Secure Software Systems Engineering: The Secure Tropos Approach

Haralambos Mouratidis
School of Computing, IT and Engineering, University of East London, U.K.
Email: haris@uel.ac.uk

Abstract— **This paper discusses the secure Tropos methodology. This is the first paper in the literature that discusses all the aspects of the methodology as it has evolved over the last 10 years. In particular, the paper discusses the Secure Tropos modeling language, the security aware process of the methodology, and it also introduces the secTro, an automated tool to support the methodology.**

Index Terms—**Secure Tropos, Secure Software Systems Engineering, Security Requirements, Secure Design**

I. INTRODUCTION

It is widely recognized that security is an important aspect of any software system that stores and/or handles sensitive and confidential information. It is therefore expected that software system developers are able to develop and deploy very secure systems. However, this is not always the case. In fact, current surveys have indicated that we are far from developing acceptable secure software systems. One of the main reasons for this situation is that many software system developers do not always have a strong background in computer security and lack expertise in secure software systems development. Nevertheless, in practice, they are asked to develop software systems that require security features. Without appropriate modelling languages and methodologies to guide them during the development process, security is usually considered as an afterthought, meaning that security enforcement mechanisms have to be fitted into a pre-existing design, leading to serious design challenges and conflicts that usually translate into the emergence of information systems afflicted with security vulnerabilities. These vulnerabilities are often the major cause of system security disasters, and adjustments that are usually very expensive.

Research has shown that such vulnerabilities can be reduced and in many cases eliminated if the overall system development considers security aspects in a coherent way. Considering security throughout the development stages of a software system helps to limit the cases of security related conflicts by avoiding them and/or isolating them very early in the development process, saving therefore considerable time and money. The security analysis and implementation process should be similar to what happens when discussing functional software system requirements; where one does not get immediately trapped into discussions about programming languages and /or coding techniques.

It is therefore essential for security to be considered from the early stages and throughout the software development life-cycle. Nevertheless, to follow such paradigm, sound software engineering methodologies and practices need to be developed that support the simultaneous analysis of both security and software requirements; their transformation to an appropriate design and the implementation of that design.

This paper presents an effort to develop a methodology that incorporates security concerns in a structured and coherent way at all the stages of software systems design and development. The paper is structured as follows. Section II discusses and compares related work, while Section III presents the Secure Tropos methodology. Section IV discusses areas of future work and it concludes the paper.

II. RELATED WORK

The last few years a considerable number of works aiming to introduce security considerations during the various stages of the software systems development process have been presented in the literature. Anton et al. [1] propose a set of general taxonomies for security and privacy, to be used as a general knowledge repository for a (security) goal refinement process. Schumacher and Roedig [2] apply the pattern approach to the security problem by proposing a set of patterns, called security patterns, which contribute to the overall process of security engineering. Although useful, these approaches lack the definition of a structured process for considering security. A well defined and structured process is of paramount importance when considering security during the development stages of software systems.

On the other hand, a number of researchers model security by taking into account the behaviour of potential attackers. Van Lamsweerde and Letier [3] use the concept of security goals and anti-goals. Anti goals represent malicious obstacles set up by attackers to threaten the security goals of a system. Crook et al. [4], 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.

Similarly, Lin et al. [5], 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. An important limitation of all these approaches is that security is considered as a vague goal to be satisfied whereas a precise description and enumeration of specific security properties is still missing.

Differently, another "school of thinking" indicates the development of methods to analyse and reason about security based on the relationships between actors (such as users, stakeholders and attackers) and the system. Liu et al. [6] analyse security requirements as relationships amongst strategic actors by proposing different kinds of analysis techniques to model potential threats and security measures. Although a relationship based analysis is suitable for reasoning about security, an important limitation of existing approaches is that each of them only guides the way security can be handled within a certain stage of the software development process.

Another direction of work is based on the extension of use cases and the Unified Modelling Language (UML). In particular, McDermott and Fox [7] adapt use cases to capture and analyse security requirements, and they call the adaption an abuse case model. An abuse case is defined as a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders of the system. Similarly, Sindre and Opdahl [8] define the concept of misuse case, the inverse of use case, which describes a function that the system should not allow. They also define the concept of mis-actor as someone who intentionally or accidentally initiates a misuse case and whom the system should not support in doing so. Jurgens proposes UMLsec [9], an extension of the Unified Modelling Language (UML), to include modelling of security related features, such as confidentiality and access control. Lodderstedt et al. [10] also extend UML to model security. In their work, they present a security modelling language called SecureUML. They describe how UML can be used to specify information related to access control in the overall design of an application and how this information can be used to automatically generate complete access control infrastructures. An important limitation of all the use-case and/or UML related approaches 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.

III. SECURE TROPOS

A. History and Motivation

The creation and development of the Secure Tropos methodology was initiated in 2000 as a PhD project [11]. Since then, the methodology has undergone a number of developments. These include the development of a modelling language, the development of a security-aware process, and the development of an automated tool to support it. The main motivation behind the creation of Secure Tropos was the lack of a methodology to support the capturing, analysis and reasoning of security requirements from the early stages of the development process. One of the first main dilemmas faced was whether a new methodology should be developed from scratch or an existing methodology should be extended. Following a detailed analysis of that issue [11] it was decided to extend the Tropos methodology [12].

A number of important limitations of the Tropos methodology were identified, with respect to security modelling and analysis. Although a detailed discussion of these limitations is outside the scope of this paper and can be found in the literature [11], we outline them below to enable readers of this paper to understand the context of the extensions provided by Secure Tropos:

- The Tropos methodology uses the concept of soft-goal to model security requirements, similarly to the way that it handles any non-functional requirements. The concept of soft-goal is "used to model quality attributes for which there are no a priori, clear criteria for satisfaction, but are judged by actors as being sufficiently met" [13]. However, as it becomes evident in an increasing number of research works, security requirements are better defined in terms of constraints on a system's functions [14] [15]. Differently than quality properties, which represent characteristics of the system that its stakeholders care about, security requirements represent rules or conditions imposed to the system that are (theoretically) non negotiable. This is problematic because the concept of a soft-goal captures qualities but it fails to capture constraints.
- The usage of soft-goals to model general non-functional requirements, although it allows developers to define together security and other functional and non-functional requirements, it does not help to clearly identify security requirements. Such a distinction is made even harder by the lack of definition of the Tropos concepts, such as goals, tasks, and dependencies, with security in mind. This is problematic since it does not allow a clear understanding of the security requirements of the system and how these might conflict with functional and non-functional requirements.
- There are limitations regarding the process of modelling, analysing and reasoning about security requirements. In fact the Tropos methodology process for dealing with security requirements is quite ad hoc. Developers are allowed to capture security requirements with the aid of soft-goals, and then propagate them throughout the development stages. However this process is neither clear nor well guided. It is

unclear how developers can systematically capture security requirements (expressed as soft-goals) and how they can develop a design that successfully meets those requirements in a systematic way.

- The methodology does not provide any process to allow developers to reason about the consequences of the application of a particular design to their system and also fails to provide a process that allows developers to evaluate the developed security solution.
- The methodology fails to integrate security modelling during the early requirements analysis stage. However, all the actors play an important role with respect to the security of the system and all of them should be analysed with security in mind.

Therefore, the Secure Tropos methodology was developed to fulfil these limitations. Secure Tropos initially extended the Tropos methodology into two directions: concepts/language and process. Later a number of new models were also added to support the further analysis and design of security requirements. The rest of this section discusses the modeling language and the security aware process of Secure Tropos and it introduces secTro, a tool that supports the methodology.

B. Secure Tropos Modelling Language

Secure Tropos, as an extension of the Tropos methodology, uses a number of concepts found in Tropos:

- An actor [13] represents an entity that has intentionality and strategic goals within the multiagent system or within its organisational setting. An actor can be a (social) agent, a position, or a role.
- A (hard) goal [13] represents a condition in the world that an actor would like to achieve. In other words, goals represent actor's strategic interests. In Tropos, the concept of a hard-goal (simply goal hereafter) is differentiated from the concept of soft-goal. A soft-goal is used to capture non-functional requirements of the system, and unlike a (hard) goal, it does not have clear criteria for deciding whether it is satisfied or not and therefore it is subject to interpretation [13]. For instance, an example of a soft-goal is "the system should be scalable". According to Chung et al. [16], the difference between a goal and a soft-goal is underlined by saying that goals are satisfied whereas soft-goals are satisficed .
- A plan represents, at an abstract level, a way of doing something [12]. The fulfilment of a task can be a means for satisfying a goal, or for contributing towards the satisficing of a soft-goal. In Tropos different (alternative) tasks, that actors might employ to achieve their goals, are modelled. Therefore developers can reason about the different ways that actors can achieve their goals and decide for the best possible way.
- A resource [12] presents a physical or informational entity that one of the actors requires. The main concern when dealing with resources is whether the resource is available and who is responsible for its delivery.
- A dependency [13] between two actors represents that one actor depends on the other to attain some goal, execute a task, or deliver a resource. The depending actor is called the depender and the actor who is depended upon is called the dependee. The type of the dependency describes the nature of an agreement (called dependum) between dependee and depender. Goal dependencies represent delegation of responsibility for fulfilling a goal. Soft-goal dependencies are similar to goal dependencies, but their fulfilment cannot be defined precisely whereas task dependencies are used in situations where the dependee is required to perform a given activity. Resource dependencies require the dependee to provide a resource to the depender. By depending on the dependee for the dependum, the depender is able to achieve goals that it is otherwise unable to achieve on their own, or not as easily or not as well [13]. On the other hand, the depender becomes vulnerable, since if the dependee fails to deliver the dependum, the depender is affected in their aim to achieve their goals.

The Secure Tropos modeling language enhances the above concepts by defining extensions with security in mind and by adding a number of new concepts. In particular, the following concepts are introduced by Secure Tropos:

- Security Constraint. The main concept introduced by Secure Tropos is the concept of Security Constraint. Security Constraints are used, in the Secure Tropos methodology, to represent security requirements. A Security Constraint is a specialisation of the concept of Constraint. In the context of software engineering, a constraint is usually defined as a restriction that can influence the analysis and design of a software 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. In other words, constraints can represent a set of restrictions that do not permit specific actions to be taken or prevent certain objectives from being achieved. Often constraints are integrated in the specification of existing textual descriptions. However, this approach can often lead to misunderstandings and an unclear definition of a constraint and its role in the development process. Consequently, this results in errors in the very early development stages that propagate to the later stages of the development process causing many problems when discovered; if they are

discovered. Therefore, in the Secure Tropos modelling language we define security constraints, as a separate concept. To this end, the concept of security constraint has been defined within the context of Secure Tropos as: *A security condition imposed to an actor that restricts achievement of an actor's goals, execution of plans or availability of resources*. Security constraints are outside the control of an actor. This means that, differently than goals, security constraints are not conditions that an actor wishes to introduce but it is forced to introduce.

- Secure Dependency. A Secure Dependency introduces one or more Security Constraint(s) that must be fulfilled for the dependency to be valid. In the Secure Tropos methodology we distinguish among three types of secure dependencies: *dependee secure dependency*, *depender secure dependency*, and *double secure dependency*. In terms of the modeling language, different Secure Dependency types are defined using Depender and Dependee attributes of Security Constraints.
- Secure Goal. A secure goal represents a strategic interest of an Actor with respect to security. In the Secure Tropos context, strategic interest means a course of action that an actor needs to follow to satisfy one or more security constraints. The satisfaction of one or more security constraints by a secure goal is defined through a *Satisfies* relationship. It is worth stating that a secure goal does not define operational details of how a security constraint can be satisfied, since operational alternatives can be considered.
- Secure Plan. A secure plan represents a particular way for satisfying a secure goal. In the context of Secure Tropos, this means a specific and defined action that an actor executes to operationalise a secure goal.
- Secure resource. A secure resource is defined as an entity that is security critical for the system under development.
- Attack. In secure Tropos an attack is an action that might cause a potential violation of security in the system (this definition has been adopted by Matt Bishop's definition of a computer attack).
- Attacker. An attacker represents a malicious actor that has an interest to attack the system. As defined in Tropos, an actor has intentionality and strategic goals within the system. In the case of an attacker, the intentionality and strategic goals are related to breaking the security of the system.
- Secure Capability. A secure capability represents the ability of an actor to achieve a secure goal, carry out a secure plan and/or deliver a secure resource.
- Threat. Threats represent circumstances that have the potential to cause loss; or problems that can put in danger the security features of the system.
- Security features represented security related features that the system-to-be must have. Examples of security features are privacy, availability, and integrity.
- Protection objectives represent a set of principles or rules that contribute towards the achievement of the security features. These principles identify possible solutions to the security problems and usually they can be found in the form of the security policy of the organisation. Examples of protection objectives are authorisation, cryptography and accountability.
- Security mechanisms represent standard security methods for helping towards the satisfaction of the protection objectives. Some of these methods are able to prevent security attacks, whereas others are able only to detect security breaches. It must be noted that furthered analysis of some security mechanisms is required to allow developers to identify possible security sub-mechanisms. A security sub-mechanism represents a specific way of achieving a security mechanism. For instance, authentication denotes a security mechanism for the fulfilment of a protection objective such as authorisation. However, authentication can be achieved by sub-mechanisms such as passwords, digital signatures and biometrics.

D. Modelling Diagrams

The above concepts support the construction of five different diagrams: *Security Enhanced Actor Diagram (SEAD)*, *Security Enhanced Goal Diagram (SEGD)*, *Architectural Style Selection Diagram (ASSD)*, *Security Attack Scenarios Diagram (SASD)*, *Security Reference Diagram (SRD)*. The rest of this section describes these diagrams.

1) Security Enhanced Actor Diagram

A Security Enhanced Actor Diagram (SEAD) identifies and analyses actors of the environment, actors of the system and dependency relationships between them. The diagram enables software system developers to understand the security concerns of each actor and model these concerns with appropriate security constraints. Figure 1 illustrates a simple Security Enhanced Actor Diagram.

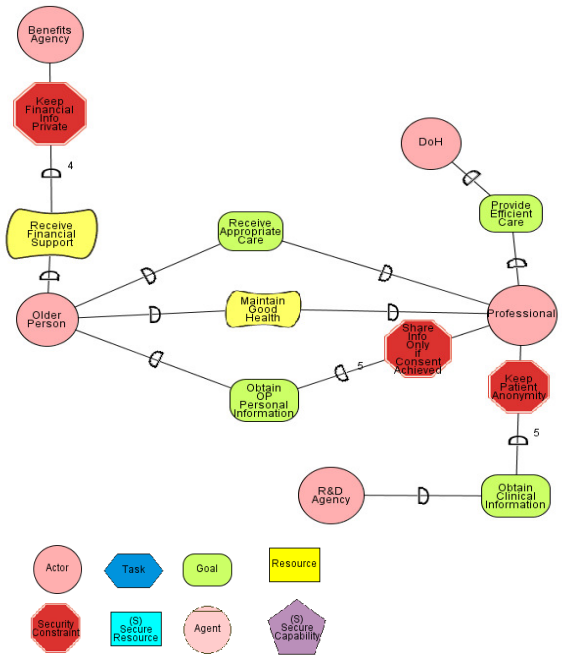

Fig.1 : Security Enhanced Actor Diagram

2) Security Enhanced Goal Diagram

SEGD allows a deeper understanding of how the actors reason about goals to be fulfilled, security constraints to be operationalised, plans to be performed and availability of resources. It completes the actor model with the reasoning that each actor makes about its internal (secure) goals, security constraints, (secure) plans and (secure) resources.

In the goal model, elements are linked by the means-ends, decomposition and contribution relationships. The means-ends relationship permits to link a means (plan/goal/resource) with an end (goal). The decomposition relationship permits to define a structure for a plan. A contribution relationship describes a positive or negative impact that one element has on another. Restricts relationships are used to model the connection between security constraints and the entities restricted. Figure 2 illustrates a simple Security Enhanced Goal Diagram (SEGD).

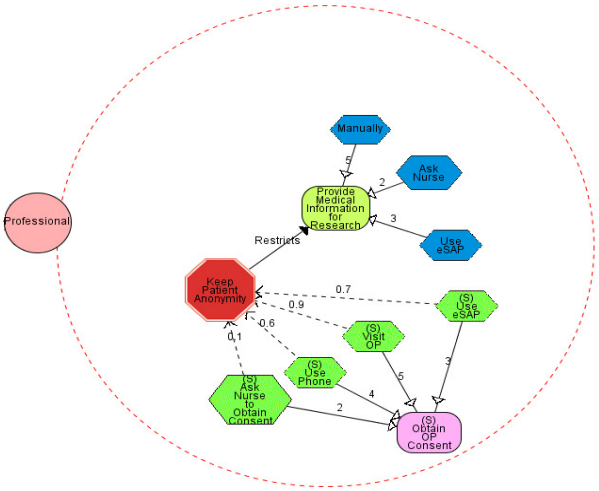

Fig.2: Security Enhanced Goal Diagram

3) Architectural Style Selection Diagram (ASSD)

This diagram is used to model architectural styles, security properties and security requirements of the system under development and the different contributions that each architectural style has on the security properties and the security requirements of the system [17]. In this diagram, a hexagon represents a security solution, while an emboldened cloud represents a non-functional requirement of the system. Links represent contributions and weights represent the degree of satisfiability [18] of the architectural style (for example Client/Server – Mobile Agents) to the various nodes. The diagram is used to perform an analysis based on an independent probabilistic model, which uses the measure of satisfiability proposed by Giorgini et al. [18]. Satisfiability represents the probability that a non-functional requirement will be satisfied. Therefore, the analysis involves the identification of specific non-functional requirements and the evaluation of different architectural styles against these requirements. Figure 3 illustrates a simple Architectural Style Selection Diagram.

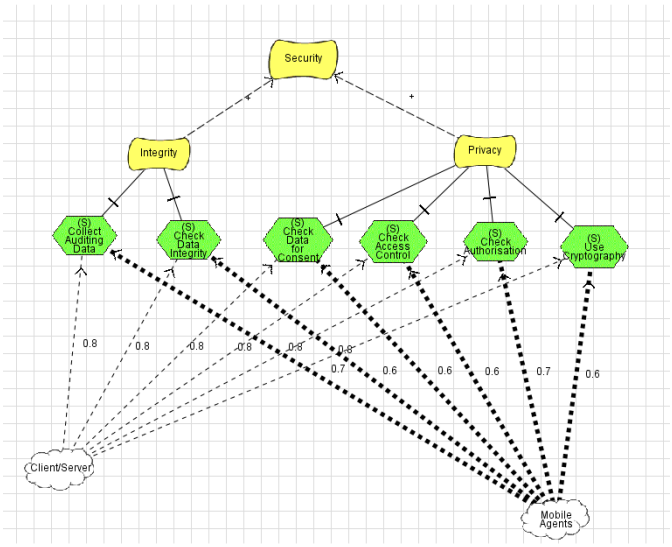

Fig.3: Architectural Style Selection Diagram

4) Security Attack Scenarios Diagram

This diagram models possible attackers, the resources of the system that are attacked and the actors of the system related to the attack (Figure 16). In particular, an attacker is modelled as an actor and its intentions are modelled as goals and tasks. Attacks are depicted as dash-lined links, called attack links, which contain the "attacks" tag, starting from one of the attackers' goals and ending on the attacked resource. Moreover, the system's actor secure capabilities are modelled and links are employed to indicate which of these capabilities help towards the prevention of the attackers' goals. Figure 4 illustrates a simple SASD.

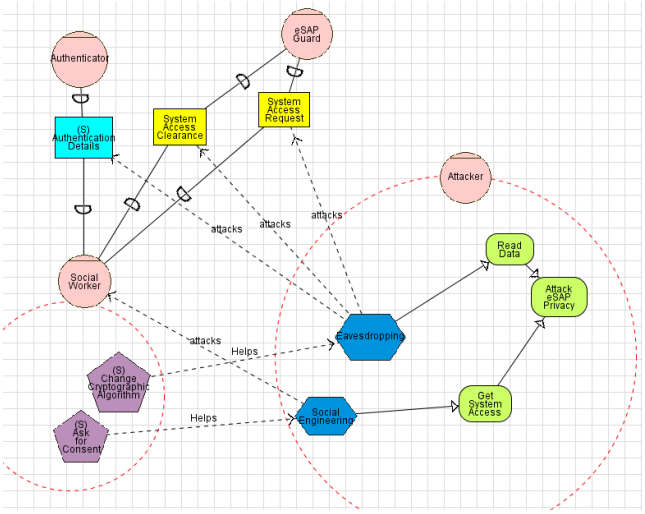

Fig.4: Security Attack Scenarios Diagram

5) Security Reference Diagram

The security reference diagram represents relationships between security features, threats, protection objectives, and security mechanisms. A security reference diagram is constructed after analysing the security requirements of the system-to-be and its environment. The main purpose of the security reference diagram is to allow flexibility during the development stages of a software system and also to save time and effort. Many systems under development are similar to systems already in existence. Therefore the security reference diagram can be used as a reference point that can be modified or extended according to specific needs of particular systems.

The security reference diagram is a graph that consists of a set of labelled nodes and a set of labelled directed edges, each of which connects a pair of nodes (Figure 5). Formally, this is represented as a special case of a labelled directed diagram. To control the non-deterministic derivation process during the construction of the security reference diagram, priority rules have been defined [11] and should be used by the developer. Figure 5 illustrates a security reference diagram.

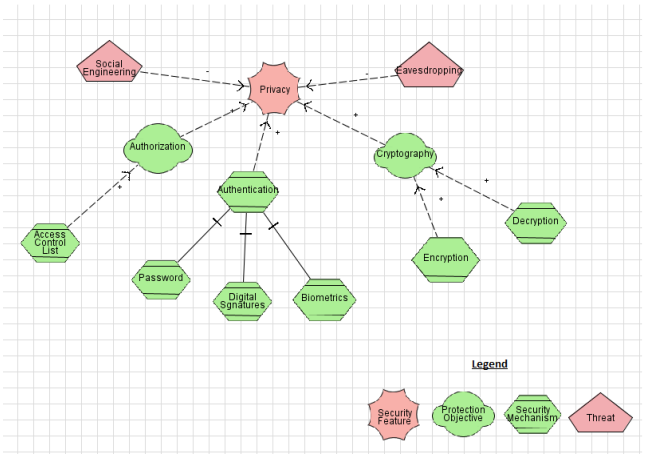

Fig.5: Security Reference Diagram

The analysis done during the construction of the security reference diagram can be used later in the development process to identify security constraints that must be introduced to the system-to-be (by taking into account the security needs of the system) and also by identifying possible means (security mechanisms) that contribute towards the satisfaction of the security constraints that are introduced to the system.

It is also worth mentioning, that the notation of the security reference diagram can be adapted to reflect the notation of the methodology that the diagram is integrated. This is very useful since it allows developers to work with well-known concepts and allows them to use the same concepts throughout the development process.

E. Security aware process

There are three main aims when considering security issues throughout the development stages of a software system: (i) identify the security requirements of the system; (ii) develop a design that meets the specified security requirements; and (iii) validate the developed system with respect to security. With the above in mind, the security-oriented process in Secure Tropos is one of identifying the security requirements of the software system, transforming these requirements to a design that satisfies them and validating the developed system with respect to security.

To achieve the above, the secure Tropos process consists of four main stages: *Security Analysis of the System Environment*; *Security Analysis of the System*; *Secure System Design*; *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 6. 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 [12] 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 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 [12]. However, such techniques are combined with a number of security-related techniques such as attack trees [19] and security reference diagrams [11]. The Secure Goal Introduction 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 6. 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. 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, security constraints to be operationalised, plans to be performed and availability of resources. 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.

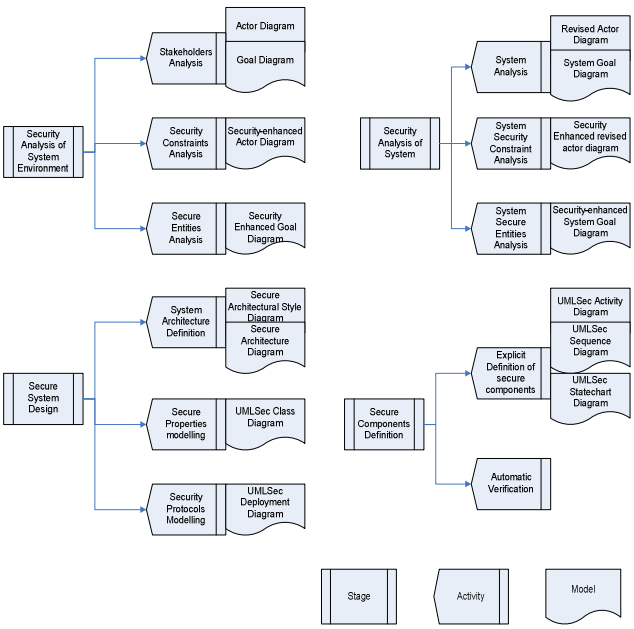

Fig.6: Secure Tropos security aware process

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 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.

E. Automated Tool Support: SecTro

Initial limited computer-aided support for the Secure Tropos methodology was provided by the OME tool (http://www.cs.toronto.edu/km/ome/). One of the main problems of that support was the lack of notation to support the Secure Tropos concepts such as security

constraints and the various new diagrams required by Secure Tropos. Therefore, an ad-hoc development was supported were Tropos models were developed and later they were enhanced using other software tools.

In order to overcome such limited support, secTro was developed to supports Secure Tropos. secTro 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. Moreover, the tool enables users to export created diagrams as PNG images. Figure 7 illustrates the main workspace of the tool.

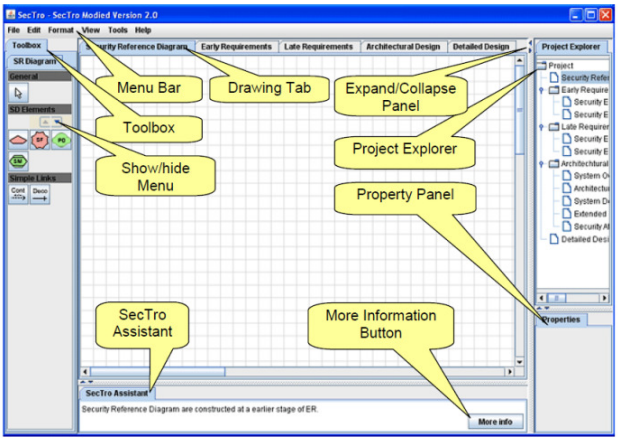

Fig.7: SecTro workspace

Apart from allowing users to create Secure Tropos diagrams, the tool also supports automatic generation of a number of templates and diagram components required by the methodology, such as Security Attack Scenarios and Capability diagrams (see Figure 8). This reduces the development time and restricts user errors.

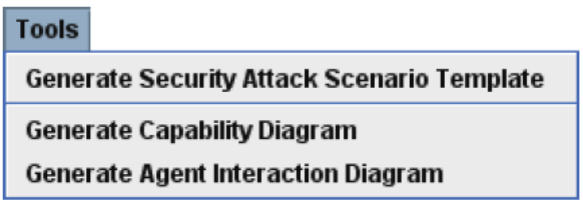

Fig.8: SecTro tools

IV. CONCLUSIONS / FUTURE WORK

Secure Tropos demonstrates a number of novel features, such as (1) allowing developers not only to model but also to reason about the technical as well as the social issues of security; (2) allowing developers to represent security concerns at different levels of software description; and (3) allowing developers to verify at the design stage, whether the developed solution satisfies the security requirements of the system.

Nevertheless, more work is required and the following areas have been identified as important for the further development of the methodology.

- Further evaluation of the methodology in large commercial projects. It is important that the methodology is used to support the analysis and development of large software systems within industrial environments. This would enable us to test the usability of the methodology and how easily the methodology's modeling language and processes can be understood and successfully employed by software engineers.
- secTro tool. It is important that further enhancements take place in the secTro tool to support the integration with other relevant tools (such as better integration with UMLsec tools) as well as introduce new automated processes that support the further automation of secure Tropos procedures. Currently, the development of the models at the later stages of the development process mostly takes place manually but it is envisaged that the tool will be able to automate the construction of such of these models based on the information provided and models constructed by the users in the early stages of the development process.

ACKNOWLEDGMENT

The author wishes to thank everyone that has been involved in the development of any parts of the Secure Tropos methodology.

REFERENCES

[1] A. I. Anton, J. B. Earp, A requirements taxonomy for reducing web site privacy vulnerabilities, Requirements Engineering, 9(3):169-185, 2004.

[2] M. Schumacher, U. Roedig, Security Engineering with Patterns, in the Proceedings of the 8th Conference on Pattern Languages for Programs (PLoP), Illinois – USA, 2001.

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

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

[5] L.C. Lin, B. Nuseibeh, D. Ince, M. Jackson, J. Moffett, Analysing Security Threats and Vulnerabilities Using Abuse Frames, Technical Report 2003/10, The Open University, 2003.

[6] L. Liu, E. Yu, J. Mylopoulos, Security and Privacy Requirements Analysis within a Social Setting, In Proceedings of the 11th International Requirements Engineering Conference, pp. 151-161, IEEE Press, 2003.

[7] J. McDermott C. Fox, Using Abuse Care Models for Security Requirements Analysis, Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.

[8] G. Sindre, A.L. Opdahl, Eliciting security requirements with misuse cases, Requirements Engineering, 10(1):34-44, 2005.
[9] J. Jurjens. Secure Systems Development with UML, Springer Verlag, 2004.
[10] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-Based Modelling Language for Model-Driven Security, in Proceedings of the UML'02, LNCS 2460, pp. 426-441, Springer-Verlag, 2002.
[11] 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", 2004, PhD Thesis, University of Sheffield.
[12] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, TROPOS: An Agent-Oriented Software Development Methodology, in Journal of Autonomous Agents and Multiagent Systems, 8(3):203-236, 2004.
[13] E. Yu, Modelling Strategic Relationships for Process Reengineering, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.
[14] H. Mouratidis and P. Giorgini (eds), "Integrating Security and Software Engineering: Advances and Future Vision" Idea group, IGI Publishing Group, 2006.
[15] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, Security Requirements Engineering: A Framework for Representation and Analysis, IEEE Transactions on Software Engineering, 34(1): 133-153, January 2008
[16] 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
[17] H. Mouratidis, P. Giorgini, and G. Manson, "When Security meets Software Engineering: A case of modelling secure information systems" Information Systems 30(8) pp. 609-629, Elsevier, 2005.
[18] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani. "Reasoning with Goal Models", in the Proceedings of the 21st International Conference on Conceptual Modeling (ER2002), Tampere, Finland, October 2002.
[19] B. Schneier, Secrets & Lies: Digital Security in a Networked World, John Wiley & Sons, 2000.

Haralambos Mouratidis is a Principal Lecturer in the School of Computing, IT and Engineering (CITE) at the University of East London, where he also leads the Computer Science Field. Haralambos is co-director of the Distributed Software Engineering Research Group at CITE and his research interests are related to secure software systems engineering and information systems development. He has published more than 80 refereed papers in relevant journals and conferences and he has received national and international funding for projects related to the above research areas.