Designing Secure Socio-Technical Systems with STS-ml

Elda Paja[1], Fabiano Dalpiaz[2], and Paolo Giorgini[1]

[1] University of Trento, Italy – {elda.paja, paolo.giorgini}@unitn.it
[2] University of Toronto, Canada – dalpiaz@cs.toronto.edu

Abstract. A Socio-Technical System (STS) is an interplay of humans, organizations and technical systems. STSs consist of interacting actors, which depend on one another to achieve their objectives. In previous work, we have proposed STS-ml, a security requirements modelling language (using *i**-like primitives such as actor, goal, delegation) for the design of secure STSs. STS-ml represents security requirements as constraints over the interactions (goal delegation and document exchange) among actors in the STS. In this work, we present the current version of STS-ml, which introduces further modelling primitives as well as sophisticated reasoning mechanisms to detect conflicts in security requirements.

1 Introduction

Socio-Technical Systems (STSs) are an interplay of social (human and organisations) and technical subsystems, which interact with one another to reach their objectives, making an STS a network of social relationships [1]. Each subsystem is a participant of the STS, acts according to its business policy (it is autonomous), and interacts with others. When relying upon others for carrying out tasks and manipulating information, a participant would like its own security requirements to be fulfilled. For example, if a buyer sends its personal data to a seller, the buyer may require the data to be used only for shipment purposes.

To deal with security in the early phases of STS design, we have previously proposed STS-ml [2] (Socio-Technical Security modelling language), an actor- and goal-oriented security requirements modelling language. STS-ml is based on the idea of relating security requirements to *interaction*. As such, the language allows stakeholders to express *security needs* over their interactions to constrain the way interaction is to take place, as in the buyer/seller example above.

STS-ml specifies security requirements as *social commitments*, promises with contractual validity made by an actor to another. One actor commits to another that, while delivering some service, it will comply with the required security needs. In the example above, a security requirement is that the seller commits not to disclose personal data to other parties.

These specifications guide the design of a STS that satisfies the security requirements. However, in certain cases, the specification may be *inconsistent*, i.e., one or more requirements might be conflicting. It is, thus, not possible to implement a STS that satisfies all requirements of the specification.

Coping with such conflicts at requirements-time avoids developing a non-compliant and hard-to-change system. We propose to rely on automated reasoning to identify and resolve these conflicts. This choice is justified by our gathered evidence [5] that requirements models are large and that even skilled analysts would be unable to identify all the conflicts in a model.

2 Objectives of the research

We aim at creating a framework that supports the identification of inconsistencies (conflicts) among security requirements in a STS-ml specification. We focus on two types of conflicts: (1) *among security requirements*: two or more security requirements cannot be implemented by the same system. For instance, access to some information may be granted from one stakeholder, and prohibited from another. The different authorisations are conflicting, and one of them needs to be relaxed and adapted, perhaps by negotiating the needs of the authorisee; and (2) *between actors' business policies and security requirements*: an actor's policy may specify that some information shall be accessed, while no authorisation is granted by the information owner. We provide examples in Sec. 3.1.

While analysts may detect some conflicts by looking at the graphical models, others are harder to spot—especially in large models—and require computer-aided support. To such extent, we have formalised the semantics of the STS-ml primitives and that of its *security requirements*. This effort enables us developing automated reasoning techniques for the identification of conflicts. Our longer-term objective is to support the resolution of conflicts too.

3 Scientific contributions

STS-ml is an actor- and goal-oriented security requirements engineering method. It is composed of the modelling language and its support tool STS-Tool [3]. The contributions of this research are as follows:

- a revised version of the STS-ml language, including a wider set of security requirements;
- a formal framework to identify conflicts among security requirements, as well as among actors' business policies and the security requirements they are required to comply with;
- an implementation of the framework in disjuctive Datalog. STS-Tool supports the graphical visualisation of conflicts in STS-ml diagrams.

3.1 The STS-ml framework for security requirements engineering

STS-ml has been first proposed in [2], here we present the current version of STS-ml. STS-ml includes high-level organisational concepts such as actor, goal,

[3] `http://www.sts-tool.eu`

delegation, etc. Security requirements in STS-ml models are mapped to *social commitments* [4]—contracts among actors—that actors in the STS shall comply with at runtime. STS-ml modelling consists of three complementary views of the same model, namely *social*, *information*, and *authorisation view* (see Fig. 1), so that different interactions among actors can be analysed by concentrating on a specific view at a time. Interview consistency is ensured by STS-Tool [4].

Example 1 (Travel Planning). A tourist wants to organise a trip using a Travel Agency Service (*TAS*). *TAS* allows end users to read about various destinations, book flights and hotels, hire a car, etc., and uses the *Amadeus flight service* to book flight tickets. To book hotels, the *Tourist* has chosen to directly contact the *Hotel* himself, without interacting with *TAS.*

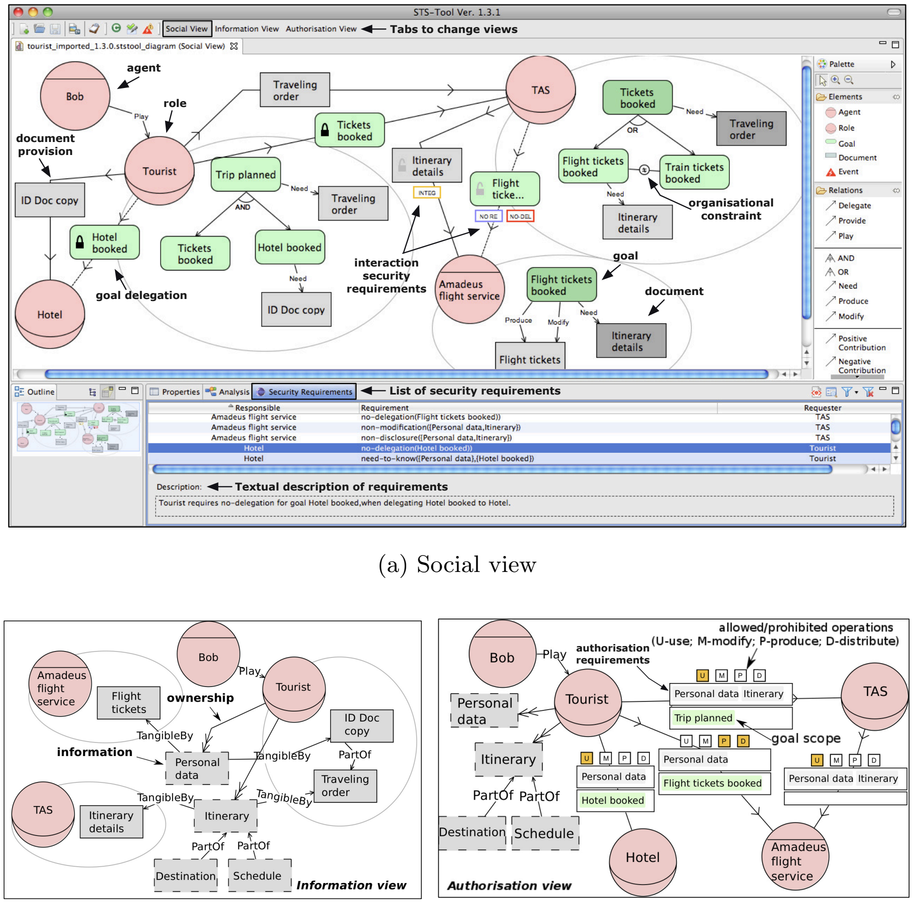
The *social view* (Fig. 1a) represents actors as intentional and social entities. Actors are intentional as they aim to attain their goals, and they are social, for they interact with others by delegating goals and exchanging documents. STS-ml supports two types of actors: agents—concrete participants, and roles—abstract actors, used when the actual participant is unknown. In our example, we represent the *TAS* as a role, and the *Amadeus flight service* as an agent, as it refers to a specific flight service. Actors may possess documents, they may *use*, *modify*, or *produce* documents while achieving their goals. For instance, *Tourist* wants to achieve the goal *trip planned*, for which it needs to both have *tickets booked* and *hotel booked.* To book the hotel it needs document *ID Doc copy.*

The *information view* (Fig. 1b) gives a structured representation of actors' information and documents, and the way they are interconnected. Information can be represented by one or more documents (through TangibleBy), and on the other hand one or more information entities can be part of some document. For instance, information *Personal data* is represented by both *ID Doc copy* and *Flight tickets* documents. We keep track of how information and documents are interconnected to identify which information actors manipulate, when they *use*, *modify*, *produce*, or *distribute* documents to achieve their goals. We take a pessimistic approach assuming that whenever a document is altered, the information it makes tangible is also altered. For instance, the *Amadeus flight service* modifies information *Personal data* when modifying document *Flight tickets.*

The *authorisation view* (Fig. 1c) shows the authorisations actors grant to others over information, specifying which operations they are allowed (prohibited) to do, for which goals (scope), and whether authorisation can be further transferred or not. For instance, *Tourist* authorises *TAS* to use (U selected) information *Personal data* and *Itinerary* in the scope of the goal *Tickets booked* granting a transferrable authorisation (authorisation's arrow line is continuous). Through its three views, STS-ml supports different requirements types:

- *Business policies* are expressed by specifying actors, their goals, delegations, document provisions, and how actors manipulate documents to fulfil goals. In a nutshell, they are represented by an actor's goal model.

[4] http://www.sts-tool.eu



Responsible	Requirement	Requester
Amadeus flight service	no-delegation(Flight tickets booked))	TAS
Amadeus flight service	non-modification({Personal data,Itinerary})	TAS
Amadeus flight service	non-disclosure({Personal data,Itinerary})	TAS
Hotel	no-delegation(Hotel booked))	Tourist
Hotel	need-to-know({Personal data},{Hotel booked})	Tourist

Tourist requires no-delegation for goal Hotel booked,when delegating Hotel booked to Hotel.

(a) Social view

(b) Information view

(c) Authorisation view

Fig. 1: Multi-view modelling for the travel planning scenario

- *Interaction (security) requirements* are security constraints on goal delegations and document provisions. STS-ml supports *non-repudiation*, four types of *redundancy*, *no-redelegation*, *availability* and *integrity-of-transmission* interaction security requirements. For instance, *TAS* requires the *Amadeus flight service* not to repudiate the delegation of goal *Flight tickets booked*, and also not to redelegate this goal. The *Amadeus flight service*, on the other hand, requires that the integrity of document *Itinerary details* is preserved.
- *Authorisation requirements* determine which information can be used, how, for which purpose, and by whom; at the same time they define also prohibitions over the same information. In this category, STS-ml supports *non-reauthorisation*, *need-to-know* of information, *non-usage*, *non-modification*,

non-production and *non-disclosure* of information. While authorising *TAS* to use (U selected) information *Personal data* and *Itinerary* in the scope of the goal *Tickets booked*, the tourist is prohibiting *TAS* to perform the rest of operations, requiring *non-modification*, *non-production* and *non-disclosure* of the given information. Moreover, since authorisation is limited to a goal scope, this expresses a *need-to-know* requirement over information, demanding it to be used only for the specified scope;

- *Organisational constraints* determine constraints on the adoption of roles and the uptake of responsibilities. STS-ml supports *separation* and *binding of duties* both over roles and over goals. For instance, a separation of duty is expressed between goals *Flight tickets booked* and *Train tickets booked* requiring *TAS* not to ever achieve both these goals.

Together, interaction requirements, authorisation requirements, and organisational constraints are the security requirements of STS-ml. A STS-ml model is consistent if the actors' business policies comply with the security requirements.

3.2 Detecting conflicts in security requirements

We are concerned with the verification of two types of inconsistencies that relate to security requirements: (i) identifying conflicts among security requirements, that is, verifying whether the simultaneous specification of different security requirements brings up conflicts; and (ii) identifying conflicts raised by the specification of the security requirements with respect to an actor's business policy. For each of these two categories, we identify conflicts in our running example, and provide insights on how they could be fixed. Here we provide just the intuition behind the identification of these conflicts. The formal framework, its implementation in disjunctive Datalog, and experimental results from an industrial case study on eGovernment are presented in a technical report (see [3]).

The first challenge is to obtain a set of security requirements that is consistent, which enables us to build a secure socio-technical system (that violates no security requirement). The focus of STS-ml is mainly on information security; thus, we have begun by tackling authorisation conflicts. Detecting conflicts among other types of security requirements is still work in progress.

For instance, the *Tourist* does not authorise the *Amadeus flight service* to use his/her *Personal Data*, while *TAS* authorises the *Amadeus flight service* to use *Tourist*'s *Personal Data*. This situation represents an authorisation conflict for the *Amadeus flight service*. Specifically, this conflict relates to the "use" operation. Our framework supports also conflicts about modification, production, distribution, and *authorisation transferibility* (i.e., an actor is granted authorisation to perform operations but not that of further distributing the authorisation).

The second challenge is to verify if one or more security requirements conflict with the private business policy of an actor (as dictated by its goal model). For instance, suppose the *Hotel*'s business policy requires handling all bookings by means of a reservation system service. When *Tourist* specifies a *no-redelegation* security requirement over the delegation of the goal *Hotel booked*, there is a conflict between what the security requirement demands the Hotel to do, and what its internal requirement dictates. Let us show another example of this kind of conflict. The *Amadeus flight service* modifies *Flight tickets* when achieving goal *Flight tickets booked*. *Flight tickets* makes tangible tourist's *personal data*, for which no authorisation on modification (M) is granted, instead a *non-modification* authorisation requirement is specified. The *Amadeus flight service* might need to modify *Flight tickets* to accommodate changes in the itinerary of the *Tourist*, so either the *Tourist* decides he/she does not want the flight tickets modified, or he/she should grant the permission to the *Amadeus flight service*.

4 Conclusions, ongoing and future work

STS-ml is a framework to model and reason over security requirements for STSs. The current version of the framework is a result of an iterative approach, of various evaluations conducted with industrial partners of the FP7 European project Aniketos. The framework has proven suitable to model real case studies spanning from eGovernment to Air Traffic Management Control. STS-ml supports a rich set of security requirements, for which we can identify conflicts.

Ongoing work includes different directions: (i) developing a new version of the tool with improved usability, and (ii) exploring how STS-ml can inform later phases in the design of secure STSs, e.g., the definition of access control policies.

Future work includes: (i) devising other reasoning techniques for more sophisticated reasoning, (ii) exploring techniques for conflict resolution, and (iii) reducing the learning curve for STS-ml, also via self-learning mechanisms.

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

References

1. F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Adaptive Socio-Technical Systems: a Requirements-driven Approach. *Requirements Engineering*, 18(1):1–24, 2013.
2. F. Dalpiaz, E. Paja, and P. Giorgini. Security Requirements Engineering via Commitments. In *Proceedings of the First Workshop on Socio-Technical Aspects in Security and Trust (STAST)*, pages 1–8, 2011.
3. E. Paja, F. Dalpiaz, and P. Giorgini. Identifying Conflicts in Security Requirements with STS-ml. TR DISI-12-041, University of Trento, `http://disi.unitn.it/~paja/tr-identifying-sec-conflicts.pdf`, 2012.
4. M. P. Singh. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law*, 7(1):97–113, 1999.
5. S. Trösterer, E. Beck, F. Dalpiaz, E. Paja, P. Giorgini, and M. Tscheligi. Formative User-Centered Evaluation of Security Modeling: Results from a Case Study. *International Journal of Secure Software Engineering*, 3(1):1–19, 2012.