F. Moyano, C. Fernandez-Gago, and J. Lopez, "Towards Engineering Trust-aware Future Internet Systems", *3rd International Workshop on Information Systems Security Engineering (WISSE 2013)*, LNBIP vol. 148, pp. 490-501, 2013.
http://doi.org/10.1007/978-3-642-38490-5
NICS Lab. Publications: https://www.nics.uma.es/publications

Towards Engineering Trust-aware Future Internet Systems*

Francisco Moyano, Carmen Fernandez-Gago, Javier Lopez
Network, Information and Computer Security Lab
University of Malaga, 29071 Malaga, Spain
{moyano, mcgago, jlm}@lcc.uma.es

June 24, 2013

Abstract

Security must be a primary concern when engineering Future Internet (FI) systems and applications. In order to achieve secure solutions, we need to capture security requirements early in the Software Development Life Cycle (SDLC). Whereas the security community has traditionally focused on providing tools and mechanisms to capture and express hard security requirements (e.g. confidentiality), little attention has been paid to other important requirements such as trust and reputation. We argue that these soft security requirements can leverage security in open, distributed, heterogeneous systems and applications and that they must be included in an early phase as part of the development process. In this paper we propose a UML extension for specifying trust and reputation requirements, and we apply it to an eHealth case study.

1 Introduction

Security is a crucial concern that must be addressed in order to guarantee the successful deployments of FI scenarios [19]. These scenarios usually comprise a huge number of heterogeneous, geographically distributed entities, including human users, which must interact to provide services. The complexity of managing security in these scenarios is aggravated by their dynamic nature, with devices changing, appearing and disappearing along the system lifetime. In these complex and open scenarios, more flexible security solutions, namely soft

*This work has been partially funded by the European Commission through the FP7/2007-2013 project NESSoS (www.nessos-project.eu) under grant agreement number 256980. The first author is funded by the Spanish Ministry of Education through the National F.P.U. Program.

security mechanisms [14], are required as a complement to the traditional hard security ones: confidentiality, integrity and availability.

Trust is a soft security mechanism that can leverage the security of a system. Even though there is not any accepted definition of trust, it is agreed that it can improve decision-making processes under risk and uncertainty, improving in turn systems security. Reputation, which is a concept strongly related to trust, can also help in this task. We argue that increasing security in FI applications entails that trust relationships between actors, applications and system environments cannot be taken for granted any more and must be explicitly specified from the very beginning in the Software Development Life Cycle (SDLC). However, security requirements engineering methods often lay trust aside and focus on specifying hard security requirements, such as confidentiality or authorization [7][8]. Even when some social aspects are beginning to be captured at the requirements stages [10][18], the approach towards analysing trust is still naive. This could explain why trust and reputation models have been traditionally added after-the-fac in an ad-hoc fashion, limiting their re-usability and presenting scalability problems [4].

We advocate that a comprehensive analysis of trust and reputation during the initial stages of the SDLC is required. The contribution of this paper is twofold. First, we provide an extension to UML in order to help requirements engineers and software designers to have a clearer understanding of the trust and reputation requirements of the system-to-be; second, we analyse, by means of an eHealth case study, how we can apply this UML profile as well as some considerations when designing trust-aware systems. We choose UML because it is a *de facto* standard in the industry and because other relevant security-oriented profiles exist that could be potentially integrated with ours.

The rest of the paper is structured as follows. Section 2 reviews some related work. In Section 3 we provide a domain analysis in the field of trust and reputation. The extensions performed on UML are explained in Section 4, whereas Section 5 applies these extensions to an eHealth scenario. Finally, Section 6 presents the conclusion and some lines for future research.

2 Related Work

There are several works that consider security requirements at the early stages of the SDLC. Some of these works focus on detecting possible attacks on the system [16][15]. In others, the emphasis is on modeling security requirements, such as confidentiality or authorization. This is the case of UMLsec [7] and SecureUML [8], two UML profiles that include security constraints and annotations into the diagrams. Other works aim to integrate the notion of risk into the requirement analysis stage [9] in order to assess whether the risk level of some unwanted incidents is beyond an acceptable threshold.

The contributions mentioned up to now focus on hard security requirements or risk, but they usually lay trust aside. In addition to traditional policy languages for distributed trust management [2][6], there are other works that focus

on trust in early stages of the SDLC. Mouratidis and Giorgini [10] present Secure Tropos, a methodology that extends the Tropos methodology in order to enable the design of secure systems. Actors in Tropos may depend on other actors in order to achieve a goal, and these social relationships are captured by the methodology. In a similar direction, Lamsweerde and Letier present KAOS [18], a comprehensive goal-oriented methodology to elicit the requirements of a socio-technical system. All these contributions put forward the idea of capturing social aspects, but the notion of trust and its influence on the information systems is barely explored. This is partially covered by Pavlidis, Mouratidis and Islam [12], who include trust-related concepts in Secure Tropos.

The work by Chakraborty and Ray [3] bridges a gap between traditional security requirements modeling and soft-security considerations by incorporating the notion of trust levels into the traditional Role-Based Access Control model. These levels are measured by means of a trust vector, where each component in the vector is a factor that influences trust, such as knowledge or experience.

In general, the aforementioned works usually fail in capturing and making explicit all the trust relationships, and above all, how trust and reputation can be used by the system-to-be. The closest contribution to our work is the one by Uddin and Zulkernine [17], who present a UML profile for trust called UMLtrust. They provide extensions, as we do, to some UML diagrams in order to represent trust information. Their approach and focus is, however, different than ours. First, their primary concern is reasoning about *trust scenarios*, without making explicit which are the trust relationships in the system. Also, they do not address reputation, whereas it is a primary concern for us. We also provide more details on how trust and reputation can be computed and the factors (e.g. variables and attributes) that will be taken into account for this computation. We also show how trust can influence at the infrastructure level by means of deployment diagrams. However, our trust analysis is in general at a higher level of abstraction, without delving into the details of class attributes and methods, which is something that UMLtrust requires. As a conclusion, we think that both works are complementary and can help each other in providing a more comprehensive vision of trust in the system for designers and developers.

3 Trust and Reputation

There are many definitions of trust, and one often cited is the one by Gambetta [5]: *'trust is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action[...]'*. Reputation is defined by the Concise Oxford Dictionary as 'what is generally said or believed about a person or the character or standing of a thing', being a more objective concept that trust.

A trust conceptual model, adapted from the one in [11], is shown in Figure 1. Many of these concepts are included in the UML profile presented in the next section. Just to mention some details, it is important to differentiate between a trust factor, a variable and an attribute. A factor is an application-independent

attribute (e.g. the trustor's disposition to believe in others' capability). An attribute captures the notion of factor and extends it with application-specific information too (e.g. how much a customer has paid for a book). At the software level, one or more attributes are represented by variables, which forms the core of a trust metric. The dimension of a trust or reputation value refers to whether the final score is just one number (or label), or a tuple of numbers or labels. Finally, for simplicity sake, some arrows are not depicted. For example, there would be an arrow from *Reputation* to *Value* and from *Variable* to *Value*.

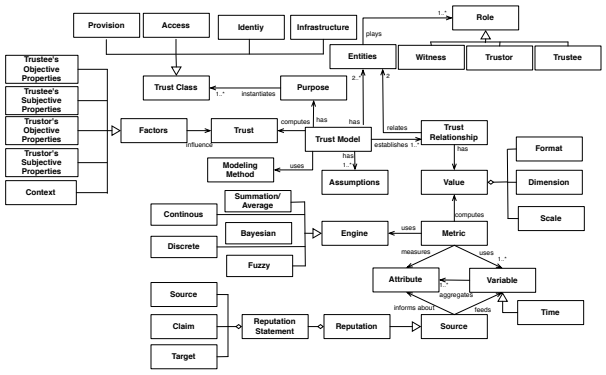

Figure 1: Trust Conceptual Model (summarized)

4 UML Profile for Trust and Reputation

This section presents an extension of UML that aims to ease the specification and initial design of trust-aware systems and applications. We consider indispensable the use of some behavioural diagram, such as activity diagrams, even though we do not propose extensions for them. This diagram should represent the interaction patterns between the trust and business logic of the application, and should make clear which actor initiates a trust event, how this event is triggered and what their consequences are[1].

Each extended diagram is further explained in the following sections.

4.1 Use Case Diagram

The goal of use case diagrams in the context of trust is to depict, at a glimpse, the trust relationships that exist between the different actors in the system. There is however more interesting information that we can represent in this diagram. For example, we could make explicit which actors can make a claim about which other actors, thus incorporating reputation information in the diagram. The extensions performed on the use case diagram are summarized in Table 1.

[1] A trust event is any occurrence in the system that triggers a reputation or trust relationship update.

Table 1: Use Case Diagram Extensions

Stereotype	Base Class	Explanation
Trustor	Actor	Actor playing the trustor role
Trustee	Actor	Actor playing the trustee role
Witness	Actor	Actor playing the witness role
Source	Actor	Actor capable of making a claim
Target	Actor	Actor capable of receiving a claim
Trusts	Connector	Trust relationship
Claims	Connector	Source makes a claim about a Target
Decides	Connector	Use case affected by a trust/reputation decision

Trustor, *trustee*, *witness*, *source* and *target* are roles that actors can play in the system. Trust relationships are made explicit by means of the extension *trusts*, whereas *claims* represent that a given *source* can make a claim about a given *target*. As the ultimate goal of trust is aiding in making a decision, we also add the *decides* connector, which captures the idea that a use case can be affected by trust or reputation information. An actor could perform the same use case in different ways (or even could decide not to perform it at all), and this decision can be influenced by trust or reputation information.

In addition to the previous UML extensions, we define two adornments: *decision criteria* and *context*. The former is used to annotate the *decides* relationship between an actor and a use case, and it specifies whether the decision in that use case is based on trust or reputation. The latter annotates *trusts* and *claims* relationships and specifies their context. This captures the idea that trust and reputation are context-dependent.

4.2 Class Diagram

Class diagrams can provide more insight in certain aspects of trust and reputation. The stereotypes used to extend class diagrams are depicted in Table 2. We find the same stereotypes that in the use case diagram extension regarding the roles of the actors. Also, we find *TrustRelationship*, which represent the trust relationship between a *trustor* and a *trustee*, and *Claim*, which captures the notion of a claim made by a *source* entity about a *target* entity. We add also three important notions for the evaluation of trust and reputation, namely *TrustEngine*, *ReputationEngine* and *Variable*. They represent how trust and reputation are computed, and the variables considered for such computation.

Tagged values are used in order to define more precisely the aforementioned concepts. The list of tagged values is shown in Table 3. Just to mention some of them, *subjective properties* refer to a list of beliefs of the trustor regarding a trustee, whereas *objective properties* represent a list of trustee's properties that can be (more) objectively measured (e.g. reliability or certification by a Trusted Third Party). *Dimension* is the number of components of a trust or reputation value, and *how* specifies whether the value of a variable is explicitly assigned (interactively) by the actor or is monitored by another system.

Table 2: Class Diagram Extensions

Stereotype	Base Class	Explanation
Trustor	Class	Actor playing the trustor role
Trustee	Class	Actor playing the trustee role
Witness	Class	Actor playing the witness role
Source	Class	Actor capable of making a claim
Target	Class	Actor capable of receiving a claim
TrustRelationship	Class	Trust relationship between trustor and trustee
Claim	Class	Claim that a source makes about a target
TrustEngine	Class	Engine in charge of updating a trust relationship
ReputationEngine	Class	Engine in charge of computing a target's reputation
Variable	Class	Variable used by a trust or reputation engine

Table 3: Tagged Values for Class Diagrams

Value	Class	Explanation
type	Trustor, Trustee, Witness, Source, Target	The type of actor (i.e. human, system)
subProp	Trustor	Subjective properties
objProp	Trustee	Objective properties
context	TrustRelationship, Claim	Context
dimension	TrustRelationship, Claim	Dimension of a trust relationship or a claim
scale	TrustRelationship, Claim, Variable	Upper and lower bounds
default	TrustRelationship	Default value
format	TrustRelationship, Claim	Quantitative vs. qualitative
display	ReputationEngine	Visualization by user actors
engine	Engine	Type of computation engine
variables	Engine	List of variables used by the engine
attribute	Variable	Attribute(s) captured by the variable
source	Variable	System or actor that triggers the variable update
how	Variable	Assigned vs. monitored

Note that some of these tagged values could be almost directly mapped to attributes of design classes, whether others are just informative and require further refinement at design stage. For example, *attribute* represents the attribute(s) captured by a variable. This information might be useful for aiding designers to keep in mind what the variable actually should represent, but could hardly be directly mapped to a class attribute.

4.3 Deployment Diagram

Deployment diagrams are useful as they represent the software from the infrastructure point of view, and they show valuable information in terms of trust and reputation. Very often, trust and reputation must be considered not only at the application level (trust among actors or among software components), but also at the infrastructure level [13]. Platforms and networks can trust each other and they can even hold reputation values. This is particularly useful when designing large-scale distributed systems, where a given processing node (e.g. a mobile phone or a server) can choose among different nodes in order to collaborate or

communicate information.

The extensions performed on deployment diagrams are shown in Table 4. We can specify which node acts as reputation manager in a centralized reputation model. Reputation managers compute reputation, store it, and distribute it (or just publish it) when necessary. The *decides* stereotype captures the decision process made by one entity (processing node) when communicating with other processing nodes. As in the case of use case diagrams, this stereotype can be adorned in order to make explicit whether this decision is based on trust or reputation with *decision criteria*. Finally, we also add a tagged value *entities* to specify the reputation of which entities the reputation manager will store.

Table 4: Deployment Diagram Extensions

Stereotype	Base Class	Explanation
ReputationManager	Node	Node that acts as reputation manager
decides	Connector	Trust-based decision

The next section puts all the concepts discussed in this section together by applying them to an eHealth scenario.

5 Case Study

In order to consider trust and reputation requirements early in the SDLC, we will present in this section how we can apply the information provided in Section 4 to a real scenario. The case study comes from the NESSoS project[2] and belongs to an eHealth scenario as described in a project deliverable [1].

The case study presents a patient monitoring scenario, which aims to collect health-related data independently of the location of the patient. This is useful for patients, who can receive immediate feedback under critical situations and be assisted by physicians at any moment and place. In order to make this scenario feasible, the patient must wear a device capable of measuring vital signs (e.g. blood pressure). This device must be able to send this information to other systems that will show it to physicians for monitoring purposes.

The goal is to build a web application through which the physician and the patient can interact in a trusted way. In this application, the physician can add and remove a wearable device to the system, start the process to assign the device to a patient, configure both critical and uncritical alerts, ask patient consent to use his data for research purposes, create an advice for the patient based on the patient's data, demand an immediate reading from the wearable and start the process to change a patient's wearable. Patients can configure uncritical alerts, ask for second opinions (to other physicians), accept or deny consent, show the physician's advices, complete the device assignment process started by the physician and demand a physician change.

[2]http://www.nessos-project.eu

Even though there are important hard security requirements, the application must also be trusted, in the sense that physicians and patients must be confident that the application is performing well and that they can trust the information provided by other entities. We propose using the aforementioned UML profile in order to consider trust and reputation requirements early in the SDLC.

A possible trust-aware use case diagram is shown in Figure 2. We state that there is a trust relationship between the patient and the physician. The patient plays a *trustor* role and the physician plays a *trustee* role. In addition, there is a *trusts* connector, which is adorned by the *context* where this trust relationship is set, namely *monitoring*. There is another trust relationship between the physician (who therefore also plays a *trustor* role) and the wearable. The patient also plays the *source* role and can therefore make claims (*claims* connector) about the physician, who plays in this case the *target* role.

Up to now, we have defined the main actors, the trust roles they can play, and the trust relationships and possible claims that the application considers. We also need to include for which purpose this information is going to be used, and this is the role of the *decides* connector. Just to mention two examples, the patient may decide to ask another physician for second opinion. In order to decide who this other physician is, he uses reputation information about the physician (annotation *decision criteria*). Also, the physician may ask for a new wearable if his trust in the actual wearable falls below a certain threshold. Thus, we are using trust and reputation to help actors to make decisions at runtime.

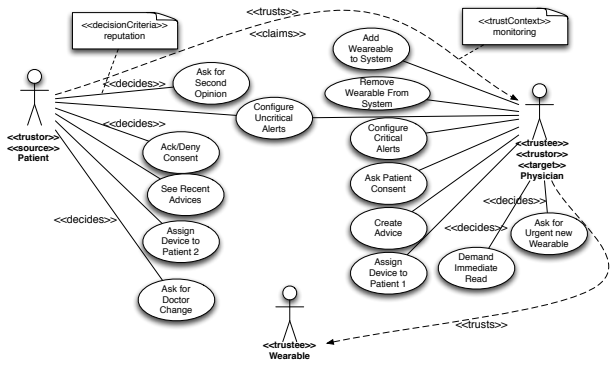

Figure 2: Trust-aware Use Case Diagram

Claims and trust relationships can be further refined in trust-aware class diagrams, as shown in Figure 3 and Figure 4[3]. Regarding the patient monitoring relationship, we specify the context of this relationship (which should be consistent with the context in the use case diagram), the dimension and format, which are one and numeric in this case, the scale, which is the interval $[0, 1]$, and the default value, which is 0.5. Thus, every trust relationship between a patient and a physician will be assigned by default (i.e. at bootstrap phase)

[3]We do not depict the trust relationship physician-wearable due to space limitations.

the value 0.5 and will take values between 0 and 1 along the application life. Also, we specify some information regarding the trustor and the trustee. In this relationship, the trustor is a human actor and has a subjective property that influences on the trust relationship: *capability belief*. This means that the belief that the patient has in the capability of the physician must be considered when setting the trust relationship, as stated also by the *trust engine* that updates the trust relationship. This engine uses a continuous engine (meaning that it will yield a continuous value by aggregating continuous variables). The list of variables used by the engine are the reputation of the trustee, the belief of the trustor, and the trustor's quality feedback, which is represented by the claim in Figure 4. Note that in the case of a claim, the *reputation engine* computes reputation for a given target, and not for a trust relationship. The reputation engine gathers the claims that different patients make about a given physician and computes a final reputation using an average. In addition to the claims, *time* is also used to derive this reputation value, which will be displayed by a *3 stars* notation.

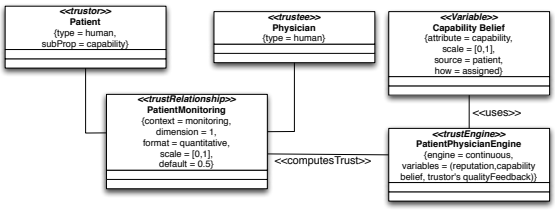

Figure 3: Patient-Physician relationship

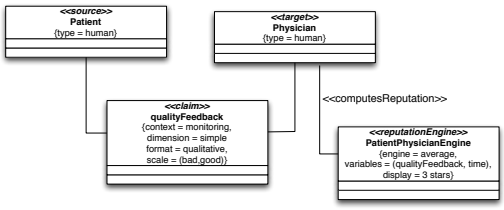

Figure 4: Quality Feedback Claim

For every variable defined in the engine, we can define a new *Variable* stereotype and specify some of the important properties of them. Figure 3 shows that capability belief, assigned by the patient, will take a value in the interval $[0, 1]$ and should capture the attribute *trustor's capability belief*. In the *physician-wearable* relationship[3], the physician triggers a system that monitors the wearable reliability variable, where the attributes captured by this variable are reliability and accuracy of the provided data.

Note that from the class diagrams information, especially after identifying

the variables that we need, we can go back to the use case diagram during the second iteration and add new use cases that should be included. We do not depict them due to space limitations but basically, the patient should have means of rating a physician and to set the physician preferences. This last use case captures the capability belief, as the preference list will likely be made by the patient in terms of this capability belief about the physicians. Finally, the physician should be able to measure the wearable reliability.

How the business and trust layers of the application interact may be a valuable information for designers. This can be depicted by a behavioural diagram, such as an activity diagram. The goal is to represent which actor can trigger a trust event and how, and what are the consequences of that trust event. We propose using swim lanes in order to make clearer the responsibilities of actors, the business logic and the trust logic in the whole application. Figure 5 depicts the trust event triggered when the patient asks for a second opinion.

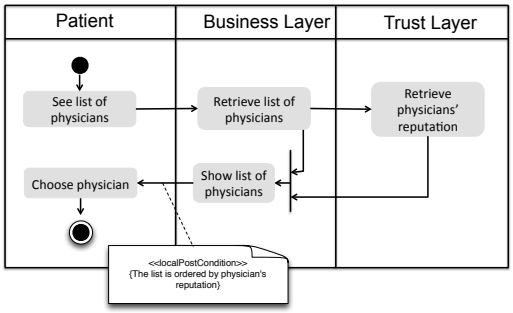

Figure 5: Activity Diagram for Use Case *Ask for Second Opinion*

The basic deployment for this application, without considering trust information, consists of a sensor that communicates with a wearable, which in turn, aggregates the information and sends it to a front-end server running the application. This front-end server will send the information to a back-end server that will store it into the patient's Electronic Health Record (EHR) and that executes a configuration application (i.e. to configure certain aspects of the application) only available to administrators. Figure 6 shows a trust-aware deployment diagram. The wearable device can decide, based on the front-end server reputation, to which server to send information. The same happens between the front-end server and the back-end server. Of course we are assuming that the final deployment will consist of, at least, two front-end servers and two back-end servers. Otherwise, a decision is not possible. We can also make explicit on which node the reputations for different entities in the system will be stored (i.e. assuming a centralized reputation model). In this case, a node is reserved to play the role of a reputation server that will store the reputation values for physicians, the front-end servers and back-end servers.

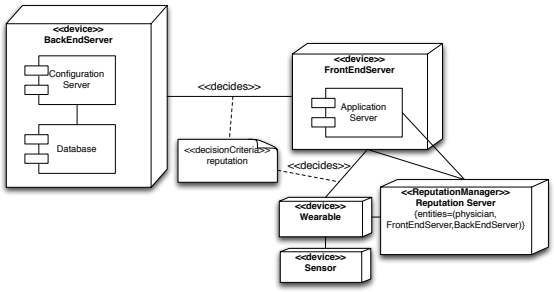

Figure 6: Trust-aware Deployment Diagram

6 Conclusion

Trust and reputation can be powerful mechanisms to improve security in complex, distributed systems and applications. We have proposed an extension to UML and some design guidelines to help requirements engineers and software designers to have a clearer view on trust requirements. This is not a straightforward task, as the concept of trust itself is difficult to grasp, and as there is an important gap between the social notion of trust and its software representation.

Our goal with this paper has been to continue bridging this gap, even though much work still remains to be done. First, the profile should be further extended in order to represent policies, credentials and trusted third parties, which constitute the roots of many trust management systems nowadays. The profile should also allow representing how trust information can be propagated between actors in the system. Trust derivation from lower software abstractions (e.g. trust among components) to higher level abstractions (e.g. trust among processing nodes), if possible at all, is an interesting field that requires much further exploration. Finally, there is a need for defining the semantics and constraints of each syntactic element. Tool support is then required to check compliance with these constraints and to derive design patterns and code from the specification. In this direction, how to integrate our approach with existing frameworks (e.g. UMLsec) should be analysed.

References

[1] Initial version of two case studies, evaluating methodologies. NESSoS Deliverable 11.3, October 2012.

[2] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 164–, Washington, DC, USA, 1996. IEEE Computer Society.

[3] Sudip Chakraborty and Indrajit Ray. Trustbac: integrating trust relationships into the rbac model for access control in open systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, SACMAT '06, pages 49–58, New York, NY, USA, 2006. ACM.

[4] Randy Farmer and Bryce Glass. *Building Web Reputation Systems*. Yahoo! Press, USA, 1st edition, 2010.

[5] Diego Gambetta. Can We Trust Trust? In *Trust: Making and Breaking Cooperative Relations*, pages 213–237. Basil Blackwell, 1988.

[6] Tyrone Grandison. *Trust management for internet applications*. PhD thesis, University of London, July 2002.

[7] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK, UK, 2002. Springer-Verlag.

[8] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, London, UK, UK, 2002. Springer-Verlag.

[9] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. *Model-Driven Risk Analysis - The CORAS Approach*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[10] Haralambos Mouratidis and Paolo Giorgini. Secure Tropos: a Security-Oriented Extension of the Tropos Methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2):285–309, 2007.

[11] Francisco Moyano, Carmen Fernandez-Gago, and Javier Lopez. A Conceptual Framework for Trust Models. In *9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012)*, volume 7449 of *Lectures Notes in Computer Science*, pages 93–104, Vienna, Sep 2012. Springer Verlag.

[12] Michalis Pavlidis, Haralambos Mouratidis, and Shareeful Islam. Modelling Security Using Trust Based Concepts. *IJSSE*, 3(2):36–53, 2012.

[13] Sarvapali D. Ramchurn, Dong Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *Knowl. Eng. Rev.*, 19(1):1–25, March 2004.

[14] Lars Rasmusson and Sverker Jansson. Simulated social control for secure internet commerce. In *Proceedings of the 1996 workshop on New security paradigms*, NSPW '96, pages 18–25, New York, NY, USA, 1996. ACM.

[15] Bruce Schneier. Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal*, 1999.

[16] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, January 2005.

[17] Mohammad Gias Uddin and Mohammad Zulkernine. Umltrust: towards developing trust-aware software. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 831–836, New York, NY, USA, 2008. ACM.

[18] Axel van Lamsweerde and Emmanuel Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Trans. Softw. Eng.*, 26(10):978–1005, October 2000.

[19] Dirk van Rooy and Jacques Bus. Trust and privacy in the future internet - a research perspective. *Identity in the Information Society*, 3(2):397–404, August 2010.