Towards a Comprehensive Framework for Secure Systems Development

Haralambos Mouratidis[1], Jan Jürjens[2], and Jorge Fox[2]

[1] Innovative Informatics, School of Computing and Technology, University of East London, UK
haris@uel.ac.uk
[2] Software and Systems Engineering, TU Munich, Germany
juerjens@in.tum.de, fox@in.tum.de

Abstract. Security involves technical as well as social challenges. In the development of security-critical applications, system developers must consider both the technical and the social parts. To achieve this, security issues must be considered during the whole development life-cycle of an information system. This paper presents an approach that allows developers to consider both the social and the technical dimensions of security through a structured and well defined process. In particular, the proposed approach takes the high-level concepts and modelling activities of the secure Tropos methodology and enriches them with a low level security-engineering ontology and models derived from the UMLsec approach. A real case study from the e-commerce sector is employed to demonstrate the applicability of the approach.

1 Introduction

Security related challenges and problems fall into two categories: *technical challenges*, i.e. those related to the available technology and the infrastructure of information systems, and *social challenges*, i.e. those related to the impact of the human factor on the security of a system. To be able to develop secure information systems, both dimensions should be considered simultaneously. Consider for instance, a typical social engineering attack on health information systems. Social engineering is a non-technical kind of intrusion that relies on human interaction and involves tricking other people (doctors, or nurses in the case of medical records) to break normal security procedures. A private detective (or someone interested in obtaining personal health information) calls in a health professional's office or a hospital, introduces herself as a doctor in an emergency hospital and asks information about the medical record of a particular patient [22]. This example shows that considering only the technical dimension of security, will not produce the desirable output.

To enable developers to deal with both dimensions, research has shown that security should not be considered in isolation but within the context of the development process employed to develop the system [7][13][16][8]. However, it has remained true over the last 30 years, since the seminal paper [16], 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 subgoals [17], as well as a large body of security engineering

knowledge accumulated [1]. In contrast, today ad hoc development leads to many deployed systems that do not satisfy important security requirements. Thus a sound methodology supporting secure systems development is needed. Such a methodology should take into account not only the technical problems but also the social dimension of developing secure information systems.

Our goal is to work towards the development of such methodology. This paper presents an approach for modelling secure information systems, which takes the high-level concepts and modelling activities of the secure Tropos methodology [3] and enriches them with a low level security-engineering ontology and models derived from the UMLsec [11] approach. More concretely, we present an approach that integrates two complementing security-oriented approaches: secure Tropos and UMLsec. Section 2 provides an overview of secure Tropos and UMLSec, and section 3 discusses their integration. Section 4 illustrates the enhanced framework with the aid of a use case, and section 5 concludes the paper.

2   An Overview of Secure Tropos and UMLsec

Secure Tropos [13][14] is a security-oriented extension of the well known[1] Tropos methodology. Tropos provides support for four phases [3]: *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.

Secure Tropos introduces security related concepts to the Tropos methodology, to enable developers to consider security issues throughout the development of informat-ion systems. 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 [13]. Additionally, secure Tropos defines secure dependencies. A *secure dependency* introduces security constraint(s) that must be fulfilled for the dependency to be satisfied [14]. 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.

UMLsec is an extension of UML [15] for secure systems development. Recurring security requirements, such as secrecy, integrity, and authenticity are offered as

---

[1] In the requirements engineering area.

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». 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 [11]. The extension has been developed based on experiences on the model-based development of security-critical systems in industrial projects involving German government agencies and major banks, insurance companies, smart card and car manufacturers, and other companies. There have been several applications of UMLsec in industrial development projects.

3   Integration 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 and identified. Then, the methodology continues to 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. However, the developers of the methodology do not focus on the detailed security specification of each component of the system. The UMLsec approach is on the other side of the spectrum. It 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. We believe that integrating these two approaches will lead us to a complementary approach for secure information

systems development, which will consider the two dimensions of security. In particular, we have identified individual strengths of such integration, 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 automatic validation/verification of security properties. ***Combinational***, both of the approaches are extensions of well-known approaches (Tropos and especially UML) and this makes the approach easier accessible to a large number of researchers/developers. Also, the strength of secure Tropos (requirements analysis) is complementary to the strengths of UMLsec (design) 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.

As mentioned above, secure Tropos is particularly focused on the *Early Require-ments Analysis*, *Late Requirements Analysis*, and *Architectural Design*, whereas for the *Detailed Designed* stage the methodology is mainly based on UML diagrams with minor extensions to indicate some security issues [13]. On the other hand, the strength of UMLsec can be found on the *architectural* and *detailed* design stages, while some weak support for *late requirements* can be introduced using use case diagrams. Therefore, the integration of the two methods provides a framework of particular strength throughout all the development stages as shown in Figure 1.

However, the integration of secure Tropos and UMLsec is not straightforward and we had to deal with various challenges. To overcome these, a set of mapping guidelines and steps were defined and the secure Tropos development process was redefined and enriched with extra methods and procedures.
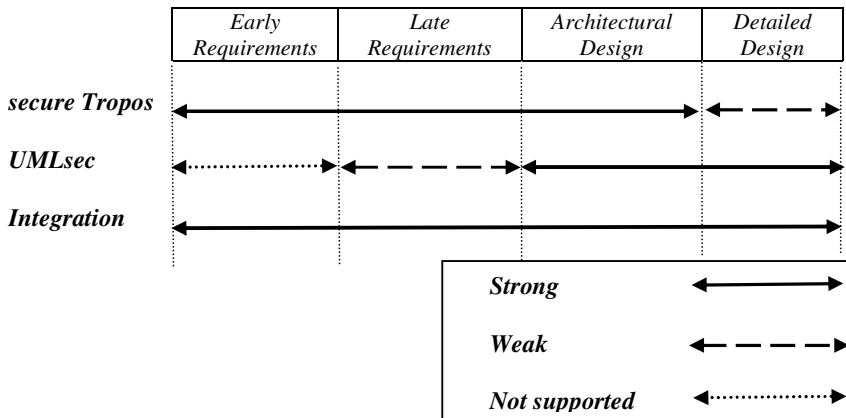

Fig. 1. Secure Tropos and UMLsec coverage of development phases

3.1   Mapping the Secure Tropos Models to UMLsec Models

As it was mentioned above, the appropriate stage of the secure Tropos development process for the integration is the *architectural design* stage. However, different concepts and notations are used by secure Tropos and UMLsec. So, the first challenge involved the definition of a set of 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: Map the Secure Tropos Analysis Model to UMLsec Class Diagram

- **Step 1. Identify the UMLsec classes:** For every actor on the secure Tropos actor diagram a class is created on the UMLsec class diagram. In case there are sub-actors, these are mapped into the UMLsec class diagram as an inheritance relationship pointing from the sub-actor class to the main actor class.
- **Step 2. Identify the operations of the UMLsec classes:** The capabilities of each of the actors mapped into the UMLsec classes are added on the corresponding class as operations.
- **Step 3. Identify the attributes of the UMLsec classes:** Resources related to each of the actors are mapped to attributes on the UMLsec diagram. This is not a 1-to-1 mapping, meaning that a UMLsec class will not have exactly the same number of attributes as the secure Tropos actor counterpart. The reason for this is that 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 class diagram.
- **Step 4. Identify associations:** In order to identify any associations between the UMLsec classes, the dependencies of the secure Tropos actor diagram are taken into account. Each dependency might provide an association. However, this is not a strict rule, and in fact in some cases developers will identify one association for a number of dependencies. This is again due to the reason that secure Tropos models are analysis models and UMLsec are design models so they contain more information.
- **Step 5. Identify UMLsec stereotypes:** UMLsec stereotypes are identified through the secure dependencies. The type of the 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. The classes corresponding to critical actors are indicated with the <<critical>> stereotype.

Guideline 2: Map the Secure Tropos Analysis Model to UMLsec Deployment Diagram

The actor diagrams of the secure Tropos methodology contain two types of actors, external and internal, without differentiating between them. However, in UMLsec deployment diagrams, nodes and components need to be defined together with their communications and any security related stereotypes. The following steps are defined:

- **Step 1. Identify UMLsec nodes and components:** Define at least one "user" and one "system" nodes. A "user" node represents one or more external actors of the system, whereas the "system" node represents the system. External actors should be

modelled as components on the appropriate "user" node, whereas system's internal actors must be modelled as components of the "system" node.

- **Step 2. Mode of communication:** Identify the mode of communication between the different nodes and use UMLsec stereotypes to denote that mode. For example, if the internet is used as the mode of communication between user node X and system node Y, then the <<internet>> UMLsec stereotype should be employed to denote that communication.
- **Step 3. Identify the necessary security stereotypes:** Consider the security constraints from the secure Tropos model. At least one UMLsec stereotype should be identified for each security constraint. 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.

3.2   The New Process

In a nutshell, the redefined secure Tropos process, allows developers initially to employ secure Tropos concepts and modelling activities to identify and analyse the security requirements of the system-to-be. Then, a combination of secure Tropos and UMLsec is employed to determine a suitable architecture for the system with respect to the identified security requirements, and identify the components of the system along with their secure capabilities, protocols, and properties. During the last stage UMLsec is used to specify in detail the components, which were identified in the previous stage, with respect to security.

In particular, during the *Early Requirements Analysis* the security needs of the stakeholders are analysed and a set of security constraints are imposed to the actors that satisfy the identified security needs. Moreover, security goals and entities are identified, for each of the participating actors, to satisfy the imposed security constraints. To achieve this, developers employ a set of different, but related, modelling activities defined by secure Tropos and its diagrammatic notations, such as actor's and the goal diagrams [13]. During the *Late Requirements Analysis,* the security requirements of the system are identified taking into account the security needs of the stakeholders as well as their security constraints (identified during the analysis of the previous stage). The output of this stage will be the definition of the system's security requirements together with a set of security constraints, along with the system's security goals and tasks that allow the satisfaction of the security requirements of the system.

The main aim of the *Architectural Design* is to define the architecture of the system with respect to its security requirements. To achieve this, initially secure Tropos notation together with a set of security patterns [13] are used to determine the general architecture and the components of the system, then the secure Tropos models are mapped to UMLsec models and in particular UMLsec Class and Deployment diagrams. These are used to model the security protocols and properties of the architecture.

During *Detailed design*, UMLsec is used to specify in detail the components of the system identified in the previous stage. For this reason, UMLsec activity diagrams are used to define explicitly the security of the components, UMLsec sequence diagrams are used to model the secure interactions of the system's components, e.g., to determine if cryptographic session keys exchange in a key exchanged protocol remain confidential

in considering 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. Figure 2 illustrates the redefined development process. Highlighted in italic are the new activities due to the integration of the two approaches.
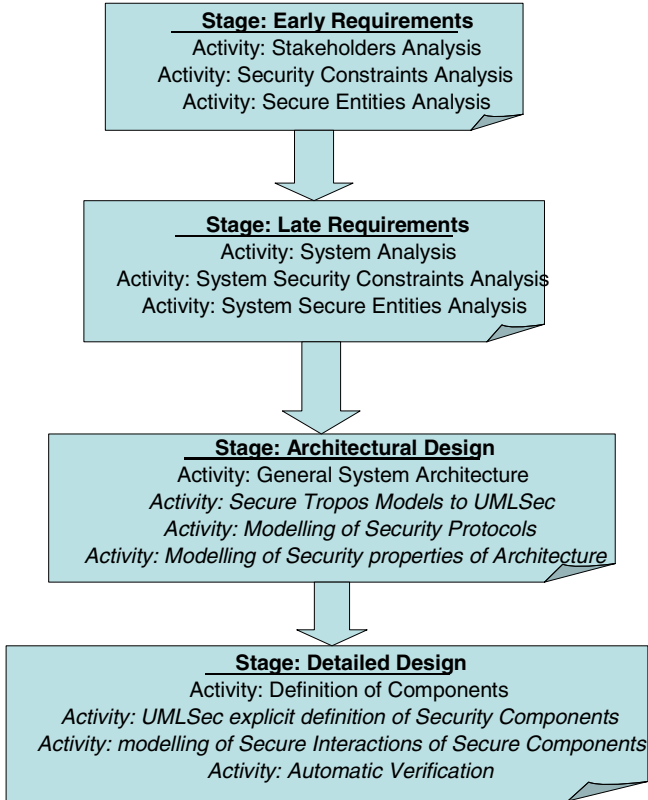
Stage: Early Requirements
Activity: Stakeholders Analysis
Activity: Security Constraints Analysis
Activity: Secure Entities Analysis

Stage: Late Requirements
Activity: System Analysis
Activity: System Security Constraints Analysis
Activity: System Secure Entities Analysis

Stage: Architectural Design
Activity: General System Architecture
Activity: Secure Tropos Models to UMLSec
Activity: Modelling of Security Protocols
Activity: Modelling of Security properties of Architecture

Stage: Detailed Design
Activity: Definition of Components
Activity: UMLSec explicit definition of Security Components
Activity: modelling of Secure Interactions of Secure Components
Activity: Automatic Verification

Fig. 2. The redefined development process

4   Case Study

To demonstrate our approach, we employ a case study from the e-commerce domain: The Common Electronic Purse System (CEPS) [4]. 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[2].

Amongst others, the following participants are defined in a CEP transaction [4]: the ***Scheme Provider***, the authority responsible for establishing an infrastructure for the overall functionality and security of the CEP system and enforcing the operating

---

[2] 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.

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.

4.1 Early Requirements

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 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 3, 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.
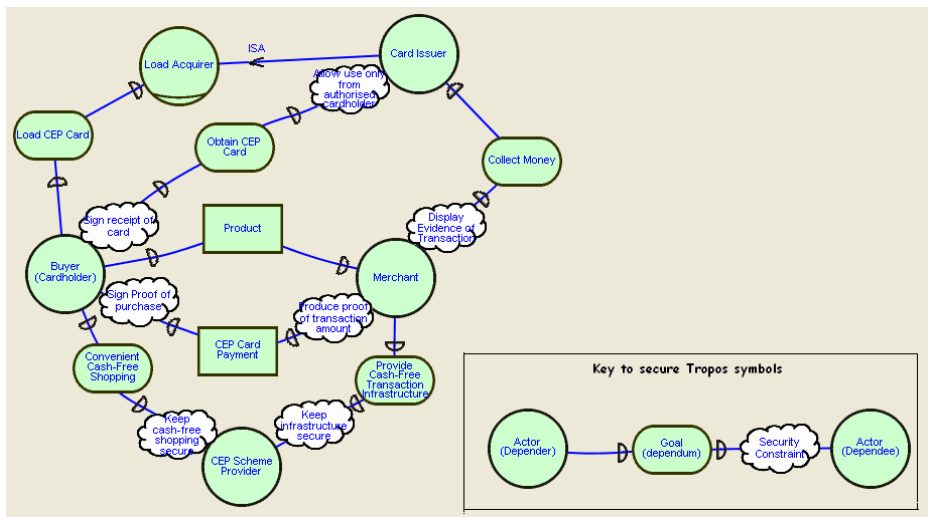

Fig. 3. Actor diagram of the CEP System

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 [13]. 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[3].

4.2   Late Requirements Analysis

During the late requirements analysis, 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 4). 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

Fig. 4. Actor diagram including the CEP System

---

[3] 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 [13].

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.

However, at this stage, the security constraints are defined at a high level which makes it impossible (and impractical) 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 can 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) be 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.

4.3 Architectural Design

During the architectural 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 Section 3.1 the first step on this process is to map the Secure Tropos analysis model to the UMLsec class diagram. Following the first four steps described in section 3.1 the UML classes are identified as shown in Figure 5.
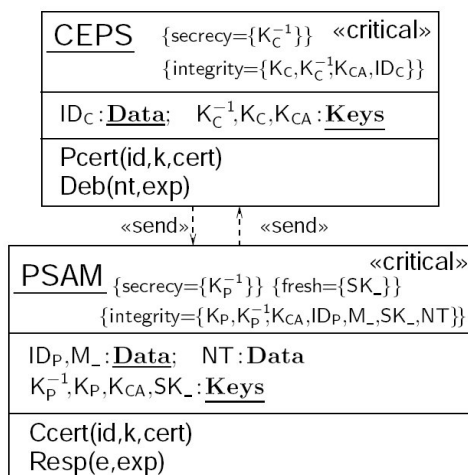
In particular, as our analysis has shown, the participants involved in the off-line purchase transaction protocol are the customer's card and the merchant's POS device. The POS device contains a *Purchase Security Application Module* (PSAM) that is used to store new and processed data. As indicated in 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 2 for {fresh}).

Following the steps of the second guideline provided in section 3.1, the deployment diagram of figure 6 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.
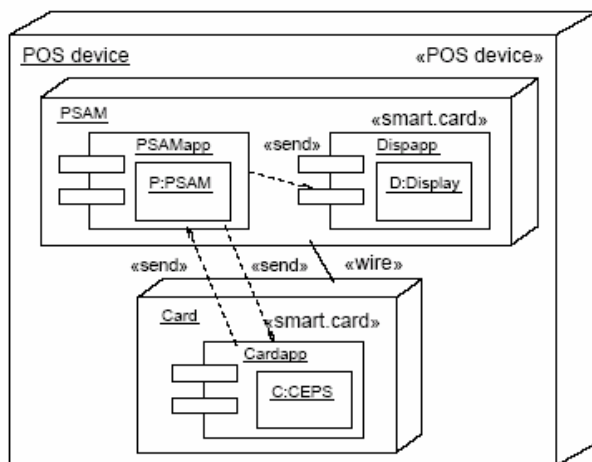
As shown in Figure 6, this is achieved by using a smart card with an integrated display as the *PSAM*. Furthermore, to satisfy the rest of the security constraints of our analysis, our design makes sure that the *PSAM* cannot be replaced without being noticed.

4.4 Detailed Design

The next step on the development involves the detailed design of each of the system components.

CEPS $\{secrecy=\{K_C^{-1}\}\}$ «critical»
$\{integrity=\{K_C,K_C^{-1},K_{CA},ID_C\}\}$

$ID_C:\mathbf{Data};$ $K_C^{-1},K_C,K_{CA}:\mathbf{Keys}$

Pcert(id,k,cert)
Deb(nt,exp)

«send»   «send»

PSAM $\{secrecy=\{K_P^{-1}\}\}$ $\{fresh=\{SK\_\}\}$ «critical»
$\{integrity=\{K_P,K_P^{-1},K_{CA},ID_P,M\_,SK\_,NT\}\}$

$ID_P,M\_:\mathbf{Data};$ $NT:\mathbf{Data}$
$K_P^{-1},K_P,K_{CA},SK\_:\mathbf{Keys}$

Ccert(id,k,cert)
Resp(e,exp)

Fig. 5. Partial UMLSec diagram for the presented case study

Fig. 6. Deployment diagram of the case study

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*[4]. 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 following diagram for the purchase protocol:

---

[4] Due to lack of space we illustrate only sequence diagrams.

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.
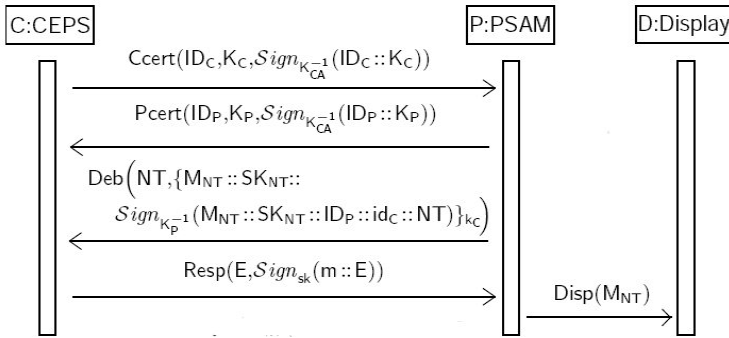

Fig. 7. UMLSec sequence diagram for the purchase protocol

4.5 Discussion

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 cardholder 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.

In our design such attack will fail. Our analysis and design improves the initial CEPS specification by securing the communication link between the *PSAM* and the *Display*, and by making sure that the *PSAM* cannot be replaced without being noticed. This will guarantee that the *Display* cannot anymore be manipulated, which means that if the *PSAM* received less money than expected, it would be noticed immediately.

5 Conclusions

Because of their wide-spread use in security-critical applications, information systems have to be secure. Unfortunately, the current state of the art in the development of

security-critical information systems is far from satisfactory. A sound methodology to consider the technical as well as the social dimension of security is needed.

Towards this goal, we have presented the integration of two prominent approaches to the development of secure information systems: secure Tropos and UMLsec. The main feature of our proposal is the integration of the strong parts of each of these approaches, namely the socially oriented part of the secure Tropos methodology and the technical part of the UMLsec. This achieves several goals. First of all, developers are able to consider security both as a social aspect as well as a technical aspect. As it was argued in the introduction, this is important when developing information systems. Secondly, the approach allows the definition of security requirements in different levels and as a result it provides better integration with the modelling of the system's functionality. Thirdly, security is not considered in isolation but simultaneously with the rest of the system requirements. Fourthly, the integration allows the consideration of the organisational environment for the modelling of security issues, by facilitating the understanding of the security needs in terms of the security policy and the real security needs of the stakeholders, and then it allows the transformation of the security requirements to a design that is amenable to formal verification with the aid of automatic tools. It is worth mentioning at this point, that advance tool support is provided to assist with our approach [10]. The developed tool can be used to check the constraints associated with UMLsec stereotypes mechanically, and it uses 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 UMLsec model, where the weaknesses that were found are highlighted. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes.

To demonstrate the practical applicability and usefulness of our approach we have applied it to the CEP case study. The results are promising since our analysis in fact improves the security of the system. A large number of research efforts related to our work has been presented in the literature [2][5][6][8][9][12][19][23]. However, our work is different in two main points. Existing work is mainly focused either on the technical or the social aspect of considering security, and it presented approaches applicable only to certain development stages. In contrast our approach considers security as a two dimensional problem, where the technical dimension depends on the social dimension. Moreover, our approach is applicable to stages from the early requirements to implementation.

References

1. Anderson, R., *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, New York, 2001.
2. Basin, D., Doser, J., Lodderstedt, T., Model Driven Security for Process Oriented Systems. In Proceedings of the 8th ACM symposium on Access Control Models and Technologies, Como, Italy, 2003
3. Bresciani, P. Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A., 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

4. 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].

5. Crook, R., Ince, D., Lin, L., Nuseibeh, B., Security Requirements Engineering: When Anti-requirements Hit the Fan, In Proceedings of the 10th International Requirements Engineering Conference, pp. 203-205, IEEE Press, 2002

6. Cysneiros, L.M. Sampaio do Prado Leite, J.P., Nonfunctional Requirements: From Elicitation to Conceptual Models. IEEE Trans. Software Eng. 30(5): 328-350 (2004)

7. Devanbu, P., Stubblebine, S., Software Engineering for Security: a Roadmap. In Proceedings of ICSE 2000 ("the conference of the future of Software engineering"), 2000.

8. Giorgini, P., Massacci, F., Mylopoulos, J., Requirements Engineering meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard, in Proceedings of the International Conference on Conceptual Modelling (ER), LNCS 2813, pp. 263-276, Springer-Verlag, 2003.

9. Hermann, G. Pernul, G., Viewing business-process security from different perspectives. International Journal of electronic Commence 3:89-103, 1999

10. Jürjens, J., Shabalin, P., Tools for Critical Systems Development with UML (Tool Demo), UML 2004 Satellite Events, Nuno Jardim Nunes, Bran Selic, Alberto Silva, Ambrosio Toval (eds.), LNCS, Springer-Verlag 2004E. [Accessible at http://www.UMLsec.org. Protected content can be accessed as user: Reader, with password: Ihavethebook]. Available as open-source.

11. Jürjens, J., Secure Systems Development with UML, Springer, March-Verlag, 2004

12. McDermott, J., Fox, C., Using Abuse Case Models for Security Requirements Analysis. In Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.

13. Mouratidis, H., 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

14. Mouratidis, H., Giorgini, P., Manson, G., Integrating Security and Systems Engineering: towards the modelling of secure information systems. In Proceedings of the 15th Conference on Advanced Information Systems (CaiSE 2003), Velden –Austria, 2003

15. Object Management Group, OMG Unified Modeling Language Specification v1.5, March 2003. Version 1.5. OMG Document formal/03-03-01.

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

17. Schneider, F., editor. Trust in Cyberspace. National Academy Press, Washington, DC, 1999. Available as http://www.nap.edu/readingroom/books/trust/.

18. Schneier, B., Secrets & Lies: Digital Security in a Networked World, John Wiley & Sons, 2000

19. Schumacher, M., Roedig, U., Security Engineering with Patterns. In Proceedings of the 8th Conference on Pattern Languages for Programs (PLoP 2001), Illinois-USA, September 2001

20. Schumacher, M., Security Engineering with patterns. In LNCS 2754, Springer-Verlag, 2003

21. Shamir, A., Crypto Predictions. In 3rd International Conference on Financial Cryptography (FC 1999), 1999.

22. The Economist, Digital rights and wrongs, July 17, 1999

23. van Lamsweerde, A., Letier, E., Handling Obstacles in Goal-Oriented Requirements Engineering, Transactions of Software Engineering, 26 (10): 978-1005, 2000

24. Viega, J., McGraw, G., Building a Secure Software. Addison-Wesley, Reading, MA, 2002.