An Ontology for Modelling Security: The Tropos Approach

Haralambos Mouratidis[1], Paolo Giorgini[2], Gordon Manson[1]

[1]*University of Sheffield, Computer Science Department, UK*
{haris, g.manson}@dcs.shef.ac.uk
[2]*University of Trento, Department of Information and Communication Technology, Italy*
paolo.Giorgini@dit.unit.it

Abstract. It has been argued that security concerns should inform all the stages of the development process of an agent-based system. However, this is not the case since current agent-oriented methodologies do not, usually, consider security concepts within their modelling ontology. In this paper we present extensions to the Tropos ontology to enable it to model security.

1. INTRODUCTION

Following the wide recognition of multi-agent systems, agent-oriented software engineering has been introduced as a major field of research. Many agent-oriented software engineering methodologies have been proposed [1,2] each one of those offering different approaches in modelling multi-agent systems.

It has been argued [3] that security issues should inform all the stages of the development of agent-based systems. However, usually, this is not the case. One of the reasons is the lack of concepts and notations employed by the current methodologies to help towards the inclusion of security within the development stages. In other words, agent oriented software engineering methodologies do not, usually, integrate security concepts within their ontology.

In this paper we describe how the Tropos ontology has been extended to consider security issues. Section 2 of the paper provides an overview of Tropos ontology, and Section 3 identifies the need to extend the methodology to consider security issues. Section 4 describes the newly introduced (security) concepts and Section 5 concludes the paper and presents directions for future work.

2. TROPOS

Tropos [2] is an information system development methodology, tailored to describe both the organisational environment of a system and the system itself,

employing the same concepts throughout the development stages. Tropos ontology is described at three levels of granularity [4] and is inspired by social and organisational structures. At the first level (lowest), Tropos ontology adopts components from the i* modelling framework [5], which is based on the concepts of actors, goals, soft goals, tasks, resources, and social dependencies. Social dependencies represent obligations or agreements, called dependum, between two different actors called depender and dependee. To partially illustrate the modelling of the social dependencies between actors consider the eSAP System [6]. Such a system, involves four actors [6], namely *Older Person*, *R&D Agency*, *Benefits Agency*, *Department of Health* (DoH), and *Professional* (Figure 1).

The depender is the depending actor and the dependee is the actor who is depended upon. For example in Figure 1, the *Older Person* depends on the *Professional* to fulfil the *Receive Appropriate Care* goal dependency. For this dependency, the *Older Person* is the depender, the *Professional* the dependee and the *Receive Appropriate Care* goal the dependum. Actors have strategic goals and intentions within the system or the organisation and represent (social) agents (organisational, human or software), roles or positions (represent a set of roles). A goal represents the strategic interests of an actor. In *Tropos* we differentiate between hard (only goals hereafter) and soft goals. The latter having no clear definition or criteria for deciding whether they are satisfied or not. A task represents a way of doing something. Thus, for example a task can be executed in order to satisfy a goal. A resource represents a physical or an informational entity while a dependency between two actors indicates that one actor depends on another to accomplish a goal, execute a task, or deliver a resource.

At the second level, Tropos ontology provides a set of organisational styles inspired by organisation theory and strategic alliances [4]. These styles are used to describe the overall architecture of the organisational context of the system or its architecture. The last element of the Tropos ontology consists of social patterns [4]. These patterns, unlike organisational styles, are focused on the social structure necessary to achieve a particular goal instead of the overall goals of the organisation.

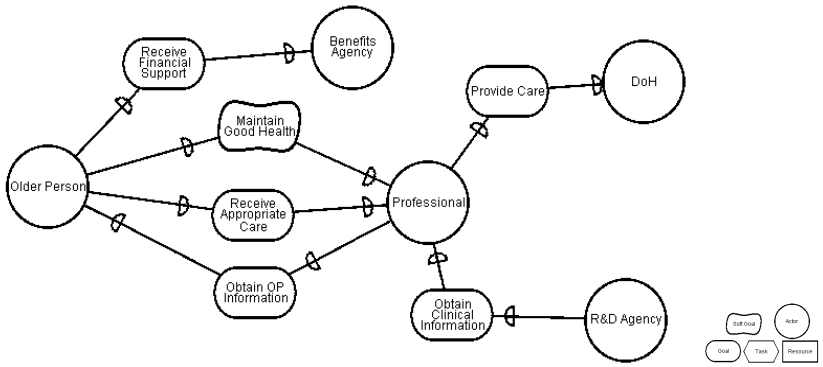

Fig. 1. The social Dependencies between the stakeholders of the eSAP system.

In addition to the graphical representation, *Tropos* provides a formal specification language called Formal *Tropos* [7]. Formal Tropos compliments i* by defining a textual notation for i* models and allow us to describe dynamic constraints among the different elements of the specification in a first order linear-time temporal logic [7].

3. (LACK OF) SECURITY ONTOLOGY IN TROPOS

As we have been argued in a previous paper [3], the Tropos methodology needs to be extended in order to adequately model security. The current Tropos ontology provides developers the ability to model security requirements as soft goals. The concept of a 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" [5]. However, security requirements may relate to system's quality properties, or alternatively may define constraints on the system [8]. Qualities are properties or characteristics of the system that its stakeholders care about, while constraints are restrictions, rules or conditions imposed to the system and unlike qualities are (theoretically) non negotiable. Thus, although the concept of a soft goal captures qualities, it fails to adequately capture constraints [3]. Security constraints might affect the analysis and design of the system, by restricting some alternative design solutions, conflict with some of the requirements of the system, and also by refining some of the goals of the system or introducing new ones that help the system towards the satisfaction of the constraint.

We believe the current Tropos ontology must be extended towards three main directions. Firstly, the concept of a security constraint must be introduced, as a separate concept, next to the existing concepts of Tropos. Secondly, existing concepts such as goals, tasks, resources, must be defined *with* and *without* security in mind. For example a goal should be differentiated from a *secure goal*, the latter representing a goal that affects the security of the system. Thirdly, security-engineering concepts such as security features, protection objectives, security mechanisms and threats, which are widely used in security engineering, must be introduced in the Tropos ontology, in order to make the methodology applicable by software engineers as well as security engineers. In this paper, due to lack of space, we only present the extensions towards the first two directions. Readers interested in how security-engineering concepts are integrated within Tropos methodology should refer to [3].

4. SECURITY CONCEPTS

Security Constraints

We define *security constraint* as a constraint that is related to the security of the system. Since constraints can influence the security of the system either positively (e.g., *Allow Access Only to Personal Record*) or negatively (e.g., *Send Record Plain Text,* not encrypted), we further define positive and negative *security constraints*, respectively.

In the early requirements analysis security constraints are identified and analysed according to the constraint analysis processes we have proposed in [9]. *Security constraints* are then imposed to different parts of the system, and possible conflicts between security and other (functional and non functional) requirements of the system are identified and solved. To identify these conflicts we differentiate between *security constraints* that contribute positively or negatively to the other requirements of the system. We consider a *security constraint* contributing to a higher level of abstraction.

This means we are not taking into consideration specific security protocols that should be decided during the implementation of the system, and that most of the times restrict the design with the use of a particular implementation language.

Secure Entities

The term *secure entities* involves any *secure goals*, *tasks* and *resources* of the system. A *secure entity* is introduced to the actor (or the system) in order to help in the achievement of a *security constraint*. For example, if a health professional actor has the security constraint *Share Info Only If Consent Obtained,* the *secure goal Obtain Patient Consent* can be introduced to this actor in order to help in the achievement of the constraint.

A *secure goal* does not particularly define how the *security constraint* can be achieved, since (as in the definition of goal, see [5]) alternatives can be considered. However, this is possible through a *secure task*, since a task specifies a way of doing something [5]. Thus, a *secure task* represents a particular way for satisfying a *secure goal*. For example, for the secure goal *Check Authorisation,* we might have secure tasks such as *Check Password* or *Check Digital Signatures*.

A resource that is related to a *secure entity* or a *security constraint* is considered a *secure resource*. For example, an actor depends on another actor to receive some information and this dependency is restricted by a constraint *Only Encrypted Info*.

Secure Dependencies

A *secure dependency* introduces *security constraint*(s), proposed either by the depender or the dependee in order to successfully satisfy the dependency. For example a *Doctor* (depender) depends on a *Patient* (dependee) to obtain *Health Information* (dependum). However, the *Patient* imposes a *security constraint* to the *Doctor* to share *health information only if consent is achieved*. Both the depender and the dependee must agree in this constraint for the secure dependency to be valid. That means, in the depender side, the depender expects from the dependee to satisfy the *security constraints* while in the dependee side, a secure dependency means that the dependee will make an effort to deliver the dependum by satisfying the *security constraint*(s). There are two degrees of security: *Open Secure dependency* (normal dependency) and *Secure dependency*. In an *Open Secure Dependency* [3] some security conditions might be introduced but if the dependee fail to satisfy them, the consequences will not be serious. This means that the security of the system will not be in danger if some of these conditions are not satisfied. On the other side, there are three different types of a secure dependency [3], *Dependee Secure Dependency*, *Depender Secure Dependency*, and *Double Secure Dependency*.

Taking as an example the eSAP system illustrated in Section 2, the social dependencies between the actors of the system can be modelled now taking into account the security constraints between them as shown in figure 3. The *Older Person* depends on the *Benefits Agency* to *Receive Financial Support*. However, the *Older Person* worries about the privacy of their finances so they impose a constraint to the *Benefits Agency* actor, to keep their financial information private. The *Professional* depends on the *Older Person* to *Obtain Information*, however one of the most important and delicate matters for a patient (in our case the older person) is the

privacy of their personal medical information, and the sharing of it. Thus most of the times the *Professional* is imposed a constraint to share this information if and only if consent is achieved. In addition, one of the main goals of the *R&D Agency* is to *Obtain Clinical Information* in order to perform tests and research. To get this information the *R&D Agency* depends on the *Professional*. However, the *Professional* is imposed a constraint (by the Department of Health) to *Keep Patient Anonymity*.

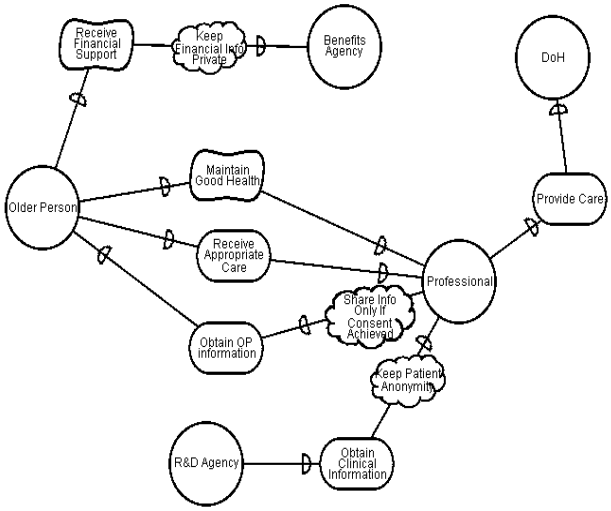

Fig. 3. Social dependencies between the eSAP stakeholders

The security constraints imposed at each actor can be further analysed by identifying which goals of the actor they restrict [9]. For example, the *Professional* actor has been imposed two *security constraints* (*Share Info Only If Consent Obtained* and *Keep Patient Anonymity*). During the means-end analysis [1] of the *Professional* actor we have identified the *Share Medical Info* goal. However, this goal is restricted by the *Share Info Only If Consent Obtained* constraint imposed to the *Professional* by the *Older Person*. For the *Professional* to satisfy the constraint, a secure goal can be introduced such as *Obtain Older Person Consent*. However this goal can be achieved with many different ways, for example a *Professional* can obtain the consent personally or can ask a nurse to obtain the consent on their behalf. Thus a sub-constraint can be introduced, *Only Obtain Consent Personally*. This sub constraint introduces another *secure goal Personally Obtain Consent*. This goal can be divided into two sub-tasks *Obtain Consent by Mail* or *Obtain Consent by Phone*.

Formal Tropos

Formal *Tropos* [7] complements graphical *Tropos* by extending the *Tropos* graphical language into a formal specification language [7]. The language offers all the primitive concepts of graphical *Tropos*, supplemented with a rich temporal specification language, inspired by KAOS [10], that has formal semantics and it is amenable to formal analysis. In addition, Formal *Tropos* offers a textual notation for i* models and allows the description of different elements of the specification in a

first order linear-time temporal logic. A specification of formal *Tropos* consists of a sequence of declarations of entities, actors, and dependencies [7].

Formal *Tropos* can be used to perform a formal analysis of the system and also verify the model of the system by employing formal verification techniques such as model checking to allow for an automatic verification of the system properties [7].

As with the graphical *Tropos*, Formal *Tropos* has not been conceived with security on mind. Thus, Formal *Tropos* fails to adequately model some security aspects (such as secure dependencies and security constraints). Extending Formal Tropos will allow us to perform a formal analysis of our introduced concepts and thus provide formalism to our approach. Towards this direction, we have extended Formal *Tropos* grammar [9] and below we present an example in which the secure dependency *Obtain OP information* between the *Older Person* and the *Professional* (Figure 3) is specified

Entity HealthInformation
Attribute constant Record: Record

Entity Record
Attribute constant
content: CarePlan ,accessControl: Boolean
patient: Patient, consent: boolean
Security Constraint
∃hi:HealthInformation ((hi.record=self) → self.accessControl)

Actor Professional
Attribute patients: PatientList
Goal provideCare
Creation condition
∃ p: Patient (In(p,self.patients) ∧ ¬p.helthOK)

Actor Older Person
Attribute healthOK: boolean
Goal MaintainGoodHealth
Creation condition ¬self.healthOK
Security constraint
∀ rec: Record ((rec.patient=self) → rec.accessControl)

Dependency ObtainOPInformation
Type Goal
Security Type Dependee
Mode Achieve and Maintain
Depender Professional
Dependee Older Person
Attribute constant
Creation condition
In(self.dependee,self.depender.patients) ∧self.dependee.healthOK
Security Constraint for depender
∀ rec: Record ((rec.patient=dependee) ∧ rec.consent)

5 Conclusions and future work

In this paper we have presented extensions to the Tropos ontology to enable it to model security issues. Concepts and notations were introduced to the existing graphical *Tropos* and also Formal Tropos grammar was extended to provide formalism for our newly introduced concepts.

During the process of extending the *Tropos* ontology we have reach some useful conclusions. By introducing the concepts of security constraints, functional, non-functional and security requirements are defined together, however a clear distinction is provided. In addition, by considering the overall software development process it is easy to identify security requirements at the early requirements stage and propagate them until the implementation stage. This introduces a security-oriented paradigm to the software engineering process. Also the iterative nature of the methodology along with the security concepts allows the redefinition of security requirements in different levels therefore providing a better integration of security and system functionality.

Our extensions only apply to the first level of the Tropos ontology. Future work involves the expansion of our approach to the other two levels of the Tropos ontology. We aim to provide a set of organisational styles and a pattern language that will help developers to consider security throughout the development of an agent-based system.

References

1. C. Iglesias, M. Garijo, J. Gonzales, "A survey of agent-oriented methodologies", *Intelligent Agents IV*, A. S. Rao, J. P. Muller, M. P. Singh (eds), Lecture Notes in Computer Science, Springer-Verlag, 1999
2. J. Castro, M. Kolp and J. Mylopoulos. "A Requirements-Driven Development Methodology," In *Proc. of the 13th Int. Conf. On Advanced Information Systems Engineering (CAiSE'01)*, Interlaken, Switzerland, June 2001
3. H. Mouratidis, P. Giogini, G. Manson, "Modelling Secure Multiagent Systems", (to appear) in the Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, July 2003
4. A. Fuxman, P. Giorgini, M. Kolp, J. Mylopoulos, "Information Systems as Social Structures", In Proceedings of the *Second International Conference on Formal Ontologies for Information Systems (FOIS-2001),* Ogunquit, USA, October 17-19, 2001
5. E. Yu, "Modelling Strategic Relationships for Process Reengineering", *PhD thesis*, Department of Computer Science, University of Toronto, Canada 1995
6. H. Mouratidis, I. Philp, G. Manson, "Analysis and Design of eSAP: An Integrated Health and Social Care Information System", In the Proceedings of the 7th International Symposium on Health Information Management Research (ISHIMR2002), Sheffield, July 2002
7. A. Fuxman, M. Pistore, J. Mylopoulos, P. Traverso, "Model Checking Early Requirements Specification in Tropos", In the Proceedings of the 5th Int. Symposium on Requirements Engineering, RE' 01, Toronto, Canada, 2001
8. I. Sommerville, "Software Engineering", sixth edition, Addison-Wesley, 2001
9. H. Mouratidis, "Extending Tropos Methodology to Accommodate Security". Progress Report, Computer Science Department, University of Sheffield, October 2002
10. A. Dardenne, A. Van Lamsweerde, S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*, 20, pp 3-50, 1993