Modelling Security Requirements in Socio-Technical Systems with STS-Tool

Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti and Paolo Giorgini

Department of Information Engineering and Computer Science,
University of Trento, Italy
{paja,dalpiaz,poggianella,roberti,giorgini}@disi.unitn.it

Abstract. Security Requirements Engineering (SRE) deals with the specification of security requirements for the system-to-be starting with the analysis of security issues as soon as in the early requirements phase. STS-ml is an actor- and goal-oriented requirements modelling language for Socio-Technical Systems (STSs), which represents the security needs the stakeholders express as constraints over the interactions between actors. In this paper, we present STS-Tool, the security requirements engineering tool that supports STS-ml. STS-Tool allows for modelling a socio-technical system at a high level of abstraction, expressing constraints (security needs) over the interactions between the actors in the STS, and deriving security requirements in terms of social commitments (promises with contractual validity). It offers multi-view modelling, allowing designers to focus on a different perspective at a time, while promoting modularity.

1 Introduction

Socio-Technical Systems (STSs) are complex systems in which social actors interact with one another and with technical components to fulfil their goals. Each participant is autonomous, and the system is defined in terms of the interactions among actors, which may be: *social reliance*, actors rely on others to achieve their goals, and *information exchange*, actors exchange relevant information. In such systems, many security issues arise from the interaction between actors, and on how the exchanged information is manipulated. Therefore, *social* aspects are a main concern when analysing security.

The importance of considering security from a social and organisational perspective is widely recognised in literature [4,6,7,11]. However, such approaches either rely on high-level concepts that are hard to map to technical requirements (e.g. [4,7]), or suggest purely technical security mechanisms (e.g. [3]). In our view, SRE should start from high-level concerns and refine them into requirements for the system-to-be.

Goal-oriented approaches to security requirements engineering seem to be appropriate for designing secure STSs, since they build upon the concepts of intentional and social actors, who have objectives to achieve and interact with others to achieve them. Existing approaches, such as Tropos [1], Secure Tropos [8], and SI* [5], enable representing actors and their dependencies, in an organisational perspective, but they make the assumption that actors will behave as depicted in the model. Given that the participating actors in an STS are mutually independent—thus, their behaviour is not disclosed to others and they cannot be controlled—, we cannot make such assumption. Instead, the best a designer can do is to allow actors to specify security constraints over their interactions. We refer to these constraints as *security needs* to distinguish from the general security requirements of the system-to-be.

Based upon these principles, we have previously proposed STS-ml (Socio-Technical Security modelling language) [2], an actor- and goal-oriented modelling language that supports the modelling and analysis of security requirements for STSs. In this paper, we present STS-Tool [1], a security requirements engineering tool for STS-ml. The tool offers a graphical modelling environment to allow the definition of the system in terms of actors and their interactions.

The rest of the paper is organised as follows. Sec. 2 briefly outlines the STS-ml language. Sec. 3 presents the main features of STS-Tool. Sec. 4 describes a possible usage scenario. Sec. 5 presents conclusions and future work.

2 STS-ml

STS-ml builds on top of Tropos [1] and its security-oriented extension [5]. It revises the high-level organisational concepts from Tropos, maintaining a minimal set of concepts including actor, goal, delegation, etc., and uses the concept of *social commitment* among actors, to specify security requirements.

The particularity of STS-ml is that it allows actors to express *security needs* over interactions to constrain the way interaction is to take place. This is important, because the actors are mutually independent, and it is when they enter interactions that they might want to express their concerns regarding security. For instance, in e-commerce, a buyer would want a seller not to disclose its credit card details to other parties, and to use this information strictly to perform the payment of the acquired goods.

Social commitments [9] are promises with contractual validity that actors make and get from one another, to achieve their objectives. Formally, commitments are a quaternary relation *C(debtor, creditor, antecedent, consequent)* between a debtor and a creditor (both being actors), in which the debtor commits to the creditor that, if the antecedent is brought about, the consequent will be brought about. In STS-ml, we consider commitments about security-related properties. This concept is used to offer a guarantee that the debtor acknowledges the specified security need by making a commitment, and will behave as required by the security need by bringing about the commitment. For this, whenever a security need is specified from one actor to the other, a commitment on the other direction is expected from the second actor to satisfy the security need. For instance, in e-commerce, the provider commits to prospective buyers that their credit card details will not be disclosed to other parties, and will be used only for the payment of their acquired goods.

The outcome of STS-ml is a security requirements specification expressed in terms of commitments, in which the debtor actor is *responsible* for the satisfaction of the security requirement, whereas the creditor actor is the *requestor*. Fig. 1 outlines STS-ml: the specifications of security requirements for the system-to-be are derived once the modelling is done and the security needs imposed by the actors are expressed. STS-ml

[1] STS-Tool is available for download at `http://www.sts-tool.eu`

supports multi-view modelling: interactions among actors can be represented by focusing on orthogonal views. As shown in Fig. 1, STS-ml consists of three different views: *social*, *authorisation*, and *information*. The security needs are expressed in the operational view (Fig. 1), which consists of the three aforementioned views. The operational view is automatically mapped to the specification *security requirements* for the system-to-be, which supports the security needs expressed in the operational view.

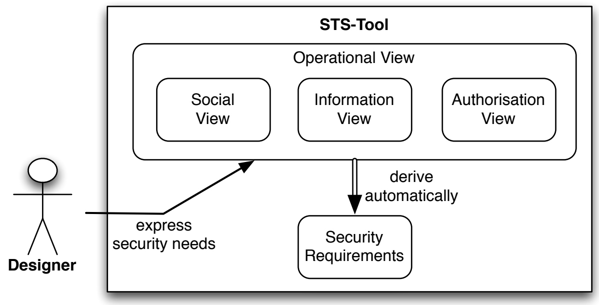


Fig. 1: From the operational view to security requirements

The *social view* represents actors as intentional and social entities. Actors are intentional as they have goals they want to achieve, and they are social, because they interact with others to get things done, mainly by *delegating goals*. Actors may possess documents, they may *use*, *modify*, or *produce* documents while achieving their goals, and they may *distribute* documents through *document provision* to other actors.

The *information view* gives a structured representation of the information and documents in the given setting. Information can be represented by one or more documents (made tangible by), and on the other hand one or more informations can be part of some document. It is important to keep track of how information and documents are interconnected, to be able to identify which information actors manipulate, while using, modifying, producing, or distributing documents for achieving their goals.

The *authorisation view* shows the permission flow from actor to actor, that is, the authorisations actors grant to others about information, specifying the operations actors can perform on the given information, namely *use*, *modify*, *produce*, and *distribute*. Apart from granting authority on performing operations, we consider also whether authority to further give authorisations is granted.

Following our intuition of relating security to interactions, we allow stakeholders to express their security needs over *goal delegations* and *authorisations* regarding information. Once the modelling is done, and all the security needs are specified, the list of of security requirements can be automatically derived from the operational view.

3 STS-Tool

STS-Tool is a modelling tool for STS-ml. It is a standalone application written in Java, and its core is based on Eclipse RCP Engine. It is distributed as a compressed archive for multiple platforms (Windows 32 and 64 bits, Mac OS X, Linux), and is freely available for download. STS-Tool has the following features:

- *Supports specification of projects*: the socio-technical security models are created within the scope of project containers. A project contains a set of models. Each project refers to a certain scenario. Typical operations on projects are supported: create, save, load, modify, rename.
- *Diagrammatic*: the tool enables the creation (drawing) of diagrams. Diagrams are created only within a project. Apart from typical create/modify/save/load operations, the following is also supported:
  - Export diagram to different file formats (png, pdf, etc.);
  - Provide *different views* on a diagram, specifically: *social view*, *information view*, *authorisation view*. Each view shows specific elements and hides others, while keeping always visible elements that serve as connection points between the views (e.g. roles and agents). Inter-view consistency is ensured by for instance propagating insertion/deletion of certain elements to all views.
- *Consistency checking*: the tool helps to create diagrams that follow the semantics of the modelling language, thus improving consistency and validity.
- *Generating requirements documents*: the tool allows the generation of requirements documents that contain the list of security requirements derived from the model in terms of social commitments. Moreover, this document contains information describing the models, which is customisable by the designer. The designer can select which concepts or relations he wants more information about.

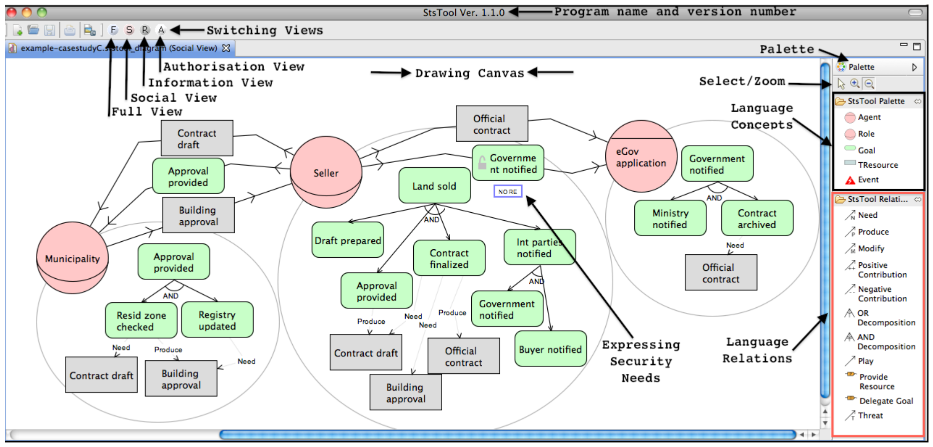
4 Modelling with STS-Tool

We will demonstrate the features of STS-Tool by modelling an illustrative example from a case study on e-Government.

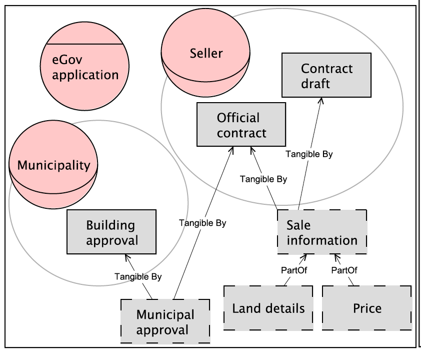
Example 1 (e-Government). Land selling involves not only finding a trustworthy buyer, but also exchanging several documents with various governmental bodies. The seller needs the municipality to certify that the land is residential zoning. The land selling process we consider is supported by an eGov application, through which the official contract (including the municipalitys certification) is sent to the ministry (who has the right to object) and is archived.

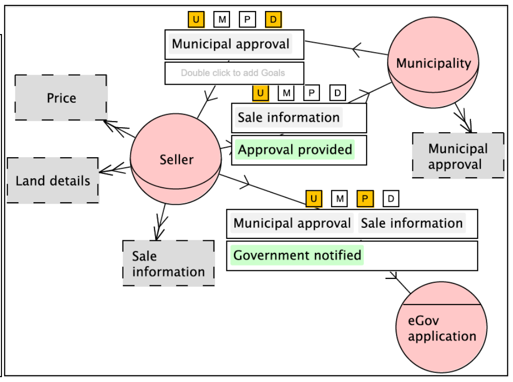
Fig. 2 shows the three orthogonal views supported by the tool, namely *social*, *information*, and *authorisation* view, together with the list of derived security requirements (*commitments view*). We present here the steps to follow for performing the modelling of our e-Government example to show how the tool facilitates and supports the modelling process:

1. *Building the Social View*: we start the modelling with the representation of the roles and agents present in the scenario. For this, we switch to *Social View* (Fig. 2a), and select these concepts from the Palette. In our example, we represent the *Municipality*, and the *Seller* as roles, whereas the *eGov application* as agent. When first created, roles and agents come together with their rationale (open compartment), so



(a) Social view





(b) Information view

(c) Authorisation view

Fig. 2: Multi-view modelling for the eGovernment scenario

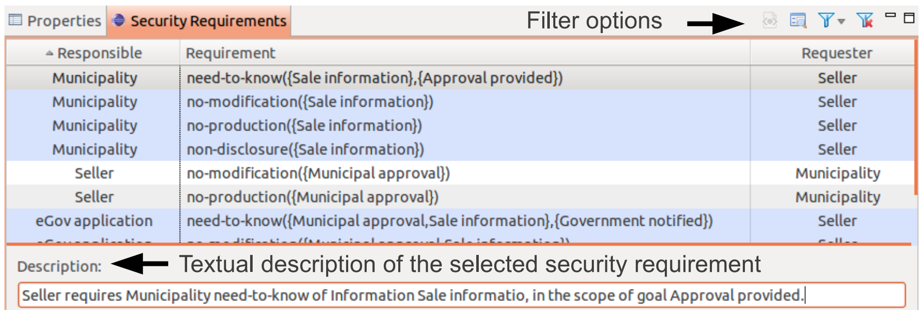
that we can specify goals or documents they have. The rationales can be hidden or expanded, to give the possibility to focus on some role/agent at a time. Actors want to achieve one or more goals. We place actor goals within their rationale: the seller has goal *Land sold*. Goals are refined by *AND/OR-decompositions* obtaining goal trees: *Land sold* is the root goal to be fulfilled. The tool facilitates a correct modelling of goal trees, by not allowing goal cycles.For some goals, actors need to rely on others through goal delegation. When drawing a delegation, the tool makes sure that the actor does have a goal before allowing to draw the goal delegation relationship. Then, the delegated goal is automatically created within the compartment of the delegatee. If a role/agent is delegated the same goal from different roles/actors, the tool maintains one copy of the goal within the delegatee's rationale. Following the semantics of the language, once a goal delegation is drawn from a delegator to a delegatee, the tool does not allow a delegation (or delegation chain) that ends up to the delegator, that is, delegation cycles are also not allowed in the tool.

2. *Are there any Security Needs?*: the designer analyses delegations, to see if any of the supported security needs applies over goal delegations. In Fig. 2a, the *Seller* requests *eGov application* not to repudiate the delegation of goal *Government notified*. To specify this using the tool, the designer clicks on the delegated goal, to have a drop down list of security needs and selects the desired ones. Some of the security needs are mutually exclusive; for these, the tool allows the selection of only one security need. Once the security need is selected, a locker appears on the goal to show that security needs have been specified, and the list of specified security needs appears below the goal, represented with distinguishable labels and different colours.
3. *Refining the Social View*: to achieve their goals, actors need, modify, and produce documents. For instance, the *Seller* needs document *Contract draft* to achieve goal *Contract finalised* (Fig. 2a). To model this, we choose the concept Document from the palette, name it *Contract draft* and then select the relation *Need* from the Palette and connect the goal with the document. The tool helps the designer by allowing this relation to be drawn only starting from the goal to the resource, not vice-versa.
4. *Analyse information*: we switch to *Information View* and *represent informations* and *documents*, relating them together. The tool inherits the roles/agents together with the documents from the social view, so the designer needs just specify how the different documents are interconnected (PartOf) and what information they represent (TangibleBy). For instance, *Official contract* and *Contract draft* contain (make tangible) *Sale information* (Fig. 2b). The tool allows TangibleBy to be drawn only from informations to documents, whereas PartOf to be drawn only between informations or documents respectively. Cycles of PartOfs are not allowed by the tool.
5. *Further refine the Social View*: the designer switches back to the *Social View* to represent information exchange. The tool allows to draw *document provisions* starting only from an actor that produces the document or is in possession of that document. In Fig. 2a, the *Seller* produces document *Official contract* and provides it to the *eGov application*, which needs this document to achieve goal *Contract archived*. Similarly, the designer represents the other interactions with *Municipality*.
6. *Model ownerships*: switch to the *Authorisation View* and define who are the owners of the different informations. The tool inherits roles/agents from the other views and the informations from the information view, so the designer just needs to link the roles/agents with the information, using the *Own* relation from the Palette. In our example, the *Seller* is the owner of *Sale information*.
7. *Model authorisations*: starting from information owners, we draw the authorisations they grant to other actors. For this, the relation Authorisation is selected from the Palette and is drawn starting from one actor to another. This action creates on the canvas an authorisation box that includes labels for the four supported operations (use-U, modify-M, produce-P,distribute-D), which the designer can select by clicking on the label. Below, there are two boxes, which specify that the designer should double click to respectively add a set of informations, and a set of goals. In our example, the *Seller* authorises the *Municipality* to use *Sale information* in the scope of goal *Approval provided* (Fig. 2c). Security needs over authorisations

are specified implicitly from the granted authorisation, so the designer needs not do anything, apart from specifying authorisations. For instance, the *Seller* requires the *Municipality* not to disclose *Sale information*, since the label 'D' for the operation distribute is not selected.

This modelling process (steps 1–7) is iterative. The views can be further refined, depending on the level of detail that is needed. The changes in one view have effects on other views. As described above, the different roles/agents are maintained throughout the views, so the addition/deletion of some role/agent would affect the other views. However, even in these cases, the tool provides support by checking that a role/agent is deleted only when it does not have any interactions with other roles/agents.

Once the modelling is done, and all security needs have been expressed, the tool allows the *automatic derivation* of security requirements. The security requirements are listed and they can be sorted or filtered according to their different attributes: Responsible, Requirement, and Requestor (Fig. 3). For instance, filtering the security requirements with respect to the Responsible actor, gives an idea of who are the actors responsible to satisfy the requirements, while filtering them according to the Requirement, groups together requirements that refer to the same type of security need. Finally, a textual *Description* is provided for every selected security requirement.



Responsible	Requirement	Requester
Municipality	need-to-know({Sale information},{Approval provided})	Seller
Municipality	no-modification({Sale information})	Seller
Municipality	no-production({Sale information})	Seller
Municipality	non-disclosure({Sale information})	Seller
Seller	no-modification({Municipal approval})	Municipality
Seller	no-production({Municipal approval})	Municipality
eGov application	need-to-know({Municipal approval,Sale information},{Government notified})	Seller

Description: Textual description of the selected security requirement

Seller requires Municipality need-to-know of Information Sale informatio, in the scope of goal Approval provided.

Fig. 3: Security requirements via commitments

At the end of this process, the tool allows designers to export models and generate automatically a *security requirements document*, which helps them communicate with stakeholders. This document is customisable: designers can choose among a number of model features to include in the report (e.g., including only subset of the actors).

5 Conclusion and future work

Our work on the STS-ml and tool is ongoing as part of the European research project Aniketos[2]. We are iteratively evaluating our language and tool on case studies from different domains, namely, telecommunications, air traffic management control, and

[2] `http://www.aniketos.eu/`

e-Government. These case studies offer different complexities, sizes and operational environments, so they prove suitable for our needs. The current version of the tool is a result of an iterative development process, where the release of internal versions of the tool has been followed by evaluation activities [10].

Future work about STS-Tool includes (i) embedding automated reasoning capabilities to identify inconsistencies and conflicts between requirements; and (ii) implementing a plugin management system that allows for adding functionalities to STS-Tool.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 257930 (Aniketos) and 256980 (NESSoS).

References

1. Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
2. Fabiano Dalpiaz, Elda Paja, and Paolo Giorgini. Security Requirements Engineering via Commitments. In *Proceedings of the First Workshop on Socio-Technical Aspects in Security and Trust (STAST'11)*, pages 1–8, 2011.
3. Donald G. Firesmith. Security Use Cases. *Journal of Object Technology*, 2(3):53–64, 2003.
4. Paolo Giorgini, Fabio Massacci, and John Mylopoulos. Requirement Engineering meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *LNCS*, pages 263–276. Springer, 2003.
5. Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Modeling Security Requirements through Ownership, Permission and Delegation. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE 2005)*, pages 167–176. IEEE Computer Society, 2005.
6. C.B. Haley, R. Laney, J.D. Moffett, and B. Nuseibeh. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.
7. Lin Liu, Eric Yu, and John Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting. In *Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE 2003)*, pages 151–161. IEEE Computer Society, 2003.
8. 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.
9. Munindar P. Singh. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
10. Sandra Trösterer, Elke Beck, Fabiano Dalpiaz, Elda Paja, Paolo Giorgini, and Manfred Tscheligi. Formative User-Centered Evaluation of Security Modeling: Results from a Case Study. *International Journal of Secure Software Engineering*, 3(1):1–19, 2012.
11. Axel van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pages 148–157. IEEE Computer Society, 2004.