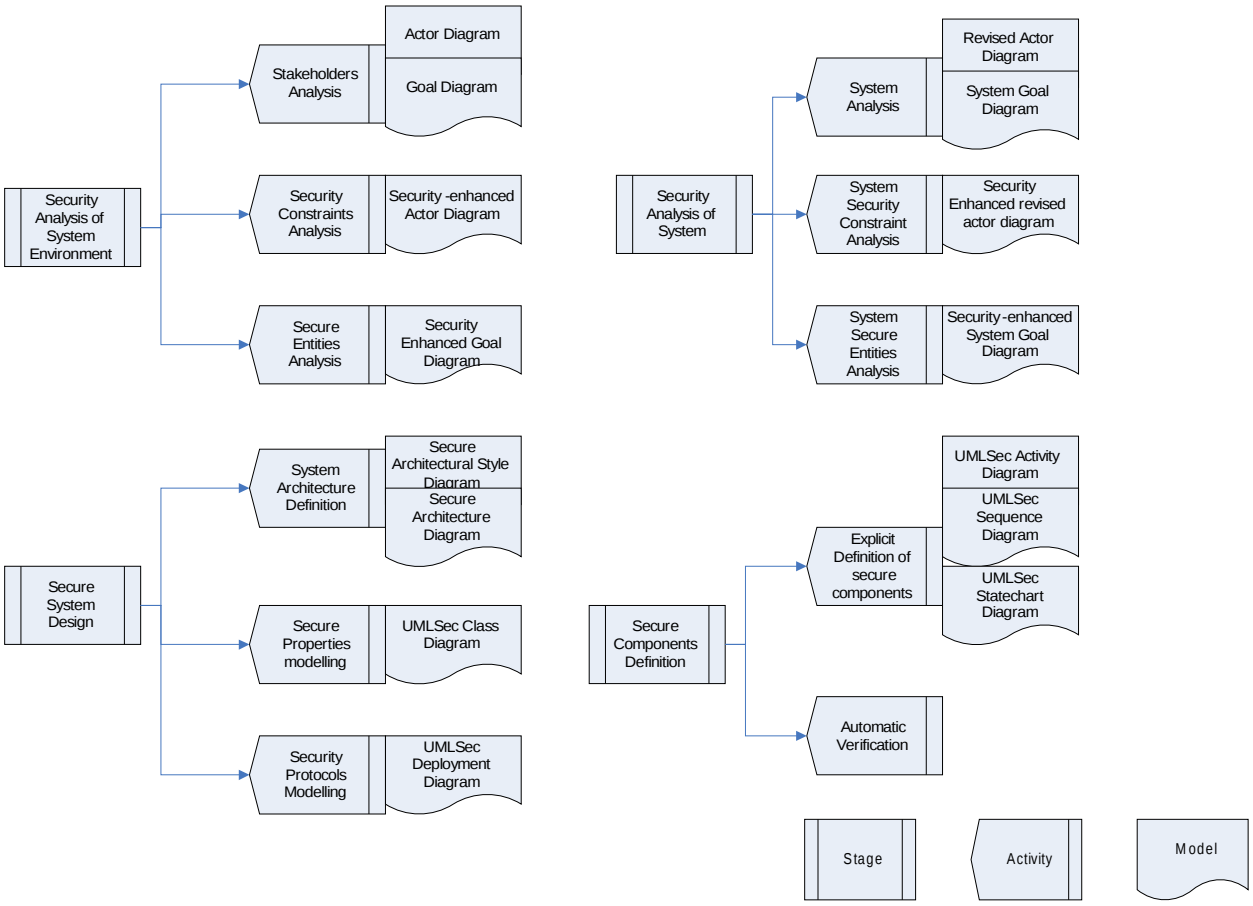

Figure 3: The stages of the process

5.1.2 Security Analysis of System

The main aim of this stage is to understand the technical dimension of security. For this stage, activities similar to the previous stage are employed but now the focus is on the system rather than its environment. In particular, the security requirements of the system are identified taking into account the security needs of the stakeholders as well as their security constraints. The output of this stage is the definition of the system's security requirements together with a set of security constraints, along with the system's security goals and entities that allow the satisfaction of the security requirements of the system.

5.1.3 Secure System Design

The main aim of this stage is to define the architecture of the system with respect to its security requirements. To achieve this, a combination of Secure Tropos and UMLsec models are employed. Actor, Goal and secure architectural style models of Secure Tropos together with a set of security patterns [5] are used to determine the general architecture and the components of the system, whereas UMLsec Class and Deployment diagrams are used to model the security properties of the data structures and architecture. It is at this stage of the development process that the translation from the Secure Tropos to UMLsec models takes place according to the guidelines and steps defined below. It is also worth mentioning that the functionality of the SecTro tool, which supports the development of the Secure Tropos models, to automatically derive XML code from the corresponding Secure Tropos models together with the functionality of the UMLSec tool to accept XML input, enables us to speed up the process of translating the Secure Tropos models to UMLSec models.

5.1.4 Secure Components Definition

During this stage UMLsec is used to specify in detail the components of the system identified in the previous stage. To achieve this, UMLsec activity diagrams are used to define explicitly the security of the components and UMLsec sequence diagrams are used to model the secure interactions of the system's components (for example, to determine if cryptographic session keys exchanged in a key exchange protocol remain confidential in view of possible adversaries). UMLsec statechart diagrams are used to specify the security issues on the resulting sequences of states and the interaction with the component's environment. Moreover, the constraints associated with UMLsec stereotypes are checked mechanically, based on an XMI representation of the UML models and using sophisticated analysis engines such as model-checkers and automated theorem provers. The results of the analysis are given back to the developer, together with a modified model, where the weaknesses that were found are highlighted.

5.2 Translation Guidelines and Steps

Our work towards the solution of the second challenge involved the definition of a set of translation guidelines to map the Secure Tropos analysis and early design models to UMLsec models. The following guidelines and steps were identified towards this direction.

Guideline 1: Translation of the Secure Tropos analysis models to UMLsec class diagrams

Step 1. Actor mapping to UMLsec classes: To map Secure Tropos actors to UMLsec models, the following translation rules are followed:

- Every actor on the Secure Tropos security enhanced actor diagram is mapped as a class on the corresponding UMLsec class diagram.
- In case of sub-actors (described in the corresponding security-enhanced goal diagrams), these are mapped into the UMLsec class diagram as an inheritance relationship pointing from the sub-actor class to the main actor class.

Consider for instance an example where a security-enhanced actor diagram defines a set of actors (along with the appropriate relationships) such as *Lecturer*, *Administrator*, *Student*, *Registration System*, *Student Records System* and so on. For each one of these actors a security-enhanced goal diagram is constructed to represent the internal functionalities (goals, tasks, security constraints etc) of these actors. Consider for example the *Student Records System*, its security-enhanced goal diagram will represent its functionalities and any sub-actors used to satisfy some of these functionalities. Examples of such sub-actors are *Qualifications Analysis* actor –which has functionalities to analyse the qualifications of a student- and *Module Confirmation* actor – which has functionalities to confirm the modules a student needs to attend according to their selected programme. Following the above mapping the *Student Records System* will be represented as a class in the UMLsec class model and the *Qualifications Analysis* and *Module Confirmation* actors will be represented as sub-classes.

Step 2. Map actor capabilities to UMLsec classes: To map actor capabilities on the corresponding UMLsec classes, the following translation rule is followed:

- Each capability of every actor defined in the Secure Tropos models, i.e. the security enhanced actor and security-enhanced goal diagrams, is mapped as an operation on the corresponding UMLsec class.

For instance, in the example above the capabilities identified for the *Records System*, the *Qualifications Analysis* and *Module Confirmation* actors will be mapped as operations to the corresponding classes of these actors to the UMLsec model.

Step 3. Map actor resources to UMLsec classes: To map actor resources to corresponding UMLsec classes, the following translation rule is followed:

- Each resource allocated or used by every actor defined in the Secure Tropos models is mapped as an attribute on the corresponding UMLsec class diagram.

It is worth stating that in some cases this is not a 1-to-1 mapping, meaning that a UMLsec class will not have exact the same number of attributes as the Secure Tropos model counterpart. The reason for this is that Secure Tropos models are mainly analysis models, whereas the UMLsec model is a design model. Therefore, it is up to the developers to identify additional attributes according to the identified operations, by following the same process followed when identifying attributes for a class on any UMLsec class diagram. For instance, in the above example the *Qualifications Analysis* actor will require access to resources such as students' qualifications and the universitity's acceptable qualifications information. These two, amongst others, represent resources for the actor as modelled in the security-enhanced goal diagram. These will be translated to attributes in the UMLsec diagram.

Step 4. Map actor dependencies to associations: The following translation rule is followed:

- Every dependency, between two actors, defined in the secure Tropos models is mapped into one association between the corresponding UMLsec classes.

It is worth stating however, that in some cases developers will identify one association for a number of dependencies. This is due to the fact that Secure Tropos models hold analysis information whereas UMLsec models hold design information. Looking at the above example, the security-enhanced actor diagram might represent a dependency between the *Records System* and the *Qualifications Analysis* actors. This is the case, since the *Records System* actor depends on the *Qualifications Analysis* actor to provide information about a student's qualifications before the student is allowed to register. Such dependency is modelled as an association in the UMLsec model.

Step 5. Map critical actors to UMLsec classes. As described in previous sections, in secure Tropos diagrams a secure dependency is defined between two actors and it indicates that one actor needs to satisfy a security constraint for the dependency to be valid. Therefore, by

analysing each secure dependency we can identify the critical actors of the system (the ones that need to satisfy security constraints). In Secure Tropos, the type of a secure dependency indicates whether an actor is critical for the security of the system or not. Actors are considered critical when a security constraint is imposed to them. To map critical actors from Secure Tropos to UMLsec, the following translation rules are followed:

- Every actor, in a secure Tropos model, that is imposed a security constraint as part of a secure dependency is considered critical.
- For every identified critical actor, the UMLsec class, corresponding to that actor, is assigned the <critical>> stereotype.

Looking at the above example, if a secure dependency exists between the *Records System* and the *Qualifications Analysis* actors with a security constraint imposed to the *Qualifications Analysis* actor, then that actor will be modelled as <<critical>> in the UMLsec model.

Guideline 2: Translate the Secure Tropos analysis model to UMLsec deployment diagram

Step 1. Map actors to UMLsec nodes and components: Secure Tropos recognises two types of actors, internal and external. An internal actor represents a core actor of the system, while an external actor represents an actor that interacts with the system. Therefore, the following translation rules are followed:

- Every external actor defined in the Secure Tropos analysis is modelled as a "user" node in UMLsec.
- Every internal actor defined in the secure Tropos analysis models is modelled as a "system" node in UMLsec.

At least one "user" and one "system" nodes need to be defined. Going back to the above example, a *Student* actor represents an external actor of the system (interacts with the system) while the *Records System* represents an internal actor of the system.

Step 2. Map Dependencies to modes of Communication. The following translation rule is followed:

- For every dependency defined in the secure Tropos models, a mode of communication is modelled in the corresponding UMLsec model.

To denote the appropriate mode of communication, UMLsec stereotypes are used, such as those described in section 3. In the above example, if the communication between the *Student* actor and the *Records System* actor takes place over the Internet, then the communication mode between the user node "Student" and the system node "Records System" should be denoted using the <<Internet>> UMLsec stereotype.

Step 3. Map Secure Tropos security constraints to UMLsec security stereotypes:

- For every security constraint identified in the secure Tropos models, at least one UMLsec security stereotype should be modelled in the corresponding UMLsec model.

It should be noted that the mapping is not one-to-one, meaning that more than one stereotypes will, usually, result from one security constraint. Using the above example, assuming a security constraint is imposed to the Records System actor to only allow access to authorised student actors, a security stereotype must be used to denote that security constraints in the UMLsec model.

5.3 Semi-Automatic Support

The proposed methodology is supported in a semi-automatic way by a set of tools. In particular, the secTro tool is employed to support developers during the security analysis of

System Environment; Security Analysis of System and the initial steps of the Secure System Design stages, while the UMLsec tool is mainly used to assist developers during the later stages of the Secure System Design and Secure Components Definition stages.

secTro (Figure 4) is a platform independent analysis and modelling tool that supports the security related concepts and notations provided by the Secure Tropos methodology. The tool has been developed following an iterative approach and it is based on JAVA. The tool allows developers to model the system under development and its environment and it supports the capture of properties of the various models, such as security enhanced actor diagram and security enhanced goal diagram, and of their components. These are represented as XML type specifications. Once developers are satisfied with the developed models, the represented XML specifications are fed – manually- to appropriate tools, such as tefkat[4], along with the transformation rules to derive the UMLsec models. These are then fed into the UMLsec tools for the analysis of the security properties of the system under development and the components definition.

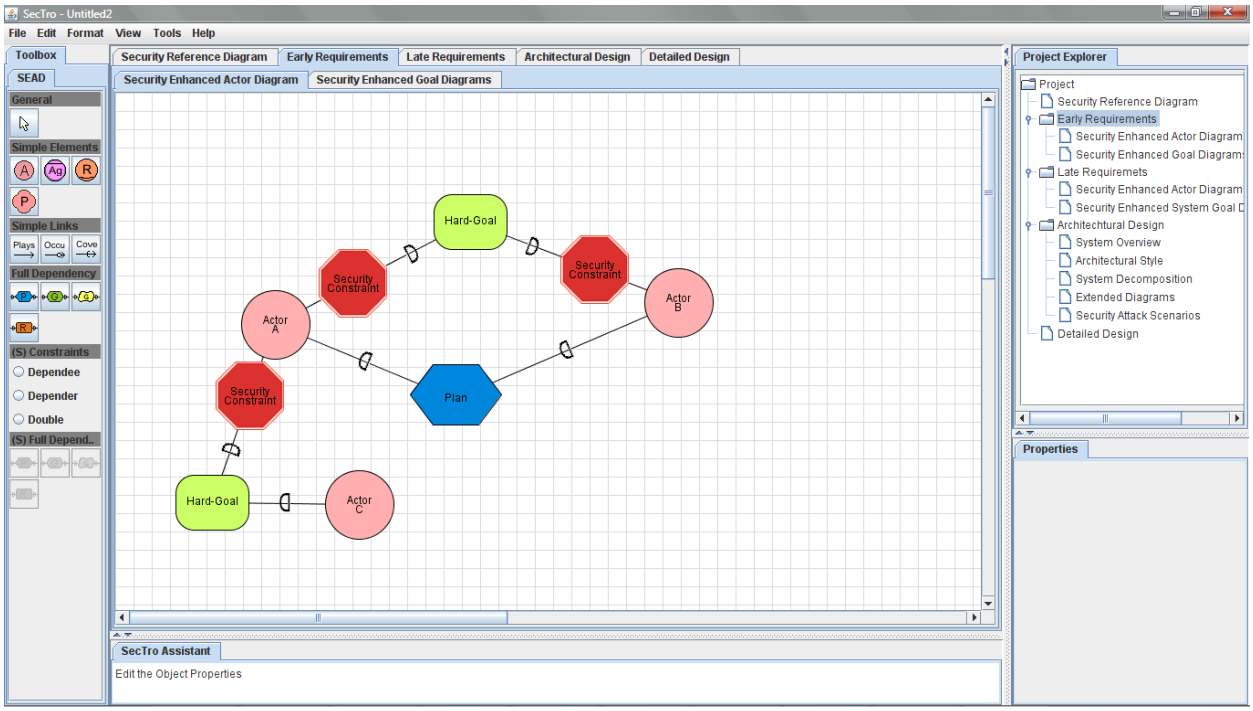

Figure 4: SecTro Screen

The UMLsec set of tools [7] (see Figure 5) generate logical formulas formalizing the execution semantics and the annotated security requirements. Automated theorem provers and model checkers automatically establish whether the security requirements hold. If not, a Prolog-based tool automatically generates an attack sequence violating the security requirement, which can be examined to determine and remove the weakness. This way we encapsulate knowledge on prudent security engineering as annotations in models or code and make it available to developers who may not be security experts. Since the analysis that is performed is too sophisticated to be done manually, it is also valuable to security experts.

[4] http://tefkat.sourceforge.net/

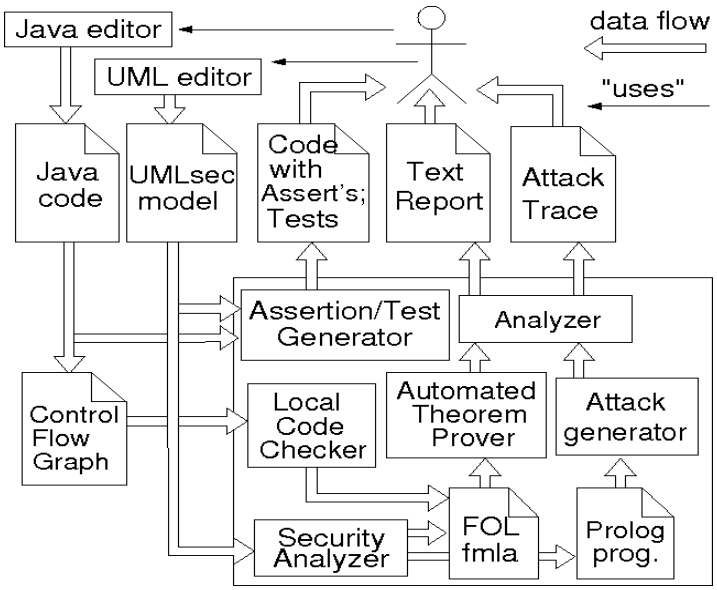

Figure 5: UMLsec Tools

6 Case Study

To demonstrate the validity of our approach, we present an application to the electronic purse standard Common Electronic Purse Specification (CEPS) [22] developed by Visa International and other companies. CEPS proposes the use of stored value smart cards, called electronic purses or CEP cards, to allow cash-free point-of-sale (POS) transactions offering more fraud protection than credit cards[5]. Amongst others, the following participants are defined in a CEP transaction [22]: the ***Scheme Provider***, the authority responsible for establishing an infrastructure for the overall functionality and security of the CEP system and enforcing the operating rules and regulations of the scheme; the ***Card Issuer***, the organisation responsible for the provision and distribution of smart cards containing a CEP application (electronic purses), and the management of the funds pool; the ***Cardholder***, the person who uses the card for making purchases; the ***Load Acquirer***, the entity responsible for establishing business relationships with one or more scheme providers to process load and currency exchange transactions, and settle unlinked transactions; the ***Merchant***, who is responsible for the use of a POS device to accept CEP cards for payment of goods and services; the ***Merchant Acquirer***, the entity responsible for establishing a business relationship with one or more scheme providers to process POS transactions, and settle POS transactions. Moreover, the merchant acquirer is responsible for the provision and distribution of Purchase Secure Application Modules (PSAMs) that interact with terminals for conducting transactions at the point of sale.

6.1 Security Analysis of System Environment

Initially, the main actors of the system are identified together with their dependencies and their security constraints. In particular, a CEP based transaction, although it provides many

[5] Credit card numbers are valid until the card is stopped, enabling misuse. In contrast, electronic purses can perform cryptographic operations which allow transaction-bound authentication.

advantages, over a cash transaction, for both the buyer and the merchant; it is much more complex. In a normal operating scenario of the CEPS scheme, the *Cardholder* loads his/her card with money. During the post-transaction settlement, the *Load Acquirer* sends the money to the relevant *Card Issuer*. The *Cardholder* buys a product from a *Merchant* using his/her card. In the settlement, the *Merchant* receives the corresponding amount of money from the *Card Issuer*. It is worth mentioning that card issuers can take on the roles of load acquirers. As shown in Figure 6, the *Merchant* depends on the *Buyer* (known as the cardholder on the CEP scheme) to pay using the *CEP Card*, on the *CEP Scheme Provider* to provide the cash free transaction infrastructure and on the *Card Issuer* to collect the money.

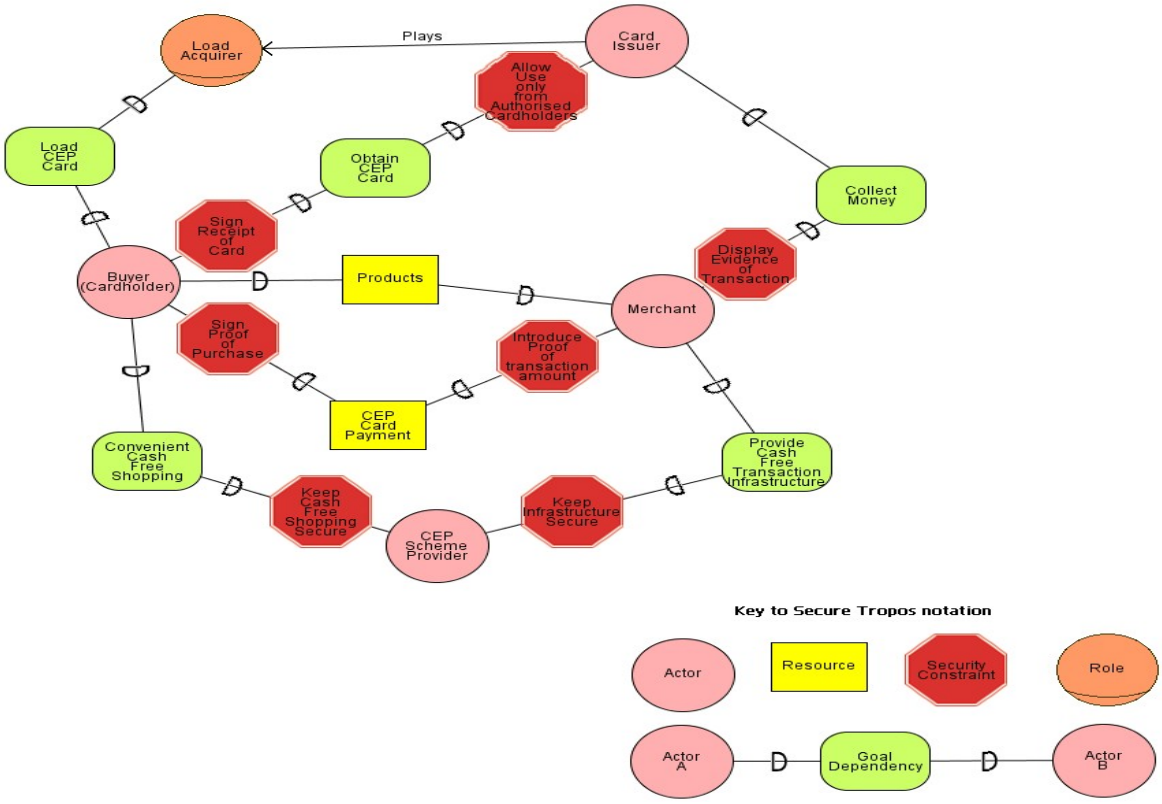

Figure 6: Actor Diagram of the CEP System

On the other hand, the *Buyer* depends on the *Card Issuer* to obtain a CEP enabled card, on the *Load Acquirer* to load the card and on the *CEP Scheme Provider* for convenient cash free shopping. As part of these dependencies, security related constraints are introduced, imposed by the different actors and the environment 5.1.4. For instance, the *Buyer* imposes to the *Card Issuer* the *Allow use only from authorised cardholder* security constraint as part of the Obtain CEP Card dependency. In turn, and in order to satisfy this constraint, the *Card Issuer* imposes two security constraints, one to the *Buyer* (*sign receipt of card*) and one to the *Merchant* (*Display evidence of transaction*). On the other hand, the *Merchant*, to satisfy the security constraint imposed by the *Card Issuer*, imposes two security constraints to the *Buyer* (*sign proof of purchase*) and the *CEP Scheme Provider* (*Keep infrastructure secure*). Apart from defining the dependencies and the security constraints of these dependencies, Secure Tropos allows developers to analyse each actor internally[6]. Figure 7 below illustrates part of the XML code

[6] Due to lack of space we do not illustrate in this paper the internal analysis of the actors. The modelling activities used for this can be found in [5].

generated by the SecTro tool, which represents the model shown in Figure 6.

```
<plays-link name="Plays Link">
<source name="Card Issuer" type="actor" />
<destination name="Load Acquirer" type="role" />
  </plays-link>
<dependency-link name="Collect Money">
<type="hard-goal" />
<source name="Merchant" type="actor"
<destination name="Card Issuer" type="actor" />
<security-constraint name="Display Evidence of Transaction" />
<security-constraint restricts="Merchant" />
  </dependency-link>
```

Figure 7: Partial XML Representation of the Actor Diagrams for the CEP System

6.2 Security Analysis of System

During this stage, the system is introduced as another actor who has a number of dependencies with the existing actors, and it accepts a number of responsibilities delegated to it by the other actors. For instance, for the CEP case study, the *CEP Scheme Provider* delegates the responsibility for administering the CEP transactions to the *CEP System*, whereas the *Merchant* delegates the CEP transaction resource to the *CEP System* (cf. Figure 8). With respect to security, since dependencies are delegated from the actors to the *CEP System*, possible security constraints regarding those dependencies are also delegated. In our case study, the *CEP Scheme Provider* actor together with the administer CEP transactions goal, delegates the *Keep transactions secure* security constraint on the *CEP system* actor. This means, that the *CEP System* is responsible now for satisfying that security constraint.

On the other hand, the introduction of the *CEP system* introduces new dependencies between the system and the existing actors. For example, the *CEP System* depends on the *Merchant* to get information regarding the transactions, such as the product information, the amount and so on. The *CEP System* also depends on the Buyer to get payment details such as the Buyer's card and account number. Moreover, these new dependencies impose extra security constraints on the *CEP System*. For instance, the *Buyer* wants their payment details to remain private so a security constraint is imposed to the *CEP System* from the *Buyer* as part of the *Get Payment Details* secure dependency. Similarly, the *Merchant* imposes a security constraint on the *CEP System* for the *Get Transaction Information* secure dependency.

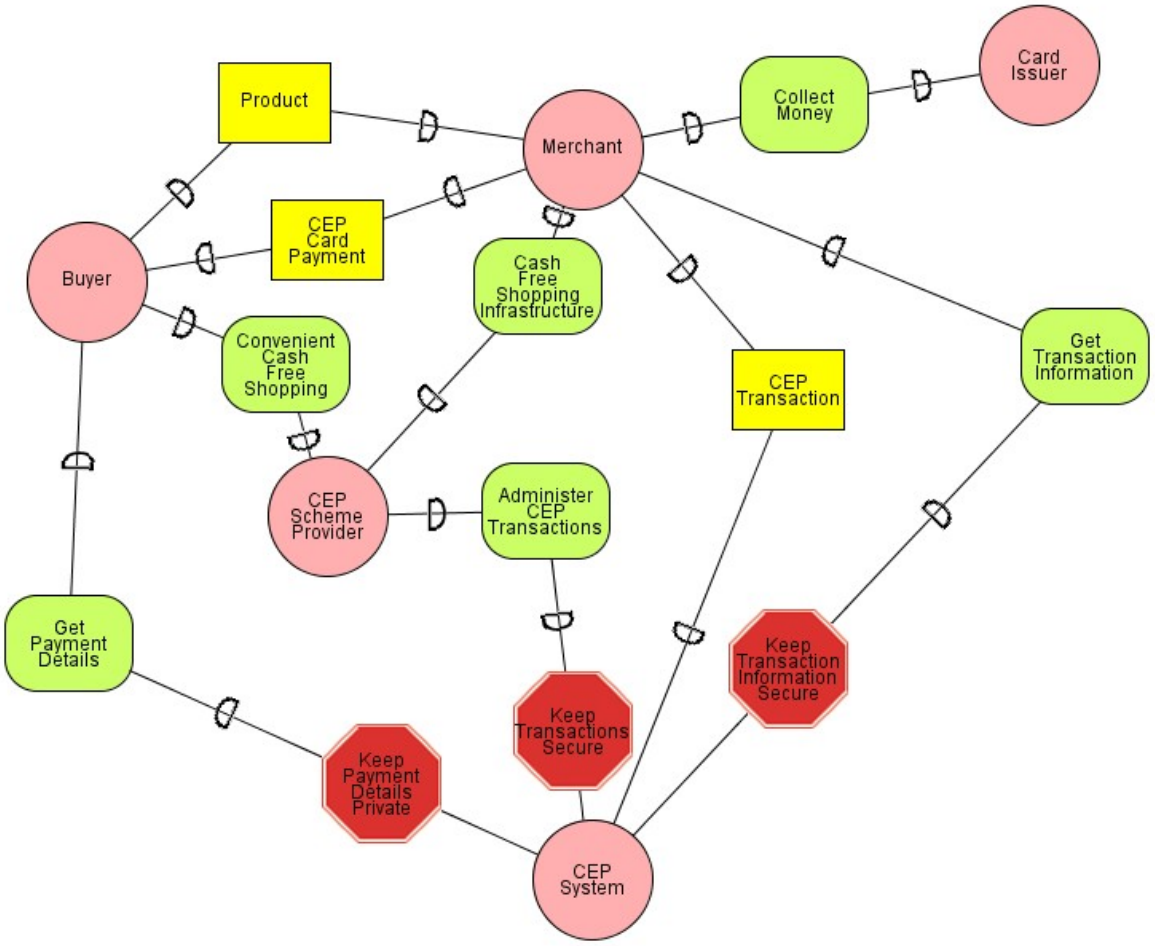

Figure 8: Actor Diagram including the CEP System

However, at this stage, the security constraints are defined at a high level which makes it hardly possible to truly understand the security implications of the imposed security constraints to the *CEP System*. Moreover, the system itself has not been defined in such a detail that it would allow developers to further analyse the security constraints. Therefore, the next step involves the internal analysis of the CEP system actors following the same analysis techniques used during the early requirements stage.

Due to lack of space, we focus our analysis for the rest of the case study to a central part of the *CEP System*, the purchase transaction. This is an off-line protocol that allows cardholders to use their electronic *CEP card* to pay for products. The internal analysis of the system for the purchase transaction results in the identification of the following main goals of the system: *process transaction data*, *store transaction data*, *adjust credit balance*, *display transaction details* and *provide proof of transaction*.

From the security point of view, secure goals are identified to satisfy the security constraints imposed initially from the other actors to the system. Moreover, the internal analysis of the system helps to identify security constraints that were not identified during the previous analysis or define in more details some existing security constraints. For instance, the *Keep transactions secure* security constraint imposed by the *CEP Scheme Provider* to the *CEP System* can now (that the system's goals have been identified) further defined. For example, related to the purchase transaction, the *Keep transaction secure* security constraint can be further refined to

constraints such as *keep transaction private*, *keep transaction available* and *keep integrity of the transaction*. These security constraints introduce more security constraints on the system such as *obtain user's authorisation details*, *authenticate all transactions* and so on. When all the goals, secure goals, entities and secure entities have been identified, the next stage of the process is the architectural design.

6.3 Secure System Design

The architecture of the system is defined with respect to its security requirements, and potential sub-actors are identified and the responsibility for the satisfaction of the system's goals and secure goals is delegated to these actors. Furthermore, the interactions of the newly identified sub-actors and the existing actors of the system are specified. In our case study, the sub-actors of the system, related to the purchase transaction, are the *Point-Of-Sale (POS) Device*, *the Purchase Security Application Module (PSAM)*, and the *Display*. Therefore, these actors are delegated responsibility for the system's goals (such as *Adjust Credit Balance*, *Process Transaction Data* and *Display Transaction Details*) and secure goals (such as *Perform Integrity Checks*, *Ensure Data Availability* and *Perform Cryptographic Procedures*). Moreover, this process allows developers to identify security constraints that could not be identified earlier in the development process. For instance, the *Merchant* and the *Buyer* now depend on the *POS Device* to deliver the resource *Proof of Transaction*. However, both these actors impose, as part of the *Proof Transaction* dependency, the security constraint *tamper resistant* to the *POS Device*. The *Buyer* imposes that constraint because he/she does not want to be charged more than the transaction amount, and the *Merchant* because he/she wants to make sure they will get the money displayed on the transaction. On the other hand, the *POS Device* actor, in turn, imposes that security constraint to the other actors involved with the resource proof of transaction, i.e. the *PSAM* and the *Display*. Therefore, security goals are introduced to the *PSAM* and the *Display* to satisfy the *tamper resistant* security constraint.

Moreover, a new actor is identified that interacts with the system, the *CEP Card*. In particular, the *Buyer* depends on the *CEP Card* actor to *pay for goods*. However, the *Buyer* imposes two security constraints to the *CEP Card* actor, to *verify the transaction* and to *be tamper-resistant*. Therefore, secure goals are identified for the *CEP Card* actor to satisfy these two security constraints. When all the security constraints and secure goals have been identified the next step in the development process involves the use of UMLsec to define more precisely some of the security related attributes of the identified actors. As indicated in the previous section the translation of Secure Tropos models to UMLsec models follows a set of guidelines and rules. In particular, the first step on this process is to translate the Secure Tropos analysis model to the UMLsec class diagram. Following the translation rules described in the previous section the UMLsec classes are identified. For example, the CEPS and PSAM actors have been translated to corresponding classes in the UMLsec model. Moreover, capabilities, dependencies and criticality of these actors have also been translated to the corresponding UMLsec model. A partial representation of the translated UMLsec class diagram is shown in Figure 9.

In particular, as our analysis[7] has shown, the participants involved in the off-line purchase transaction protocol are the customer's card and the merchant's POS device. The POS device

[7] It is worth stating that we focus our UMLsec analysis only on the security requirements without detailed explanation on how UMLsec supports the simultaneous analysis of security and functional requirement since this process has been extensively published in the literature 5.1.45.1.4.

contains a *Purchase Security Application Module* (PSAM) that is used to store new and processed data. As indicated in our analysis, the *PSAM* is required to be tamper-resistant. Moreover, following step 5 of our guidelines, UMLsec stereotypes are identified. For example, the sessions keys *SK* on the *PSAM* object are required to be fresh, therefore this is indicated using the {fresh} tag of UMLsec (see section 3 for {fresh}).

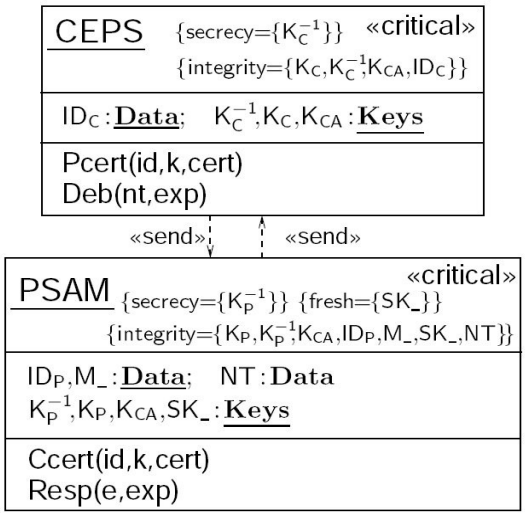

Figure 9: Partial UMLsec diagram for the CEP case study

Moreover, at this point in the case study, it is important, from the security point of view, to understand the physical distribution of the actors of the system, in order to specify in more detail the security attributes of the system's communication links. Following the steps of the second set of guidelines provided in the previous section, the deployment diagram of Figure 10 is constructed. To satisfy the security constraint *tamper resistant*, identified during the previous stage, for the *PSAM*, the *Display* and the *POS device*, the communication link between the *PSAM* and the *Display* is secured.

As shown in Figure 10, this is achieved by using a smart card with an integrated display as the *PSAM*. Furthermore, to satisfy the rest of the security constraints of our analysis, our design makes sure that the *PSAM* cannot be replaced without being noticed.

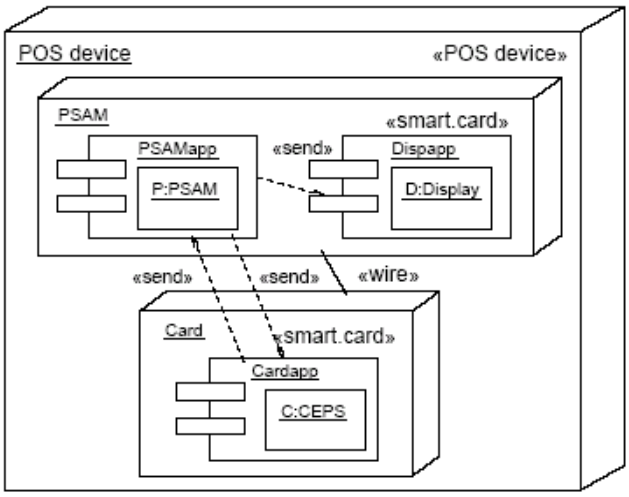

Figure 10: Partial Deployment diagram for the CEP case study

6.4 Secure components definition

The next step on the development involves the detailed design of each of the system components. During this stage each of the components identified in the previous stages is further specified by means of *Statechart Diagrams*, *Activity Diagrams*, and *Sequence Diagrams*[8]. Moreover, the UMLsec stereotypes allow us to specify the security constraints linked to the information flow and the processes carried out by the components.

UMLsec sequence diagrams are used to specify the security issues on the resulting sequences of states and the interaction with the component's environment. As an example, consider the sequence diagram depicted in Figure 11 for the purchase protocol:

At the beginning of its execution in the *POS device*, the *PSAM* creates a transaction number NT with value 0. Before each protocol run, NT is incremented. If a certain limit is exceeded, the *PSAM* stops functioning, to avoid rolling over of NT to 0. Note that here we assume an additional operation, the +, to build up expressions. The protocol between the *Card* C, the *PSAM* P, and the *Display* D starts after the *Card* C is inserted into a *POS* device containing P and D and after the amount M is communicated to the *PSAM* by typing it into a terminal assumed to be secure. Each protocol run consists of the parallel execution of the card's and the PSAM's part of the protocol. Both check the validity of the received certificate. If all the verifications succeed, the protocol finishes, otherwise the execution of the protocol stops at the failed verification.

[8] To keep the paper length to a reasonable size, we illustrate only sequence diagrams.

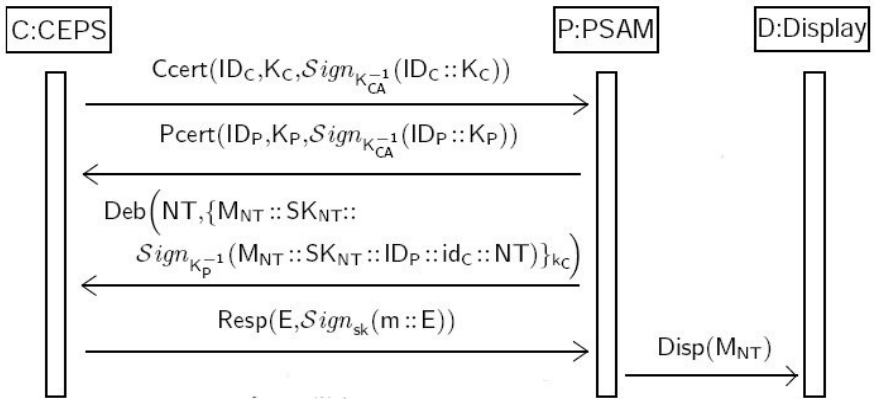

Figure 11: UMLsec Sequence Diagram for the Purchase Protocol

6.5 Verification

As mentioned above, it is important that an automated security analysis is performed to verify whether the security properties hold in the proposed design. We use the UMLsec tool framework to perform an automated security analysis of the above case-study. According to the assumptions of the CEPS, we need to consider a threat scenario where the attacker is able to access the POS device links, and can access other PSAMs over the Internet, but is not able to tamper with the smart cards. That is, we need to consider the insider attacker. Given this threat scenario, the tool needs to analyse the design for potential weaknesses for example with regards to the goal of merchant security, taking into account the fact that the CEPS may also be used over the Internet. The attacker could for example be an employee, which is a realistic scenario.

When invoking the tool with the UML diagrams including the threat scenario as an input, it generates the input for the automated theorem prover. For illustration purposes, the following is a small fragment of that input which specifies the assumptions on the initial knowledge of the adversary and the attack conjecture, in first-order logic (although this is not shown to the user and we do not have the space here to explain it in the technical details):

```
%-------- Attackers Initial Knowledge -----------

input_formula(previous_knowledge,axiom,(
 knows(conc(k_ca, conc(conc(id_a, conc(k_a, eol)), conc(inv(k_a),
conc(sign(conc(id_a, conc(k_a, eol)), inv(k_ca)), eol)))))
 )).

%----------------------- Conjecture ------------------------

input_formula(attack,conjecture,(
   knows(sign(conc(id_p, conc(id_a, conc(m_nt, conc(nt, eol)))),
inv(k_p))) )).
```

The automated theorem prover that is called by the UMLsec tool framework then investigates whether the attack conjecture can be derived from the logical formulae formalizing the assumptions on the system and the adversary. The output from the automated theorem prover is

then interpreted by the UMLsec tool, and its interpretation regarding the security requirements under analysis are output to the user. In our application, the result is that the design is secure regarding the security requirements and the threat scenario explained above.
It is also important to mention that the employment of our approach to the presented case study, identified a number of security requirements that were missing from the original specifications of the CEPS system. In particular, the original CEPS specification requires the *CEP card* and the *PSAM* to be tamper-proof but not the *POS device*. This leads to the following weakness with respect to security. The *POS device* is not secure against a potential attacker who may try to betray the *Merchant*, for example some of his/her employees, by replacing the *PSAM* and manipulating the *Display*. The idea of the attack is that the attacker redirects the messages between the *Card* C and the *PSAM* P to another PSAM P', for example with the goal of buying electronic content and let the card-holder pay for it. We assume that the attacker manages to have the amount payable to P' equal the amount payable to P. The attacker also sends the required message to the display which will then reassure the merchant that he has received the required amount.

Our goal oriented analysis identified a security constraint (and consequently a security requirement) indicating that the communication link between the PSAM and the Display needs to be secure. This security requirement was later on realized by the design, presented above, by making sure that the *PSAM* cannot be replaced without being noticed. This guarantees that the *Display* cannot anymore be manipulated, which means that if the *PSAM* received less money than expected, it would be noticed immediately.

7 Software engineering for secure systems development: Related work

We are not the first to argue for the need to consider security as part of the development process, and there is already a number of approaches towards this goal. In fact, there is a diversity of efforts ranging from the definition of security models, to the development of security pattern languages. Initial efforts to consider security in the development of software systems focused on the definition of security models. Various models [23,24,25] have been proposed based on mandatory access control (MAC), discretionary access control (DAC) and role base access control (RBCA).

Initial work on eliciting security requirements based on Goal-Driven Requirements Engineering (GDRE) produced a number of methods [26,27] and processes for reasoning about non-functional requirements, including security. Anton et al. [28], proposed a set of general taxonomies for security and privacy, to be used as a general knowledge repository for a (security) goal refinement process. A number of researchers use goal-oriented approaches to analyse and model the behaviour of potential attackers. Van Lamsweerde and Letier [11] use the concept of security goals and anti-goals. Similarly, Crook et al. [30] introduce the notion of anti-requirements to represent the requirements of malicious attackers. Anti-requirements are expressed in terms of the problem domain phenomena and are satisfied when the security threats imposed by the attacker are realised in any one instance of the problem. Lin et al. [31], incorporate anti-requirements into abuse frames. The purpose of abuse frames is to represent security threats and to facilitate the analysis of the conditions in the system in which a security violation occurs.

The development of methods to analyse and reason about security based on goals and the relationships between actors (such as users, stakeholders and attackers) and the system is also an important area of research. Liu et al. [33] have presented work to identify security requirements

during the development of multi-agent systems. Mouratidis [5] has proposed Secure Tropos to deal with the modelling and reasoning of security requirements and their transformation to a design that satisfies them. Giorgini at al. [14] have introduced an enhancement of Tropos that is based on the clear separation of roles in a dependency relation between those offering a service, those requesting the service, and those owning the very same data.

Moreover, a number of security requirements frameworks have been proposed. Mead [16] proposed the Security Quality Requirements (SQUARE) Method for documenting security requirements whereas Haley et al [29], provide an approach for security requirements elicitation, specification and analysis.

Works have also been presented that extended use cases with respect to security analysis. In particular, McDermott and Fox [12] adapt use cases to capture and analyse security requirements, and they call the adaption an abuse case model. Similarly, Sindre and Opdahl [13] define the concept of misuse case, the inverse of use case, which describes a function that the system should not allow. Alexander [32] adds Threatens, Mitigates, Aggravates links to the use case diagram. In addition, a large number of efforts in the area of secure design engineering is focused on extending existing methods and languages for software systems development. Hermann and Pernul [34] extend Entity-Relationship Diagrams and Data Flow Diagrams to include security related concepts and methods. Whitmore [35] presents a secure systems design method based on extensions to the Common Criteria. Viega and McGraw [36] proposed ten (10) principles for building secure software.

Another direction of work is based on the extension of the Unified Modelling Language (UML). Jürjens proposes UMLsec [6][19], an extension of the Unified Modelling Language (UML), to include the modelling of security related features, such as confidentiality and access control. Lodderstedt et al. [37] extend UML to model role-based access control. In their work, they present a security modelling language called SecureUML. Epstein and Sandhu [38] introduce a UML-based notion for RBAC. Koch et al. [39] describe an approach to the specification of Access Control policies in UML by means of UML class and object diagrams that can be modelled with existing UML tools. The methodology, along with the graph-based formal semantics for the UML access control specification, allows to reason about the coherence of the access control specification. [43] presents a model-based approach for the specification of user rights in the context of an object oriented use case driven development process.

A number of researchers have focused their efforts on applying the pattern approach to the security problem. For a current overview see [40]. A set of agent security patterns has also been defined by Mouratidis et al. [41]. On the other hand, Schneier [42] describes attack trees as a useful way to identify and organise different attacks in an information system.

Although important and useful the above approaches illustrate a number of important limitations. For instance, the Non Functional Requirement frameworks consider security as a vague goal to be satisfied whereas a precise description and enumeration of specific security properties is still missing. An important limitation of the use-case related approaches and some of the security requirements efforts is that they do not support the modelling and analysis of security requirements at a social level but they treat security in system-oriented terms. In other words, they lack models that focus on high-level security requirements, meaning models that do not force the designer to immediately go down to security requirements. On the other hand, other approaches to security requirements, such as the i* based approaches and the security requirement framework approaches, only guide the way security can be handled within a certain stage of the software development process and they fail to provide a structure process to assist software engineers throughout the development.

Moreover, the secure systems design methods, such as the UML-based and the security pattern

approaches, are mainly solutions focused on the later stages of the software development process. Such approaches are useful for recording the results of decisions, but do not offer support for arriving at those decisions [9]. Moreover, they only focus on the technical dimension of security and neglect the social aspects of it.

8 Conclusions

A large number of Goal-Oriented Requirements Engineering approaches have been proposed in the literature, which focus on eliciting security requirements. However, most of these approaches provide little help as to how security requirements, once elicited, can be realized in the design stage and how the developed design can be verified against the security requirements of the system. To overcome these limitations, in this paper, we have presented the integration of two prominent approaches, a Goal-Oriented Security Requirements Engineering approach called Secure Tropos and a Model-Based Security Engineering approach called UMLsec. For security-critical systems, the proposed approach allows one to elicit and reason about security requirements from early on in the development process, within the development context, and in a seamless way through the development cycle. Then, one can check that the system fulfills the relevant security requirements on the design level by analyzing the model. Moreover, one can also use our analysis techniques and tools within a traditional software engineering context, or where one has to incorporate legacy systems that were not developed in a model-based way. Here, one starts out with the source code. Our tools extract models from the source code, which can then again be analyzed against the security requirements. Moreover, the integration of the two approaches allows software engineers to incorporate the configuration data (such as user permissions) in the analysis, which is very important for security but often neglected.

Although the results from the integration of the two approaches are promising, there are a number of areas that need further work. Currently, the translation from the secure Tropos models to the UMLsec models is not entirely automated. It is important however, that fully automated support is provided to assist developers and save time especially in large-scale projects. Moreover, although the case study used for the evaluation of the approach is appropriate for this type of evaluation, it is important that we employ the methodology to larger scale projects to understand its scalability limitations. It is also important to allow developers, without any previous experience in the methodology, to use the methodology and report back. This will allow us to understand any limitations of our work from an external point of view.

References

[1] H. Mouratidis, P. Giorgini, Integrating Security and Software Engineering: Advances and Future Visions, Idea Group Publishing, 2006.

[2] J. Saltzer and M. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278–1308, September 1975.

[3] R. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, New York, 2001.

[4] H. Mouratidis, J. Jürjens, J. Fox, Towards a Comprehensive Framework for Secure Systems Development, CAiSE 2006, Lecture Notes in Computer Science 4001, pp. 48-62, Springer-Verlag, 2006

[5] H. Mouratidis. A Security Oriented Approach in the Development of Multiagent Systems: Applied to the Management of the Health and Social Care Needs of Older People in England. PhD thesis, University of Sheffield, U.K., 2004

[6] J. Jürjens, Secure Systems Development with UML, Springer-Verlag, 2004

[7] J. Jürjens, P. Shabalin, Tools for Secure Systems Development with UML. International Journal on Software Tools for Technology Transfer, Springer, Volume 9, Numbers 5-6 / October, 2007, pp. 527-544.

[8] H. Mouratidis, P. Giorgini, Integrating Security and Software Engineering: An Introduction, in Integrating Security and Software Engineering: Advances and Future Actions, pp. 1-14, Idea Publishing Group, 2006.

[9] E. Yu, L. Liu, J. Mylopoulos, A social Ontology for Integrating Security and Software Engineering, in Integrating Security and Software Engineering: Advances and Future Actions, pp. 70-105, Idea Group Publishing, 2006
[10] P. Devanbu, S. Stubblebine. Software Engineering for Security: a Roadmap. In Proceedings of ICSE 2000 ("the conference of The future of Software engineering"), 2000.
[11] A. van Lamsweerde, E. Letier, Handling Obstacles in Goal-Oriented Requirements Engineering, IEEE Transactions on Software Engineering, Special Issue on Exception Handling, Vol 26, n° 10, October 2000, pp. 978-1005
[12] J. McDermott, C. Fox. Using Abuse Care Models for Security Requirements Analysis. In Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.
[13] G. Sindre, A. L. Opdahl, Eliciting Security Requirements with Misuse Cases, *Requirements Engineering*, 10(1):34-44, January 2005
[14] P. Giorgini, F. Massacci, J. Mylopoulos and N. Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. In Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05), IEEE Computer Society Press, 29 August - 2 September 2005
[15] W. Stallings, Cryptography and Network Security: Principles and Practice, Prentice Hall, 2nd ed. 1999.
[16] N.R. Mead, Identifying Security Requirements Using the Security Quality Requirements Engineering (SQUARE) Method, in Integrating Security and Software Engineering, pp. 44-69, Idea Publishing Group, 2006.
[17] H. Mouratidis, P. Giorgini, and G. Manson. When Security Meets Software Engineering: A Case of Modeling Secure Information Systems. In Information Systems, Elsevier, Vol. 30, Issue 8, pp. 609-629, Elsevier,2005
[18] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A Perini. TROPOS: An Agent Oriented Software Development Methodology. In Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers Volume 8, Issue 3, Pages 203-236, 2004
[19] Jan Jürjens: Sound methods and effective tools for model-based security engineering with UML. ICSE 2005, ACM, pp. 322-331, 2005
[20] Jan Jürjens: Security Analysis of Crypto-based Java Programs using Automated Theorem Provers. ASE 2006, IEEE Computer Security, pp. 167-176, 2006
[21] W. Muhanna, W., An Object-Oriented Framework for Model Management and DSS Development, *Decision Support Systems*, **9**:2, pp. 217-229, 1993
[22] CEPSCO, Common Electronic Purse Specifications, Business Requirements ver. 7, Functional Requirements ver. 6.3, Technical Specification ver. 2.2. Available from http://www.cepsco.com, 2000.
[23] E. D. Bell, L. J. LaPadula,, Secure Computer Systems: Mathematical Foundations, MITRE Corporation, 1973
[24] D. F. C. Brewer, M. J. Nash, The Chinese Wall Security Policy, IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY, May, Oakland, California, pp 206-214, 1989
[25] K. J. Biba, Integrity Considerations for Secure Computer Systems, MTR-3153, The MITRE Corporation, 1977.
[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] E. Yu, L. Cysneiros, Designing for Privacy and Other Competing Requirements, 2nd Symposium on Requirements Engineering for Information Security (SREIS' 02), Raleigh, North Carolina, 15-16 November, 2002.
[28] A. I. Antón, J. B. Earp, A Requirements Taxonomy to Reduce Website Privacy Vulnerabilities, Requirements Engineering Journal, Springer Verlag, 9(3), pp. 169-185, August 2004.
[29] C. B. Haley, R. Laney, J. D. Moffett, B. Nuseibeh, Arguing Satisfaction of Security Requirements, in Integrating Security and Software Engineering: Advances and Future Visions, pp. 16-43, Idea Publishing Group, 2006
[30] R. Crook, D. Ince, B. Nuseibeh, Modelling Access Policies Using Roles in Requirements Engineering , Information and Software Technology, 45(14):979-991, Elsevier, November 2003
[31] L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett, Introducing Abuse Frames for Analysing Security Requirements , Poster Paper, Proceedings of 11th IEEE International Requirements Engineering Conference (RE'03), 371-372, Monterey, USA, 8-12 September 2003.
[32] I. Alexander, Misuse Cases: Use Cases with Hostile Intent , IEEE Software, Vol 20, No 1, pp. 58-66, 2003
[33] L. Liu, E. Yu, J. Mylopoulos, Analyzing Security Requirements as Relationships Among Strategic Actors, in the Proceedings of the 2nd Symposium on Requirements Engineering for Information Security (SREIS'02), Raleigh, North Carolina, October 2002.
[34] G. Hermann, G. Pernul. Viewing business-process security from different perspectives. International Journal of electronic Commence 3:89-103, 1999
[35] J. J. Whitmore, A method for designing secure solutions, IBM Systems Journal, 40(3), pp. 747-768, 2001
[36] J. Viega and G. McGraw. Building a Secure Software. Addison-Wesley, Reading, MA, 2002.
[37] T. Lodderstedt, D. A. Basin, J. Doser, SecureUML: A UML-Based Modeling Language for Model-Driven Security, Proceedings of the 5th International Conference on the unified Modelling Language, Lecture Notes in Computer Science, 2460, pp. 426-441, 2002.
[38] P. Epstein, R. Sandhu, Towards a UML based approach to role engineering, Proceedings of the 4th ACM workshop on role-based access control, Virginia, USA, pp. 135-143, 1999.
[39] M. Koch, F. Parisi-Presicce, Access Control Policy Specification in UML, Proceedings of the Critical Systems Development with UML, Satellite Workshop of UML, pp. 63-78, 2002
[40] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad, Security Patterns: Integrating Security and Systems Engineering, Willey, 2005

[41] H. Mouratidis, M. Weiss, P. Giorgini, Modelling Secure Systems Using an Agent-Oriented Approach and Security Patterns, *International Journal on Software Engineering and Knowledge Engineering*, 16(3), 471-498, 2006
[42] B. Schneier. Secrets & Lies: Digital Security in a Networked World, John Wiley & Sons, 2000
[43] Ruth Breu, Gerhard Popp, Muhammad Alam: Model based development of access policies. STTT 9(5-6): 457-470 (2007)