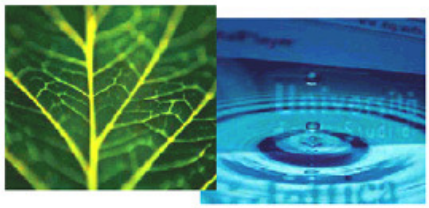
PhD Dissertation



International Doctorate School in Information and Communication Technologies

DISI - University of Trento

STS: A SECURITY REQUIREMENTS ENGINEERING METHODOLOGY FOR SOCIO-TECHNICAL SYSTEMS

Elda Paja

Advisor:

Prof. Paolo Giorgini

Università degli Studi di Trento

May 2014

Abstract

Today's software systems are situated within larger socio-technical systems, wherein they interact — by exchanging data and delegating tasks — with other technical components, humans, and organisations. The components (actors) of a socio-technical system are autonomous and loosely controllable. Therefore, when interacting, they may endanger security by, for example, disclosing confidential information, breaking the integrity of others' data, and relying on untrusted third parties, among others. The design of a secure software system cannot disregard its collocation within a socio-technical context, where security is threatened not only by technical attacks, but also by social and organisational threats.

This thesis proposes a tool-supported model-driven methodology, namely STS, for conducting security requirements engineering for socio-technical systems. In STS, security requirements are specified — using the STS-ml requirements modelling language — as social contracts that constrain the social interactions and the responsibilities of the actors in the socio-technical system. A particular feature of STS-ml is that it clearly distinguishes information from its representation — in terms of documents, and separates information flow from the permissions or prohibitions actors specify to others over their interactions. This separation allows STS-ml to support a rich set of security requirements. The requirements models of STS-ml have a formal semantics which enables automated reasoning for detecting possible conflicts among security requirements as well as conflicts between security requirements and actors' business policies — how they intend to achieve their objectives. Importantly, automated reasoning techniques are proposed to calculate the impact of social threats on actors' information and their objectives. Modelling and reasoning capabilities are supported by STS-Tool.

The effectiveness of STS methodology in modelling, and ultimately specifying security requirements for various socio-technical systems, is validated with the help of case studies from different domains. We assess the scalability for the implementation of the conflict identification algorithms conducting a scalability study using data from one of the case studies. Finally, we report on the results from user-oriented empirical evaluations of the STS methodology, the STS-ml modelling language, and the STS-Tool. These studies have been conducted over the past three years starting from the initial proposal of the methodology, language, and tool, in order to improve them after each evaluation.

Keywords

[*Security requirements, socio-technical systems, automated reasoning, requirements models*]

Acknowledgements

Having arrived to the end of this challenging, but gratifying journey, I want to take the time to thank the people that have played a significant role in the completion of my PhD.

I am very grateful to Professor Paolo Giorgini, my supervisor, not only for his supervision and guidance, but for believing in me and pushing me to do more than I thought I could. I have learned a lot through the course of these years. Professor Giorgini has been my biggest critic, the discussions and meetings with him have been crucial to providing me the skills and preparation to present and defend my work. I am thankful also for having had the chance to travel extensively throughout my doctoral studies to present my work, giving me the opportunity to interact with some of the best researchers in our field.

I am very thankful to Professor John Mylopoulos for convincing me to embark in this exciting journey. Thank you for all the interactions and discussions, and most importantly for the opportunity to further continue my research.

Thanks to Professor Haralambos Mouratidis (University of East London), Professor Oscar Pastor (Valencia University of Technology), and Professor Fabio Massacci (Università degli Studi di Trento), for accepting the invitation to participate in my thesis committee. I am very fortunate to have had the opportunity to interact with them and to discuss my work with them over the years. I owe a special thanks to Professor Fabio Massacci for all the feedback he has provided me since the very beginning of my research, throughout the various presentations and discussions, as well as for his advice on choices I had to make.

I am extremely thankful to my main collaborator and co-author, Dr. Fabiano Dalpiaz. It has been a real pleasure working with you and I hope we continue our collaboration, which has been very fruitful. I have learned a lot from you during these years, you have been a great example to me, thank you. Most importantly, I consider you a very good friend.

I am indebted to Mauro Poggianella, who has developed STS-Tool. Thank you for all the great work and above all for your patience. Apparently, even computer scientists, worse, requirements engineers, sometimes make bad customers. ;-)

I am very thankful to two professors I have closely collaborated and worked with, Professor Alex Borgida (Rutgers University) and Professor Travis Breaux (Carnegie Mellon University). Thank you Alex for all the discussions, your feedback, and for scrutinizingly reviewing the formalisation of STS-ml. Travis Breaux hosted me during my visit at Carnegie Mellon University. Thank you for the great opportunity. Above all, thank you for the warm hospitality. Looking forward to continuing our collaboration.

I am grateful to the industrial partners of the EU Funded FP7 Project Aniketos, who have used the modelling framework since the beginning and have provided continuous feedback for its improvement. Above all, thank you for participating in the evaluation workshops. In

particular, thanks to Stéphane Paul and Per Håkon Meland for the constructive criticism, and thanks to Sandra Trösterer and Elke Beck for all the help and support in organising the empirical evaluation workshops.

Thanks to Dr. Raian Ali and Dr. Amit K. Chopra. Although our collaboration was brief, you have provided me with useful suggestions and directions. For this I am thankful. A special thanks goes to Dr. Federica Paci, I have benefited much from the discussions together, and I hope that, finally, we will have a chance to collaborate. Thanks to Dr. Jennifer Horkoff for the suggestions and for proofreading parts of this thesis.

I am grateful to my research group in Trento for the stimulating discussions during our seminars, but also for the coffee breaks, lunches, and dinners together. These gatherings have served as great opportunities not only to discuss research, but also to socialise. Sorry for not mentioning any names, I am afraid I would forget to mention someone. I consider myself fortunate to have worked and studied in such an interactive and supportive group.

Finally, thanks to all my friends. Thanks to my good old friends home for the support and the words of encouragement, even though we are far. Thanks to the friends I have made in Pittsburgh: you made my stay very nice, I hope we keep in touch. Thanks to the good friends I have made here in Trento, these years would have not been the same without you. In particular, I want to thank Davide, my best friend, you are family to me. Thanks to Dome and Luigi for their love and support, I feel extremely lucky to have you in my life.

Thanks to my parents Bardha and Bardhi, and to my brother Geni, for being always so proud, and for the immense support and encouragement through the course of these years.

Last but not least, thank you Gerti, my love, for the great patience and the unconditional love. Thank you for always believing in me, for standing by me and for bearing with me all these years, I know it has not been easy. Above all, thank you for not thinking it twice about joining me in Trento, and for always supporting my choices and putting my career first. This achievement is dedicated to you.

To all of you, from the bottom of my heart, *Thank you*, *Grazie*, *Faleminderit*,
Elda

The work compiled in this thesis has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 257930 (Aniketos) and 256980 (NESSoS).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The failure of large complex systems to meet their deadlines, costs, and stakeholder expectations are not, by and large, failures of technology. Rather, these projects fail because they do not recognize the social and organisational complexity of the environment in which the systems are deployed. The consequences of this are unstable requirements, poor systems design and user interfaces that are inefficient and ineffective.

Baxter and Sommerville [2011]

Most of today's software systems are part of larger systems, that include not only technical components, but also humans and organisations. These larger systems are socio-technical systems [Emery, 1959; Sommerville et al., 2012; Dalpiaz et al., 2013a], and they consist of autonomous subsystems (*participants*) that interact—by exchanging messages, socially relying on one another and exchanging information—to achieve their objectives. Examples include virtual communities, e-commerce, smart cities, and healthcare systems. Socio-technical systems open up new challenges for software engineers when dealing with security, for the threats analysts need to consider are not just technical, but also social, and should consider the underlying business processes of an organisation.

In this chapter, we discuss the problem of developing secure socio-technical systems from a requirements engineering perspective. Specifically, we tackle the problem of specifying the security requirements a secure socio-technical system should satisfy. This phase (requirements engineering) is relevant for the design of secure socio-technical systems. We present the problem and motivate its importance together with the challenges we face in Section 1.1. We motivate the focus on model-driven—and in particular goal-oriented—security requirements engineering in Section 1.2. We present the research roadmap for this thesis presenting the research questions, the corresponding expected outcomes, and evaluation actitivies in Section 1.3. Additionally, this chapter presents the motivating scenario we will use throughout the thesis to illustrate our approach in Section 1.4, the contributions of the work in Section 1.5, the thesis

outline in Section 1.6, and finally, the list of publications this research work has produced in Section 1.7.

1.1 The security problem in socio-technical systems

Socio-technical systems stand at the core of how people work and collaborate with others while using the technical systems to get things done. For instance, in a healthcare socio-technical system, organisations such as hospitals and laboratories interact with one another and with humans such as doctors and nurses, to provide healthcare services to patients; all interact with hardware equipment and software within the system: doctors use medical equipment to visit patients and prescribe medications; citizens provide their personal data to be hospitalised and receive healthcare; hospital staff enters patient data in a database, etc. Socio-technical systems exist because their underlying (component) subsystems cannot do everything on their own (even though they are operationally independent), quite often they need to rely on others and interact with them to achieve their desired objectives. For instance, consider the classical interaction between doctors and patients. Patients need to rely on doctors to receive healthcare services, while doctors need to have as much information as possible about the patients to provide them the best healthcare services, that is, doctors need to rely on patients to provide this information.

But, socio-technical systems are decentralised and their component subsystems are autonomous and loosely controllable [Northrop et al., 2006]. This raises a number of security issues, especially when interaction involves exchange of information. For instance, patients would want their medical history to remain confidential and be consulted only by the said doctor or specialized physicians that would provide them with healthcare [Sankar et al., 2003]. Any disclosure of this information to unauthorised parties would have unpleasant effects on the patients, causing them embarrassment or fear of discrimination [Appari and Johnson, 2010]. Therefore, a question security designers need to answer is: "*How can patients' confidential information, namely their medical history and current conditions, be protected?*".

Let us analyse this problem. Patient confidential information might be protected from a technical point of view, because of high security protection on the server where the information is stored, or because of security mechanisms implemented in the devices used to access such information, which may significantly reduce the possibility of malicious outsiders to access the information. However, this information would not be protected from authorised users such as nurses or medical personnel disclosing this information either accidentally or on purpose. Indeed, internal attacks caused by participants of the system itself, exploiting loopholes in the system, are often more harmful than external attacks [Massacci and Zannone, 2008; Colwill, 2009; Roy Sarkar, 2010].

Therefore, in addressing the security problem in socio-technical systems, we need to account

for the various *stakeholders* (participants) [1] in a socio-technical system (e.g., in a healthcare socio-technical system, patients, doctors, nurses, etc.), as well as the *interactions* they enter, be these to achieve some objective (e.g. receive healthcare) or to exchange information (e.g. provide medical history), in order to identify where and how things could go wrong and result in security violations (e.g. nurse disclosing information).

In this thesis, we argue that security issues arise mostly because of interaction, which is the glue that holds together the overall socio-technical system. This is under the assumption that as long as participants do not interact, that is, the fulfilment of their objectives is solely dependent on their own capabilities and the protection of information is their liability (e.g. patient keeps information to himself), then no harm could happen (e.g. information is not disclosed to unauthorised parties).

Nevertheless, capturing these interactions is only one facet of dealing with security. We need to be able to capture what are the concerns stakeholders have when entering these interactions. This is because it is when stakeholders interact with others, relying to fulfil a desired objective or provide relevant information, that they become concerned about these *assets*, i.e., their objectives and information. As such, one might want to specify how this information is to be manipulated by others. For instance, celebrities might want to protect embarrassing health information from being transmitted to the media and made public.

Therefore, one of the challenges we need to address is *relating security to interaction*, providing a way to represent and capture the interactions among the various participants of a socio-technical system, as well as providing a way to allow them to express their concerns with respect to security over the interactions they enter. This is important to understand *why* security mechanisms are needed, to protect stakeholders' assets, and *what* security concerns they have, expressing their needs with respect to security over the interactions about their given assets.

The security problem in traditional software systems has been analysed and tackled by considering technical security mechanisms to overcome possible security breaches and protect the system from malicious attackers [Liu et al., 2003; Jürjens, 2002]. However, in tackling the security problem in socio-technical systems we cannot rely on technical mechanisms alone. Instead, a thorough analysis of social and organisational aspects is required for the specification of a secure socio-technical system.

Consider that healthcare is opening the door to the use of tablet devices in examining patients' medical records, exchanging information among physicians as well as accessing the healthcare system of the hospital [Lewis, 2012]. While this evolution maximises efficiency, it poses challenges about information security. In many cases physician-owned devices are used.

[1] Note that we use the terms *stakeholder* and *participant* interchangeably, because they refer to different facets of the same entity. Indeed stakeholders are the subjects that express their needs on the system-to-be on behalf of the actual participants of the socio-technical system.

These devices may leave the hospitals together with their owners, and cannot be controlled the same way as corporate computers. What if a doctor shares sensitive information with a friend instead of a specialised physician? What if the doctor needs to lookup medical records when called at a restaurant? Even worse, what if he forgets the device at a restaurant table? Even though accidental, this action results in disclosing confidential patient information to unauthorised parties. These new ways of operating (new organisational business processes) open up new challenges when specifying a secure socio-technical system, for they pose severe *social* and *organisational threats* to the system-to-be.

Social concerns are related to security violations caused by the involved humans or organisations participating in the socio-technical system, while *organisational* concerns are related to loopholes in the underlying business processes that could be exploited by the participants themselves. Thus, another challenge we need to face is adequately capture these social and organisational aspects underlying socio-technical systems that could be exploited, resulting in security violations.

Considering the autonomous and heterogeneous nature of stakeholders in a socio-technical system, they have different objectives for entering the system [van Lamsweerde et al., 1998]. Security concerns and needs are no exception: eliciting stakeholders' objectives (from multiple stakeholders) together with their security needs will potentially lead to inconsistencies. For instance, the need of a doctor to have a patient's detailed medical history (for a better diagnosis) does not necessarily comply with the will of the patient to keep some of this information private (to himself) or at least not to be inserted in the hospital's information system. As such, requirements analysts need to account for these divergences, identify possible conflicts, and try to reconcile the different needs stakeholders have while respecting their autonomy. Socio-technical systems open up new challenges for requirements analysts and security engineers in reconciling participants' needs with respect to security without affecting their autonomy.

Scientific relevance. Summarising, the challenges security requirements engineers [2] face when dealing with the security problem in socio-technical systems, emerging from the very nature of socio-technical systems and their participants, are:

C_1 representing stakeholders as the main components of a socio-technical system together with their *assets*,

C_2 capturing *security on interactions*,

C_3 capturing stakeholders' *security needs* over their interactions,

C_4 capturing and analyzing *social* and *organisational* concerns, and representing *social threats*,

[2] A role covered by an expert in both requirements engineering and security, otherwise it requires two experts: a requirements analyst and a security engineer.

C_5 identifying potential conflicts resulting from stakeholders' different concerns and reconciling them without affecting stakeholders' autonomy.

Societal relevance. Dealing with security is an important activity, for security violations may have severe repercussions. In the healthcare scenario, security breaches do not only affect patients (should their security needs of maintaining information confidential be violated), but they might have major consequences on hospital staff and the hospital itself. The hospital risks a lawsuit at the very least, with potential financial losses. Hence, dealing with security in socio-technical systems is an important and critical issue not to be overlooked, considering also the societal value and impact these systems have. In doing so, we need to address the challenges we laid down.

1.2 Security requirements engineering to the rescue

Before the rise of security requirements engineering [Mouratidis et al., 2003; Massacci and Zannone, 2008], security was typically left as an afterthought, resulting in substantial monetary expenditures to cover the damage in the case of security breaches, as well as challenges faced in integrating security measures and mechanisms into the rigid architecture of an already operating system as acknowledged by [Anderson, 2008; Johnstone, 2009]. Security mechanisms might not fit in the pre-existing architecture or they might even be in conflict with the design under consideration [Giorgini et al., 2006; Mouratidis and Giorgini, 2007b].

These issues have lead to *security by design*, which pressed on the necessity to deal with security issues as early as possible [Devanbu and Stubblebine, 2000; Dubois and Mouratidis, 2010], already during the requirements engineering phase, to inform the later phases such as design, implementation, and so on. Devanbu and Stubblebine [2000] are among the pioneers in the requirements engineering community to recognise the importance of considering security as part of the early stages of software development. Security requirements engineering, in particular, is now well established within requirements engineering, considering the plethora of proposals of security requirements frameworks (see [Mellado et al., 2010] for a review). What is interesting is that most of these frameworks advocate the use of models to consider security issues when developing secure systems. Model-driven security requirements engineering offers the advantages of precisely documenting and analysing security requirements together with design requirements [Basin et al., 2011]. This allows the security requirements engineer to adequately capture security requirements and account for them early in system development, while integrating and accommodating them along design requirements.

A requirements engineering process [Sommerville and Sawyer, 1997] encompasses requirements *elicitation* (discovering, documenting and understanding users' needs), analysis (refining users' needs), *specification* (clearly documenting users' needs), *verification* (ensuring require-

ments are consistent and clear), and *management* (coordinating and documenting the previous activities). A security requirements engineering process should, similarly, start with the elicitation of stakeholders' security *needs* and terminate with the specification of the security *requirements* for a system-to-be. The needs of the stakeholders motivate *why* the system has to be secure, and explain *what* are the security requirements for the system. Subsequently, these requirements lead to the design and implementation of security mechanisms (which define *how* requirements are satisfied by the system). For instance, a requirement for preserving the integrity of transmission of information is fulfilled by integrity assurance mechanisms such as data replication, mirroring, and checksumming; a requirement for the authentication of users to access information is fulfilled by authentication mechanisms, which vary from system to system: in case of a webmail system a username and password may be used, badges may be used to access laboratory spaces, while iris and face recognition may be used to enter military installations or facilities. Which security mechanism is more appropriate, should, however, become clear from and be the logical consequence of the modelling and analysis processes that are intrinsic to the security requirements engineering process [Giorgini et al., 2003], refining high-level needs of the *why* dimension and understanding the *how* dimension.

Following on [Giorgini et al., 2003], we advocate that a further step needs to be taken: distinguish among different types of assets stakeholders have and wish to protect, making information a first-class citizen. This differentiation will allow analysing how these assets are used or manipulated and exchanged with others, what are the security needs the owners have with respect to their assets, etc. This differentiation is core to a clear understanding of *why* security is needed, *what* we need to protect and *what* security needs, to then answer *how* these security needs can be satisfied. For instance, this distinction allows us to represent patients as the owners of their personal data and medical records, and the fact that they want to protect information about any eventual infective disease. Patients may specify that this information should be accessible only to their curing physicians, not to other medical staff, and can be used only for the purpose of providing the best healthcare service. To insure that patient security concerns are satisfied, we need to introduce authentication mechanisms for the first concern so that the curing physicians identify themselves to access patient information, and access control mechanisms to ensure information is not used for any other purposes than the specified one for the second concern, respectively.

Looking at existing work, traditional approaches to requirements engineering treated security as a particular type of non-functional requirements [Yu and Cysneiros, 2002; Liu et al., 2002; Bresciani et al., 2004], introducing quality constraints under which the system must operate. The limitation of these approaches is that they do not offer support for identifying who is concerned about which assets, that is, whose assets are at stake, what are the actual concerns in protecting these assets, who are authorised or unauthorised stakeholders, and so on. Morevoer,

security requirements are captured at a very high-level of abstraction and then suddenly mapped to security mechanisms, without a clear understanding of how these were derived. Instead, as we illustrated, the modelling and analysis of security features should naturally guide the identification and integration of security solutions into a software system design [Giorgini et al., 2003]. Therefore, it is hard, not to say impossible, to address all challenges C_1–C_5 discussed in Section 1.1 related to the specification of secure socio-technical systems by adopting existing approaches.

A considerable number of mainstream approaches to security requirements engineering explicitly captures security requirements [Firesmith, 2003; Mead et al., 2005; McDermott and Fox, 1999; Sindre and Opdahl, 2005; van Lamsweerde, 2004; Haley et al., 2008]. However, these approaches consider security in system-oriented terms, that is, they are suitable for the design of software systems, since the system is treated as a monolithic entity focusing strictly on technical mechanisms, and the only interactions considered are among end users and the system. This perspective fails to capture the social and organisational aspects of socio-technical systems, the interactions among stakeholders, and their needs with regard to security.

Other security requirements engineering frameworks and methodologies have been proposed that allow modelling organisations and stakeholders, recognising the importance of considering security from a social and organisational perspective [Yu and Liu, 2001; Liu et al., 2003; Giorgini et al., 2003, 2005a; Mouratidis and Giorgini, 2007a]. These approaches use goal-orientation, based on Yu's *i** [Yu, 1995], by modelling a socio-technical system as a set of goal-oriented actors (representing stakeholders) that are *intentional*—they have objectives—and *social*—they interact with others to achieve their desired objectives. In particular, the approaches proposed by Giorgini et al. [2005a] and Mouratidis and Giorgini [2007a] go a step further, distinguishing between actors that own a service or a resource (asset) from actors requiring a service/resource, from actors entitled to do any of these. This is in line with our view, which considers stakeholders interacting with one another to reach their objectives (services) and exchange information (resources), while distinguishing among their proprietary assets and the security concerns they have with respect to protecting these assets.

Thus, goal-oriented approaches to security requirements engineering [Giorgini et al., 2005a; Mouratidis and Giorgini, 2007a; Liu et al., 2003] are a good starting point, for they explicitly acknowledge the importance of social factors. However, existing approaches express security requirements at a very high level of abstraction (e.g., [Liu et al., 2003; Giorgini et al., 2005a]), which makes them difficult to operationalise to technical requirements for the system-to-be. Also, their underlying ontologies are not expressive enough to effectively represent real-world security requirements.

We want to be able to express real-world security requirements, that is, as close as possible to the way stakeholders would express their concerns regarding security (as illustrated above).

The importance of tackling this issue has emerged from collaborations and interaction with industrial practitioners, who require a fine-grained specification of security requirements to develop security into socio-technical systems.

Additionally, the practitioners community has emphasised the need for any proposed approach to the design of secure socio-technical systems to be in line with the security principles followed by the information security community [Gollmann, 2011; Pfleeger and Pfleeger, 2012; Kissel, 2011; ISO/IEC, 2005]. Guided by this pressing need, we have studied the extensive work in security and the various security standards proposed such as ISO 27002 [ISO/IEC, 2005] or COBIT 5 [cob, 2012]. These standards provide an extensive list of concepts or high-level requirements as good security practices to be followed, however, they fail to provide any guidelines or methodological tool to support decision-making [Zannone, 2007].

1.3 Research Roadmap

We have discussed the challenges we have to face in proposing a methodology to effective security requirements engineering for socio-technical systems. What is missing is a systematic process (methodology) to specify security requirements for socio-technical systems that not only acknowledges underlying social and organisational factors, but also allows one to capture security requirements over interactions as a result of capturing participants security needs. Ideally, such a process should narrow the gap in the requirements analysis process and facilitate operationalisation of security requirements and the introduction of security mechanisms that will satisfy the given security requirements. We will consider the main security aspects common to the prominent security standards in eliciting the list of security requirements the methodology should support, while capturing stakeholders security needs. Our aim is to build on top of the existing approaches to goal-oriented security requirements engineering, extending, revising and refining them in order to overcome the highlighted limitations and address their shortcomings in the design of secure socio-technical systems.

1.3.1 Research Questions

Our overarching objective is that of *providing a comprehensive methodology to design secure socio-technical systems starting from early requirements engineering*.

Following the motivations and challenges discussed in Section 1.1 and 1.2, the overall objective of this thesis will be achieved by addressing the following research questions:

RQ_1 *How can we help security requirements engineers in specifying a secure socio-technical system?*

RQ_2 *How to effectively capture security requirements for socio-technical systems?*

RQ_3 *How to perform analysis on security requirements?*

To satisfy the identified research questions, we propose a methodology (systematic approach) acompanied by a modelling language and a formal framework that allow modelling and reasoning about security requirements in order to specify secure socio-technical systems. To facilitate the evaluation of the proposed methodology, in Table 1.1 we identify several success criteria for each research question, as follows:

Table 1.1: Research questions and success criteria

RQ.	**Success Criteria**
RQ_1	$SC1$ Have a systematic approach that is
	$SC1.1$ *Able to conduct modelling and analysis activities in few steps*
	$SC1.2$ *Applicable to different domains, and different socio-technical systems*
	$SC1.3$ *Usable by both researchers and practitioners*
RQ_2	$SC2$ Have a modelling approach that is
	$SC2.1$ *Able to capture stakeholders' security requirements (starting from security needs)*
	$SC2.2$ *In line with the terminology in international security standards*
	$SC2.3$ *Equipped with a formal semantics*
RQ_3	$SC3$ Have an analysis approach that supports
	$SC3.1$ *The definition of a set of properties to be verified over security requirements models*
	$SC3.2$ *Automated analysis for conflict identification (with tool support)*
	$SC3.3$ *Analysis results are provided in acceptable time*

Addressing RQ_1. In order to address RQ_1, *How can we help security requirements engineers in specifying a secure socio-technical system?*, we need to have a systematic approach which supports modelling and analysis activities to specify a secure socio-technical system, and is:

$SC1.1$ *Able to conduct modelling and analysis activities in few steps.* We will provide a comprehensive methodology that supports modelling and analysis activities for the specification of secure socio-technical systems. The methodology will cover the entire security requirements engineering phase, starting from the representation of stakeholders and their

assets (addressing C_1), the elicitation of stakeholders' needs, their representation in models, their analysis, until the specification of the security requirements for the system to-be. The proposed methodology will support the modelling of social threats and analyse their impact over stakeholders' assets (addressing C_4).

As such, the proposed methodology should take into account these questions to guide the modelling activities: "*Who are the stakeholders participating in the given socio-technical system?*", "*What are their valuable assets?*", "*What are the interactions they enter?*", "*What are their security needs over these interactions?*", "*What kind of security needs?*", "*Are there any social threats affecting stakeholders assets?*", etc. Most importantly, the process followed by the methodology will facilitate the work of security requirements engineers by iteratively and incrementally building the requirements models, and performing the modelling and analysis activities in few steps.

$SC1.2$ *Applicable to different domains, and different socio-technical systems*. This success criteria is important to prove the generality of the proposed methodology, which should support the modelling of various socio-technical systems, capturing different security requirements types, while offering good coverage for different domains.

$SC1.3$ *Usable by both researchers and practitioners*. It is important that the methodology finds applicability not only among researchers, but also among practitioners. This adoption is important to understand the limitations of the methodology while being applied to realistic settings, and guide further improvements.

Addressing RQ_2. In order to address RQ_2, *How to effectively capture security requirements for socio-technical systems?*, we need to have a modelling approach which is:

$SC2.1$ *Able to capture stakeholders' security requirements (starting from security needs) along system and stakeholders' requirements*. We will provide a design-time modelling language that will support the modelling and derivation of security requirements for the system-to-be. The language should address the distributed nature of socio-technical systems, while respecting the autonomy of their participants. Therefore, we need to represent stakeholders (participants of a socio-technical system) as intentional entities (they have objectives to achieve and for this they participate in the socio-technical system), as well as social entities (they often depend on one another to achieve their goals). Participants are generally heterogeneous and autonomous, therefore, their behaviour is usually unknown and non-controllable. Thus, the best a designer can do is allow them specify constraints over their interactions (addressing C_2). We refer to these constraints as *security needs* to distinguish from the general security requirements for the system-to-be. The modelling language will reflect such intuition by relating security requirements to interaction, and

capturing security requirements as relationships between socio-technical system participants, where a *requester* actor requires a *responsible* actor to comply with a security need (addressing C_3). The outcome of the language will be a security requirements specification composed of all security requirements derived by participants' security needs.

$SC2.2$ *In line with the terminology in international security standards*. The modelling approach will follow security principles recognised by the information security community, while defining the set of supported security requirements types. This is important for the satisfaction of $SC1.3$ to help adoption by practitioners, who need to comply with international security standards.

$SC2.3$ *Equipped with a formal semantics*. Requirements analysis is concerned with the identification of inconsistencies and possible conflicts among requirements. This is particularly important when dealing with security, since violation of security requirements may have severe consequences, such as privacy law infringement and monetary sanctions. Hence, as part of analysis, we want to know whether the expressed security needs might be in conflict or potentially violated. Therefore, some of the questions we need to consider in devising adequate analysis techniques are: "*Are there any conflicts among the specified security needs?*", "*Can the stakeholder satisfy the required security needs?*", "*If not, why not?*", "*Is the model well-formed?*", "*What is the impact of social threats over stakeholders' assets?*", etc.

Addressing RQ_3. In order to address RQ_3, *How to perform analysis on security requirements?*, we need to have an analysis approach which supports:

$SC3.1$ *The definition of a set of properties to be verified over security requirements models*. We are primarily concerned with the consistency of security requirements specifications, to avoid the existance of conflicting security requirements. As Jureta et al. [Jureta et al., 2010] point out in their core modelling language proposal, Techne, "*As the shift from software and hardware moved towards socio-technical systems, requirements engineering must account for the variously inconsistent expectations of stakeholders*. Our modelling language and its reasoning approach, too, should account for participants' different views. Being specified independently by different participants, the various security needs (and as a consequence security requirements) one should comply with are likely to conflict. The same applies to ones objectives and the security needs one should comply with, thus leading to inconsistent specifications that cannot be satisfied by an implemented socio-technical system (at least one requirement would be violated).

$SC3.2$ *Automated analysis for conflict identification (with tool support)*. The detection and handling of conflicts between requirements is a hard task [Finkelstein et al., 1994] as goal-

models tend to become huge and complex, and it often requires the usage of automated reasoning techniques. Our proposed modelling language can be no exception, especially because it aims to support rich security requirements, which are quite expressive, but complex, and real-world models, which are typically large [Trösterer et al., 2012]. Therefore, automated analysis will be required to handle these large models, identify possible well-formedness issues, verify whether there are potential conflicts among requirements or potential violations of security requirements (addressing C_5). In order to perform automated analysis, the language constructs and relationships, together with the set of security requirements to be verified, will be represented in a formal language.

$SC3.3$ *Analysis results are provided in acceptable time*. The employed techniques will be devised such that they scale well (give an answer in acceptable time) with increasing model sizes.

1.3.2 Evaluation activities

The proposal of the methodology and its constituent components, as requirements engineering artefacts, need to be evaluated to verify whether they satisfy the established success criteria.

From the three research questions, we can identify three main artefacts of this research, a tool-supported methodology for security requirements engineering, a modelling language and a formal framework to support automated analysis.

The following are the evaluation activities considered for the evaluation of each and every expected artefact:

E_1 *Self-evaluation study*: via application scenarios (in Software Engineering terminology widely known as *case study*): the method designer applies the methodology or other artefacts to a real life scenario. The scenario can be constructed on the basis of interviews and discussions with stakeholders and domain experts, as well as on the basis of supplementary material (documents, deliverables, etc.). Such an activity has the objective of demonstrating the applicability of the methodology and other artefacts in the given domain and to the given scenario, while showing that they offer a good coverage by capturing the most important aspect of the scenario.

E_2 *Case study*: the method designer interacts with end-users of the artefact and consults supplementary documents to have a good understanding of the scenario. However, the end-users are the ones using the artefact (methodology, language, or analysis; all tool-supported) and provide feedback. Method designer collects feedback through interviews, discussions, and provided reports.

E_3 *Empirical study*: the method designer conducts empirical evaluation study with end users (or representative end-users) of the artefact (apply methodology, use modelling language

and analysis). Such an evaluation activity is important, for not only it shows the applicability of the artefact to various application domains and scenarios, but it also allows to measure other qualities of the methodology, language and analysis, such as, usefulness, usability, ease of use, and intention to adopt [Moody, 2003].

E_4 *Formal semantics*: the method designer proposes a formal framework to allow defining properties of the artefact (mainly modelling language and its constituent constructs), in order to verify behaviors and characteristics displayed by the artefact (models in our case).

E_5 *Scalability study*: the purpose of this activity is to evaluate the scalability of the automated analysis techniques supported by the modelling language, implemented by the tool, with the help of real scenarios with increasing model size.

Table 1.2: Fulfilment of success criteria via evaluation activities

	Methodology			Language			Analysis		
	SC1.1	SC1.2	SC1.3	SC2.1	SC2.2	SC2.3	SC3.1	SC3.2	SC3.3
E_1 *Self-evaluation study*	✓	✓		✓	✓				
E_2 *Case study*	✓	✓		✓	✓				
E_3 *Empirical study*	✓	✓	✓		✓				
E_4 *Formal semantics*	✓					✓	✓	✓	
E_5 *Scalability study*									✓

In Table 1.2 we present the evaluation activities for the validation and evaluation of each and every artefact produced by this work, namely methodology, language, and analysis. We show for each evaluation activity, what are the success criteria (for each artefact) that the said evaluation activity contributes to fulfil.

1.4 Motivating scenario: Red Cross Blood Transfusion Centre

We use a scenario from the Healthcare domain to illustrate the need for our work. In this section we describe the case study in more detail in order to introduce the main participants, why they participate in the healthcare socio-technical system, to help us illustrate the methodology, the modelling language, and tool features in the next chapters.

The blood transfusion service has been considered by various researchers working in the privacy and security domain such as [Butch, 2002; Zhu et al., 2006; Mohammed et al., 2009],

to offer solutions in protecting healthcare data, while ensuring availability of necessary information. We will consider variants of this scenario throughout the thesis to illustrate the methodology and introduce the modelling language.

Alice is a blood donor that periodically donates blood through the Red Cross Blood Transfusion Centre (BTC). Of course, the Red Cross BTC has numerous other donors. The centre is responsible for collecting and examining the blood collected from the donors, and then to distribute it to different hospitals. The Red Cross BTC is responsible for ensuring all donors are eligible to donate blood. Elaborate test results, however, are performed at specialised laboratories. For instance, Alice has to take infectious disease testing at ModernLabs.

The hospitals have many patients that need appropriate healthcare. Hospitals collect and maintain the health records of their patients, and transfuse the blood to them when needed. All information used and maintained throughout the transfusion procedure, including the type of operation, participated medical practitioners, and reason of transfusion, is clearly documented and stored in the database of each corresponding hospital. The hospital relies on physicians to provide healthcare services to patients. Physicians of the hospital access this information to provide adequate medical health advice and treatments to the patients.

Patient information as well as the blood usage listings have been made accessible to the Red Cross BTC so that this institution can perform certain statistical analysis and auditing tasks. The objectives of the statistical analysis and auditing procedures are to improve the estimated future blood consumption in different hospitals and to make recommendations on the blood usage in future medical cases.

The Hospital Authority sets the regulations to be in place for the privacy of patients' records. The Red Cross BTC submits reports to the Hospital Authority. Referring to the privacy regulations, such reports have the purpose of keeping patients' privacy protected. The data published along with the Privacy Aware Health Information Sharing Service gets refined in a way to meet certain privacy criteria.

The analysis of the case study confirms the challenges discussed in Section 1.1, while identifying new, more specific, challenges and requirements that should be considered and addressed to enable a successful security requirements engineering process:

1. The system is defined by the *interaction* among social and technical participants (stakeholders). The Red Cross BTC system is not monolithic: its operation depends by the succesful interaction among donors, physicians, laboratories, patients, hospitals, hospital authority and the Red Cross BTC center.

2. Consider and analyse *social aspects* in order to identify security issues arising from social threats as opposed to strictly technical ones. Medical personnel of the hospital may disclose details of the health status of a patient to his spouse without the permission of the

former, while the patient may have required this information to be discussed and shared only with him, not with any family members.

3. The participants aim to protect their *assets*, namely *informational and intentional assets* [3]. Donors are concerned with the protection of the confidentiality of their medical history (informational asset). The Red Cross BTC is concerned with ensuring that the express courier successfully handles the blood transportation process towards the various hospitals, i.e., fulfills *blood transported* (intentional asset). To address this issue, we need to support a thorough analysis of participants relevant assets, being these informational or intentional.

4. Every participant/stakeholder has its own expectations on security, *security needs*, which constrain the way they would like others to behave when it comes to the assets they want to protect. For example, donors allow the Red Cross BTC to use their data for approving them as donors and for any statistical analysis needed over the collected blood, but they do not want the involvement of third parties, even of research centres. Celebrities being hospitalised, in particular, want the information related to their conditions to remain confidential and not made public.

5. The autonomy of participants makes the specification of secure socio-technical systems a challenging task. What if a nurse accesses information on celebrities heath status and transmits it to the media? A secure interaction among participants is key to the proper functioning of socio-technical systems. Thus, the interaction among participants (subsystems) of a socio-technical system requires guarantees from a security point of view that user needs (in this case not disclosing donors' confidential health information to third parties and not transmitting celebrities health status to the media, respectively) will be satisfied. To address this challenge, we need to show how stakeholders *security needs* lead *to* a security requirements *specification*, and how this specification can be satisfied.

6. The participants may *grant* or *deny permission* over their informational assets. Patients prohibit (deny permission) third party research centres to access their personal data and health status.

7. *Conflicts concerning security* are possible, due to the autonomy of the participants and their different concerns (needs). For instance, the Red Cross BTC wants to externalise the statistical analysis performed on donors, blood types, and hospital requests—this violates the donors' expectation that such data is not disclosed.

[3] Distinction discussed in Chapter 3 and 4

1.5 Overview and Contributions

Our approach to specifying secure socio-technical systems is based on a comprehensive tool-supported methodology, a security requirements modelling language, which offers modelling and automated analysis capabilities integrated in the supporting toolset.

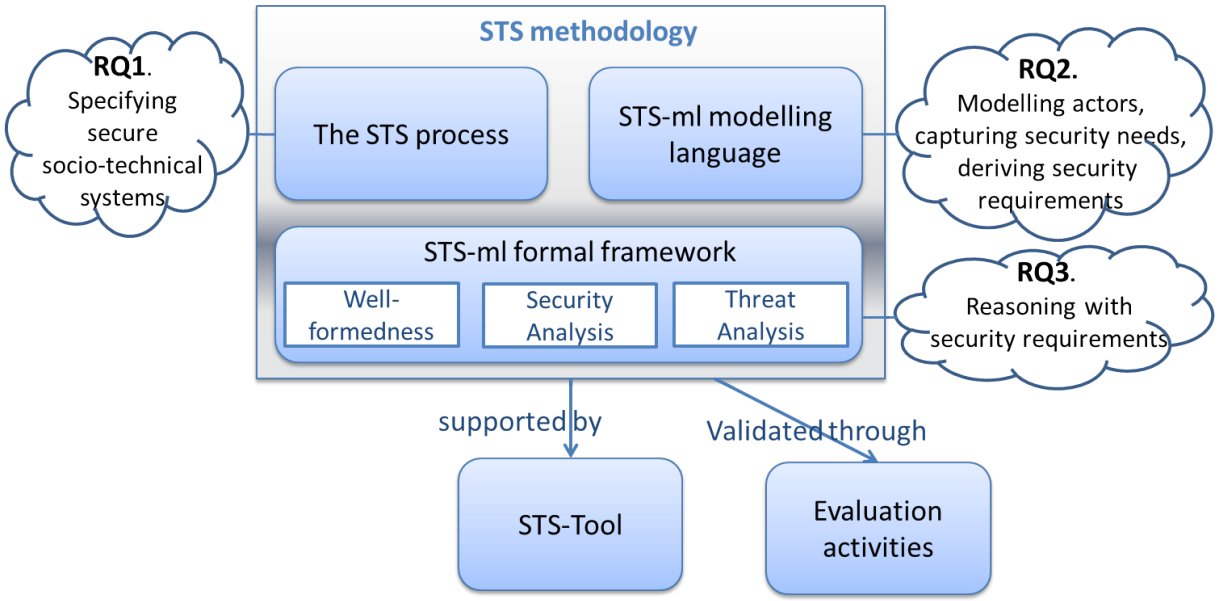


Figure 1.1: Overview of our approach to specifying secure socio-technical systems

Figure 1.1 presents an overview of our proposed approach to specifying secure socio-technical systems. Specifically, the contributions of the thesis are as follows:

1.5.1 The STS methodology

The *STS methodology* for security requirements engineering of socio-technical systems guides the modelling and analysis of secure socio-technical systems through the STS process. The methodology addresses the challenges laid down in Section 1.4 and makes the following contributions beyond the state of the art:

- *Social threats*: security analysis often considers the perspective of possible malicious users attacking the system by exploiting system vulnerabilities [Liu et al., 2003; Jürjens, 2002]. In the same spirit, STS methodology does not overlook threats, and supports the identification of social and organisational threats, which do not necessarily exploit technical vulnerabilities of a software system. However, the methodology assumes that the represented events threatening actors' assets and the identification of the assets they threaten are the result of risk analysis activities (following the *identification phase* of some

risk analysis method [Tixier et al., 2002]), which is out of the scope of this work. STS methodology focuses in offering a way to represent social threats affecting stakeholders' assets and analyse how they threaten the rest of their assets, while leaving to the security requirements engineers the choice among CORAS [Lund et al., 2010], OCTAVE [Albert and Dorofee, 2001], or any other risk analysis methodology that better suits their needs.

- *From needs to specifications*: unlike other goal-oriented approaches, the methodology covers the entire security requirements engineering phase, starting from the elicitation of stakeholders, their important assets, threats (mainly social) affecting stakeholders' assets, stakeholders' security needs, their representation in models, their analysis, until the specification of the security requirements for the system to-be.

- *Security standards*: the methodology is aligned with the principles adopted by the community working on (information) security. Although there is no agreed upon taxonomy of security principles, this proposal relies on six principles acknowledged by mainstream taxonomies [Gollmann, 2011; Pfleeger and Pfleeger, 2012; Kissel, 2011]: confidentiality, integrity, availability, authenticity, reliability, and accountability. STS methodology refines its supported security needs from the principles of confidentiality, integrity, availability, accountability, authenticity, and reliability, supporting in this way a rich set of security requirements. Differently from security standards, such as ISO 27002 [ISO/IEC, 2005], which provides a general model and guidelines for establishing, implementing, operating, monitoring, reviewing, and improving an information security management system, or COBIT 5 [cob, 2012], which provides best practices for Information Technology security governance and management, the STS methodology provide a detailed account of the concrete steps a security requirements engineer should undertake for the specification of a secure system, while making use of the rich concepts of the modelling language and being facilitated by the CASE tool.

1.5.2 The STS-ml modelling language

The modelling language includes a rich set of security requirements that address the major aspects of information security. STS-ml builds upon other goal-oriented modelling languages such as SI* and Secure Tropos [Giorgini et al., 2005a; Mouratidis and Giorgini, 2007a; Liu et al., 2003], but the proposed language revises the high-level organisational concepts, maintaining a minimal set of concepts including role, agent, goal, delegation, etc., while distinguishing information flow from permission and prohibition flow, including authorisation modelling, and supporting a significantly richer set of security requirement types. More specifically, the STS-ml modelling language makes the following contributions beyond the state of the art:

- *Social/organisational perspective*: inspired by goal-oriented languages such as SI* [Giorgini et al., 2005a] and Tropos [Bresciani et al., 2004], the methodology models socio-technical systems as sets of goal-oriented actors that play different organisational roles and that interact in order to fulfil their objectives. However, the STS-ml is a more expressive modelling language, and supports a richer set of security requirements.

- *Security on interactions*: security requirements in the STS-ml modelling language arise from the expectations regarding security over the interactions among the actors (e.g., the citizens expect the hospital not to disclose their medical records to third parties).

- STS-ml distinguishes between *information* (e.g. ideas) and its representation via *documents* (e.g. email), which enables exchange and manipulation of information. This distinction is in line with the fact that information can be made available in various forms, and each and every form needs to be protected in order to protect the information it represents [ISO/IEC, 2005]. This conceptual difference provides STS-ml with greater expressiveness, compared to SI* and Secure Tropos, when specifying security requirements over information.

- *Separation of concerns*: considering the effort in building goal models, the STS-ml modelling language wants to facilitate the work of security requirements engineers offering them the possibility to create different models that focus on different perspectives of the system under development, such as stakeholders' interactions, their assets, etc., which all together form the model of the system-to-be. This is a distinguishing feature of the modelling language, as, differently from the approaches it builds upon, it separates social and organisational concerns (objectives, interactions) from information ownership, structure and flow, from permission and prohibition flow. Specifically, the STS-ml modelling language supports *multi-view modelling*, separating the social interactions among stakeholders captured in the *social view*, from their information and representation captured in the *information view*, from the permissions and prohibitions flow captured in the *authorisation view*. This crisp separation of concerns highlights the need to represent and analyse separately these different aspects in socio-technical system. Moreover, it stands at the basis of having a richer and more expressive modelling language.

- STS-ml acknowledges the importance of representing *information owners* as in SI* [Giorgini et al., 2005a, 2006]. However, following the distinction between information and documents, ownership is modelled over information, to represent the rightful participant(s) owning an information entity. These are the ones concerned with what happens to their proprietary information and may express their concerns (needs) with respect to security when information is manipulated or exchanged via documents.

- STS-ml explicitly distinguishes *information flow* from *permission* and *prohibition flow*. Similarly to SI* [Giorgini et al., 2005a, 2006], information owners are the legitimate stakeholders to grant or deny rights to others over their information. However, in SI* these permissions and prohibitions are modelled similarly to information flow itself. STS-ml offers the possibility to model and visualise authorisations stakeholders grant or deny to others. Authorisations can be seen as a visualisation of access control policies specified by the stakeholders themselves over their proprietary information. They define operations one can or cannot perform over *what information*, for *what purpose*, and whether one can *further authorise* others or not. As such, they offer greater expressiveness and can support a richer set of security requirements over information.

- *Formal semantics*: security requirements models tend to be large, and cannot be effectively analysed manually (demonstrated by our experience in [Trösterer et al., 2012]). The formal framework lays down formal foundations (semantics) that support the execution of automated analysis techniques to verify well-formedness of the models, to identify conflicts among security requirements and among stakeholders' objectives and security requirements, and to determine the impact of threats over stakeholders' assets. Tool support is provided to automate analysis techniques and visualize results.

1.5.3 Automated analysis techniques

The *automated analysis techniques* are devised for detecting conflicting requirements at design time. We propose a formal framework to perform automated analyses over the created models, which allows to:

- Perform *well-formedness* checks: given that goal models tend to become huge and complex, well-formedness analysis allows the security requirements engineer to build syntactically correct models, following the syntax of the modelling language.

- Identify security requirements conflicts and possible violations of security requirements. This is known as *security analysis* and is intended to identify conflicts arising due to (i) simultaneous security requirements a given participant needs to satisfy, but are conflicting with one another, and (ii) the participant's own objectives and the security requirements imposed to him by others being in conflict. These conflict identification techniques are implemented in Disjunctive Datalog.

- Calculate of the impact of social threats over stakeholders' assets. The representation of social threats alone would not be enough to know the impact they have over stakeholders' assets. Therefore, we calculate the trace of a given threat, starting from the asset it threatens to the rest of stakeholders' assets.

1.5.4 The STS-Tool

The *STS-Tool* supports modelling socio-technical systems, while capturing security needs, and automated analysis over security requirements models. STS-Tool is an Eclipse RCP standalone application, which has the following main features:

- *Multi-view modelling*: differently from existing tools to security requirements engineering, such as SI* [4] or ST-Tool [5], which support modelling stakeholders and their interactions to identify security requirements violations, the STS-Tool supports separating the social interactions among stakeholders captured in the *social view*, from their owned information and representation captured in the *information view*, from the permissions and prohibitions flow captured in the *authorisation view*.

- *Automatic derivation of security requirements* once the modelling is done and security needs are expressed.

- *Automatic generation of a security requirements document*, which provides a description of the various views, the list of security requirements, and automated analysis results among other detail. This is an important output that the security requirements engineer can use to communicate with stakeholders and eventually resolve the identified security issues whenever possible.

1.5.5 Evaluation of the methodology, language, and tool

The methodology, language and tool are evaluated through a number of activities, as discussed in Section 1.3.2. We have taken advantage of an application scenario (E_1, Section 8.1) and two industrial case studies spanning different domains, such as Air Traffic Control Management and e-Government (E_2, Section 8.3).

These activities aim at assessing both the effectiveness of the methodology, language, and tool in modelling security requirements in various domains, while identifying possible security issues (supported by E_4, see Chapter 6), as well as the usefulness and scalability of the automated reasoning techniques in providing answers in acceptable time even for models with growing size (E_5, see Section 8.2).

Most importantly, we have conducted empirical studies with domain experts, practitioners, and Masters and PhD students to further evaluate the effectiveness of the proposed methodology (E_3, Chapter 9).

[4] `http://sistar.disi.unitn.it/index.php/SI*_Tool`
[5] `http://sesa.dit.unitn.it/sttool/install.php`

1.6 Organisation of the thesis

The rest of the thesis is organised as follows.

- Chapter 2 presents the state of the art work for this thesis. We review methodologies and languages for requirements (Section 2.1) and security requirements engineering (Section 2.2), as well as techniques and methods about reasoning with requirements and security requirements (Section 2.3). We consider approaches beyond security requirements engineering in goal-oriented terms (Section 2.4), reviewing approaches from business process modelling, privacy modelling, policy specification languages, and international security standards.

- Chapter 3 introduces the STS security requirements engineering methodology for socio-technical systems. It introduces the process followed by the STS methodology, and provides a high-level description of the activities supported by the methodology to guide modelling and automated analysis over the created models till the specification of security requirements.

- Chapter 4 presents the primitives of the STS-ml modelling language and the list of supported security requirements types, which are illustrated with the help of the motivating scenario.

- Chapter 5 presents the models that can be created when using the proposed modelling language, STS-ml, what language primitives are used and what security requirements can be captured in each, while highlighting the importance and focus of each and every model.

- Chapter 6 introduces the formal framework (Section 6.1) that defines the semantics of the primitives of the modelling language, and details the automated analysis techniques (Section 6.2) that can be performed over the created models. In particular, it shows how *security analysis* is used to identify conflicts among security requirements and among participants' objectives and security requirements with which they have to comply.

- Chapter 7 presents the CASE tool, namely STS-Tool, which supports the activities of the process followed the STS methodology, while implementing the modelling primitives of the security requirements modelling language, and integrating the formal framework. STS-Tool not only facilitates modelling and automated analysis activities, but it also supports the automatic derivation of security requirements starting from participants' security needs, and the generation of a security requirements document.

- Chapter 8 introduces an application scenario and two case studies that are used to show how security requirements engineers can use the proposed methodology, illustrating step

by step the methodology in action to build and analyse the STS-ml models. We report on scalability results of the analysis techniques applied on the case studies in Section 8.2. In particular, Section 8.3.3 reports on the application and adoption of the methodology, language and tool by practitioners.

- Chapter 9 evaluates the STS methodology through empirical studies. We present the results of several empirical user studies conducted through the course of this research work, while reflecting the impact each has had in the evolution of the methodology, the STS-ml modelling language, and the various releases of STS-Tool.

- Chapter 10 provides an extensive discussion on how the evaluation activities have contributed to the fulfillment of success criteria for each research question, to then conclude. Follows a discussion of ongoing and future work beyond the contributions presented in this thesis, and potential new lines of research.

1.7 Published papers

Published work related to this thesis is listed here. They are divided in two categories: (i) refereed, with subcategories for books, journal, conference, and workshops; and (ii) un-refereed, presenting book chapters.

1.7.1 Refereed

International Journals

1. Per Håkon Meland, Elda Paja, Erlend Andreas Gjære, Stéphane Paul, Fabiano Dalpiaz, and Paolo Giorgini (2014), *Threat Analysis in Goal-Oriented Security Requirements Modelling*, International Journal of Secure Software Engineering, (*To appear*).

2. Elda Paja, Amit K. Chopra, and Paolo Giorgini (2013), *Trust-based Specification of Sociotechnical Systems*, Data and Knowledge Engineering (DKE) Special Issue ER 2011, Elsevier Science Publishers, Volume 87, September 2013, pages 339—353.

3. Sandra Trösterer, Elke Beck, Fabiano Dalpiaz, Elda Paja, Paolo Giorgini, and Manfred Tscheligi (2012), *Formative User-Centered Evaluation of Security Modeling: Results from a Case Study*, International Journal of Secure Software Engineering 3 (1) pages 1—19.

International Conferences

4. Elda Paja, Fabiano Dalpiaz and Paolo Giorgini (2013), *Managing Security Requirements Conflicts in Socio-Technical Systems*, In Proceedings of the 32nd International Conference on Conceptual Modeling, ER 2013, pages 270—283, Hong Kong.

5. Amit K. Chopra, Elda Paja, and Paolo Giorgini, *Socio-Technical Trust (2011): An Architectural Approach*, In Proceedings of the 30th International Conference on Conceptual Modeling, ER 2011, pages 104—117, Brussels, Belgium.

International Workshops and Demos

6. Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti, and Paolo Giorgini, *Specifying and Reasoning over Socio-Technical Security Requirements with STS-Tool*, In Proceedings of the 32nd International Conference in Conceptual Modelling - Workshops, pages 504—507, (ER'13 Workshops), Hong Kong.

7. Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini, *Designing Secure Socio-Technical Systems with STS-ml*, 6th International i* Workshop (iStar'13), Valencia, Spain.

8. Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti, and Paolo Giorgini, *STS-Tool: Specifying and Reasoning over Socio-Technical Security Requirements*, 6th International i* Workshop, pages 79—84, (iStar'13), Valencia, Spain.

9. Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti, and Paolo Giorgini, *STS-Tool: Socio-Technical Security Requirements through Social Commitments*, In Proceedings of the 20th International IEEE Conference on Requirements Engineering, pages 331—332, (RE'12 Demo and Posters), Chicago, IL, USA.

10. Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti, and Paolo Giorgini, *STS-Tool: Using Commitments to Specify Socio-Technical Security Requirements*, In Proceedings of the 31st International Conference on Conceptual Modeling Workshops, pages 396—399. (ER'12 Demostrations), Florence, Italy.

11. Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti, and Paolo Giorgini, *Modelling Security Requirements in Socio-Technical Systems with STS-Tool*, In Forum of the 24th International Conference on Advanced Information Systems Engineering, pages 155—162, (CAiSE'12 Forum), Gdańsk, Poland.

12. Elda Paja, Paolo Giorgini, Stephane Paul, and Per Hakon Meland, *Security Requirements Engineering for Business Processes*, In Proceedings of the 1st International Workshop on Alignment of Business Process and Security Modelling, pages 79—84, (ABPSM'11), Riga, Latvia.

13. Fabiano Dalpiaz, Elda Paja, and Paolo Giorgini, *Security Requirements Engineering via Commitments*, 1st Workshop on Socio-Technical Aspects in Security and Trust, pages 1—8, (STAST'2011), Milan, Italy, 2011.

14. Fabiano Dalpiaz, Elda Paja, and Paolo Giorgini, *Security Requirements Engineering for Service-Oriented Applications*, 5th International i* Workshop (iStar'11), Trento, Italy, 2011.

15. Elda Paja, Fabiano Dalpiaz, Paolo Giorgini, Stéphane Paul, and Per Håkon Meland, *Modelling Trust and Security Requirements: the Air Traffic Management Experience*, In Proceedings of iStar Showcase 2011, London, UK.

1.7.2 Non-refereed

16. Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini, *The Socio-Technical Security Requirements Modelling Language for Secure Composite Services*, In Secure and Trustworthy Service Composition: The Aniketos Approach, Springer, 2014, (To appear).

17. Elda Paja, Mauro Poggianella, Fabiano Dalpiaz, Pierluigi Roberti, and Paolo Giorgini, *Security Requirements Engineering with STS-Tool*, In Secure and Trustworthy Service Composition: The Aniketos Approach, Springer, 2014, (To appear).

18. Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini, *STS-Tool: Security Requirements Engineering for Socio-Technical Systems*, In Advances in Engineering Secure Future Internet Services and Systems, volume 8431 of LNCS, Springer, 2014, (To appear).

19. Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini, *Identifying Conflicts in Security Requirements with STS-ml*, Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, (December 2012).

1.7.3 Under preparation

Books

20. Fabiano Dalpiaz, Elda Paja, and Paolo Giorgini (2014), *Security Requirements Engineering. Designing Secure Socio-Technical Systems*, Accepted for publication, MIT Press.

International journals

21. Elda Paja, Fabiano Dalpiaz, Paolo Giorgini, Alexander Borgida, *STS: A Security Requirements Methodology for Socio-Technical Systems*, *Sumbitted to* ACM Transactions on Software Engineering and Methodology.

Chapter 2

State of the art

The state of the art in the area of security requirements engineering is quite broad. However, as discussed in Chapter 1, our focus lies mainly in model-driven solutions for the design of secure socio-technical systems. Therefore, we present here approaches for requirements engineering in goal-oriented terms (see Section 2.1), to then discuss approaches for security requirements engineering in Section 2.2, where we provide a comprehensive survey of security requirements methodologies (see Section 2.2.1) and of security requirements modelling languages (see Section 2.2.2). Apart from modelling aspects, our methodology aims to provide analysis support, and thus, we consider works about reasoning with requirements in Section 2.3, investigating in particular approaches for conflict identification (see Section 2.3.1) and for reasoning with security requirements (see Section 2.3.2).

To provide a comprehensive overview of the state of the art in specifying security requirements, we consider approaches beyond security requirements engineering in goal-oriented terms in Section 2.4, namely business process modelling (see Section 2.4.1), privacy modelling (see Section 2.4.2), and policy specification languages to support enforcement (see Section 2.4.3). Finally, in Section 2.5 we discuss international security standards to determine the terminology to be adopted for the security requirements types STS supports.

2.1 Goal-oriented requirements engineering

Requirement Engineering (RE) is concerned with the elicitation of *what* a system should do. Goal-Oriented Requirements Engineering (GORE) has emerged as a distinguishable approach in Requirements Engineering that captures the rationale behind requirements for the system-to-be, answering the *why* and *how* questions repeatedly. Goal-oriented approaches, as the name indicates, build around the notion of *goal*, which refers to desired objectives the organization or the stakeholders of the system under development want the system to accomplish. Goals are used to elicit, elaborate, structure, specify, analyze, negotiate, document and modify require-

ments [van Lamsweerde, 2001]. The importance of considering goals in early requirements is acknowledged by numerous works [Mylopoulos et al., 1992; Dardenne et al., 1993; Mylopoulos et al., 1999; van Lamsweerde, 2001]. Goals gained prominence because of several advantages they offer, such as providing a precise criteria for completeness of requirements specification (with respect to a set of goals), provide the rationale for requirements (useful to explain requirements to stakeholders), providing support in choosing among alternatives as well as identifying requirements conflicts and resolving them eventually [van Lamsweerde, 2001].

Many GORE approaches have been proposed, see [Lapouchnian, 2005], and [van Lamsweerde, 2001] for an overview. We focus here on approaches that acknowledge the importance of *social* factors in designing socio-technical systems in order to establish the ones best suited for our needs.

The *i** [Yu, 1995] modeling framework was developed by *Eric Yu* to model and reason about organizational environments and their information systems. It addresses the need to model and analyze the reasons behind stakeholders requirements and interests during early phase requirement engineering. The basic concept in *i** is that of an *intentional actor*, taking into consideration the fact that Actors in an organizational environment have to achieve goals, are equipped with certain abilities, have beliefs, etc. All of these are intentional properties that characterize an Actor. In *i** actors are further specialized into agents, roles, and positions [Yu, 1997] to represent the roles they play in the organization or the positions they might occupy. In such an environment, it is obvious that actors interact with one another to accomplish their goals, perform tasks, or access system resources. They could also depend on each other for the same reasons. Dependency relations help actors obtain results that would be difficult, if not impossible for them to achieve alone, without delegating to other actors. Anyhow, this exposes them to become vulnerable awaiting for the depended-on actor to deliver. The *i** modeling language is at the base of the framework. The basic constructs offered by this language are: *actor*—together with its associations, *intentional elements*—goals, tasks, and resources, *strategic dependencies*, *decomposition links*, *means-end links*, *contribution links* among intentional elements. The use of *i** for requirements engineering considers two different levels of modeling: (i) *Strategic Dependency* (*SD*) *Model*, and (ii) *Strategic Rationale* (*SR*) *Model*. The *SD Model* describes an organizational environment in terms of actors (and their objectives) and their social interdependencies, while not not exhibiting any details of actors' internal rationales. This model allows representing intentionality behind processes in the organization, while highlighting what is important for the actors themselves. The *SR Model*, on the other hand, goes in depth analyzing the internal rationale of actors, in terms of intentional elements—goals, tasks, resources, softgoals. By making explicit the knowledge about on actors' rationallity, SR models are useful to reason about actors' behavior.

Tropos [Bresciani et al., 2004] is an agent-oriented software development methodology cov-

ering the full range of software development phases. The Tropos methodology was designed to support analysis and design activities throughout the development process. Special focus is dedicated to early requirements analysis, taking advantage of *i**. The concepts of intentional actor, goal, dependency, decomposition, contribution, etc., are defined and modeled here too, although applying some constraints to the underlying conceptual model. Modeling agents, their beliefs and desires, tasks and methods, captures important elements of an organization and the way its members interact and collaborate with one another. Tropos addresses primarily the need to develop robust and reliable software that operates in evolving and dynamic organizational environments, as such it describes both the system and the organization. The main advantage of this is that, by doing an earlier analysis, one can capture not only the *what* or the *how*, but also the *why* a piece of software is developed. Ignoring the *why* perspective is indeed one of the major limitations of languages adopted by practitioners such as UML. This, in turn, supports a more refined analysis of the system dependencies and, in particular, for a much better and uniform treatment, not only of the system's functional requirements, but also of the non-functional requirements. Yet, this methodology lacks the ability to capture at the same time functional and security issues [Giorgini et al., 2003], which are very important at this level. The only way to model security issues in terms of non-functional requirements through the use of softgoals. Although softgoals can support security related analysis during requirements analysis, they lack detail to support analysis in the later development stages, for it is hard to distinguish among security and other requirements of the system [Mouratidis and Giorgini, 2007a]. Thus, Tropos cannot capture security requirements properly. Consequently, Tropos has to be enriched with additional constructs to consider them.

2.2 Security requirements engineering

2.2.1 Security requirements engineering methodologies

The requirements engineering community has acknowledged the importance and necessity of considering security since the early stages of the software systems development [Devanbu and Stubblebine, 2000; Dubois and Mouratidis, 2010].

We contrast our approach with other methodologies for security requirements engineering, highlighting the ones that detect violations of security requirements.

We summarise the main methodologies in Table 2.1, a thorough review can be found in [Mellado et al., 2010], while a comparison of security requirements engineering methods can be found in [Fabian et al., 2010]. The columns of the table—most of which derived from the challenges listed in Section 1.4—indicate the methodology name, the taken perspective (organisational or system-centred), the support of refining security needs into specifications, the formality of the approach, the support to informational and intentional assets, the inclusion

Table 2.1: Comparison with other security requirements methodologies

Methodology	**Perspect.**	**Needs-to-specs**	**Formal**	**Informat. assets**	**Intent. assets**	**Threats**	**Conflict identif.**
Anti goals	Sys		•		•	•	•
Haley et al.	Sys	•	•	•		•	•
Liu et al.	Org	•	•	•	•	•	
Mellado et al.	Sys	•		•		•	
SecReq	Sys	•	•	•		•	
Secure Tropos	Org	•		•	•	•	
Security trade-offs	Org	•	•	•	•	•	
SQUARE	Sys	•		•	•	•	
STRIDE	Sys			•	•	•	
UMLSec	Sys	•	•	•		•	•

of threats, and the identification of conflicts. Our proposed methodology takes an organisational perspective and supports all the listed features, as shown in Section 1.5.

Anti-goals [van Lamsweerde, 2004] abstract abuse/misuse cases to the intentional level. The proposed method promotes security engineering at the application layer. Security requirements are specified by incrementally building two models: a model of the system-to-be and an anti-model. The former model specifies a set of security goals, making use of specification patterns to elicit candidate security requirements. The anti-model captures how the security goals in the first model could be endangered, deriving the vulnerabilities and capabilities needed to achieve the anti-goals of the security goals. Anti-goals are refined in threat trees, whose leaf nodes represent either vulnerabilities observable by the attacker or anti-requirements implementable by the attacker. The model of the system-to-be is then enriched with new security requirements which represent the countermeasures to apply to the anti-model. In contrast to this approach, our focus is at the organisational level and on protecting the social interactions among actors. In our view, the security specification patterns could be expressed as security needs one actor could impose while interacting with others.

Breaux and Antón [2008] present a methodology to systematically extract security (legal) requirements from regulation texts. They introduce a method to acquire and present data requirements, accompanied with a method for assigning and managing priorities between them,

to ensure law compliance and avoid inappropriate information disclosure. The acquisition consists of extracting a hierarchy of stakeholders based on the specialisation relationships used to define them in the regulations, and then for each stakeholder group, the rules that apply in a given situation are identified. Stakeholders' rights—actions they are permitted to perform, and obligations—actions they are required to perform, are extracted from regulations, together with a set of applicable constraints. Finally a series of patterns are applied to determine data access rules. Prioritization takes advantage of exception rules, and a set of patterns for identifying priorities between data requirements while preserving low compliance (thereby preventing improper information disclosure). Though relying on contractual rules to ensure compliance, they focus only on data usage restrictions.

Problem frames [Jackson, 2001] were extended to support security requirements [Haley et al., 2008]. After constructing the system context with problem frames, security requirements are defined as constraints over functional requirements, and a structure of satisfaction arguments is built to verify the correctness of security requirements. The argument fails if the security requirements are not satisfiable in the context, or the context is insufficient to develop the argument. This approach focuses mainly on system requirements, without considering the social perspective, while ours is centred on user-specific security requirements emerging from the interaction among actors.

An extension [Liu et al., 2003] of *i** [Yu, 1995] enables dealing with security and privacy requirements. The methodology defines security and privacy-specific analysis mechanisms to identify potential attackers, derive threats and vulnerabilities, thereby suggesting countermeasures, and specifying access control. The latter aims at bridging the gap between security requirement models and security implementation models. The security specific analysis steps are integrated into the requirements engineering process, in order to anticipate potential attackers and seek countermeasures for system protection when needed. This approach is intended to provide mechanisms that explicitly relate social concerns with technologies and policies addressing these concerns, while putting together the organisational perspective with the attacker perspective. Our approach represents social threats exploiting actors' assets and uses social relationships to propagate the threat impact over the entire model; moreover, we support a significantly larger set of security requirements, while differentiating information flow from permissions/prohibitions represented through sophisticated authorisations.

Mellado et al. [2007] propose a standard-based security requirements engineering process named SREP to deal with security requirements at an early stage. SREP follows the same steps (activities) proposed by SQUARE, but the process is standardised by integrating knowledge from the Common Criteria (ISO/IEC 15408) and the Systems Security Engineering Capability Maturity Model (ISO/IEC 21827) into the software lifecycle model, particularly in eliciting security requirements. SREP reuses security requirements based on a security resources repos-

itory containing threats, assets, countermeasures, and requirements specifications. Unlike ours, this approach is not based on requirements models. A future direction could include integrating our work in a standard-based methodology like Mellado et al.'s; to do so, their methodology needs to be extended to consider social aspects, in addition to system-related ones.

The SecReq methodology [Houmb et al., 2010] guides the security requirements engineering process to support security-non-experts. SeqRep supports iteratively: (i) detecting security issues at an early stage using heuristic requirements (through HeRA), (ii) refining them based on the Common Criteria (ISO 14508), and (iii) tracing security requirements back to design (using UMLsec analysis). The first two techniques are combined to support non-experts in writing better requirements, starting from functional requirements and using heuristics to search for keywords or patterns that indicate security-related issues. The search is performed over the Common Criteria security requirements knowledge. The methodology supports security requirements analysis and tracing capabilities through HeRA and UMLsec. Their approach, similarly to ours, supports refining security needs into specifications, however our methodology is targeted to security engineers. As for security-non-experts, as the evaluation activities showed, there is a need for extensive training to use the STS methodology properly. SecReq provides guidelines for secure systems development based on security expertise derived from security standards, our methodology considers core security properties in information security to protect actors informational and intentional assets, and importantly it considers security issues emerging from interaction (social, organisational perspective).

Secure Tropos, based on SI*, is an agent-oriented security requirements engineering methodology, was developed to model functional and security requirements of socio-technical systems [Giorgini et al., 2005a]. The methodology allows for the design of secure information systems starting since the initial development stages (early requirements). The SI* [Giorgini et al., 2005a, 2006] modeling language is adopted for the acquisition of security requiremens, relying on organisational concepts and explicitly acknowledging that the system involves the interaction among a number of actors socially depending on one another. It builds on *i** and adds security-related concepts to capture security at the early requirements stage, among which delegation and trust of execution and permission. Our approach supports a larger set of security requirements, binds all requirements to interactions, and clearly separates between security requirements and business policies.

Secure Tropos by Mouratidis and Giorgini [2007a] builds on the Tropos methodology [Bresciani et al., 2004] and models security concerns from early requirements to design. The framework expresses security requirements as *security constraints*, considers potential threats and attacks, and provides methodological steps to validate these requirements and overcome vulnerabilities. Their analysis identifies secure goals and entities to guarantee the satisfaction of security constraints, and gives secure capabilities to agents to guarantee the satisfaction of secu-

rity entities. Secure Tropos, however, suffers from these limitations: (i) security constraints are expressed over goal dependencies, similarly to our approach, however they express constraints related to information that goals might need or create; and (ii) in Secure Tropos no relations between goals and resources are shown, only goal-task means-ends relationships, while secure resources are only considered after the requirements engineering phase, during architectural design, as needed to satisfy security constraints. In STS, we clearly separate between security constraints (needs) over goal delegations, which impose constraints on the delegation itself of the goal being delegated, and the security constraints over information, which are expressed over information exchange and through authorisations, representing permission/prohibition flow. This allows us to be more expressive and support more types of security requirements. We represent how actors manipulate information when fulfilling their goals. This is important to identify whether information is being manipulated in compliance with the security constraints imposed by the information owner. By defining a formal semantics, we can identify violations of security requirements, and as a result of security needs, apart from identifying conflicts among security requirements.

Elahi's work [Elahi and Yu, 2007] extends the *i** framework by supporting security trade-off analysis. The authors propose a conceptual modelling technique to reach a good enough security level in a multi-actor setting. This technique offers the possibility to assess the impact of assessing security mechanisms on actors' goals and threats. Vulnerabilities refer to the deficiencies in the structure of goals and activities of intentional agents. However, differently from STS-ml, they do not take into account vulnerabilities related to actors interaction.

SQUARE [Mead et al., 2005] is a 9-steps methodology/process to elicit, categorise and prioritise security requirements for information technology systems and applications. The steps include identifying safety and security goal to elicit safety and security requirements, being so goal-based at the early stage. SQUARE distinguishes requirements from other types of constraints, it includes risk assessment, prioritisation, and requirements inspection. The final output provided by SQUARE is a security requirements document that is designed to satisfy the security goals of the organisation. SQUARE is a generic methodology that defines the key steps to follow. Our approach, conversely, is less broad but provides concrete models, tooling, and reasoning techniques.

The STRIDE [Hernan et al., 2006] model uses threat modelling for the design of secure systems by methodologically breaking down the system into components, analysing each component for susceptibility to threats, and mitigating threats, so to discover design problems potentially leading/allowing security breaches and correct these design-level security problems. The considered security properties are a subset of those supported by our methodology, while the threats are purely technical (we consider social threats too). However, the idea of potential security issues and threats arising when components are put together is in-line with our idea of

security issues arising over interactions.

UMLsec [Jürjens, 2002] extends UML to develop security-critical systems. The methodology helps specify security requirements focusing mostly on authenticity, secrecy, and integrity. Security issues are analysed by representing the behaviour of potential attackers (adversaries), and modelling specific types of attackers (stereotypes). Basic security requirements such as integrity are provided/supported via stereotypes and tags, standard extension mechanism in UML. The UMLsec profile has been extended to UMLseCh profile [Jürjens et al., 2011] for specifying one or more evolutions on a model and verifying that the system remains secure (compliant to the security requirements expressed in UMLsec) despite evolution. Stereotypes are used to define what specific elements change in a model (are added, deleted, or substituted), while constraints in first-order logic coordinate and define the different evolution paths. Unlike UMLsec, our approach takes an organisational stance on security requirements engineering, and supports threats too. The support to evolution of UMLseCh could be adapted for STS-ml in future work.

2.2.2 Security requirements modelling languages

We summarise the principal security requirements modelling languages in Table 2.2. We use the same dimensions for the comparison as we did for the methodology, for the modelling language will support activities of the methodology and because the considered modelling languages support the specification of security requirements, even though not explicitly providing a systematic approach (method) to conduct this process.

Abuse cases [McDermott and Fox, 1999] extend use cases to capture and analyse security requirements. An abuse case specifies a type of interaction between a system and one or more actors, where the results of the interactions are negative/harmful. It includes a range of security concerns that might be abused, as well as a description of the harm that might be caused.

In a similar spirit, misuse cases [Sindre and Opdahl, 2005] exploit use cases to represent sequences of actions that a system or other entities can perform, interacting with *misusers* of the entity and causing harm if the sequence is allowed to complete. These approaches exploit negative scenarios to elicit and analyse security requirements. We focus on how actors should interact, and define a set of security requirements for protecting their interaction. As such, the approaches are complementary.

SecureUML [Lodderstedt et al., 2002] is a modelling language extending UML designed for integrating the specification of role-based access control into application models, while providing additional support for specifying authorisation constraints (expressed in OCL). The formal semantics in terms of a model transformation to UML/OCL is provided in [Brucker et al., 2006]. The idea of the transformation is that of substituting the security model build in SecureUML with a model for an explicit enforcement mechanism specified in UML/OCL. This approach, unlike ours, takes a system-centric stance, and supports only authorisation requirements.

Table 2.2: Comparison with other security requirements modelling languages

Language	**Perspect.**	**Needs-to-specs**	**Formal**	**Informat. assets**	**Intent. assets**	**Threats**	**Conflict identif.**
Abuse/misuse cases	Sys	•				•	
SecureUML	Sys		•	•			
Secure use cases	Sys	•		•		•	
Security patterns	Sys			•		•	
SI*	Org		•	•	•		•

Security use cases [Firesmith, 2003] are used to analyse and specify requirements that the application shall successfully protect itself from relevant security threats. They are driven by misuse cases [Sindre and Opdahl, 2005], and are based on an analysis of the assets and services to be protected, while considering the security threats from which the said assets and services should be protected. Firesmith provides guidelines to better use security use cases during requirements engineering. However, it is not clear how the analysis of assets and services to be protected is performed. It is hard to distinguish the assets, which appear to be included in the security threat itself.

Security patterns help in solving recurring security problems during the design and implementation of systems [Schumacher et al., 2005]. Security patterns they are helpful in establishing the problem of the pattern, and delegate the identification of a solution to other activities. A variant of their approach has been added to goal-oriented languages [Asnar et al., 2011]; a future direction for our approach is to integrate security patterns to solve recurrent security problems.

The SI* modelling language makes explicit who is the requester of a service (objectives), who is the legitimate owner (entitlements) and who is able to provide a service (provisioning, aka capabilities). It builds on *i** and adds security-related concepts to capture security at the early requirements stage, among which delegation and trust of execution and permission, while considering also distrust of permission and execution. The latter allow capturing negative authorisations, such as delegation denial and prohibitions. However, SI* mixes together ownership, information flow and permission/prohibition flow in the same model, which limits the ability to express explicit security requirements, especially when it comes to information security. Those can be derived from the prohibitions specified at the social level over the services actors are

not entitled to execute. Our approach clearly separates between security requirements and business policies (aka, how actors achieve their desired objectives either on their own or by relying on others via delegations), binds all requirements to interactions, and supports a larger set of security requirements.

2.3 Reasoning with requirements

2.3.1 Conflict identification

The literature in reasoning with requirements is quite broad, a thorough review and classification can be found in [Horkoff and Yu, 2011]. We limit ourselves to providing an overview of the literature on the identification of conflicting requirements to compare our reasoning techniques. The importance of identifying conflicting requirements is well-known by practitioners and has been widely acknowledged by the research community [van Lamsweerde et al., 1998; Fuxman et al., 2001]. Several formal frameworks exist, especially in goal-oriented requirements engineering.

SAT solving was applied to analyse the satisfaction or denial of goals in goal models [Giorgini et al., 2002]. Giorgini et al. define a range of satisfaction and denial evidence, from full to partial, resulting in four distinct predicates: full evidence of satisfaction (FS), partial evidence of satisfaction (PS), full evidence of denial (FD), and partial evidence of denial (PD). By introducing the positive and negative contribution relationships among goals, they can identify situations of contradictory contributions among goals. The new relationships are given both a qualitative semantics and a quantitave semantics, which based on a probabilistic model. As such, the approach includes both qualitative and quantitative analysis techniques that determine evidence of goal satisfaction/denial by using label propagation algorithms, which are proven to be sound and complete. Conflicts are identified when both positive and negative evidence exists.

In [Giorgini et al., 2005b] Giorgini et al, incorporate goal model analysis procedures introduced in [Giorgini et al., 2002] into the Tropos Framework, with the objective of making the goal analysis process concrete while coping with qualitative relationships and incosistencies among goals, in order to suggest, explore and evaluate alternative solutions. Goal analysis inlcudes forward reasoning, which is used to evaluate the impact of the adoption of different alternatives (leaf goals being fulfilled) with respect to the softgoals of the system to be. Additionally, backward reasoning is used to find the acceptable alternative (set of leaf goals) at the lowest costs, such that if achieved can guarantee the achievement of the desired root goals and softgoals. As far as the identification of conflicts is concerned, goal analysis identifies conflicts not only based on contradictory contributions to the same goal as in [Giorgini et al., 2002], but also possible due to the existence of diamonds or loops.

This approach inspired further research, such as the iterative interactive identification of conflicts [Horkoff and Yu, 2009], [Horkoff and Yu, 2010] by Horkoff and Yu for early requirements engineering. In [Horkoff and Yu, 2009] the authors propose a qualitative interactive procedure that allows users to systematically evaluate and compare the alternative actions and solutions expressed in models asking "What if?" questions. The modeler can supplement the evaluation with domain knowledge, to guide model creation and domain exploration, which includes the evaluation of alternatives. In this way the procedure can benefit from human intervention to compensate for the incomplete nature of the models, while reasoning in early stages of analysis, before concrete quantitative information is known. The procedure propagates forwards by first deciding on an alternative, second propagating satisfaction/denial labels and placing results over softgoals (often receive multiple incoming labels), and finally resolving conflicting labels over softgoals either through automatic cases or human judgment when cases do not apply. In this way, the procedure does not itself decide on an alternative, but picks a case or human judgment which leads to selecting an alternative, while evaluating the effects of alternative choices in the model.

The work described in [Horkoff and Yu, 2010] on the other hand, proposes an interactive backward procedure from ends to means to answer questions "Is this possible?", "If so, how?" and "If not, why not?". They encode i* models (used as an example modelling language) and target values into conjunctive normal form (CNF), and use a SAT solver to reason on this representation. The SAT solver is called iteratively on the CNF representation. After each iteration, their approach takes user input to take a decision in case of conflicts or multiple sources of partial evidence, by using domain knowledge, to then re-encode CNF formula removing the axioms for backward or forward propagation, and adding new axioms representing human judgment. When the SAT solver finds the answer, it returns success, and human judgment is not needed. When the SAT solver cannot provide an answer, they display UNSAT and backtrack over the last round of human judgment to find an answer. Should no more human judgment be available to backtrack over, then the procedure returns failure, for no answer was ultimately found. We also intend to adopt human judgment to resolve conflicts, but differently from Horkoff and Yu's work, this cannot be based on selecting an alternative, but requires negotiation and an analysis of trade-offs.

Fuxman et al. [Fuxman et al., 2001] translate *i** models to Formal Tropos, which supplements i* concepts with first-order linear-time temporal logic. The use of formal methods, in particular model checking techniques, intends to allow a formal and mechanized analysis of early requirements specifications to help the requirements analyst identify errors and limitations of the specification, which would be otherwise impossible in an information setting. Formal assertions are used to represent a set of required and desired constraints over the system. The framework uses an intermediate language to link Formal Tropos and model checking. Early

requirements are translated into this intermediate language and given in input to a symbolic model checker to verify contradictions in the requirements specification (consistency checking), as well as to validate formal properties (for actors, goals or dependencies). When identifying property violations, their analysis returns s concrete conflicts' scenario (counterexample) that describes the scenario violating the property, to help the user understand the problem. Similarly, our techniques visualize and describe the identified conflicts and violations to offer users a better understanding.

KAOS [van Lamsweerde et al., 1998] includes analysis techniques to identify and resolve inconsistencies that arise from the elicitation of requirements from multiple stakeholders with different viewpoints. Emphasis is put on formal analysis that identifies goal conflicts. The authors review various types of inconsistency classifying them based on the description of the requirements, identifying: *process-level deviations*, which refer to a state transition in the requirements engineering process resulting in a inconsistency between a process-level rule and a process state, *instance-level deviations*, which results in an inconsistency between a product-level requirement and a specific state of the running system, *terminology clashes*, in which the requirements specification contains different syntactic names for the same concept, *designation clashes*, when a single syntactic name in the requirements specification designates different real-world concepts, and *structure clashes*, when a single real-world concept is given different structures in the requirements specification. *Conflict* is defined among several specifications of goals/requirements enhanced with domain knowledge, which seems to play an important role in the identification and anticipation of conflicts (logical inconsistent assertions). Importantly, resolution techniques are discussed, such as finding alternative goal refinements, weakening goals, and using of divergence (a boundary conditions that makes assertions logically inconsistent) resolution heuristics. The proposed classification and resolution techniques are shown to be useful in managing inconsistencies in a real-world project, however most of the classifications and resolution strategies are specific to KAOS concepts, such as boundary conditions.

Jureta et al. [Jureta et al., 2010] propose Techne, an abstract formal requirements language, which provides ontological foundations (core concepts and relationships) and basic analysis, to serve as a basis for creating new early requirements engineering modelling languages. The language provides a minimal set (core) of components such as goasl, softgoals, quality constraints, domain assumptions, and tasks, which are necessary to define the requirements problem, define candidate solutions, model preferences and optional requirements, and use them as criteria to compare candidate solutions. Candidate solutions are consistent sets of requirements satisfying some properties. The language supports classification and relation of the information elicited from stakeholders to then model and formulate requirements for the system to be. Among relations we can spot *conflict*, which is used to model a conflict between requirements that cannot be part of the same candidate solution. Techne uses r-net visual syntax to represent require-

ments and the relations among them. Analysis in Techne aims to find candidate solutions to the requirements problem, which must be conflict-free.

Our framework takes an interaction-oriented stance to conflict identification, by checking business policies against security requirements on social relationships, as opposed to reasoning on a single goal model. An interesting research line is to integrate the discussed frameworks to detect inconsistencies among individual business policies.

2.3.2 Reasoning with security requirements

De Landtsheer and Van Lamsweerde [2005] model confidentiality claims in terms of specification patterns, representing properties that unauthorised agents should not know. Their reasoning identifies violations of confidentiality claims in terms of counterexample scenarios present in requirements models. Diagnosis algorithms are used to generate the unauthorised agents reasoning to infer knowledge that is claimed to be confidential. While their approach represents confidentiality claims in terms of high-level goals, ours represents authorisation requirements as social relationships, and we identify violations by looking at the business policies of the actors.

SI* [Giorgini et al., 2005a, 2006] proposes automated reasoning to check security properties of a model, and studies the interplay between execution and permission of trust and delegation relationships. Inconsistencies are identified as a result of considering two different levels, social and individual, based on the role-based access control model. They specify entitlements, objectives and responsibilities to roles, and then assign agents to roles. The social level represents for instance the policies that rule the organisation, while the individual level represents the concrete instance of the organisation. Therefore, inconsistencies are identified in case the concrete instance violates the specified security requirements. While we use the same reasoning engine, our formalisation of security requirements and business policies makes our reasoning task different. We take a deeper perspective on information security, representing the legitimate owners and capturing the information and permission/prohibition flow with respect to the said information. Apart from supporting a wider set of security requirements, this allows us to identify unauthorised access and usage of information and unauthorised delegation of rights among others.

2.4 Beyond security requirements engineering

2.4.1 Business processes modelling

Business process modelling is often used to capture security policies. There exists a number of approches that verify the compliance between security policies and business processes executed in a system.

Security aspects might be directly captured through business process modelling, or existing business process models might be enriched with security annotations after getting in input security requirements specifications derived from security requirements modelling languages, among others.

For instance, Wolter et al. [Wolter et al., 2008], describe an approach to integrate security goals and constraints in business process modelling together with a model-driven transformation that focuses on authorisation requirements. In a similar way, Rodriguez et al. [Rodríguez et al., 2007] introduce an extension to the *Business Process Modelling Notation* (BPMN) to allow business analysts express security needs from their perspective. In our view, security policies modelled at the business process level should be a consequence of the modelling performed at a higher level of abstraction, which provides a rationale on how the business analyst should decide upon security requirements in the business process and what security requirements are specified over the business processes of the system-to-be.

In [Menzel et al., 2009], Menzel et al. employ a model-driven approach to generate security policies based on security patterns. They provide an enhancement to BPMN to enable the assessment of risks based on the evaluation of assets and the trustworthiness of participants, and to enable the annotation of security requirements such as confidentiality or integrity.

Pavlovski and Zou [Pavlovski and Zou, 2008] extend BPMN to capture non-functional requirements related to business process models, among which security policies that apply. Their extension involves two notations: *operating condition*, which refer to constraints over activities, and *control case*, which describes the risks associated to the operating condition together with mechanisms to mitigate or reduce business risks.

Cardoso et al. [Cardoso et al., 2011] start from goal modelling to elicit business process models. Goals are considered as objectives to be achieved by the execution of a business process. The authors show how the elicitation process takes place starting from a preliminary phase to a supplementary one, which refines the goal models by using NFR (Non-Functional Requirements) catalogues. However, how goal models are related to business process models is left as future work.

For high-level business process modelling in UML, the approaches by Sindre and Opdahl [Sindre and Opdahl, 2005] related to misuse cases and UMLSec by Jürjens [Jürjens, 2002], are well-known. In Sindre [2007], Sindre proposes another technique, which complements misuse cases, to capture security issues throughout business process diagrams. The author extends UML activity diagrams by adding malicious activities and malicious actors to identify possible threats, and then adds defensive processes to mitigate the identified risks, suggesting where in the process the mitigation activities would be placed.

2.4.2 Privacy modelling

P3P [Cranor et al.] is a W3C specification for matching users' privacy concerns with providers' privacy policies. The specification offers a way for providers' an easy way (multiple choice selection) to communicate their privacy policies in machine-readable format so that they can be interpreted by web browsers. Several software agents, such as the privacy minder, AT&T/Microsoft browser helper object, AT&T usability testing prototype and AT&T privacy bird, have been developed to compare policies with user preferences, inform users and take actions based on users' preferences.

Beckers [Beckers, 2012] proposes a conceptual framework to compare various privacy requirements engineering approaches. The framework extends an existing security requirements engineering approach from [Fabian et al., 2010] and is applied to compare three privacy requirements engineering approaches, namely PriS [Kalloniatis et al., 2008], LINDDUN [Deng et al., 2011], and the Framework for Privacy-Friendly System Design [Spiekermann and Cranor, 2009]. The conceptual framework is extended with personal information of stakeholders, as well as relationships among stakeholders and their personal information and privacy goals. The framework supports these privacy goals: anonymity, unlinkability, unobservability, and pseudonymity. Importantly, they distinguish personal information from assets to be protected from a security point of view. In STS, information, and in particular personal information is considered an informational asset for the stakeholders , an asset that needs to be protected (if specified). Privacy requirements are implicitly captured in STS via authorisations that specify permissions and/or prohibitions over how information can be used or manipulated, the scope of usage (for particular goals) and the authority transferred to interacting stakeholders. This is inline with approaches that deal with privacy as part of confidentiality, wherein stakeholders' information is protected from unauthorised access [Danezis and Gürses, 2010]. The framework proposed by Beckers considers the transferral of personal information in aggregated form. In STS, we support the representation of information structure to exactly keep track of aggregated or composite information. This allows us to specify security requirements either directly over specific parts of information or on the composite information.

Costante et al. [Constante et al., 2013] propose an approach to assist both service users and service providers in composing and selecting the optimal services with respect to their privacy policies. The proposed approach supports the modelling of privacy preferences along several privacy dimensions, namely sensitivity (of data subjects), purpose, retention period, and visibility. Moreover, they propose algorithms for service composition that selects services that satisfy both users' functional requirements and their privacy requirements, to finally rank the resulting composite services with respect to the offered privacy level. STS can be seen in service oriented terms, for the interactions among stakeholders are similar to those among a service consumer (user) and a service provider. The selection algorithm supports our ideas about re-

solving potential conflicts among stakeholders' business policies (representing their functional requirements) and their security requirements (which in STS include privacy requirements too, captured through permission and prohibition modelling).

2.4.3 Policy specification languages

Policy specification languages play an important role in supporting enforcement of security policies as well as auditing of decisions.

SecPAL [Becker et al., 2010] is a declarative authorisation language to support legal compliance with evolving legislation based on formal logic. The authorisation decision is based on query evaluations using Datalog with constraints. SecPal supports fine-grained delegation control for decentralized systems, highly expressive constraints and negative conditions, offering in this way high expressiveness to specify formal authorisation policies.

IBM's Enterprise Privacy Authorisation Language (EPAL) [Ashley et al., 2003] is a W3C standard privacy specification language that supports enterprises to specify directly enforceable policies for data-handling. A policy in EPAL is typically a list of privacy rules. A rule is a statement that includes a ruling (allow, deny), a user category (e.g., sales department), an action (e.g., store), a data category (e.g., customer record), and a purpose (e.g., order processing). Conditions and obligations may be applicable too. Decision taking is based on evaluating rules and policies to verify their applicability, authorisation requests are assessed by a Policy Enforcement Point (PEP) and taken by a Policy Decision Point (PDP).

A similar standard specification language is the eXtensible Access Control Markup Language (XACML) [OASIS, 2013]. XACML is an OASIS standard that defines a declarative access control policy language implemented in XML and a processing model describing how to evaluate authorisation requests based on the rules defined in policies. Decision making is similar to EPAL: all requests for access to a resource go through a PEP; the request it translated to a request for an authorisation decision which is sent to a PDP; this latter component considers the various policies (a policy is a set of rules), evaluates the information and provides in return an authorisation decision to the PEP. The request contains the information necessary to take authorisation decisions. A XACML Engine is typically used to filter requests and apply the rules. XACML supports several rule combining algorithms to combine several rules and policy combining algorithms to deal with multiple policies. These are very useful in taking a decision whenever the different policies get conflicting evaluations. STS supports only the identification of security issues and conflicts among security requirements. Mapping the security requirements specification to a policy specification language will provide the possibility to have enforcement of the required security needs. In particular the use of policy combining algorithms will aid the process of conflict resolution at the STS level.

2.5 Security standards

In dealing with security, we want to be in line with the principles adopted by the community working on security; in particular, the one concerned with information security. We rely on this body of work to identify the core security aspects we need to consider in the design of secure socio-technical systems.

ISO-27002 [ISO/IEC, 2005] defines information security as the "*preservation of confidentiality, integrity and availability of information; in addition, other properties, such as authenticity, accountability, non-repudiation, and reliability can also be involved*". This standard provides a model for establishing, implementing, operating, monitoring, reviewing, maintaining, and improving an information security management system. Information security is a main pillar in ISO 27002, acknowledging the importance of information as an asset that needs to be suitably protected, like other important business assets. ISO 27002 states that an appropriate protection of information requires its protection in any form (information can be printed or written on paper, transmitted via email, shown on films, or spoken in conversation) or means by which it is shared or kept (paper, email, film, or voice). The standard provides a set of (11) security control objectives—considered a good starting point, as they apply to most organizations or environments—to develop specific guidance to implement information security systems. These include *risk assessment and treatment*, *security policy*, *organizing information security*, *asset management*, *communications and operations management*, *access control*, and *information security incident management* among others. The standard describes each considered control objective, and provides high-level guidelines for their implementation. However, it does not include more fine-grained details on how to implement these control objectives.

Common Criteria (CC) [cc-, 2012] is a computer security standard (ISO/IEC 15408), which provides a guide for the development, evaluation and/or procurement of IT products with security functionality. The CC addresses protection of assets from *unauthorised disclosure*, *modification*, or *loss of use*. The security categories relating to these three types of failure are commonly the CIA triad security protections, i.e., confidentiality, integrity, and availability, respectively. Common Criteria allows: (i) consumers to specify their security requirement (provides a set of common requirements for security functionality and assurance measures), (ii) developers to implement and make claims about the security attributes of their products, and (iii) evaluators to establish if the said products meet the claims, checking against evaluation assurance levels. CC is usually supplemented by other standards, such as ISO 27002, for it does not guarantee the security of products, instead it is meant to be used as the basis for evaluation of security properties of IT products. CC ensures that the process of specification, implementation, and evaluation of a product has been conducted in a rigorous and standard way.

NIST-IR 7298 [Kissel, 2013] provides a glossary of information security terms. It defines

Adequate Security as: "*Security commensurate with the risk and the magnitude of harm resulting from the loss, misuse, or unauthorized access to or modification of information. This includes assuring that information systems operate effectively and provide appropriate confidentiality, integrity, and availability, through the use of cost-effective management, personnel, operational, and technical controls.*" Although not explicitly part of its information security definition, NIST-IR considers *authentication* and *accountability* two important security goals for information security. Given its nature as a glossary, NIST-IR is not meant to provide guidance to implementing the defined security requirements.

COBIT [cob, 2012] from ISACA, defines information security as: "*Ensures that within the enterprise, information is protected against disclosure to unauthorised users (confidentiality), improper modification (integrity) and non-access when required (availability).*" However, COBIT is not meant to serve as a standard tackling information security alone, rather it aims to provide best practices for enterprises for information technology governance and management. It provides a comparison with other available standards, such as ISO27002 [ISO/IEC, 2005] and NIST-IR [Kissel, 2013], while providing directions to executive agencies on how to apply and adapt its good practices to specific contexts.

The CIA triad is a well-known information security model, acknowledging the importance of three security properties a system should satisfy, mainly Confidentiality, Integrity and Availability. However, for a complete picture other aspects or properties should be considered [Stallings and Brown, 2008]. This has given rise to alternative information security models, such as the Parkerian hexad (Confidentiality, Possession or Control, Integrity, Authenticity, Availability and Utility).

Although there is no agreed upon taxonomy of security requirements (principles), the community working in information security agrees on six core security principles a system should fulfil Gollmann [2011]; Pfleeger and Pfleeger [2012]; Kissel [2013], that we consider in our proposal. All of these proposals provide their own definitions of the identified security principles. In the following, we list the security properties we consider important in a socio-technical system, providing a definition we consider more apt for our purposes.

- *Confidentiality* covers information confidentiality and privacy [Stallings and Brown, 2008]. It assures that private or confidential information is not made available or is not disclosed to unauthorized users. Moreover, it assures that users control (or influence) what information related to them may be collected (used), and to whom that information may be disclosed [Stallings and Brown, 2008]. For example, a citizen's taxable income is typically confidential.

- *Integrity* assures that information is not changed (modified) or destroyed in an unauthorized way [Stallings and Brown, 2008]. For instance, one would want to preserve the

integrity of information related to its bank account details, that this are not falsified (modified) by any employee of the bank.

- *Availability* assures that the system works promptly, service is not denied to authorised users, and timely and reliable access to and use of information Stallings and Brown [2008]. For instance, in air traffic management control, the controllers need at all times full availability of information on incoming and outgoing traffic to manage landing and take-off of aircrafts.

- *Authenticity* is the property of being genuine and being able to be verified and trusted [Kissel, 2013]. Authenticity is ensured through authentication processes that verify whether users are who they say they are (entity authenticity [Stallings and Brown, 2008]). We use authentication mechanisms everyday when we access web information such as our email or internet banking.

- *Reliability* is concerned with the consequences of accidental errors [Gollmann, 2011]. In the era of the Internet, however, the notion of accident encompasses non-designed usages, including attackers trying to misuse the system. Mechanisms of redundancy are often employed to ensure reliability of a system. For instance, service providers often maintain communication with third party service providers to ensure service consumers have always access to their service, should their site be down (e.g., due to a DDoS attack).

- *Accountability* refers to the requirements for actions of an entity to be traced uniquely to that entity [Kissel, 2013]. The most well-known property related to accountability is that of non-repudiation, which prevents two communicating (interacting) parties from denying the communication (interaction) has taken place.

Starting from these categories of high-level security requirements, we will refine them to specific security requirements applicable to interactions among participants of a socio-technical systems. As security requirements in the STS methodology capture stakeholders' needs with respect to security, the security requirements are also such that their naming and representation is close to what stakeholders might express and require over interactions, with respect to the achievement of their goals and information.

2.6 Chapter summary

Our approach to designing secure socio-technical systems is model-driven: models are used to represent stakeholders, their objectives, security needs, and used at design time to detect conflicts among stakeholders' objectives (their business policies) and the security requirements they have to comply with. Specifically, we employ a goal-oriented approach, taking advantage

of social models in order to capture social and organisational aspects of the system to be, in line with the challenges presented in Chapter 1. Therefore, STS methodology builds on top of Tropos [Bresciani et al., 2004] and its security-oriented extension [Giorgini et al., 2005a]. This choice is related to the effectiveness of these goal-oriented approaches in the design of secure socio-technical systems [Dalpiaz et al., 2008]. They offer the adequate level of abstraction in capturing security requirements for the system-to-be, and modelling the structure of socio-technical systems. Their main limitation—that we cope with in our approach—is that their underlying ontology is not expressive enough to effectively represent real-world security requirements.

Most importantly, STS intends to be in line with the terminology used by experts in security, and for this we considered various international security standards to come up with a taxonomy of security properties our methodology should consider. This will be further explored in Chapters 3 and 4.

Chapter 3

The STS methodology for security requirements engineering

This chapter presents the STS methodology for security requirements engineering. The need for a new methodology has been extensively motivated in Chapter 1, and thus, we focus here on introducing the STS methodology. We describe its type of process, the main stages it considers, as well as the roles conducting those stages in Section 3.1. Section 3.2 describes the main phases of the process supported by the STS methodology [1], describing in detail each of them throughout the Sections 3.2.1–3.2.5.

3.1 Security requirements engineering with STS

The STS methodology encompasses the whole security requirements engineering phase which is crucial for later phases, such as design, in developing secure socio-technical systems. Unlike existing methodologies, which result in the security specification for a software system, the STS methodology produces the security specification for the overall socio-technical system, which apart from software systems includes also human and organisational stakeholders. A high-level overview of the process followed by the STS methodology is presented in Figure 3.1.

The process. In line with prominent methods in requirements engineering [van Lamsweerde, 2001; Yu, 1995; Dardenne et al., 1993], our method starts with understanding stakeholders' objectives and their needs about security (Elicitation, see Figure 3.1). Elicitation is typically performed with the help of stakeholders, through discussions and interviews to identify their needs about participating in the socio-technical system, and in STS, the focus is particularly

[1] We use method and methodology interchangeably, in line with the common use in Software Engineering, but acknowledge the distinction.

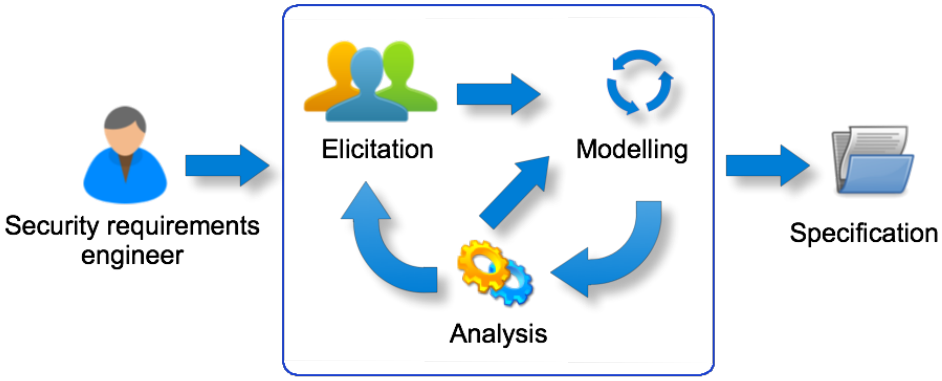


Figure 3.1: The STS methodology: an overview

on their needs about security. Security needs might also be acquired and elicited through the investigation of organisational regulatory rules, laws or any other documents providing related information. However, the details on how the elicitation stage itself is conducted are out of the scope of the STS methodology. We consider the requirements gathering and elicitation activities to have taken place, and we analyse the outcomes, which serve as input for the modelling stage.

The methodology guides requirements analysts and security engineers in modelling (Modelling, see Figure 3.1) and analysing (Analysis, see Figure 3.1) a socio-technical system (focusing on security relevant components and aspects, such as participating stakeholders, their interactions and important assets), and terminates with the definition of a security requirements specification (Specification, see Figure 3.1). These are the principal stages of the process supported by the STS methodology to be followed by the security requirements engineer to build the specification for secure socio-technical systems.

As it can be understood from Figure 3.1, STS is model-driven, and therefore, modelling is one of the core phases considered by the STS process to capture and specify security requirements. As already explained and argued in Chapter 1, STS methodology follows a goal-oriented approach. Hence, it helps address the "*why*" and "*what*" questions, that is, why security is important to stakeholders and for the socio-technical system, what security needs stakeholders' have, and ultimately helps understand the "*how*" question, that is, how the security needs can be satisfied. However, adequately answering the "*how*" question requires mapping the specification derived from STS to lower level languages such as business process modelling and policy specification languages (XACML [OASIS, 2013], SecPAL [Becker et al., 2010]).

Business process modelling helps specify security policies to be met by the business process running in the socio-technical system. Most importantly, it allows to capture temporal aspects of the activities conducted by stakeholders and their interactions with others. On the other hand, policy specification languages, together with their implementations, provide policy decision

points to dwell upon the satisfaction of the specification derived from STS. These, however, are steps to be conducted after the security requirements engineering phase, and as such are outside the scope of STS. Nevertheless, we will consider later on these developments, for they help in establishing and enforcing compliance with the security specification obtained from STS.

The STS methodology is iterative and incremental, for (security) requirements are in constant evolution [Ernst et al., 2009] and they need continuous reevaluation throughout the development process. Therefore, modelling can be refined for as long as the security requirements engineer considers that there are still stakeholders to consider, interactions to model, security needs to capture, and so on. The iterative nature of this phase is graphically shown with the circle arrows above Modelling in Figure 3.1.

Modelling is succeeded by the analysis phase. Analysis is automated through tool support, to facilitate and make feasible analysing the goal-model resulting from the modelling phase. The execution of automated analysis requires some modelling to have taken place, notice the arrow from Modelling to Analysis in Figure 3.1. However, automated analysis does not require a complete model to be executed. The requirements analyst and the security engineer can use the results of the automated reasoning to improve the STS-ml model step by step, see the arrow directed from Analysis to Modelling in Figure 3.1. Analysis may discover security issues or inconsistencies among requirements deriving from stakeholders' different views and needs. As a result, analysis may influence elicitation, requiring further investigation and discussion with stakeholders (see the arrow from Analysis toward Elicitation, Figure 3.1).

The process continues till the security requirements engineer considers to have covered all [2] important security issues when modelling the socio-technical system at-hand. Termination criteria is established by answering these questions:

- *Did I capture all important stakeholders?*
- *Did I capture all important interactions?*
- *Did I model all assets?*
- *Did I express all security needs?*

The process ends with the derivation of security requirements specification for the socio-technical system under consideration.

STS methodology can be included in the context of broader methodologies, such as those for system engineering and for software engineering. Due to the evolving nature of requirements, STS methodology is optimally used within agile methodologies that support quick response to changes. Its iterative and incremental nature makes it a good fit for agile development. In

[2] Note that "all" refers here to the overall information gathered from stakeholders.

particular, the separation of concerns principle followed for the modelling activities (see details below), allows the participation of several requirements analysts and security engineers to model various components or parts of a socio-technical system. However, given the complexity of socio-technical systems and their considerable size, resulting models tend to be of a considerable size too. This opens up issues and challenges for the integration of the models, which the supporting toolset accompanying the methodology should satisfy.

Roles. The *requirements analyst* is concerned with the identification of stakeholders, their assets and interactions, their information (asset), etc. In a nutshell, they are in charge of understanding stakeholders' needs to participate in the socio-technical systems (i.e., what are their objectives) and how they intend to accomplish their objectives. Thus, the requirements analyst is in charge of gathering stakeholders' security needs together with all the necessary information to identify their assets, why they participate in the socio-technical system, who they interact with, and so on. In STS, we assume the elicitation phase of the process has already taken place, and therefore, we take the output of this phase, which is the acquired information the requirements analyst needs for the creation of models.

The *requirements analyst* conducts modelling activities analysing the information acquired during elicitation. During the modelling phase, the requirements analyst is responsible for the representation of stakeholders, their assets and interaction with the help of a modelling notation (language). But, the perspective of the requirements analyst is not enough for an adequate security requirement engineering process. Therefore, STS considers the perspective of a *security engineer* too.

The *security engineer* is responsible for capturing stakeholders' security needs in the socio-technical system. Specifically in STS, security engineers investigate on possible security issues arising over stakeholders' social interactions, as well as on eventual threats putting at risk stakeholders' assets (and as a consequence stakeholders themselves). The first type of security needs can be discovered through interviews and discussions with stakeholders, through related documentation, as well as through the revision of the various models constructed by the requirements analyst over different iterations. The identification of threats affecting stockholders' assets, on the other hand, requires the involvement of yet another role for a clear separation of concerns, that of a *risk analyst*. However, in STS, we assume that the risk analysis activity (this activity is out of the scope of STS) has already been conducted, and support only the modelling (representation) of events threatening stakeholders' assets. That is why, this role is covered by the security engineer too.

The security engineer is concerned with the modelling of security needs, as well as with the Analysis activities, which are necessary to verify the satisfaction of the specified needs. Given that modelling and analysis activities are closely linked together, they require the collaboration

of requirements analysts and security engineers involved in the process.

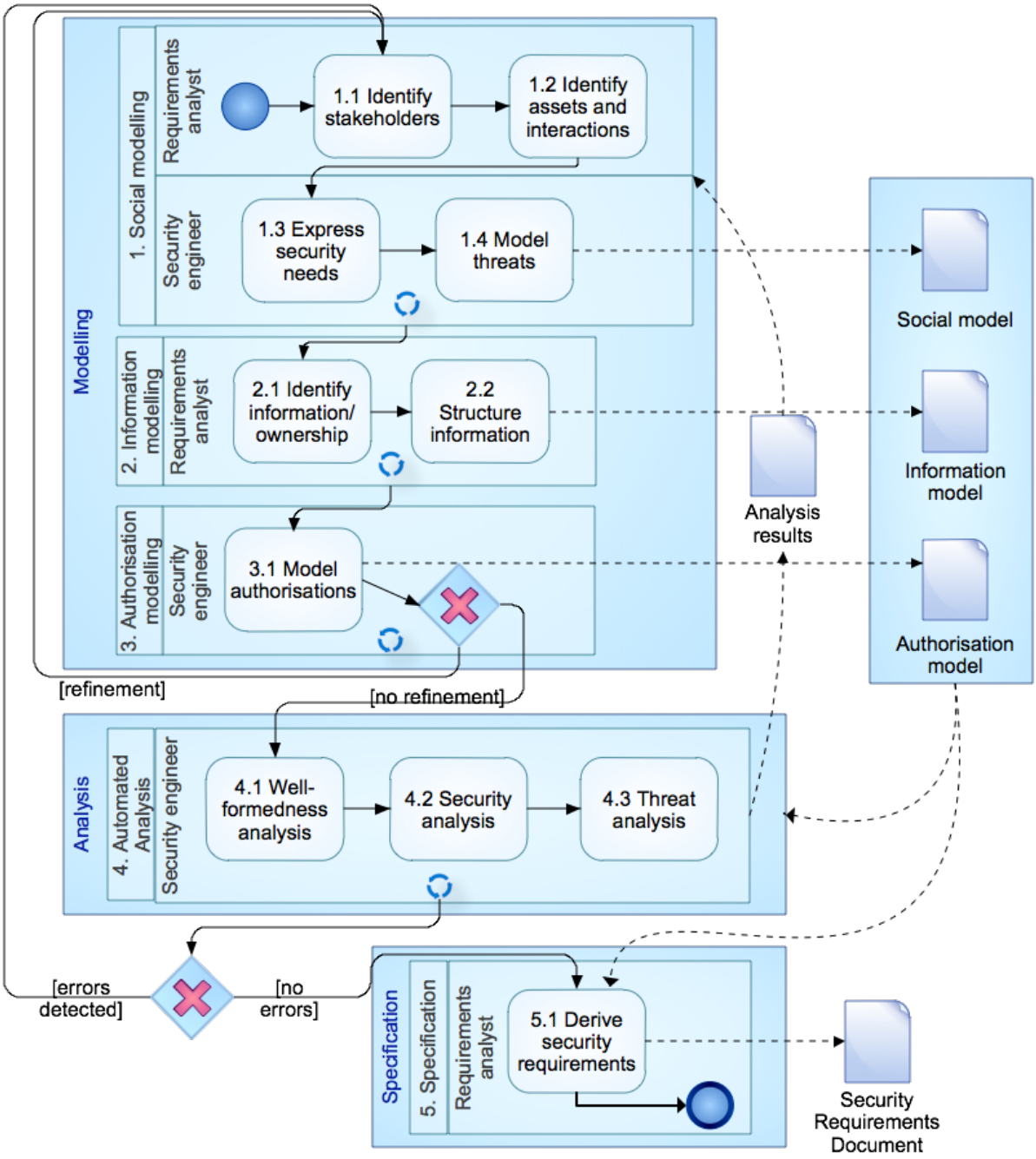


Figure 3.2: The STS methodology: the process

Note that the competences related to the roles *requirements analyst* and *security engineer* might be encapsulated in one single role, that of a *security requirements engineer* (as in Fig-

ure 3.1), who has expertise both in requirements elicitation and analysis, as well as in security. However, for the sake of clarity, in the following we keep these roles separate when describing in detail the steps of the process followed by the STS methodology, see Figure 3.2.

Multi-view modelling approach. One of the particularities of the STS methodology is that it neatly distinguishes between the social and organisational aspects, stakeholders' assets (with a special focus on *information*), and their expectations with regard to security on sharing and protecting their assets. This feature helps the requirements analysts and security engineers separate concerns, while allowing for higher expressivity in capturing security issues. The multi-view approach considers the creation of the models for the socio-technical system under consideration can be done by focusing on one perspective (view) at a time. This means that the underlying system model is one only, but STS offers different complementary views of the same model. This is done not only to support separation of concerns, but also to promote modularity and help in addressing scalability issues in constructing ever growing models for complex systems.

Due to this differentiation, the modelling activities (Modelling, Figure 3.1) in STS are specialised and tailored to support the creation of three different (sub)models when building the model for the system-to-be, which correspond to three different modelling phases, namely *social modelling*, *information modelling*, and *authorisation modelling*. Social modelling is conducted by both a requirements analyst and a security engineer, information modelling requires only the expertise of a requirements analyst, while authorisation modelling requires the expertise of a security engineer.

These modelling phases result in the creation of three complementary models: the *social model*, the *information model*, and the *authorisation model*. The overall model for the system-to-be is, in fact, composed of all these three (sub)models, which are the direct output of the modelling in three different views as supported by the process of the STS methodology. Being views over the same model, these output models should be consistent among them. This, however, needs to be addressed by the supporting toolset.

The overall detailed process followed by the methodology is illustrated in Figure 3.2 using a BPMN-like notation [Object Management Group, 2011]. The figure shows only the main backwards iterations to previous activities and phases. The BPMN diagram outlines the five phases supported by STS methodology: *social modelling*, *information modelling*, *authorisation modelling*, *automated analysis*, and *specification*, the activities within these phases, the roles that execute the activities: *security engineer* and *requirements analyst*, and the artefacts: *social model*, *information model*, *authorisation model*, *analysis results*, and *security requirements document*.

Modelling activities are supported by STS-Tool, which ensures inter-view (model) consis-

tency as we will show in Chapter 7. The changes in one model have effects on other models. For instance, the different represented stakeholders may be first captured in one model, such as the social model, but they are maintained throughout the three models, unless the requirements engineer decides to hide it in another model, such as the information model, should that stakeholder not have any informational assets.

The modelling process is iterative, and as a result Phases 1–3 in Figure 3.2 are iterative too, see the circle arrows in the corresponding swimlanes. Each modelling phase produces an output model. The output models—*social model*, *information model*, and *authorisation model*—can be further refined, depending on the level of detail that is needed. This is, however, performed by iterating over the corresponding modelling phases.

The process supported by the STS methodology (Figure 3.2) aims at providing guidelines (think of a checklist) for the requirements analysts and security engineers to follow when specifying secure socio-technical systems. Therefore, the presented steps are not prescriptive. Moreover, the order of the modelling phases, as well as that of the activities within each modelling phase is not prescriptive too. The requirements analyst may decide to represent stakeholders' informational assets before modelling social interactions among them for instance. The security engineer may work in parallel with the requirements analyst, however, in the first iteration the representation of security needs by the security engineer requires the modelling of stakeholders' interactions and of stakeholders' assets to be performed by the requirements analyst. Nevertheless, this should not be seen as a limitation, rather it is related to the incremental nature of the process.

Automated analysis. Analysis activities are supported by the *automated analysis* phase (Phase 4, Figure 3.2). As argued in Chapter 1, model-driven approaches result in large (growing) models, the analysis of which requires automated tool support. Automated analysis is based on the formalisation of the modelling notation (language) employed during the Modelling phase. Such analyses are essential to verify whether the system meets its specification, as well as to verify desired properties of the derived security requirements, such as their consistency, and other extensive reasoning.

Automated analysis in STS considers verifying the well-formedness of the constructed models (*Well-formedness analysis*), verifying the satisfaction of stakeholders' security needs over the model of the system-to-be (*Security analysis*), as well as verifying the impact of threats affecting stakeholders' assets (*Threat analysis*).

As explained above, analysis results may be used to improve the created models, and therefore, analysis activities can be iteratively executed after every modelling iteration. The process represented in Figure 3.2 suggests a sequence of execution of analysis activities, but this is not prescriptive too. As a matter of fact, only well-formedness analysis is required to be executed

first, for the other two analyses require a well-formed model. As far as security analysis and threat analysis are concerned, they could be performed in any order, as the security engineer chooses to execute them.

Specification. The security requirements specification is supported by the *Specification* phase (Phase 5, Figure 3.2). The derivation of security requirements specification is also a tool supported activity, otherwise it would be a tedious and error-prone activity, not to say unfeasible to be conducted manually over growing (in size) goal models at every iteration. The security requirements specification contains the list of security requirements the system-to-be and its participating stakeholders should satisfy for the given system to be secure.

The specification phase is also iterative and incremental. Following the overall process of the STS methodology depicted in Figure 3.1, it is good practice to generate the security requirements document at the end of the modelling and automated analysis activities. The document enumerates the security requirements while providing textual descriptions that detail what the system or stakeholders should (not) do to comply with the given security requirements.

However, the security requirements engineer may use the details provided by this document to further improve the models and iteratively run analysis till a point is reached, in which stakeholders agree on the specified security requirements.

Indeed, the security requirements document facilitates performing these iterations, for it is tailored to provide details on the various models, elements in each model, as well as on the analysis results. As such, it provides extensive details on the detected errors and conflicts identified during the automated analysis phase. Therefore, the document can help the requirements analyst when communicating with stakeholders, to better capture their needs regarding security, as well as to resolve conflicts between security requirements, before an error-free model (and as a result a consistent security requirements specification) is obtained. We want, however, to highlight the fact that it is in practice quite impossible to solve all conflicts [Ernst et al., 2012], and in many cases the security requirements engineer may decide to stick with a specification that does not meet (satisfy) all security requirements, where the design alternatives to complying with all security requirements may result too costly.

In specifying security requirements, the STS methodology aims to comply with the terminology that security experts are familiar with. Unfortunately, there is no agreement upon a reference taxonomy of security requirements and mechanisms that we could adopt. As discussed in Chapter 2, [Stallings and Brown, 2008] state that: "*although the CIA triad to define security objectives is well established, some in the security field feel that additional concepts are needed to present a complete picture, and two of the most commonly mentioned are authenticity and accountability*". Based on this, we propose a classification of the main aspects of security that combines ideas from multiple sources [Kissel, 2011; Gollmann, 2011; Pfleeger

and Pfleeger, 2012], starting from the basic CIA triad and considering other security concepts deemed most important, namely *authenticity*, *reliability* and *accountability*. See Table 3.1 for an overview.

Table 3.1: Security principles employed by the STS methodology

Principle	**Definition**
Confidentiality	Assures that private or confidential information is not made available or disclosed to unauthorised users, and that users control (or influence) what information related to them may be collected, used, and to whom it is disclosed [Stallings and Brown, 2008, Chapter 1].
Integrity	Assures that information is not changed (modified) or destroyed in an unauthorised way [Stallings and Brown, 2008, Chapter 1]
Availability	Assures that a system works promptly, service is not denied to authorised users, and timely and reliable access to and use of information [Stallings and Brown, 2008, Chapter 1].
Authenticity	The property of being genuine and being able to be verified and trusted [Kissel, 2011]. Authenticity is ensured through authentication processes that verify whether users are who they say they are (entity authenticity [Stallings and Brown, 2008, Chapter 1]).
Reliability	It is concerned with the consequences of accidental errors [Gollmann, 2011], but it also includes non-designed usages performed by attackers.
Accountability	It refers to the requirements for actions of an entity to be traced uniquely to that entity [Kissel, 2011] (e.g., non-repudiation of a communication that took place).

Starting from these high-level security properties we define a taxonomy of security requirements types to be supported by the STS methodology [3]. This taxonomy refines the core (selected) security principles in information and computer security [4], and serves as a checklist for security engineers to keep track of the security requirements types to be considered, and whether they are applicable or not to the system at-hand. This is important to understand the security needs expressed by stakeholders, and map them to one of the types defined as part of this classification proposed in the taxonomy. While some security requirements types apply in all domains (e.g., confidentiality), others are domain-specific (e.g., reliability requirements are applicable to systems offering real-time services for instance).

[3] The taxonomy is introduced in Chapter 4.

[4] Details on the refinement of security properties to security requirements will be provided in Chapter 4.

In the rest of the sections, 3.2.1–3.2.5, the five phases of the process supported by STS methodology are described.

3.2 STS phases

The requirements analyst(s) and the security engineer(s) should conduct the phases detailed below, when following the process supported by STS methodology.

3.2.1 Phase 1. Social modelling

Security is mainly concerned with the protection of stakeholders' assets. Unsurprisingly, our methodology starts with a study of the context—led by the requirements analyst—which encompasses the identification of the stakeholders in the socio-technical system (Activity 1.1), and the identification of their assets and the interactions among actors (Activity 1.2).

We support two types of assets: *informational*, referring to stakeholders' information that they want to protect and *intentional*, referring to the stakeholders' objectives that they want to achieve. Information is represented in various forms. In STS, we consider information to be represented within documents that contain information. Hence, in this phase, we identify the various documents (representing the said information) the stakeholders have and exchange with others. The protection of documents as assets becomes evident when considering the contained confidential information. In our motivating scenario, an informational asset is the patients' medical status, while an intentional asset for the Red Cross BTC is to collect and examine blood.

After the context has been defined, the security engineer leads the conduction of Activity 1.3: the stakeholders' security needs on their interactions are elicited and expressed. For instance, Alice may want to *take tests* only at a specialised laboratory, such as ModernLabs, before filing an application as a donor and it may require that all tests are examined in that laboratory, not any other.

Considering the possible attackers that intend to exploit vulnerabilities of a system is an important activity of security analysis, and socio-technical systems are no exception. Hence, STS does not overlook threats, and supports the identification of social and organisational threats, which do not exploit technical vulnerabilities. Activity 1.4 supports exactly this, the threats that can potentially affect stakeholders' assets are modelled. Potential threats, such as for instance, *test results lost* might threaten the possibility of delivering (as a result, receiving) test results on time. This threat affects ModernLabs in providing a timely service to its patients, and the patients themselves, in this example Alice.

Unlike traditional methodologies, these activities are conducted by considering the interactions among the stakeholders and systems in the socio-technical system, as opposed to consid-

ering the technical system alone. Note that threats are of social and organisational nature, they do not necessarily exploit technical vulnerabilities of a software system. Let us reconsider the threat *test results lost*. This threat might occur as a result of a staff member forgetting test results at a printer; it is not necessarily the result of someone intruding in the system and stealing the results.

The social modelling phase is conducted iteratively in order to refine initial models. The outcome of this phase is the *social model*, which is presented in Section 5.2.

3.2.2 Phase 2. Information modelling

An important distinction that we make in STS is that between *primary* and *supporting assets*. In Phase 1, the analysts look at the informational assets in terms of concrete exchanges of documents. These documents are supporting assets, and their relevance from a security standpoint is due to the information (primary asset) that they represent.

The purpose of information modelling is twofold. Firstly, the requirements analyst identifies the information in the considered domain and determines which stakeholders own this information (Activity 2.1). For instance, *medical history* is an information that is owned by the *patients*, whose medical history this information reflects.

Secondly, the analyst structures the information by determining *part-of* relations and by relating information to the documents that materialise it (Activity 2.2). For instance, information *medical history* is a composite information: patients' *personal data* and *health status* are part of *medical history*. The latter is materialised by a *report* (document), which is in turn part of the document *health record*.

The outcome of this phase—conducted by the requirements analyst—is the *information model*, which acts as a bridge between the social model and the authorisation model. This model is described and illustrated in Section 5.3.

3.2.3 Phase 3. Authorisation modelling

Permissions and prohibitions are key concepts in security, for actors may permit (allow) or prohibit other actors to use (manipulate) their valuable assets. By leveraging on the information model the security engineer defines the permissions and prohibitions that the stakeholders grant (specify) one to another.

The outcome of this phase is the *authorisation model*. As we will show in Section 5.4, this model enables expressing fine-grained relationships concerning *who* can use, for *what* purpose, and *how*, documents that represent specific information. To give an example of the fine-grained relationships supported by the authorisation model, consider that Alice should provide to the Red Cross BTC her *health record* when filing an application as a donor. As such, the Red Cross

BTC should be authorised to access such document. However, Alice may authorise the Red Cross BTC to read her health status information to approve her as a donor, but she may prohibit any modification over the said information. Also, her authorisation granting Red Cross BTC to read her health status may be limited to an objective (scope): Red Cross BTC can read the information to approve Alice as a donor, but not for any other purposes.

3.2.4 Phase 4. Automated analysis

The three models created in Phases 1–3 are tightly related, indeed they are submodels of the overall model of the system-to-be. Therefore, they may include inconsistencies, conflicts between security requirements (captured in different models, but having effects across models), and threats that endanger stakeholders' assets. The purpose of the automated analysis phase is to discover these issues, which are represented in the *analysis results* artefact. If errors are detected, then the process steps back to social, information, and authorisation modelling, till all errors are fixed. The errors are fixed by the security engineer, but may require also the involvement of the requirements analyst and stakeholders to negotiate on distinct conflicts that are not resolvable by analysing the models.

We will comprehensively describe this phase—which is led by the security engineer—, and the three activities that it consists of, in Chapter 6.

3.2.5 Phase 5. Specification

The process followed by our methodology terminates with the specification phase, led by the requirements analyst, which takes an error-free socio-technical system model and returns a security requirements specification. This document contains a specification of the security requirements for the socio-technical system under design, while describing the overall system, its stakeholders, providing details on the created models (social, information, and authorisation) and the relevant security issues. The activities of this phase are automated by the CASE Tool. In Sections 7.3 and 7.1.4, we will show how our tool enables the automated generation of the security requirements document.

3.3 Chapter Summary

The STS methodology is a security requirements engineering methodology for the design of secure socio-technical systems. Key features of STS are to analyse the problem domain in terms of both social and technical aspects, and to ascribe security requirements to the interaction among the actors in the socio-technical system.

We described the main modelling and analysis activities that guide the requirements analysts and security engineers in the design of secure socio-technical systems. This chapter serves as foundation for the rest of the thesis, as we will see how the STS methodology is used in practice to model and analyse three industrial case studies from three different domains in Chapter 8.

Chapter 4

The Socio-Technical Security Modelling Language

This chapter presents the Socio-Technical Security modelling language (STS-ml) for representing security requirements in *socio-technical* systems. In addition to presenting the language, we highlight similarities with and differences from security requirements methods and languages for *software* systems.

We present the principles that underlie STS-ml in Section 4.1. We describe the modelling primitives to represent the stakeholders in a socio-technical system (Section 4.2), the interactions among those stakeholders (Section 4.3), the events that threaten the stakeholders (Section 4.4), and the security requirements that the stakeholders want the socio-technical system to fulfil (Section 4.5).

Acknowledgement. This chapter builds on top of [Dalpiaz et al., 2014] and revises and extends [Dalpiaz et al., 2011; Paja et al., 2013b,c].

4.1 STS-ml: principles

As any other design artifact (civil buildings, bridges, cars, electronic devices, software systems, advertisements, lectures, books, etc.), modelling languages are conceived on the basis of a set of principles that constitute the requirements for the language.

STS-ml is designed with the challenges laid in Chapter 1, Section 1.5 in mind (social / organisational perspective, security on interactions, social threats, from needs to specifications, stakeholders' assets, separation of concerns, and formal semantics), which guide the conception of the modelling language. Addressing these challenges is important to ensure that the language is a good fit for its application domain (security requirements in socio-technical systems) and a good support for the STS methodology. In addition to addressing these challenges, we are

guided by other principles to guarantee that our language accommodates desired properties for (requirements) modelling languages [Loucopoulos and Karakostas, 1995]. These properties, such as abstraction, minimality, formality, traceability, and ease of analysis among others, are desiderata for modelling languages [Easterbrook, 2002]. STS-ml, too, is intended to satisfy these properties in order to avoid ambiguities, and errors in the resulting models. As a result, the set of principles used to guide the creation of STS-ml are as follows:

Principle 1 (Social and organisational perspective) *The modelling primitives have to enable creating security requirements models for a socio-technical system that encompasses humans, organisations, and software systems, in which social and organisational aspects are critical for a comprehensive security requirements engineering process. These participants interact to fulfil their own strategic objectives (humans and organisations) and requirements (software systems).*

Security requirements for software systems such as, for instance, R_1: "The system shall ensure the confidentiality of the collected donor's personal data", have to be represented by primitives that acknowledge the socio-technical nature of the setting. This means representing the existence of multiple actors (e.g., a donor, a nurse, and the hospital's information system), their interaction for the purpose of donating blood, the need of recording the donor's personal data, and the donor's desire to keep such data confidential.

Principle 2 (Multiple stakeholders) *Socio-technical systems consist of multiple participants. Each participant is an autonomous stakeholder and, as such, specifies her own (security) requirements, which may conflict with others'. The language shall therefore express the stakeholders' viewpoints, and enable the identification of conflicts.*

Security requirement R_1, expressed by donors, may conflict with the requirement of haematologists to access some details, such as age, profession, drinking/eating/smoking habits, etc. The language shall be able to represent these viewpoints, also to identify that a conflict exists.

Principle 3 (Stakeholders' assets) *The language shall support modelling at a high-level of the assets that the stakeholders care about, such as their strategic objectives and their information.*

(Security) requirements engineering is the initial stage in system design. As such, the activities in this phase are conducted at a level of abstraction that is higher than in following phases. A key principle in computer and information security is the protection of the assets that the stakeholders value. The language should support representing these assets at a high level of abstraction so to express security requirements for their protection. In the case of R_1, a donor's asset is her personal data, while a haematologist's asset is the goal of assessing the adequacy of prospective donors.

Principle 4 (Security over interactions) *The participants in a socio-technical system are autonomous and, thus, loosely controllable. It is while interacting with others—to get things done or to exchange information—that a participant is concerned with the possible consequences of relying on others and transferring information, and states/expresses security requirements over the interactions.*

Referring to requirement R_1 again, and its socio-technical interpretation that we sketched earlier, the desire of keeping the transferred data confidential is an example of a security requirement that is imposed by the donor over the interaction with the nurse. This requirement affects, in turn, the interaction between the nurse and the information system. Indeed, if the nurse does not mark the data as "confidential", or if the information system does not allow this option, the donor's need could be violated.

Principle 5 (Capturing security needs) *The language shall focus on the needs of the stakeholders, while designing socio-technical systems with security in mind, rather than on the solutions or mechanisms to address these needs.*

The language is thought for an early stage of system development. As such, its models shall express *why* security is needed [Yu and Mylopoulos, 1994], and *what* are the security needs of the stakeholders participating in the system. The language shall focus on the needs of the stakeholders, while designing socio-technical systems with security in mind, rather than on the solutions or mechanisms to address these needs. Security solutions or mechanisms are to be considered in later stages of system design. In this way the modelling language supports the STS methodology to specify security requirements starting from stakeholders' security needs. Indded, security needs are expressed by the stakeholders while interacting, and for each security need expressed by one stakeholder on an interaction with another, a security requirement in the opposite direction is derived for the satisfaction of the required security need. In this way, STS-ml supports the derivation of security requirements from the modelling of stakeholders' security needs. More details will be provided in Section 4.5.

Principle 6 (Social and Organisational Threats) *The language shall not overlook threats, and shall support the identification of social and organisational threats, which do not necessarily exploit technical vulnerabilities.*

In the example of R_1, possible threats are that the nurse forgets to mark the donor's personal data as "confidential", or that she leaves a printout of the donor's record in the waiting room. Notice that these threats are social or organisational, for they do not exploit a technical vulnerability of a software system, while putting at risk the said information.

Principle 7 (Compliance with international security standards) *Whenever possible, the primitives in the language should adhere to standards and to common terminology, so to improve comprehensibility and to avoid a steep learning curve.*

This principle applies to many fields and disciplines. In the context of this thesis, however, the challenge is to reuse the terminology used in traditional security for software systems in the context of socio-technical systems.

Principle 8 (Diagrammatic and formal language) *The modelling language shall support modelling through diagrams, and its primitives shall be equipped with a formal semantics.*

While (security) requirements engineering is often conducted through informal analysis and relies on textual requirements descriptions, we advocate a model-driven approach where models are represented through diagrams—which serve as a communication means among modellers and with stakeholders—and have an underlying semantics, which supports (i) minimising ambiguities in the interpretation of the diagrams, and (ii) performing automated reasoning, e.g., for detecting conflicts among requirements.

Principle 9 (Minimality of concepts) *The language will be composed of a minimal set of primitives, namely concepts and relationships, which are needed to capture security requirements in socio-technical systems.*

This principle requires that the chosen primitives are not redundant, also that they enable representing the important security requirements that the stakeholders want to express. Lack of minimality leads to redundancy of concepts, which leaves the security requirements engineer with more choices on representing the same thing [Easterbrook, 2002].

Principle 10 (Traceability) *The language shall ensure traceability of security requirements to their requester and to the motivations (intention to achieve goals; protection of documents or information) that originated the requirement.*

Traceability, too, is a desired property for a modelling language [Easterbrook, 2002]. For example, considering R_1, the language shall enable determining who requested the requirement (the donor), and why (the donor wants to have her blood examined, and needs to provide its personal data to such extent).

4.2 Representing stakeholders in socio-technical systems

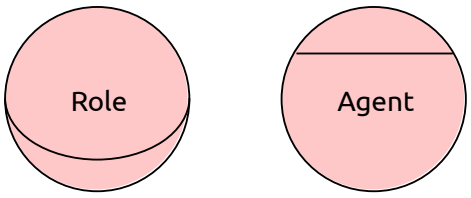
Socio-technical systems consist of multiple autonomous interacting participants or stakeholders: humans, organisations, and technical components. Software systems and infrastructures

are examples of technical components. Organisational units and employees, on the other hand, are examples of social components. As per Principle 1, the modeling language shall enable representing these participants. Their representation is crucial not only because they are the main components in a socio-technical systems, but also because from a security viewpoint, they are the subjects that express their needs on the system-to-be.

STS-ml supports the modelling of stakeholders (socio-technical system's participants) in terms of *roles* and *agents* [Yu, 1995].

A role to model a class of participants, defining a set of responsibilities for the said participants. One can think of role as a container that carries a set of responsibilities within the system. Roles are used to model participants when the actual individual(s) are unknown. Examples of roles include *patient*, *physician*, *donor*, etc. One knows that patients, physicians, donors will be part of the system, can define their responsibilities, but does not necessarily know the identity of the agents that will play those roles. Roles are an important concept in the modelling of a socio-technical system, because given the setting, designers are often unaware of who are going to be the actual participants at runtime. Specifying applications at the role level is a flexible approach that defines the requirements for an agent to adopt a role, as opposed to mandating the existence of a specific agent. At runtime, the actual participants will adopt the specified roles.

An agent, on the other hand, refers to a specific individual that will participate in the system. Additionally, it allows to represent concrete participants known to be in the system already at design time. For example, *St. John's Hospital* and *Hong Kong Red Cross* are agents that will participate in the blood transfusion and donation socio-technical system. The graphical syntax for roles and agents is represented in Figure 4.1.



(a) Role (b) Agent

Figure 4.1: Graphical representation of roles and agents

Whenever we do not need to distinguish between role and agent (for properties or relationships applying to both), we will use the general term *actor* to refer to either a role or an agent. Note that actor is an abstract concept that, unlike role and agent, has no graphical syntax in STS-ml. The notion of actor is a very generic one, for many different entities can be classified equally as actors. For instance, "Physician", "Physician Mark", and "Mark" can all be considered actors. While a complete treatment of this subject is outside the scope of this thesis (for a

comprehensive and well-founded account on the topic see [Masolo et al., 2004] and [Guizzardi, 2006]), STS-ml makes use of the concepts role and agent in the name of Principle 9: *minimality of concepts*.

We use the terms stakeholder, participant, and actor interchangeably, as they refer to different facets of the same entity. Specifically, the modeling language represents the socio-technical system in terms of actors, which correspond to the stakeholders that express their needs about the system-to-be on behalf of the actual participants in the system.

For example, a sample set of donors would be selected and constitute the stakeholder "donor", that expressed requirements on behalf of the prospective participating donors. A donor would be represented as an actor (more specifically as a role) in STS-ml.

Agents and roles can be related one to another: an agent can play (adopt) a role. For example, Mark can play role physician, while Alice can play role donor. STS-ml introduces the relationship play between an agent and a role (graphically shown in Figure 4.2).

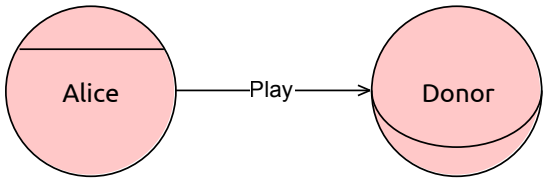


Figure 4.2: Graphical representation of a plays relationship

Some observations about the play relationship are worth. An agent can play multiple roles. In some cases, multiple roles can be played simultaneously (e.g., a physician may be both a surgeon and the head of surgery in a hospital). In other cases, an agent changes its role in the socio-technical system (e.g., Alice may play role *nurse* and, after work, switch to role *donor*).

STS-ml considers actors in a socio-technical system as peers, and therefore, it does not make use of association relationships among roles [Zannone, 2007] or agents [Yu, 1995].

4.2.1 Actors' assets

A fundamental element of security is the identification and protection of the assets that stakeholders care about. As pointed out in Principle 3, STS-ml is concerned with high-level assets that can be identified at requirements time, before design decisions about the system-to-be are made.

In STS-ml, we take the standpoint that every stakeholder, that is, every modeled actor has a set of assets that it wants to protect, its *primary assets* [ISO/IEC, 2005]. These assets become subject to vulnerabilities and many be exploited through *supporting assets* [ISO/IEC, 2005], which refer to tangible or concrete assets. For instance, a donor's personal information

and health status are her primary assets, while the donor's certificate is a supporting asset that contains this relevant information.

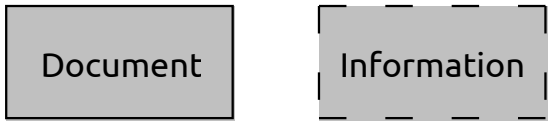
Whenever a supporting asset is handled in an unauthorised way, the primary assets it is related to could be harmed too. Therefore, the necessity of protecting supporting assets depends on the criticality of their corresponding primary assets. If the certificate did not contain any health related data or personal information, the donor would not consider its protection critical.

In STS-ml, we are concerned with the security issues actors face in the socio-technical system, and consider two main types of assets: *informational* and *intentional*. This categorization is related to the fact that actors in a socio-technical system enter with the intention of achieving their objectives, and they interact with others to achieve their objectives while exchanging related information.

Informational assets

Actors are concerned about the information that they own, which they often consider as confidential (it is a primary asset for them). STS-ml distinguishes between *information*—the data that actors own, care about, and may deem confidential—and its representation via *documents*. The latter, intended in a broad sense (e.g., an email or a text message are documents too) are the means through which actors transfer information.

Thus, in STS-ml we consider actors' information to be made available in the form of documents, while documents themselves represent transferable entities (e.g., donor certificate, identity document, driving license, an email), which contain some information (e.g., date of birth, name, surname, address, medical status). Since documents represent information, they are supporting assets. Hence, in order to protect information from misuse, it is crucial to protect the documents that make available (materialize) this information and enable its transmission [ISO/IEC, 2005]. The two concepts are illustrated in Figure 4.3. Note the similarity in the representation, the shape for both is a rectangle to denote the fact that they refer to different representations of information, but the principle of perceptual discriminability [Moody et al., 2009] with a visual distance of 1 (only the contour line has been changed to distinguish information—dashed line, from document—continuous line).



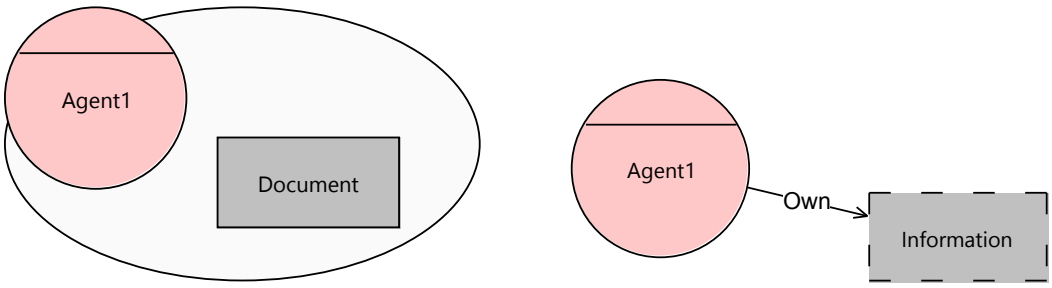
(a) Document (b) Information

Figure 4.3: Graphical representation of informational assets

In STS-ml, we use the possesses relationship to denote the fact that the actor has a document in the socio-technical system. For instance, *Red Cross BTC* possesses *donor certificates*. The graphical representation of actor possession is shown in Figure 4.4a. Actors can manipulate only (perform operations or actions over) documents that they possess. Moreover, they can get in possession of a document when the document is transferred to them from some other actor. Possession implies the actor's capability of transferring the document to other actors. In general, the actor transferring the document would not possess the document after transferal. In STS-ml, we simplify this view, keeping track only of the actors possessing the document and the operations they can do over those documents. We will explain document manipulation (aka operations actors may perform over them) and transferral in Section 4.2.2 and Section 4.3, respectively.

Information may have one or more legitimate owners. For example, a patient's personal data is owned by the patient, a credit card number is owned by the cardholder, and a bank account number is owned by its holder(s). In STS-ml, the relation owns indicates that an actor (a role or an agent) is the legitimate owner of the information it is related to, and can freely make use of it, as well as decide to transfer rights over it to others. The graphical representation of information ownership is shown in Figure 4.4b.

Note that possession is different from ownership, for it refers only to the actor having a document, not necessarily owning the information therein.



(a) Document Possession (b) Information Ownership

Figure 4.4: Graphical representation of document possession and information ownership

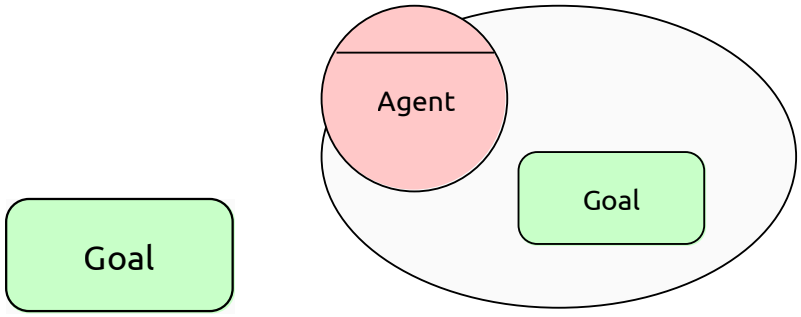
In order to talk about confidentiality and other security requirements concerning information, it is crucial to model the relationships between documents and information, which are the supporting and primary assets in the category of informational assets, respectively. We will show how STS-ml enables a fine-grained modeling of these relationships in Section 4.2.3.

Intentional assets

The behavior of agents is a result of their attempts (following their intentions) of fulfilling their objectives [1]. Thus, agents care about their intentions not to be threatened by other agents with different intentions (including attackers). A role characterizes a specific class of agents; we ascribe intentions to roles too, with the meaning that any agent that plays that role at runt-ime would inherit those intentions.

STS-ml represents intentional assets using the notion of goal, which denotes a desired state of affairs (e.g., donor approved, blood donated) for an agent or a class of agents (a role). Actors enter the socio-technical system with the intent to achieve their desired goals, thus, actors' main goals also are primary assets for them.

Goals are different from activities and processes: goals are part of the motivational component of an agent, and express *why* and *what* an agent aims to achieve, rather than *how* the agent is to achieve its objectives. Figure 4.5a shows how a goal is graphically represented, while Figure 4.5b illustrates that an actor (in the example, agent) has the intention of achieving a goal.



(a) Goal (b) Intention

Figure 4.5: Graphical representation of goals and intentions

Figure 4.6 provides examples of some of the intentions the actors in the healthcare scenario have. The agent *Red Cross BTC*, for instance, has the goals of distributing, collecting, examining blood, and performing statistical analysis among others. The *Patient* has the goal of receiving a treatment, while the *Donor* has the goal of performing blood donations on a regular basis. Notice that *Alice* is an agent that plays role *Donor*, but her goals are not the same as of the role it adopts. In other words, agents may have personal goals that differ from those of the roles they play.

[1] We consider only rational agents, i.e., agents whose actions are directed towards the fulfillment of their goals. Technical components are agents whose requirements are expressed via models from goal-oriented requirements engineering [van Lamsweerde, 2001].

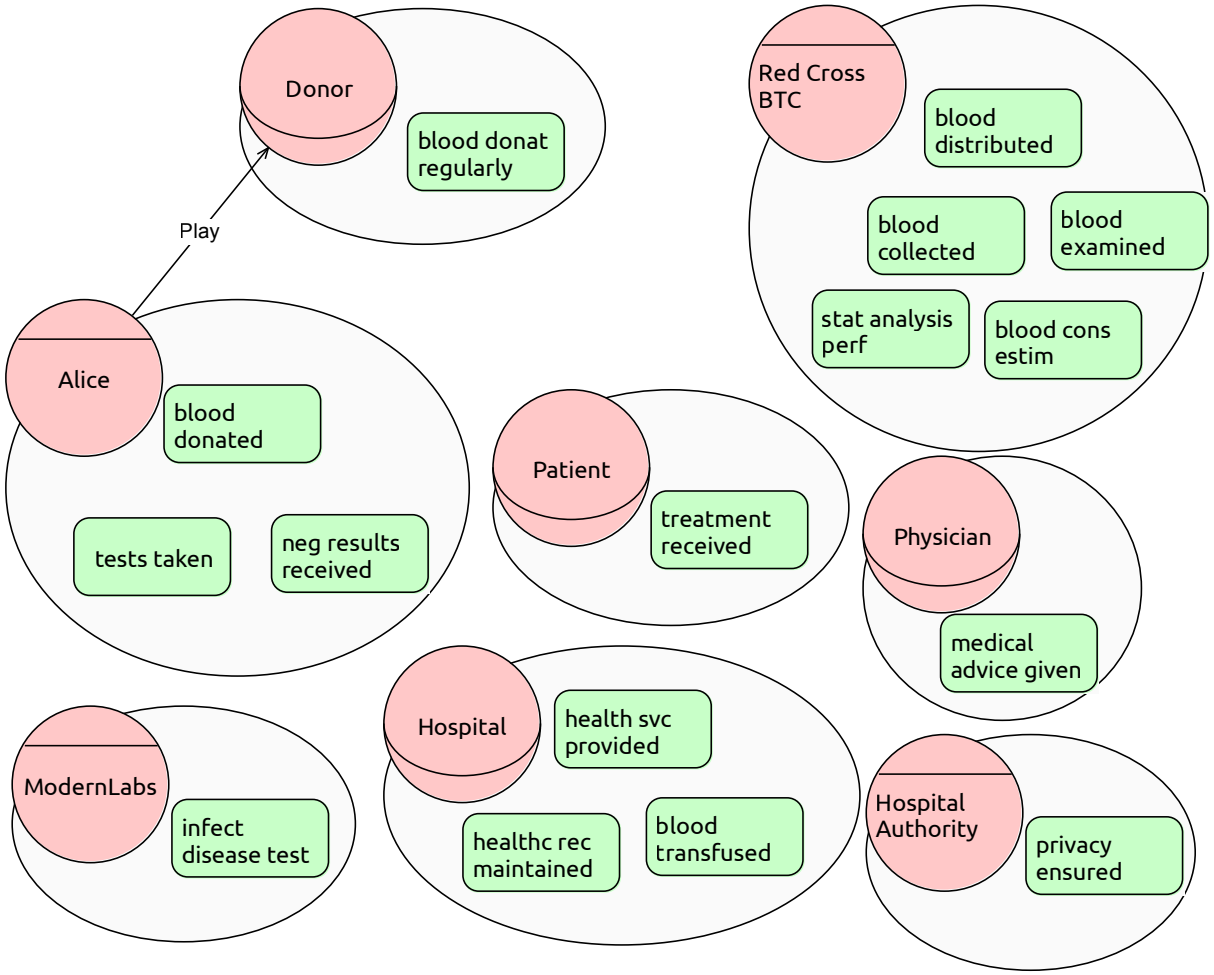


Figure 4.6: Actors' intentions in the healthcare scenario

Note that in Figure 4.6 goal labels are in passive voice. This form is chosen since a goal refers to desired state of the world the actor wants to achieve.

4.2.2 Actor models

The goals and documents of a specific actor in a socio-technical system relate one to another. There are important relationships that need to be captured, and that enable defining the rationale of that actor, i.e., how it aims to attain its goals. The purpose of this section is to explain how STS-ml supports modeling an actor's rationale.

The rationale of an actor is expressed by the elements and relationships that are contained within an *actor model*. This model consists of the actor's intended goals, its possessed documents, and the relationships between these elements that are described in this section. Graphically, an actor model consists of all elements and relationships within the boundary of the ellipse attached to the actor shape as in Figure 4.4a, Figure 4.5b, and Figure 4.6.

Relationships among goals

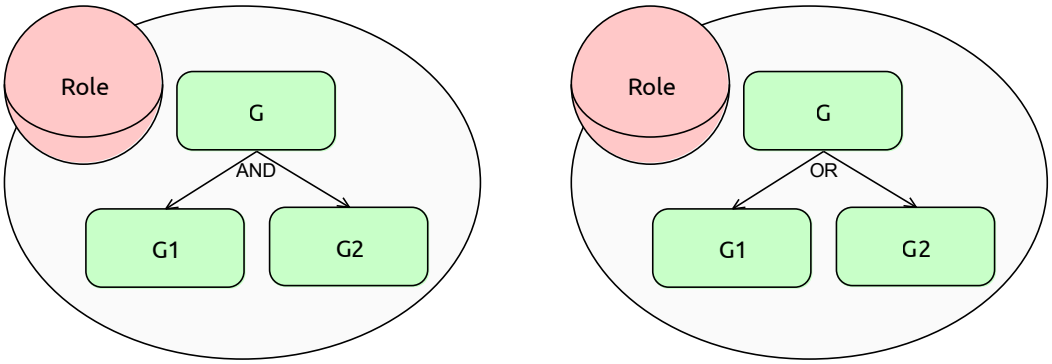
A fundamental way to relate the goals within an actor model is to refine (*decompose*) them into subgoals. Goal decomposition is widely adopted in requirements engineering (see, e.g., [van Lamsweerde, 2009; Yu, 1995; Bresciani et al., 2004]), and it allows reading goal hierarchies both top-down (to answer the question "*How* is the goal achieved?") and bottom-up (to answer the question "*Why* does this goal exist?").

Inspired by the literature in goal-oriented requirements engineering, STS-ml supports two types of goal decompositions, wherein one goal G is hierarchically refined into a non-empty set of subgoals $\{G_1, \ldots, G_n\}$. Specifically, the followin are the two relationships to support goal decomposition in STS-ml:

- and-decomposes (illustrated in Figure 4.7a): the achievement of all the subgoals implies the achievement of the decomposed goal G. For instance, the actor model of *Red Cross BTC* in Figure 4.9 includes an and-decomposition of goal *blood distributed* into subgoals *blood collected* and *blood consumption estimated*, meaning that both collection and consumption estimation are necessary for blood distribution;

- or-decomposes (illustrated in Figure 4.7b): the achievement of at least one subgoal implies the achievement of the parent goal. For example, in Figure 4.9, goal *statistical analysis performed* is or-decomposed into subgoals *on blood type eval*, *on hospital requests*, and *on donors*, referring to the different types of statistical analysis that could be performed by the Red Cross BTC.

As shown in Figure 4.9, subgoals can be further decomposed. In STS-ml, a goal has at most one parent. As a result, an actor model is a set of goal trees; each tree refines a top-level (root) goal into a set of leaf-level goals. Since the subgoals are a means to achieve actors' main (root) goals, the set of subgoals constitutes the supporting assets for the actors' corresponding



(a) And-decomposes (b) Or-decomposes

Figure 4.7: Graphical representation of goal and/or decompositions

top-level goal (primary asset). Following this intuition, the achievement of leaf-level goals is critical to the achievement of actors' top-level goals.

The refinement of goals in subgoals indicates how actors intend to achieve their root goals. Refinement continues up to the point in which there are enough details for the actor to evaluate the achievement of the goal. As an analogy, think of reducing complex problems to simpler ones. The actor needs to achieve leaf-goals first, in order to achieve their parent and ancestor goals, in order to achieve its the root goals.

Leaf-level goals in an actor model indicate the responsibilities of that actor. For a role, these responsibilities apply to every agent that adopts that role. For an agent, the responsibilities apply to the specific modeled individual.

Goal-documents relationships

Actors often need to make use of information in order to achieve their goals. As explained in Section 4.2.3, information is made accessible through documents. STS-ml supports several relationships between goals and documents, which represent the operations actors perform over documents (and, potentially, the information therein) while pursuing their goals. These relationships [2], illustrated in Figure 4.8, are:

- reads: indicates that an actor reads the content of a document while achieving a goal, thereby making the document necessary for the actor to achieve the goal. For example, in Figure 4.9, document *test results* is read by actor *Red Cross BTC* to achieve goal *donor approved*, for the adequacy of the potential donor shall be determined.

[2] We use the "-s" in the formal definition for accuracy, and omit it in the graphical notation for ease of understanding.

- modifies: indicates that an actor changes the informational content of a document while achieving a goal. For instance, in Figure 4.9, document *blood bank* is modified by the *Physician* while achieving goal *transfusion needed*.

- produces: indicates that an actor creates a new document while achieving a goal. For instance, in Figure 4.9, document *donor certificate* is produced by *Red Cross BTC* while achieving goal *donor approved*, as a proof that a certain individual has been recognized as a certified donor.

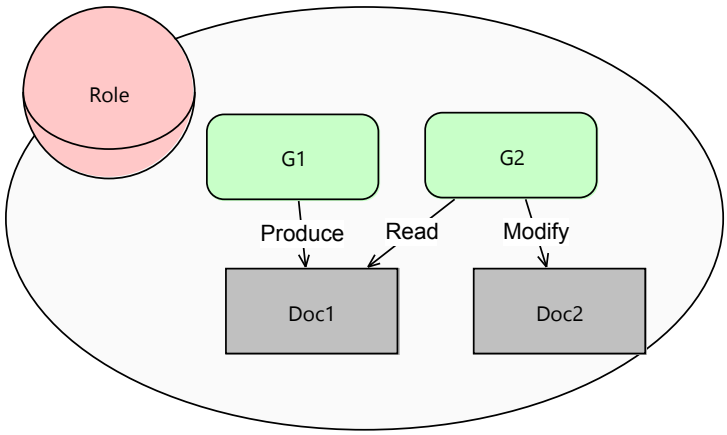


Figure 4.8: Graphical representation of goal-document relationships

Figure 4.9 shows an actor model for the *Red Cross BTC* that illustrates the goals of the actor (e.g., blood distributed, blood collected), its possessed documents (e.g., test results, report), decomposition relationships among the goals, as well as goal-document relationships.

As mentioned before, the availability of needed documents is crucial for the achievement of related goals. Therefore, the set of subgoals and documents necessary for the fulfilment of an actor's top-level goal are the supporting assets for that root goal (primary asset).

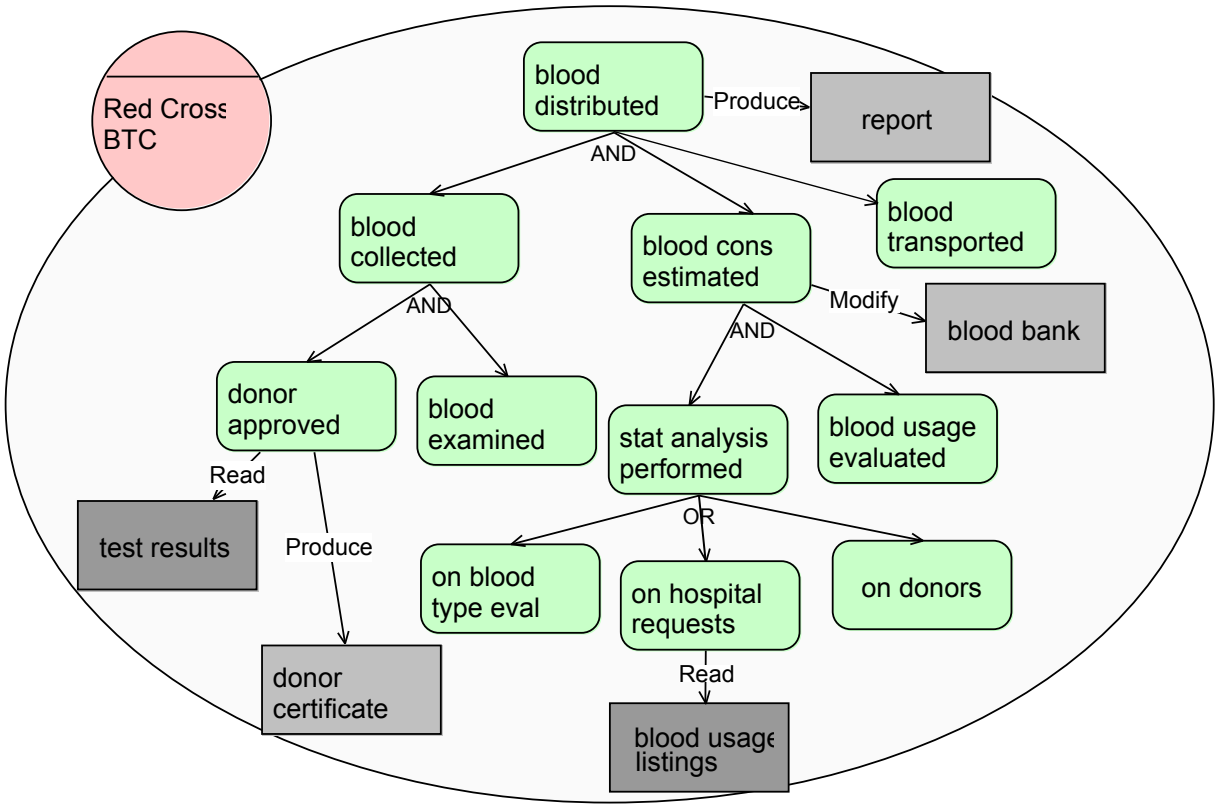
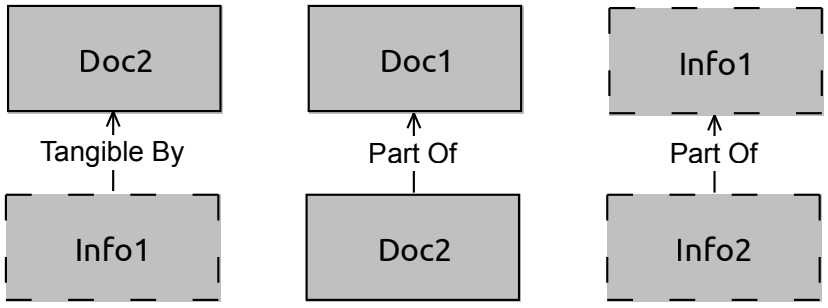


Figure 4.9: Actor model for Red Cross BTC

4.2.3 Structuring information and documents

Information is an important asset that stakeholders want to protect. Information becomes vulnerable when a document that represents it (e.g., a database record, an email, a letter, an instant message) can be accessed and modified by others. STS-ml introduces three primitives (their graphical representation is in Figure 4.10) that allow for structuring information and documents:

- tangible by is a relationship that relates an information entity and a document (illustrated in Figure 4.10a). It indicates that the specified information is represented through that document. For example, in Figure 4.11, information *medical history* is made tangible by the document *report*, while *health status* is made tangible by the *test results*.
- part-of between two documents (illustrated in Figure 4.10b) denotes that one document is essential part of another document (the latter document cannot exist without the former). For example, in Figure 4.11, the *report* created by the red cross while assessing a donor's adequacy is part of the *health record* of that potential donor.
- part-of between two information entities (illustrated in Figure 4.10c) denotes that a certain



(a) Tangible by (b) Part-of (c) Part-of

Figure 4.10: Graphical representation of part-of and tangible by

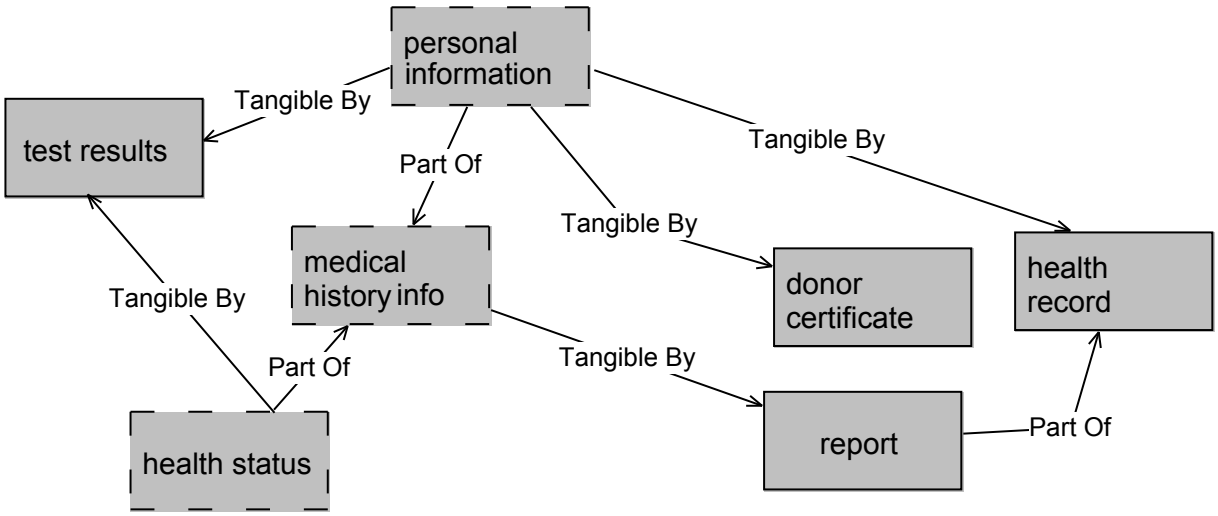


Figure 4.11: Some information and/or document relationships in the healthcare scenario

information is essential part of another information (the latter information cannot exist without the former). For example, in Figure 4.11, information *health status* is part of *medical history*. Also, *personal data* is part of *medical history*.

Note that these three relationships are of type many-to-many: an information entity can be made tangible by several document, while a document can make tangible more information entities. For instance, in Figure 4.11, information entity *personal information* is made tangible by both document *health record* and document *donor certificate*, while both information entities *personal information* and *health status* are made tangible by document *test results*.

Similarly, an information entity can be part of multiple information entities or an information entity may have multiple parts. The same applies to documents too. In Figure 4.11, for instance, both *health status* and *personal information* are part of *medical history*.

4.3 Modelling the interactions among actors

The actors in a socio-technical system are not isolated entities, but rather they interact with others either to fulfill their objectives—either because they cannot achieve them on their own or because it is more convenient or easier to rely on someone else— or to exchange information.

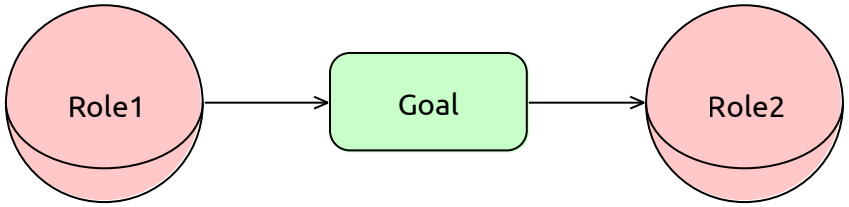
In requirements engineering, the notion of *social dependency* [Yu et al., 2011] has been suggested as a way to represent the fact that actors rely on others for the achievement of goals, the execution of tasks, and the availability of resources. An example of social dependency is for instance that of a patient depending on a doctor for performing blood transfusion (goal *blood transfused*). In STS-ml, we refine the notion of dependency into two more specific primitives that are explicitly thought for security requirements models:

- goal delegation (illustrated in Figure 4.12a): one actor (delegator) delegates to a different actor (delegatee) the fulfilment of a goal (delegatum). Delegation refines dependency by requiring the existence of an agreement between the delegator and the delegatee, and the transfer of responsibility. In STS-ml, a goal delegation results in the delegatee having the goal (graphically represented within its rationale) and being responsible to achieve it. For instance, in Figure 4.13, *Alice* delegates to *ModernLabs* goal *tests taken*. By doing so, *ModernLabs* becomes responsible for testing *Alice*'s blood sample.

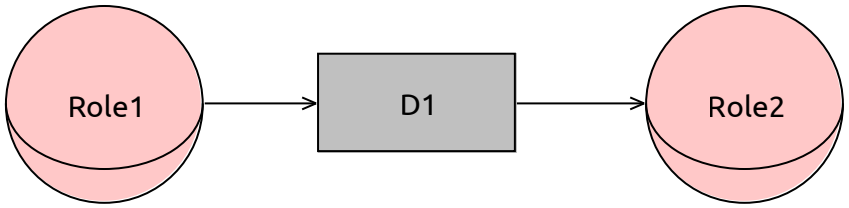
- document transmission (illustrated in Figure 4.12b): one actor (sender) transfers a document to a different actor (receiver). This refines dependency in the sense that the receiver depends on the sender for the availability of the document. The receiver possesses the document as a result of document transmission. In Figure 4.13, *ModernLabs* transmits document *test results* to *Alice*. In turn, she transmits it to actor *Red Cross BTC*.

In addition to interacting through goal delegations and document transmissions, STS-ml includes a primitive (authorisation, illustrated in Figure 4.12c) to capture two key concepts in security, i.e., *permissions* and *prohibitions*. The basic meaning is that one actor (in the illustrative relationship, *Role1*) specifies authorisations for another actor (*Role2*) on a set of information elements (*Info1* and *Info2*). For example, in Figure 4.14, Alice specifies an authorisation on information *health status* for *Red Cross BTC*, and one authorisation on *personal data* and *health status* for *ModernLabs*. Each authorisation specifies what the authorised actor can/cannot do with the information:

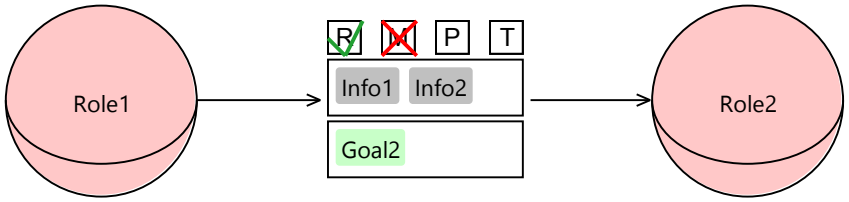
- *Allowed/prohibited operations*: they define whether the actor is permitted (green tick symbol) or prohibited (red cross symbol) to Read (R), Modify (M), Produce (P), and Transmit (T) any document that makes tangible the information. In Figure 4.12c, reading is allowed, modification is prohibited, and no authorisation is specified on production and transmission. In the healthcare scenario (Figure 4.14), the authorisation from *Alice* to



(a) Goal delegation



(b) Document transmission



(c) Authorisation

Figure 4.12: Graphical representation of goal delegation, document transmission, and authorisation

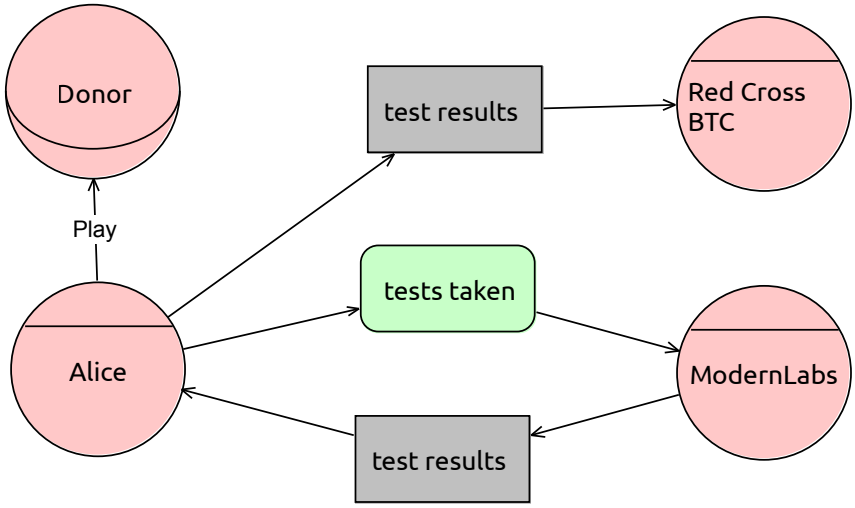


Figure 4.13: Some interactions in the healthcare scenario: Delegations and transmissions

ModernLabs grants permission to read and produce documents that make tangible her *personal data* and/or *health status*; the authorisation from *Alice* to *Red Cross BTC* permits reading but prohibits modification of documents containing her health status.

- *Information*: authorisation is granted over at least one information entity. Given the structuring of information in term of part-of relationships, authorising some actor over some information means that the actor is authorised for parts of information as well, because ownership of information propagates top-down through part-of relationships.

- *Scope of authorisation*: authority over information can be limited to the scope of a certain goal. As such, scope of authorisation defines the goals for the fulfilment of which the authorisation is granted. In other words, the authorisation is restricted to a certain purpose, and does not apply to different purposes. In Figure 4.12c, the permissions and prohibitions are restricted to the scope of goal *Goal2*. Our notion of goal scope adopts the definition in [Dalpiaz et al., 2010], which includes the goal tree rooted by that goal. As a result, if a goal is specified in the scope of authority, authority is given to make use of the information not only for the specified goal, but also for all its sub-goals. We assume that goal decompositions are part of the domain knowledge: there is no dispute between actors about how goals are hierarchically structured.

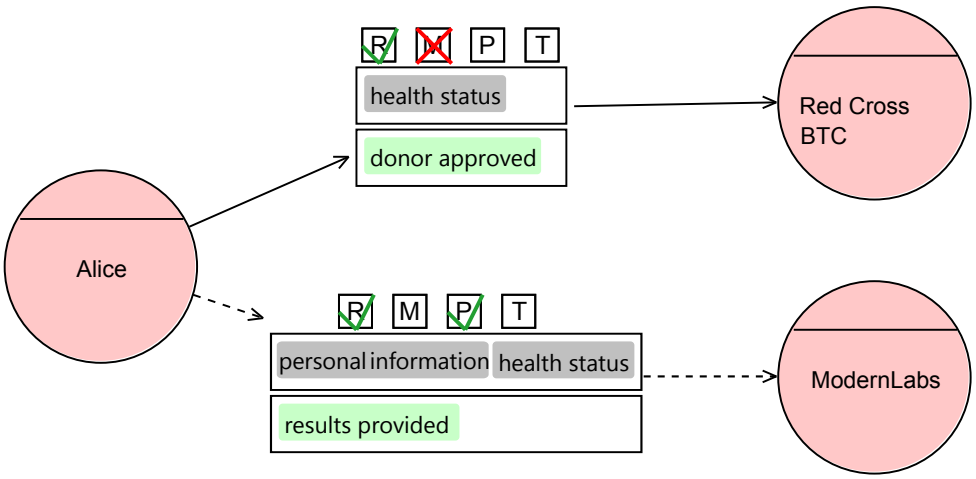


Figure 4.14: Some interactions in the healthcare scenario: Authorisations

In the healthcare scenario of Figure 4.14, the agent *Red Cross BTC* receives authorisations in the scope of goal *donor approved*; therefore, the permission to read and the prohibition to modify *Alice*'s health status apply only when *Red Cross BTC* is carrying out activities concerning the approval of *Alice* as a donor.

- *Transferability of the permissions*: it specifies whether the actor that receives the au-

thorisation is in turn entitled to transfer the received permissions or specify prohibitions (concerning the received permissions) to other actors. Graphically, transferability of the authorisation is allowed when the line connecting the two actors is solid, while it is not granted when it is dashed. In Figure 4.14, *Alice* allows the transferability of the authorisation granted to *Red Cross BTC*, while she does not allow the transferability of the authorisation granted to *ModernLabs*. Therefore, *Red Cross BTC* can authorise other actors, specifying both permissions and prohibitions concerning the reading of information *health status* in the context of goal *donor approved*.

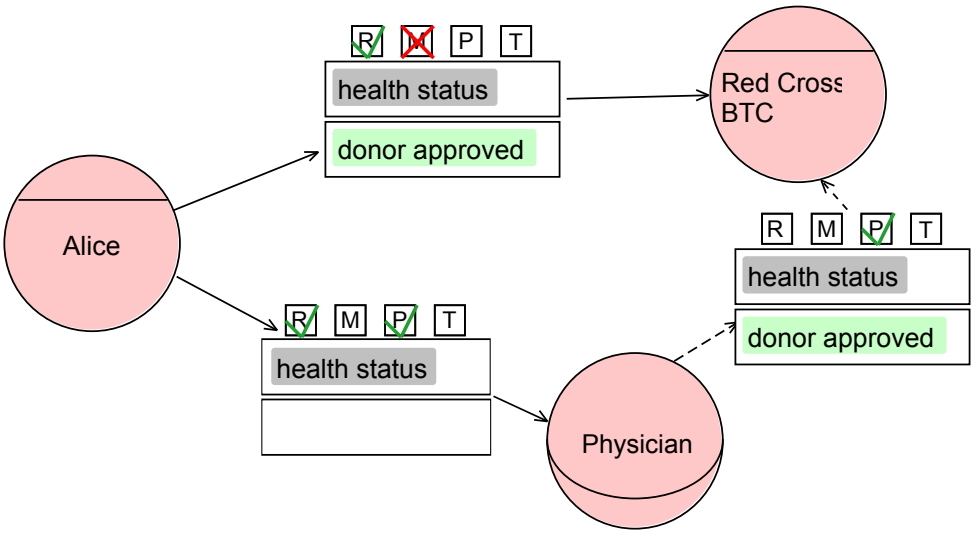


Figure 4.15: Some interactions in the healthcare scenario: summing authorisations

Note that in STS-ml authorisations are summed up, that is, whenever different authorisation relationships are drawn towards one given actor, then it is enough that at least one authorisation grants a given right (say to read) and none prohibits this right, then the authorised actor has the right to read the specified information. For instance, in Figure 4.15, actor *Red Cross BTC* obtains two authorisations for information *health status*: together, they specify that *Red Cross BTC* can read and produce documents representing *health status*, and is prohibited from modifying any document representing that information.

4.4 Events and threats

As per Principle 6, our language supports the representation of threats in terms of events that exploit the vulnerabilities of actors' supporting assets, namely *subgoals* and *documents*, in order to undermine their primary assets, *root goals* and *information* respectively. The primitives of

STS-ml are the entity Event and the relationship threatens; the latter links an event to a document or a goal.

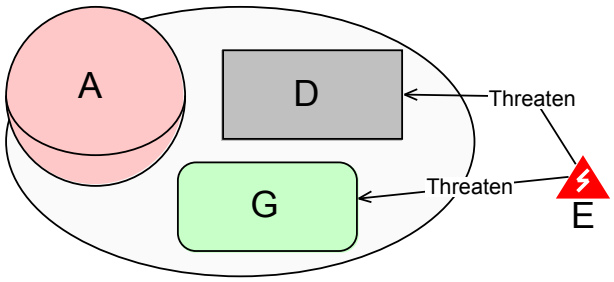
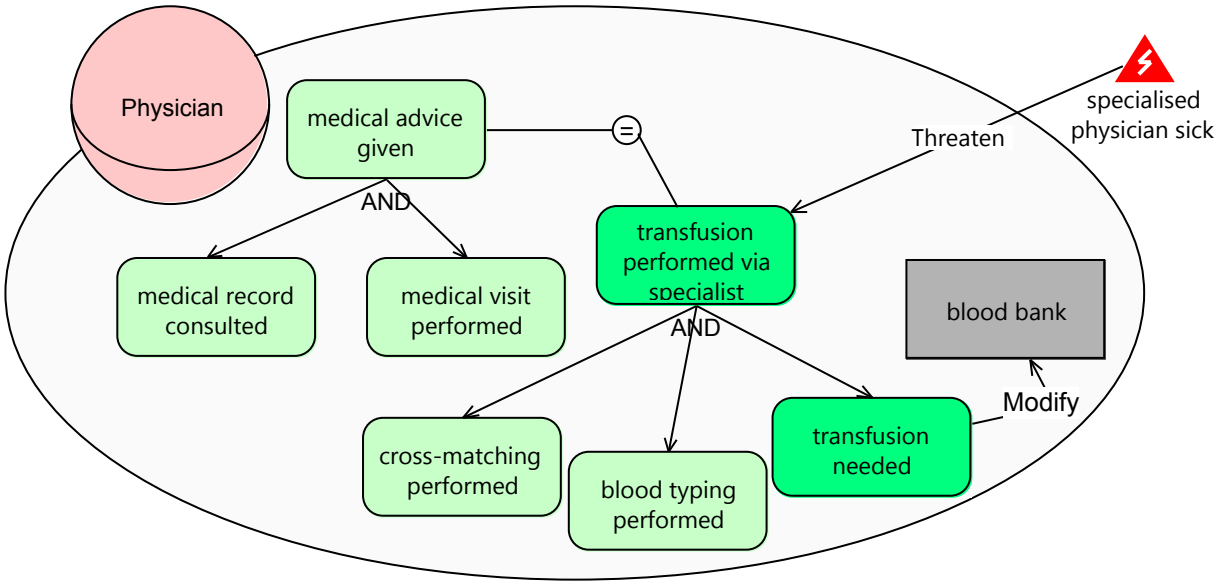


Figure 4.16: Graphical representation of events threatening actors' supporting assets in STS-ml

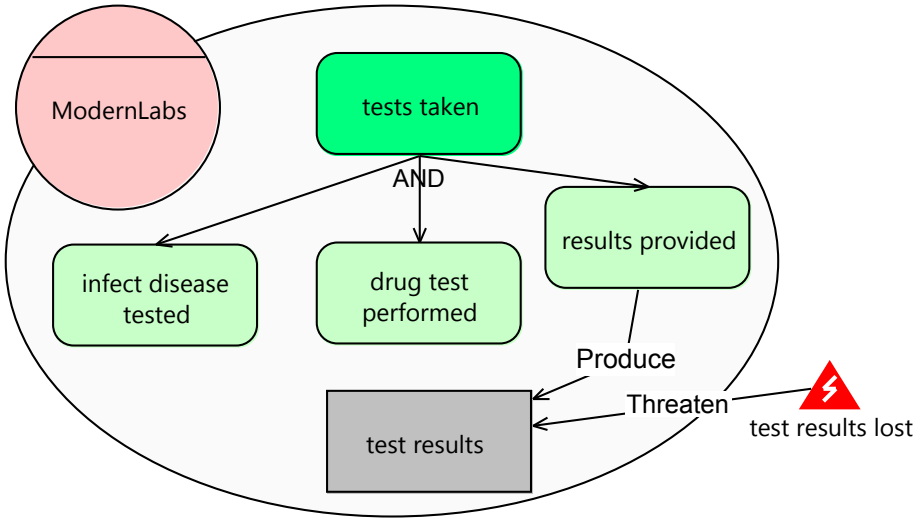
The graphical representation of events threatening actors' assets for the two categories is shown in Figure 4.16. We provide two examples of events threatening actors' assets in Figure 4.17, one for each category: (i) the event *specialised physician sick* threatens the goal *transfusion performed via specialist* (Figure 4.17a), as a specialised physician is selected to perform the transfusion procedure to the patient; (ii) the event *test results lost* threatens the document *test results* produced by ModernLabs (Figure 4.17b). This event affects ModernLabs in providing a timely service to its patients, in this scenario to Alice.

A key component of information security is to conduct risk assessment, which enables identifying events and the threatened supporting assets. In STS-ml, we assume that the represented events threatening actors' assets and the identification of the assets they threaten, are the result of risk analysis (following the *identification phase* of some risk analysis method [Tixier et al., 2002]), which is out of the scope of this work.

As we will show in Section 6.2.2, STS offers stakeholders a way to verify how the identified events threaten the rest of their assets, while leaving them the choice among CORAS [Lund et al., 2010], OCTAVE [Albert and Dorofee, 2001], or any other risk analysis methodology that better suits their needs. In this way, STS-ml does not impose a specific method in particular to security engineers, but advocates the usage of a method that allows the systematic identification of events and threatens relationships.



(a) On goals



(b) On documents

Figure 4.17: Some threats affecting actors' supporting assets is STS-ml

4.5 Specifying security requirements in STS-ml [3]

The main purpose of STS-ml is to represent and capture through its models the security needs expressed by the stakeholders of the socio-technical system (represented through the modelled actors) to then derive security requirements for the system to be. As shown earlier in this

[3]STS-ml supports a rich set of security requirements types, illustrated in Figure 4.18. We use the term "security requirements" for brevity, but we mean "security requirements types".

chapter, a preliminary step to do this is to represent the structure of a socio-technical system in terms of actors, their goals, documents and information, as well as to represent actors' social interactions.

Guided by Principle 7, we would like our language to capture security requirements using a terminology that security experts are already familiar with. As already discussed in Chapter 3, our classification consists of the following six aspects of security: *Confidentiality*, *Integrity*, *Availability*, *Authenticity*, *Reliability*, and *Accountability*. Starting from these core aspects, STS-ml supports the security requirements types in the taxonomy of Figure 4.18.

The final security requirements (end nodes) are the result of what STS-ml supports, in terms of operations one can perform over information (through documents) and actions one can execute with respect to the social interactions it is involved. Moreover, they are the result of interactions and suggestions received by practitioners with whom STS-ml has been evaluated [4] and improved through the course of its development.

We do not claim that this classification is a standard one, rather it is useful for illustrative purposes, to guide security engineers in selecting the appropriate security requirements when representing stakeholders' security needs. Additionally, this classification aims to be comprehensive, and as a result, not all security requirements types are applicable to all domains. The security engineers should evaluate whether the given security requirements adequately capture stakeholders' security needs.

Note that in STS-ml there is a one-to-one mapping of security needs and security requirements, for the latter are derived from the modelling of the former. That is, each security need expressed by an actor to another results in a security requirement from the second actor to the first for the satisfaction of the security need. Security requirements in STS-ml are specified as relationships among a *requester* actor and a *responsible* actor for the satisfaction of a *security need*. Security needs are expressed through the various security requirements types, and thus, determine the type of security requirement to be satisfied by the responsible actor.

An important advantage of having the classification of security requirements types is that it facilitates (aided by automated analysis techniques) answering questions such as: "*Are there any violations of donors' confidentiality requirements?*", "*Are Alice's interactions reliable?*", "*Is the integrity of patients' medical history preserved?*", etc.

In the following sections 4.5.1 to 4.5.6 we describe in detail what each security aspect means, explain the security requirements types for each of the considered aspects, and provide examples from the healthcare scenario to illustrate how these security requirements types are captured in STS-ml. Given the one-to-one mapping between security needs and security requirements types, for simplicity we explain directly the security requirements specified with STS-ml.

[4] More information on evaluation activities will be provided in Chapter 9

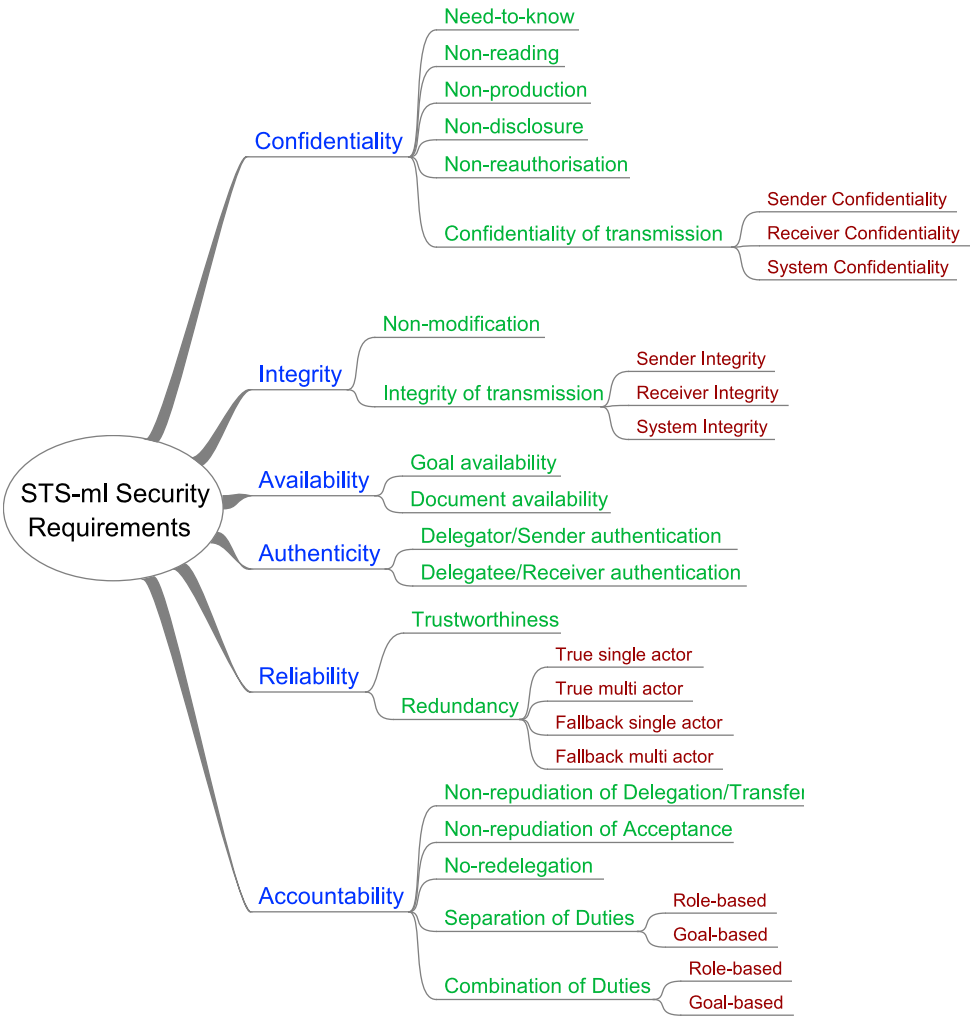


Figure 4.18: Security requirements types supported by STS-ml

4.5.1 Confidentiality

In this work, the security aspect of confidentiality encompasses both information confidentiality and privacy [Stallings and Brown, 2008]. Confidentiality assures that private or confidential information is not made available or disclosed to unauthorised users. Moreover, it assures that a stakeholder specifies what information related to her or him may be collected (used), and to whom that information may be disclosed.

An example of this category of requirements concerns sensitive information such as one's credit card number. Typically, one would like to ensure that only authorised people can access it (and for specific purposes). Additionally, one would like to be the person that specifies these authorisations.

In STS-ml, confidentiality requirements are expressed through authorisation relationships.

The language supports the following security requirements types about confidentiality (illustrated in the example of Figure 4.19): need-to-know, non-reading, non-production, non-disclosure, non-reauthorisation, and confidentiality of transmission. We discuss each in detail below.

Need-to-know

This requirement is derived from the need-to-know principle, which states that one shall have access to and rights about only the information that is necessary to accomplish one's tasks. In STS-ml, this requirement is expressed by an actor when it grants an authorisation to another actor, and the purpose of this authorisation is not empty. In other words, the authoriser does restrict the permissions that are granted to the authorisee, to one or more goals. Thus, graphically a need-to-know security requirement is expressed anytime the goal scope of an authorisation is not empty.

The authorisee can perform the permitted operations over documents representing that information only in the specified purpose, but not for achieving other goals. The prohibitions relate only to the specified goals too.

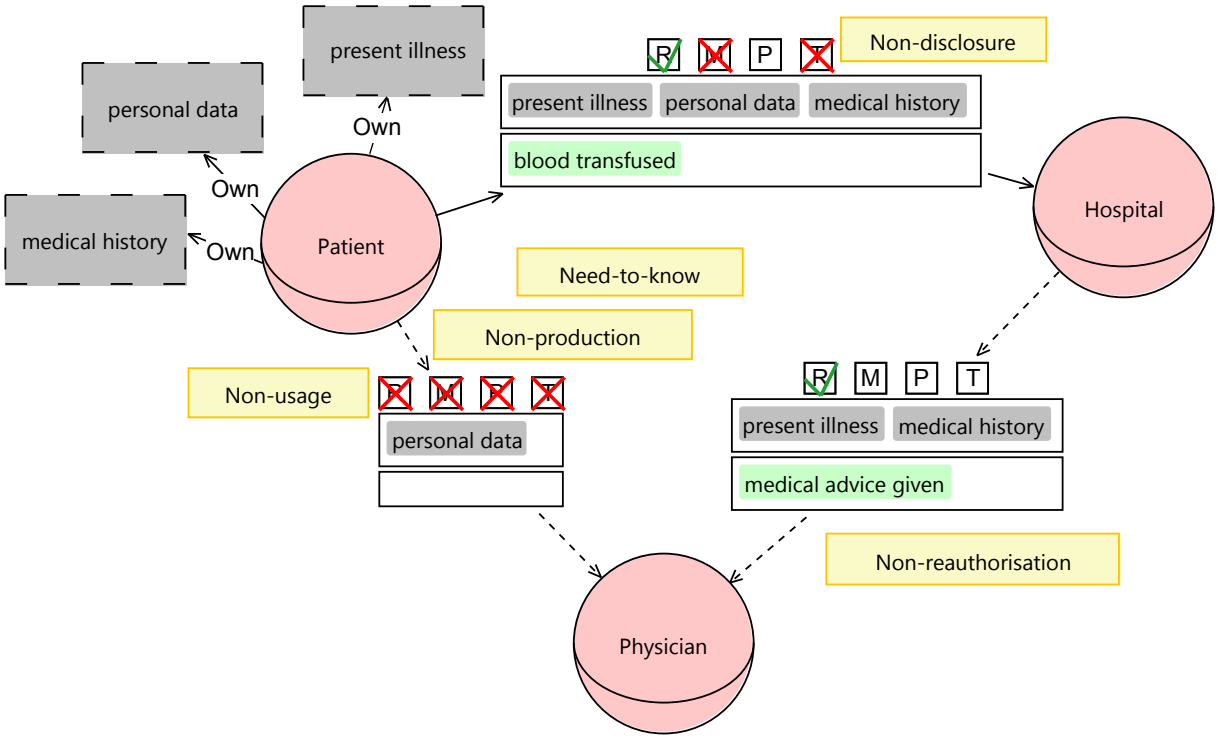
Notice that, when determining the purpose for which the permissions and prohibitions are specified, we consider not only the very goals that are in the authorisation purpose (scope), but also all their descendants in the actor model of the authorisee. For instance, if the authorisation is given on goal blood collected, and this goal is and-decomposed as in Figure 4.9, the authorisation applies to the subgoals donor approved and blood examined too.

For instance, in Figure 4.19a, the *Patient* requires the *Hospital* to ensure need-to-know over information *present illness*, *personal data*, and *medical history* in the scope of goal *medical advice given*. This means that the *Hospital* can perform the allowed operations (only read) to achieve goal *medical advice given* (including its subgoals and descendants), but not for achieving other unrelated goals. Similarly, the prohibition to modify and disclose the same information elements is restricted to the goal *medical advice given* and its descendants in the actor model of the *Hospital*.

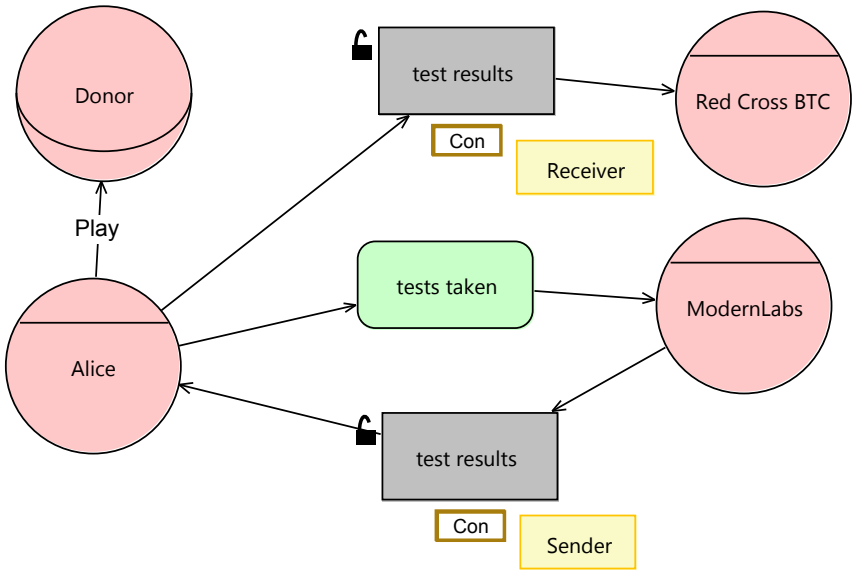
Non-reading

This confidentiality requirement indicates that the authoriser wants the authorisee not to read the information in the expressed authorisation. Non-reading is specified whenever the authoriser prohibits the read operation to the authorisee. This means that the authorisee shall not read an documents representing the given information. Graphically, this is expressed with a cross symbol over the reading (R) operation of the given authorisation relationship.

In Figure 4.19a, the patient expresses a non-reading requirement on her personal data towards the physician, given that the authorisation from the patient to the physician on infor-



(a) Non-reading, non-production, non-disclosure, non-reauthorisation, need-to-know



(b) Confidentiality of transmission

Figure 4.19: Confidentiality security requirements in STS-ml

mation personal data prohibits the read operation. Notice that the very same authorisation expresses other requirements as well that we introduce later in this section: non-distribution,

non-reauthorisation, non-modification, and non-production.

Non-production

This requirement indicates that the authoriser wants the authorisee not to produce any documents that contain any of the information for which authorisation is granted. Non-production is expressed whenever the authoriser prohibits the produce operation to the authorisee. Graphically, this is expressed with a cross symbol over the production (P) operation of the given authorisation relationship.

The produce operation is equivalent to copying the information, since it considers rewriting the information into a new document. Making unauthorised copies of information results in a threat to confidentiality [Stallings and Brown, 2008], therefore this requirement is classified under the confidentiality principle.

In Figure 4.19a, for example, the patient expresses a non-production requirement on her *personal data* to the *Physician*, by prohibiting the produce operation.

Non-disclosure

Non-disclosure is an important confidentiality requirement that considers unauthorised disclosures of information. In STS-ml, non-disclosure is expressed by an authoriser to indicate that information shall not be disclosed in an unauthorised way by the authorisee. Given that in STS-ml information can be disclosed by transmitting some document that represents the said information, this requirement is expressed whenever the authoriser prohibits the transmit operation to the authorisee. To comply with this requirement, the latter actor shall not transmit any document that contains the specified information to unauthorised actors.

Graphically, this is expressed with a cross symbol over the transmission (T) operation of the given authorisation relationship. In Figure 4.19, for instance, the *Patient* requires the hospital not to disclose information *present illness*, *personal data*, or *medical history*. Similarly, as anticipated in the section related to non-reading, the *Patient* requires the *Physician* not to disclose her *personal data*.

Non-reauthorisation

This requirement indicates that the authoriser wants the authorisee not to redistribute the received permissions to other actors. If the authorisee receives an authorisation that contains only prohibitions, then not-reauthorisation cannot be specified. An authorisee is subject to this requirement in two cases:

- explicitly, when the authorisee receives an authorisation that is non-transferable (graphically, when the line connecting the authoriser and the authorisee is dashed);

- implicitly, when no actor specifies permissions or prohibitions for performing a certain operation on a given information.

Note that the information owner has all permissions (all operations are allowed) over its own information, and prohibitions over this information do not apply to such actor. Information owners are the legitimate actors to grant authorisations to others in the socio-technical system.

Graphically, an explicit non-reauthorisation requirement is expressed with a dashed arrow line for the given authorisation relationship, to indicate that transferibility is false and authorisation chain should end with the authorisee. Implicit non-reauthorisation, on the other hand, is a result of postprocessing performed over the modelled authorisation relationship, and as such does not have a distinguishable graphical notation.

In Figure 4.19a, an example of explicit expression of this requirement is the authorisation from *Hospital* to *Physician* (dashed arrow line) for information elements *present illness* and *medical history*. Moreover, if we removed the authorisation (including only prohibitions) from *Patient* to *Physician*, we would have an example of implicit specification, because no actor would have granted permission to the physician to read the *Patient*'s *personal data*.

Notice the key difference between the explicit and the implicit cases. In the former case, an actor wants to ensure that another cannot do specific operations on some information. In the latter case, the lack of permissions is a temporary situation, which could be changed by any actor having permission and authority to transfer such permission, if they used this authority and specified an authorisation that passed such permission. For example, in Figure 4.19a, removing the authorisation between *Patient* and *Physician* would allow the hospital to pass its permission on *personal data* to the *Physician*.

We provide a thorough discussion on conflicting authorisations, their identification, and resolution in Chapter 6.

Confidentiality of Transmision

This requirement indicates that the confidentiality of information shall be preserved while it is transmitted from one actor to another. In STS-ml, information is transmitted through document transmission. The burden of ensuring confidentiality of transmission may affect either the sender, the receiver, or the socio-technical system's infrastructure:

- *Sender Confidentiality*: the sender shall ensure the confidentiality of transmission for the given document is preserved. Thus, the requirement is expressed by the receiver to require that the sender ensures the confidentiality of transmission of the document being transmitted. In Figure 4.19b, Alice requires ModernLabs to ensure the confidentiality of transmission of document *test results*.

- *Receiver Confidentiality*: the receiver shall ensure the confidentiality of transmission for the given document is preserved. Thus, the requirement indicates that the sender requires the receiver to ensure the confidentiality of transmission of the document being transmitted. In Figure 4.19b, Alice requires Red Cross BTC to ensure the confidentiality of transmission of document *test results*.
- *System Confidentiality*: the system shall ensure that the confidentiality of transmission of a document in transit is preserved. This indicates that the requirement is imposed by the socio-technical system's infrastructure itself deeming the document as important to preserve confidentiality while being transmitted.

Graphically, in Figure 4.19b, the transmission of document test results from Alice to Red Cross BTC and that from ModernLabs to Alice are annotated with a padlock and with a small rectangle with label "conf". The former annotation indicates that a security requirement applies to the document transmission (as we will see later, multiple requirements can apply to the same element). The latter annotation indicates the type of requirement: "conf" stands for "confidentiality of transmission".

4.5.2 Integrity

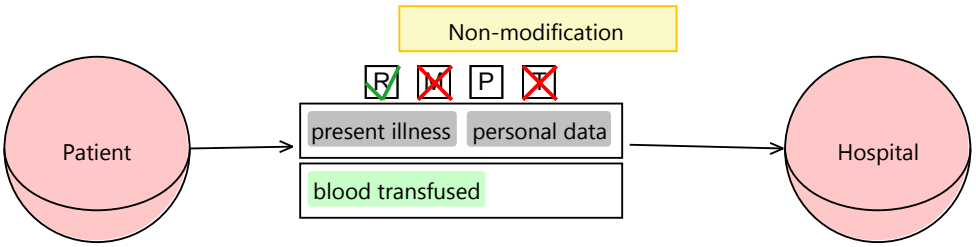
The security aspect of integrity requires assurance that information is not changed (modified) or destroyed in an unauthorised way [Stallings and Brown, 2008]. Notice that, when one refers to informational entities, it is very challenging to identify a deletion, for an information can be made tangible by many documents, and copies of these documents. Therefore, in STS-ml, we focus only on the unauthorised modification of documents that make some information tangible. In STS-ml, the following security requirements fall into the category of the integrity security principle (see Figure 4.20 illustrating them): non-modification and integrity of transmission. We discuss each in detail below.

Non-modification

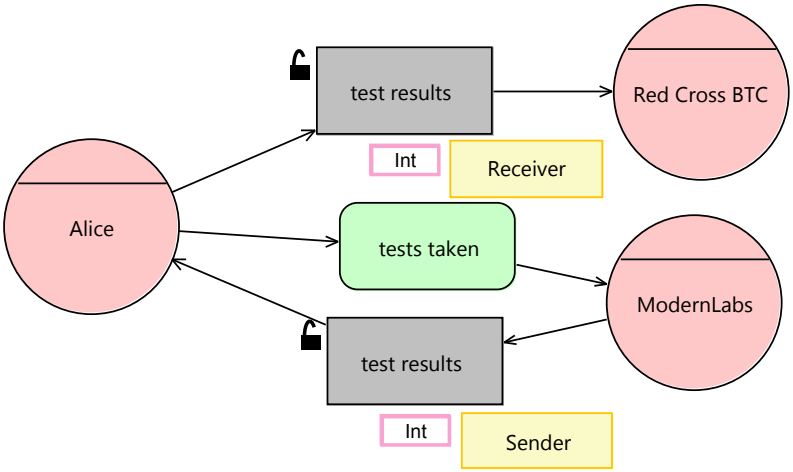
This integrity requirement requires that information is not modified in an unauthorised way. The requirement is expressed through authorisations and specifies that the authoriser wants the authorisee not to modify any document that makes tangible the information in the authorisation in the context of the specified purpose.

For instance, in Figure 4.20a, the *Patient* requires the *Hospital* that information *present illness* and *personal data* are not modified. The same requirement is expressed by the patient on the physician.

A non-modification requirement may seem over-restrictive: how can one model the fact that only the sections of the document that contain that information shall not be modified? The



(a) Non-modification



(b) Integrity of transmission

Figure 4.20: Integrity security requirements in STS-ml

part-of relationship between document is the answer. One can structure a document into sub-documents (e.g., header, body, signature), each being part of the overall document. By relating information to a specific part, the other parts can be modified without affecting the integrity of the information at hand.

Integrity of transmission

This requirement indicates that some information shall not get corrupted while it transits from one actor to another. In STS-ml, information is transmitted through document transmission. Integrity of transmission might be required by the sender, the receiver, as well as by the system (meaning the socio-technical system). Therefore, the requirement is expressed as a constraint on document transmissions , and is specialised into:

- *Sender Integrity*: the sender shall ensure the integrity of transmission for the given document. Thus, the requirement is expressed by the receiver to require that the sender ensures the integrity of transmission of the document being transmitted. In Figure 4.20b, Alice requires ModernLabs to ensure the integrity of transmission of document *test results*.

- *Receiver Integrity*: the receiver shall ensure the integrity of transmission for the given document is preserved. Thus, the requirement indicates that the sender requires the receiver to ensure the integrity of transmission of the document being transmitted. In Figure 4.20b, Alice requires Red Cross BTC to ensure the integrity of transmission of document *test results*.

- *System Integrity*: the system shall ensure that the integrity of transmission of a document in transit is preserved. This indicates that the requirement is imposed by the organisation itself deeming the document as important to preserve integrity while being transmitted.

The transmission of document test results from Alice to Red Cross BTC and that from ModernLabs to Alice are annotated with a padlock and with a small rectangle with label "Int", which indicates the type of requirement: "Int" stands for "Integrity of transmission".

4.5.3 Availability

Availability is the property that the system works promptly, service is not denied to authorised users, and timely and reliable access to and use of information is ensured [Stallings and Brown, 2008]. STS-ml does not support deletion of information to consider complete denial of service or denial of access to information. However, in STS-ml we consider potential problems in the communications between delegators and delegatees, as well as among senders and receivers of information affecting availability of goals and documents respectively. Hence, STS-ml supports two types of availability, namely document availability and goal availability.

Document availability

This requirement indicates that the actor that possesses the document has to ensure a certain level of availability to some actor that needs the document. As such, this requirement is expressed over document transmissions, and it requires the sender to guarantee an availability level expressed in percentage for the document being transmitted to the receiver. For instance, in Figure 4.21, the hospital authority requires hospitals to guarantee an availability of 99.9% for the document *registration record*, which is created whenever a patient is hospitalised.

Availability requirements are graphically expressed by annotating document transmissions with a small rectangle labeled "Ava", which stands for "Availability". Notice that, in Figure 4.21, the box showing the availability level is not part of the graphical notation. As we will show in Chapter 7, some details of the expressed requirements are shown through the STS-Tool's property view.

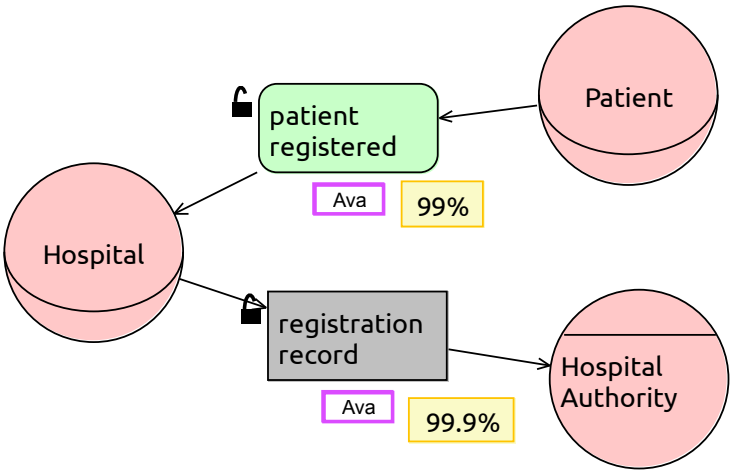


Figure 4.21: Availability security requirements in STS-ml

Goal availability

This availability requirement is expressed by a delegator to indicate that a minimum level of availability shall be provided by the delegatee of a given goal. Thus, it is expressed over goal delegations. Its graphical syntax is the same as for document availability.

For instance, in Figure 4.21, the patient request to the hospital an availability level of 99% for the delegated goal patient registered, meaning that the hospital shall guarantee that patient registration shall be ensured in 99% of the cases. Expressing an availability level is often required to accommodate unpredictable circumstances, e.g., too many concurrent patients showing up at the hospital the same day.

Notice that goal availability is highly related to the notion of service availability, where a provider specifies an uptime level for the service [5]. In service-oriented settings, availability levels often become integral part of service-level agreements between providers and consumers.

4.5.4 Authenticity

The security aspect of authenticity is the property of being genuine and being able to be verified and trusted [Kissel, 2011]. Authenticity is ensured through authentication processes and mechanisms that aim at verifying whether users are who they say they are (entity authenticity [Stallings and Brown, 2008]). This aspect of security is sometimes treated as part of integrity [PUB, 2004], although many recognize its value to be treated separately from the others [Stallings and Brown, 2008]. In STS-ml, we treat authenticity as part of security requirements

[5]The interaction between a delegator and a delegatee is similar to that of a service consumer (represented by the delegator) and a service provider (represented by the delegatee) on using/providing a service (represented by the goal).

it supports, in order to provide a comprehensive account on security requirements engineering.

In STS-ml, the authenticity requirement applies over actors' interactions related to their assets, namely *goal delegations* and *document transmissions*. We, then, specialise authenticity into two sub-requirements: the former relates to the authentication of the delegator/sender, the latter is concerned with the authentication of the delegatee/receiver.

Given a goal delegation or document transmission, one can express either delegator/sender authentication, or the delegatee/receiver authentication, or both. Graphically, authenticity requirements are expressed as an annotation of a goal delegation/document transmission with a small rectangle labeled "Auth", which stands for "Authenticity". The details concerning the exact type of the chosen authenticity requirement are not shown graphically, so as to keep the graphical notation less heavy.

Delegator/Sender Authentication

This requirement about authenticity indicates the delegatee/receiver's request that the delegator/sender shall be authenticated. This is the kind of authentication that is typically implemented in electronic commerce websites, wherein a certification authority guarantees the authenticity of the seller's website.

In the healthcare scenario, the hospital authority expresses the requirement that the transmission of document *registration record* necessitates the sender's (*Hospital*) authentication, see Figure 4.22. Moreover, the delegation of goal *tests taken* from *Alice* to *ModernLabs* includes a delegator authentication requirement: *ModernLabs* wants to ensure that the test is requested by the same person that will be tested.

Delegatee/Receiver Authentication

This type of the authenticity requirement expresses the delegator/sender's need that the delegatee/receiver is authenticated. We encounter this kind of authentication everyday when we browse the web and use our credentials (username/password) to access web information such as our email.

In the healthcare scenario, patients impose this requirement on the delegation of goal *patient registered* to *Hospital*, for they want to be certain that registration is performed by authorised hospital staff, see Figure 4.22. Also, the transmission of the *donor certificate* from *Red Cross BTC* to the *Donor* includes this requirement, which indicates that the given donors themselves are the only individuals (each his/her own) who can receive the certificate.

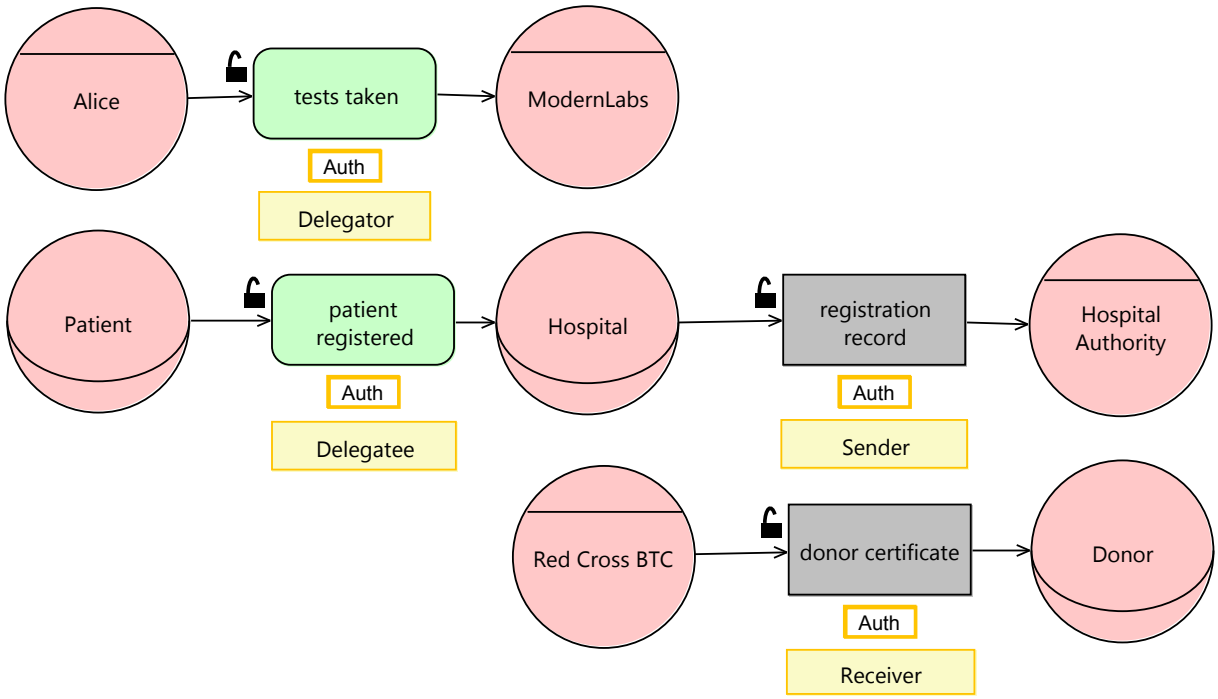


Figure 4.22: Authenticity security requirements in STS-ml

4.5.5 Reliability

Reliability is an aspect of security that is concerned with addressing the consequences of accidental errors [Gollmann, 2011]. In the age of the Internet, however, the notion of accident encompasses non-designed usages, including attackers trying to misuse the system. While reliability is sometimes treated independently from security, we include it in our supported aspects, with the intent of providing a comprehensive approach. In STS-ml, reliability is supported through the following requirements:

Trustworthiness

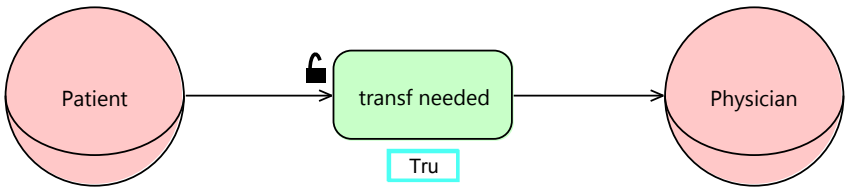


Figure 4.23: Reliability security requirements in STS-ml: Trustworthiness

It lays down a requirement on the trustworthiness of the actor one has to rely upon. It is

expressed over goal delegations by the delegator, which requires the delegatee to be trustworthy, i.e., the goal will be delegated only to trusted delegatees. This requirement implies that the delegatee shall provide a proof of trustworthiness, e.g., issued by a certification authority. For instance, in the healthcare scenario (Figure 4.23), the *Patient* imposes a trustworthiness requirement on the *Physician* with respect to *transf needed* (performing a transfusion procedure when needed).

Redundancy

This requirement is expressed on goal delegations by the delegator, who wants the delegatee to adopt redundant strategies for a delegated goal, either by using alternative internal strategies (single actor), or by relying on other actors (multi-actor). We consider two types of redundancy requirements:

1. *Fallback redundancy*: a primary strategy shall be selected to fulfil the goal, while at least another strategy shall be deployed as backup, and shall be used only if the primary strategy fails.

2. *True redundancy*: two or more different strategies shall be executed simultaneously by the delegatee to fulfill the goal.

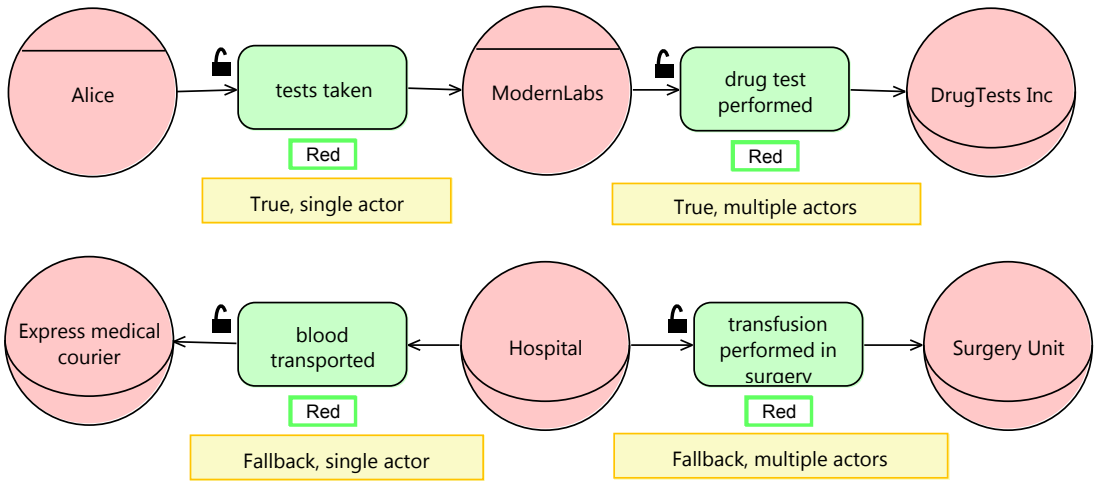


Figure 4.24: Reliability security requirements in STS-ml: Redundancy

By intertwining single/multi-actor with the redundancy types (1-2), STS-ml supports four mutually exclusive redundancy security requirements:

1. *True redundancy single actor* indicates that one actor has to deploy concurrent means for achieving the delegated goal. For example, in Figure 4.24, *Alice* requires *Modern-*

Labs to provide true redundancy for goal *tests taken*, e.g., testing the blood through two instruments.

2. *True redundancy multi actor* specifies that the delegatee shall ensure that more than one actor (possibly, including the delegatee herself) shall act concurrently towards the achievement of the delegated goal. In Figure 4.24, *ModernLabs* requires *DrugTests Inc* that concurrent ways for performing drug tests (goal *drug test performed*) are used, and multiple actors are involved.

3. *Fallback redundancy single actor* means that the same actor has to provide a fallback solution, in case the original one fails. In Figure 4.24, the *Hospital* requires the express courier to deploy a fallback solution for goal *blood transported*. A possible way to do so is to reserve a backup van, which is used only if the designated one is not working.

4. *Fallback redundancy multi actor* expresses that the delegatee shall ensure the involvement of multiple actors towards the achievement of the delegated goal. In Figure 4.24, the *Hospital* specifies this requirement to the surgery department for goal *transfusion performed in surgery*. This means that more than one physician shall be ready to perform the transfusion: one is the designated surgeon, the other ones are backup options, should the first one become unavailable.

Given that redundancy requirements also are expressed over goal delegations, they too are graphically expressed as an annotation of goal delegations. A small rectangle labeled "Red", which stands for "Redundancy" is used in this case, see Figure 4.24 showing a snippet from the healthcare scenario, illustrating the use of redundancy requirements. The details concerning the exact type of the chosen redundancy requirement are not shown graphically, in order to keep the graphical notation less heavy. The labels below the annotation "Red" are added to Figure 4.24 for illustrative purposes. As we will see in Chapter 7, these details are inserted through the property view of STS-Tool.

4.5.6 Accountability

The security aspect of accountability refers to the requirements for actions of an entity to be traced uniquely to that entity [Kissel, 2011]. STS-ml supports expressing accountability security requirements in different ways, as shown in the following sub-sections.

Non-repudiation

A key requirement related to accountability is non-repudiation, i.e., preventing either of the entities involved in a communication (the origin or the destination) from denying having participated in the communication [Stallings and Brown, 2008]. In STS-ml, communication among

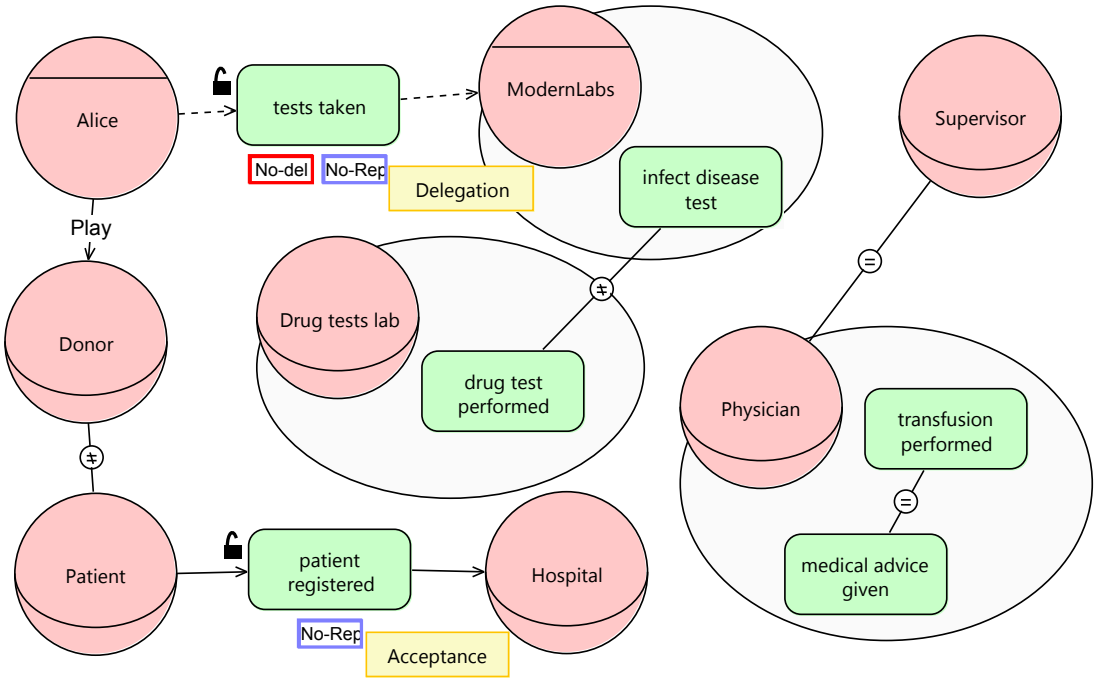


Figure 4.25: Accountability security requirements in STS-ml

stakeholders could be one of: document transmission and goal delegation. Thus, a requirement for non-repudiation is expressed with respect to transmitting information (document transmission), as well as with respect to delegating goals (goal delegation, which transfers responsibility for goal fulfilment).

Graphically, the requirement is represented as an annotation for delegations with label "No-rep", which stands for "non-repudiation". Again, we keep the annotation lightweight, without providing the details of the exact type of non-repudiation being expressed on the label over the goal delegation itself [Moody, 2009].

This requirement is specified to prevent the two interacting actors from denying having transferred the document/delegated the goal (corresponding to non-repudiation—origin) or received the document/accepted the delegation (corresponding to non-repudiation—destination) respectively. Origin and destination is security refer to the two ends of a communication. In STS-ml, we consider the two parties involved in goal delegations and document transmission respectively. Therefore, non-repudiation is specialised into these security requirements:

- *Non-repudiation of acceptance*: this indicates that the delegator/sender requires the delegatee/receiver not to repudiate the delegation/transfer of a goal/document. In Figure 4.25, the patient requires the hospital not to repudiate the acceptance of the delegation of goal patient registered.

- *Non-repudiation of delegation/transmission*: this is expressed by the delegatee/receiver to require that the delegator/sender does not repudiate the delegation/transmission of the goal/document. In Figure 4.25, ModernLabs requires Alice not to repudiate that she has delegated goal tests taken.

Not-redelegation

This requirement is expressed over goal delegations, and it is the delegator's request for the delegatee to take full responsibility of achieving the delegated goal, without relying on any other actor. The delegatee shall therefore avoid delegating the goal or any of its subgoals, should there be any. Graphically, the requirement is represented as an annotation for delegations with label "No-del", which stands for "not-redelegation".

A main reason for specifying a not-redelegation requirement concerns trust: the delegator trusts that specific delegatee for the given goal, but does not trust other actors the delegatee might want to involve. However, for the time being, in STS-ml we do not explore the interrelations between trust and security requirements. In Figure 4.25, Alice requires ModernLabs not to redelegate goal tests taken to other actors, such as third-party labs or technicians.

Separation of Duties

This accountability requirement expresses segregation of duties among different people; this is especially important when dealing with critical tasks. An example of this requirement is the procedure for opening a bank's vault, which often requires the joint effort (and information) of more than one person.

Graphically, this is represented as an arrow between two entities annotated with the "different" ($\neq$) symbol, see Figure 4.25; note that the relationship is symmetric, therefore there are no arrows pointing to the concepts it relates.

Specifically, separation of duties (SoD) comes in two versions in STS-ml:

- *Role-based SoD* defines that two roles are incompatible, i.e., when specified between the two roles, it does not allow the same agent to play both the roles. For example, in Figure 4.25, roles patient and donor are incompatible: if Alice is a donor, she cannot be a patient (blood receiver) at the same time.

- *Goal-based SoD* defines incompatible goals, i.e., an actor should not pursue and achieve both goals among which SoD is defined. For example, in Figure 4.25, ModernLabs' goal to test for infection diseases conflicts with Drug tests lab's goal of performing drug tests. This requires that different actors shall be responsible for these two tests.

Combination of Duties

This requirement originates from the *retain familiar* [Russell et al., 2004] principle, which is symmetric to separation of duties. Combination of duties expresses that some entities should be ascribed to the same agent, if one of them is already ascribed to that agent, considering the familiarity the actor has with the entity. Graphically (see Figure 4.25), this is represented as an arrow between two entities annotated with the "equal" (=) symbol, the relationship is symmetric, therefore there are no arrows pointing to the concepts it relates.

Combination of duties (CoD) comes in two versions in STS-ml:

- *Role-based CoD* defines a binding between roles, i.e., if an agent adopts either of the roles, it has to adopt (play) also the other role. In Figure 4.25, a role-based combination of duties requirement is expressed over roles supervisor and physician, i.e., a physician also has to supervise trainees in the hospital.
- *Goal-based CoD* defines that if an agent achieving one of the goals among which CoD is defined, it should achieve the other goal too. In Figure 4.25, the physician role includes a goal-based CoD for goals transfusion performed and medical advice given. This means that the same physician has to be responsible for both goals (e.g., in order to increase the patient's trust about the physician/hospital).

4.6 Chapter Summary

In this chapter, we have presented the modelling primitives of STS-ml. These constructs allow for expressing the stakeholders in a socio-technical systems, their objectives, and the security requirements that they want other stakeholders to comply with.

We have shown the security requirements that the language supports, and we have classified them in accordance with a taxonomy of aspects of security, which consists of confidentiality, integrity, availability, authenticity, reliability, and accountability.

The language complies with the ten principles that we deem important for security requirements in socio-technical systems:

1. A socio-technical perspective is provided by modelling actors (social and technical) that interact via goal delegations and document transmissions. This view follows the definition of socio-technical systems, as an interplay of social and technical subsystems, represented in STS-ml via actors.
2. The existence of multiple stakeholders is acknowledged by allowing every stakeholder to express requirements. The requirements supported by STS-ml derive primarily from stakeholders' needs.

3. Modelling relies upon high-level representation of assets, modelling actors' goals, information, and documents.

4. Security is imposed as constraints that affect different types of interactions: goal delegations, document transmission, and authorisations. This reflects the intuition that actors are concerned about their objectives (goals) and their information (represented via documents, usage dictated by authorisations).

5. The language focuses on security needs, and not on the mechanisms for fulfilling these needs. STS-ml captures stakeholders' concerns about protecting their objectives and their information while in the socio-technical system, without jumping to technical security mechanisms.

6. Representing events and how these threaten supporting assets is essential part of the language.

7. The supported security requirements are classified and presented following a taxonomy derived from existing standards. This categorisation is very important to guide requirements engineers in correctly modelling stakeholders' security needs to then derive security requirements, based on the types defined in the taxonomy.

8. Modeling is supported through diagrams (see Chapter 5), which have a formal semantics as it will be shown in Chapter 6. Modelling helps capture stakeholders' needs, however, in order to verify specific properties over the models, they need to be equipped with a formal semantics.

9. Minimality of concepts is ensured by carefully avoiding overlaps between the entities and relationships. This is important to avoid confusions while using STS-ml.

10. Security requirements can be traced back to their requester and to the goal, document, or information that originates the requirement.

This chapter provides the foundations for using the STS-ml language.

Chapter 5

Social, Information, and Authorisation Models

We show how the concepts and relationships described in Chapter 4 can be used to build STS-ml models, thereby capturing security requirements for the system at hand. Specifically, we will show what concepts and relationships are used for each and every model supported by the STS methodology as described in Chapter 3, namely the *social model* (see Section 5.2), the *information model* (see Section 5.3), and the *authorisation model* (see Section 5.4). Moreover, we will describe how to express stakeholders' security needs and capture security requirements through the three models. These models are complementary and, together, form the STS-ml model for the system-to-be. Inter-model consistency is ensured by STS-Tool (details in Chapter 7).

Acknowledgement. This chapter revises and extends [Paja et al., 2013b; Dalpiaz et al., 2011].

5.1 Multi-view modelling approach

As discussed in Chapter 3, STS methodology supports multi-view modelling, that is, STS-ml models are constructed by focusing on different views at a time. The name *multi-view modelling* derives from the fact that modelling activities supported by STS-Tool are performed over three different views in line with the major phases of the STS methodology. The creation of the models over three views results in three outcome (sub)models, namely the *social model*, the *information model*, and the *authorisation model*. Following this intuition, STS-ml primitives are combined into three complementary models, which together form the system model.

The division of the modelling activities over three different views (resulting in three models) follows the separation of concerns principle. The phylosofy of STS methodology requires the modelling of social and organisational aspects, in which we represent actors with their ob-

jectives and interactions with others, separately from the informations they own and want to protect, from the permissions and prohibitions flow. This separation facilitates also the collaboration of requirements analysts and security engineers. We describe in the following sections what primitives and relationships are used to create each model. We show the three supported models, but no complete model, because the overall STS-ml model of the system-to-be is only conceptually maintained in STS to ensure inter-model consistency with STS-Tool (details in Chapter 7).

5.2 Social model

The *social model* represents the stakeholders and their interactions. Stakeholders are represented via actors that are intentional—they have objectives they aim to attain—, and social—they interact with others to achieve their objectives.

Therefore, the social model allows to represent actors together with their intentional assets (goals) and documents, the representation of which results in constructing actor models. Apart from representing the internal rationale of the various identified actors, the social model allows to capture the different interactions each and every actor enters (if applicable). These interactions could be either to fulfill their objectives (by *delegating goals*) or to obtain information (by *exchanging documents*).

These concepts and relationships have been already introduced and extensively explained in Chapter 4, here we present which of the supported concepts and relationships are used to build the social model.

To facilitate understandability of the models we will create, we summarise the primitives in the social model in various tables:

- in Table 5.1 we present the main concepts and intentional relationships; [1])
- in Table 5.2 we present the social relationships supported by the social model, namely play, goal delegation and document transmission; and finally
- in Table 5.3 we present the event concept and the threatens relationship that allow threat modelling.

Figure 5.1 depicts a partial social model for the motivating case study. Below, we explain and illustrate the component parts of this model for each of the above concepts and relationships (Tables 5.1–5.3).

[1] In Tables 5.1–5.5, for illustrative purposes, we interchangeably use the graphical syntax of role and agent to visualise an actor, for these relationships apply to both roles and agents. The plays relationships is the exception, as in this case the difference between agents and roles is crisp.

Concepts and intentional relationships. Stakeholders (system participants) are modelled via *roles* and *agents* (see Table 5.1). In the healthcare scenario we identify various stakeholders [2], such as: Alice, donors, patients, hospitals, physicians, laboratories (ModernLabs and Drug Test Inc), Red Cross BTC, etc. We model donors, patients, hospitals, physicians through the concept of role, while we model Alice, ModernLabs, Drug Test Inc, and Red Cross BTC through the concept of agent (see Figure 5.1). The reason for this is that *roles* refer to general actors that are instantiated at run time, while *agents* refer to concrete entities already known at design time. In this scenario, we do not know the particular donor, patient or physician that will take over the role, but we do know the responsibilities encapsulated in the said roles. As far as the represented agents are concerned, we know that Alice intends to become a donor (thus, Alice is modelled as an agent, while donor as a role, and we draw a plays relationship between them), and ModernLabs is the laboratory in charge of infective disease tests, while Drug Test Inc is the laboratory specialised in drug tests. Similarly, we assume that there is only one Red Cross BTC, so we consider it as known to be part of the system already at design time.

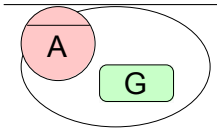
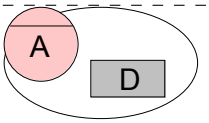
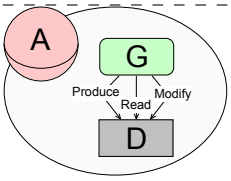
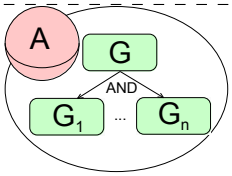
The social model supports representing stakeholders' *goals* (see Table 5.1). In the healthcare scenario, we analyse the various modelled roles and agents, to adequately represent their goals and how they intend to pursue the identified goals (considering first intentional relationships). For instance, in Figure 5.1, physicians (role Physician) intend to give medical advice to patients (goal medical advice given). To pursue this goal, physicians need to consult patients' medical records and perform a medical visit to assess current medical status. Therefore, goal medical advice given is *and-decomposed* into goal medical record consulted and goal medical visit performed. Donor has the goal of donating blood regularly (blood donat regularly). Patient has the goal of receiving medical treatment (treatment received). In the context of blood transfusion service, we consider medical treatment to be about blood transfusion. Therefore, the patient needs to get registered in order to receive treatment and the transfusion: goal treatment received is *and-decomposed* into goal patient registered and transfusion needed.

Alice has the goal blood donated, which is *and-decomposed* into test taken and neg results received. Red Cross BTC has goal blood distributed, which is *and-decomposed* into goals blood collected, blood consumption estimated, and blood transported; goal blood collected is further *and-decomposed* into goals blood examined and donor approved, while goal blood consumption estimated is *and-decomposed* into goals stat analysis performed and blood usage evaluated; finally goal stat analysis performed is *or-decomposed* into on blood type eval, on hospital requests, and on donor, see Figure 5.1. Similarly we can analyse the various goals and goal decompositions for each and every modelled actor in the social model of the healthcare scenario.

Apart from actors' goals, in the social model we can capture the documents actors possess

[2]Note that we show the modelling of stakeholders starting typically in the social model, however they could be modelled (introduced in the STS-ml model) in any of the supported models.

Table 5.1: Social model: concepts and intentional relationships

Graphical Notation	**Syntax and Description**
	Concepts
	role(R): an abstract characterization used to model a class of participants, defining a set of responsibilities for the said participants (e.g., professor, student)
	agent(A): a concrete participant known to be in the system already at design time (e.g., John, Laura)
	goal(G): represents stakeholders' objectives (e.g., exam taken)
	document(D): represents information (e.g., transcripts file)
	Intentional relationships
	wants(A, G): actor A wants to achieve goal G, models the actor's intention to achieve the goal (e.g., student wants to pass the exam)
	possesses(A, D): actor A possesses document D, models the possession of the document by an actor (e.g., professor has the exam paper)
	reads/modifies/produces (A, G, D): actor A reads / modifies / produces document D when fulfilling goal G (e.g., professor reads student exams to grade them)
	decomposes($A, G, \mathcal{S}$, DecT): A decomposes root goal G into sub-goals from $\mathcal{S}$, where $\mathcal{S}$=$\{G_1, \ldots, G_n\}$ and $\lvert\mathcal{S}\rvert \geq 2$, and the decomposition is of type DecT, such that DecT $\in$ {and, or} (e.g., a student passes the exam if she sits in for the exam and gets a grade higher than 65%)

or manipulate to achieve their goals. In Figure 5.1, Patient possesses document medical record, it reads this document to get registered at the hospital (patient registered), while it modifies personal records to achieve goal data refined. Physician modifies document blood bank when transfusion is needed (goal transfusion needed). Donor needs a donor certificate to donate blood regularly, which is represented through the read intentional relationship between goal blood donat regularly and document donor certificate. Alice reads document test results to verify if she has received negative test results. Red Cross BTC produces document report for goal blood distributed, modifies document blood bank to estimate blood consumption (goal blood consumption estimated), reads document test results to approve donors and produces document donor certificate for the same goal (donor approved), and finally it reads document blood usage listings to perform statistical analysis on hospital requests (goal on hospital requests).

Social relationships. The social model represents the social relationships between actors in the given socio-technical system. This model supports three social relationships: *play*, *goal delegation* and *document transmission* (see Table 5.2).

In the healthcare case study, we know that Alice wants to become a donor, therefore we represent this through the *play* relationship between the agent Alice and the role Donor. We do not have information about other actual participants of the healthcare socio-technical system.

An example of *goal delegation* is that of Patient delegating goal transfusion needed to Physician. This delegation results in the second actor having the goal, that is, Physician has goal transfusion needed [3], which is part of the transfusion procedure performed by a specialised physician. Goal transfusion performed via specialist is delegated to the Physician by the Hospital. Alice relies on ModernLabs to take tests, delegating goal test taken. ModernLabs delegates goal drug test performed to Drug Tests Inc, which is a laboratory specialised in drug tests. Similarly we can analyse the rest of goal delegations represented in Figure 5.1.

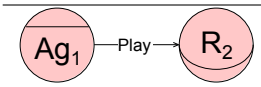
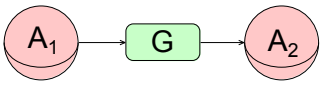
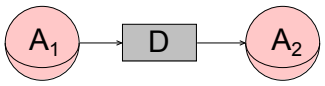
Note that in STS-ml only leaf goals (not decomposed) can be delegated.

As far as *document transmissions* are concerned, an example is that of ModernLabs transmitting the document test results to Alice, who further transmits this document to Red Cross BTC for evaluation and approval, see Figure 5.1. Red Cross BTC transmits document donor certificate to Donor, while it transmits blood bank to Hospital. The latter transmits document blood usage listings to Red Cross BTC, which needs this information to estimate blood consumption. These are just few examples of the document transmissions modelled in Figure 5.1.

Events and threats. The social model supports representing events threatening stakeholders assets. Therefore, the entity *event* and the relationship *threaten* are part of the social model,

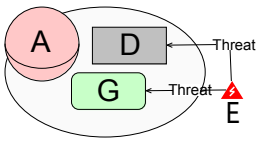
[3]Note that delegated goals are represented with a darker shade than actors' own goals to distinguish between the two types. This will become clearer in Chapter 7.

Table 5.2: Social model: social relationships

Graphical Notation	Syntax and Description
	plays(Ag_1, R_2): models the adoption of roles by agents, i.e., agent Ag_1 plays role R_2 (e.g., John plays role professor)
	delegates(A_1, A_2, G): models the transfer of responsibilities from an actor to another, i.e., actor A_1 (delegator) delegates the fulfilment of goal G (delegatum) to actor A_2 (delegatee) (e.g., professor delegates grading exams to teaching assistant)
	transmits(A_1, A_2, D): specifies the exchange of documents between two actors, i.e., actor A_1 (sender) transmits document D to actor A_2 (receiver) (e.g., teaching assistant transmits exam results to professor)

see Table 5.3. For instance, in the healthcare case study the event specialised physician sick threatens goal transfusion performed via specialist of Physician, while the event test results lost threatens document test results produced by ModernLabs (see Figure 5.1).

Table 5.3: Social model: events and threats

Graphical Notation	Syntax and Description
	event(E): represents uncertain circumstances that affect actors' assets (e.g., forgetting an important document at a shared printer)
	threaten$(E, \mathsf{wants}(A, G))$ or threaten$(E, \mathsf{possesses}(A, D))$: models the influence of an event over actors' supporting assets, i.e., event E threatens goal G or document D of actor A (e.g., disclosing confidential information to unauthorised users)

Security requirements in the social model. The social model enables specifying security requirements on the social relationships among the modelled actors. In the following, we summarise the security requirements supported by the social model from the list of STS-ml security requirements depicted in Figure 4.18, and group the set of supported security requirements per social relationship.

Over goal delegations. The following are the security requirement types expressed over goal delegations (a complete summary is in Appendix A):

1. *Non-repudiation of Delegation/Acceptance*: for instance, ModernLabs requires Alice non-repudiation of the delegation of goal tests taken, while Patient requires the Hospital non-

repudiation of acceptance of goal patient registered, see Figure 5.1.

2. *Redundancy*, in its four mutually exclusive types, true redundancy single, fallback redundancy single, true redundancy multi, and fallback redundancy multi. In Figure 5.1, Alice requires true redundancy single for goal tests taken, the Express medical courier requires to the Hospital fallback redundancy single for goal blood transported, ModernLabs requires Drug Tests Inc true redundancy multi for goal drug test performed, while the Hospital requires the Surgery Unit fallback redundancy multi for goal transfusion performed.

3. *No-redelegation*: for instance, in Figure 5.1, Red Cross BTC requires Research Center not to redelegate statistical analysis on donors.

4. *Trustworthiness*: for instance, the delegation of goal transfusion needed from Patient to Physician will take place only to trustworthy physicians, see Figure 5.1.

5. *Goal Availability*: For instance, in Figure 5.1, the Patient request to the Hospital an availability level of 99% for the delegated goal patient registered, meaning that the hospital shall guarantee that patient registration shall be ensured in 99% of the cases.

6. *Delegator/Delegatee Authentication*: in Figure 5.1, the delegation of goal tests taken from Alice to ModernLabs includes a delegator authentication requirement. The delegation of goal patient registered from Patient to Hospital , on the other hand, includes a delegatee authentication requirement.

Over document transmissions. The following are the security requirement types expressed over document transmissions (a complete summary is in Appendix A):

1. *Non-repudiation of Transmission/Acceptance*: for instance, in Figure 5.1, Hospital requires Red Cross BTC not to repudiate transmission of document blood bank, as well as the acceptance of transmission of document blood usage listings.

2. *Integrity of transmission*, in its three forms, *sender integrity*, *receiver integrity*, and *system integrity*. For instance, in the healthcare case study the Hospital Authority requires the Hospital to ensure sender integrity of document registration record, Red Cross BTC requires Research Center receiver integrity over the transmission of report, while a system integrity requirement is specified over the transmission of document report from Red Cross BTC to Hospital Authority.

3. *Document Availability*: for instance, in Figure 5.1, Hospital requires an availability level of 99.9% for the document blood bank from Red Cross BTC.

4. *Sender/Receiver Authentication*: in the healthcare scenario, in Figure 5.1, the transmission of document test results from Alice to Red Cross BTC requires sender authentication, while the transmission of document donor certificate from Red Cross BTC to Donor requires receiver authentication.

5. *Confidentiality of Transmission*, in its three forms, *sender confidentiality*, *receiver confidentiality*, and *system Confidentiality*. For instance, in the healthcare case study the transmission of document test results from Alice to Red Cross BTC requires sender confidentiality, the transmission of document test results from ModernLabs to Alice requires receiver confidentiality, while the transmission of document report from Red Cross BTC to Research Center requires system confidentiality.

Over responsibility uptake. The security requirements constraining the uptake of responsibilities, i.e., the adoption of roles and the pursuit of goals, are as follows [4] (a complete summary is in Appendix A):

1. *Separation of duties* (SoD), over roles or goals, *role-based SoD*, and *goal-based SoD* respectively. For instance, there is a separation of duties between roles Patient and Donor, while the goals infect disease tested and drug test performed are defined as incompatible, see Figure 5.1.

2. *Combination of duties* (CoD), over roles or goals, *role-based CoD* and *goal-based CoD* respectively. For instance, there is a combination of duties between roles Physician and Supervisor, while there is a goal-based combination of duties between goals medical advice given and transfusion performed via specialist, see Figure 5.1.

[4] As discussed in Chapter 4, these security requirements from organisational constraints are translated to a set of relationships, *incompatible* (represented as a circle with the unequal sign within) and *combines* (represented as a circle with the equals sign within) respectively. This is related to the fact that they are not directly expressed over a social relationship, but constrain the uptake of responsibilities of stakeholders.

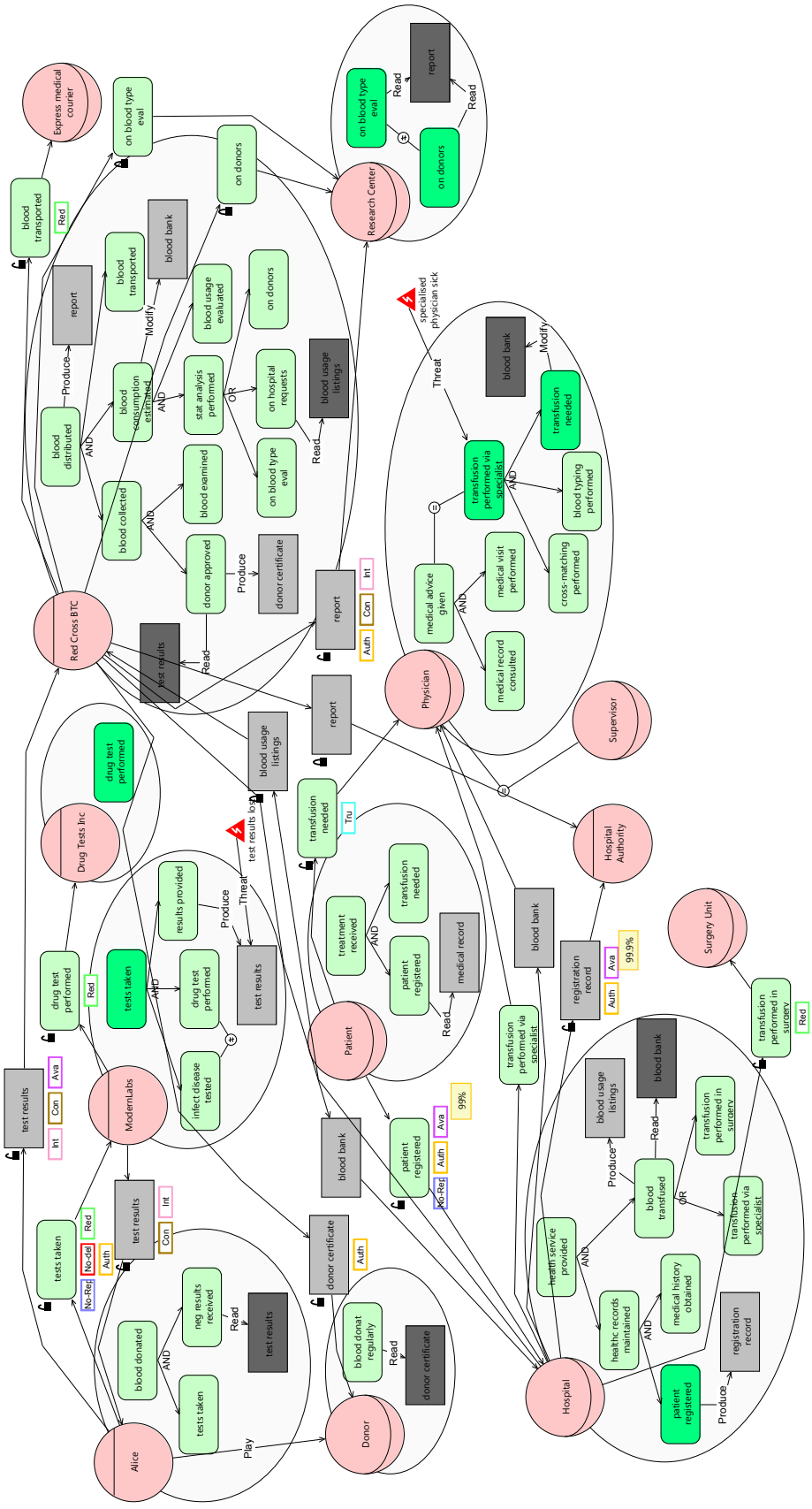


Figure 5.1: Partial STS-ml social model of the healthcare scenario

5.3 Information model

As described in Chapter 4, STS-ml distinguishes between *information*—the data that actors own, care about, and may deem confidential—and its representation via *documents*. The latter, intended in a broad sense (e.g., an email or a text message are documents too) are the means through which actors transfer information.

While the social model includes documents and their transmission, we do not know what is the informational content of the exchanged documents. This is useful in an analysis of security requirements to determine whether information was exchanged among authorised users for instance. For this, the *information model* represents the informational content of the documents in the social model. The model indicates information entities, their owners, and provides a structured representation of information and documents.

Table 5.4 summarises the concepts and relationships of the information model.

Table 5.4: Information model: concepts and relationships

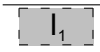
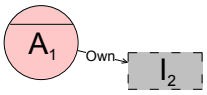
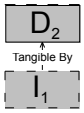
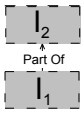
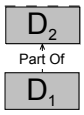
Graphical Notation	**Syntax and Description**
	information(I_1): informational entities (e.g., name, student grade)
	owns(A_1, I_2): actor A_1 is the legitimate owner of information I_2, i.e., it has full rights over that information (e.g., students own information about their personal information)
	makes-tangible(I_1, D_2): document D_2 materializes information I_1 (e.g., transcripts materialize information about course results)
	part-of-i(I_1, I_2): information I_1 is part of information I_2 (e.g., course description is part of course syllabus)
	part-of-d(D_1, D_2): information D_1 is part of information D_2 (e.g., student file is part of students registry)

Figure 5.2 represents a partial information model for the healthcare motivating case study. From the social model, we have that Patient possesses document medical record. In the information model, we can define what information is contained in this document. Looking at Figure 5.2, we can see that medical record makes tangible information personal data, which is owned by the Patient. The Patient also owns information medical history.

Similarly, Alice owns her personal information, and health status, Hospital owns information blood needs, while Red Cross BTC owns blood info. Information can be represented by one or more documents (through multiple Tangible By relationships). For instance, informa-

tion personal information owned by Alice is made tangible by both document health record and document donor certificate.

On the other hand, one or more information entities can be made tangible by the same document. For instance, health status and personal information are both made tangible by document test results of ModernLabs, see Figure 5.2.

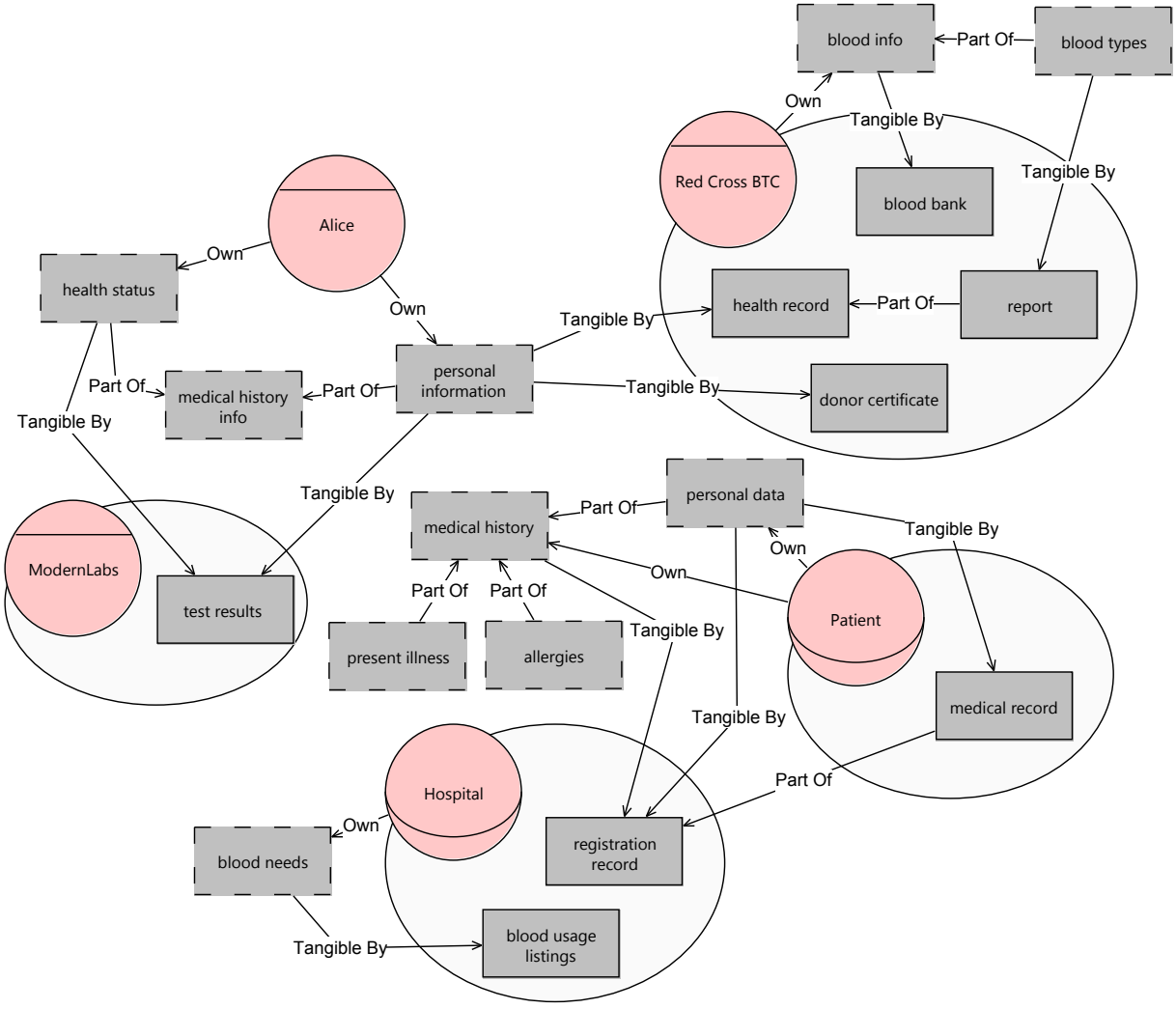


Figure 5.2: Partial STS-ml information model of the healthcare scenario

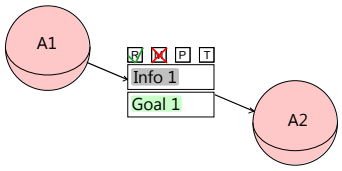
Another feature of the information model is to support composite information (documents). The structuring of information and documents is done via *part-of* relationships, allowing designers to build a hierarchy of information entities and documents, respectively. For instance, this allows representing that information health status and personal information are part of the information medical history info, information entities present illness and allergies are part of the information medical history of Patient. The same applies to documents. For instance, document

report is part of document health record in Figure 5.2.

5.4 Authorisation model

This model shows the authorisations actors grant to others over the informational entities that are represented in the information model. In STS-ml, an authorisation is a directed relationship between two actors, where one actor (*authoriser*) grants or prohibits certain rights to another actor (*authorisee*) on the usage of some information. Authorisations can be defined along four orthogonal dimensions, see Table 5.5. A partial authorisation model for the healthcare case study is shown in Figure 5.3.

Table 5.5: Authorisation Modelling: Social relationships

Graphical Notation	Syntax and Description
	authorises$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, \text{TrAuth})$: actor A_1 authorises/prohibits actor A_2 to perform operations $\mathcal{OP}$ ($\{\mathtt{R}, \mathtt{M}, \mathtt{P}, \mathtt{T}\} \cup \{\bar{\mathtt{R}}, \bar{\mathtt{M}}, \bar{\mathtt{P}}, \bar{\mathtt{T}}\}$) on the information in $\mathcal{I}$, in the scope of the goals in $\mathcal{G}$, and allows (prohibits) A_2 to transfer the authorisation to others if TrAuth is true (false); (e.g., A_1 authorises A_2 to read information Info 1 (R is checked), but prohibits modifying such information (M is crossed over), in the scope of goal Goal 1 allowing transferrability (continuous arrow line))

For instance, Alice authorises Red Cross BTC to read information health status, but prohibits modification of this information, for goal donor approved, granting a transferrable authorisation, see Figure 5.3. The authorisation relationship from Patient to Physician is the most restrictive, for it prohibits all operations and transferability of the authorisation. The authorisation from Hospital to Physician to read information present illness and medical history for goal medical advice given, but does not grant transferability of further granting permissions to other actors.

As introduced already in Chapter 4, the authorisation relationship supports a variety of security requirement types (a complete summary is in Appendix A). Here, we describe which of the supported security requirements from Figure 4.18, are expressed when modelling authorisations.

Implicitly express security needs. Security needs over authorisations are expressed by prohibiting certain operations, limiting the scope, and prohibiting reauthorisations (further transferring permissions).

Let *Auth* stand for authorise$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, \textit{TrAuth})$, where A_1, A_2 are actors, $\mathcal{I}$ is a set of information, $\mathcal{G}$ is a set of goals, $\mathcal{OP}$ is the set of allowed and prohibited operations $\{\mathtt{R}, \mathtt{M}, \mathtt{P}, \mathtt{T}\}$

$\cup \{\bar{\mathtt{R}}, \bar{\mathtt{M}}, \bar{\mathtt{P}}, \bar{\mathtt{T}}\}$, and *TrAuth* is a boolean value determining transferability:

- $\mathcal{G} \neq \emptyset \rightarrow$ *Need-to-know*: the authorisee shall not perform any operation (read/modify/produce) on documents that make some information in $\mathcal{I}$ tangible, for any goals not included in $\mathcal{G}$. The authorisation from Patient to Hospital is an example: personal data, medical history and present illness shall be read only for goal blood transfused, see Figure 5.3.

- $\bar{\mathtt{R}} \in \mathcal{OP} \rightarrow$ *Non-reading*: the authorisee shall not read documents representing information in $\mathcal{I}$. For instance, Patient requires this when prohibiting Physician to read information personal data, see Figure 5.3.

- $\bar{\mathtt{M}} \in \mathcal{OP} \rightarrow$ *Non-modification*: the authorisee shall not modify documents that include information in $\mathcal{I}$. The authorisation from Patient to Hospital prohibits the production operation, that is information personal data, medical history and present illness shall not be modified by any goal of the Hospital, see Figure 5.3.

- $\bar{\mathtt{P}} \in \mathcal{OP} \rightarrow$ *Non-production*: the authorisee shall not produce any documents that include information in $\mathcal{I}$. For example, the Patient expresses a non-production requirement on personal data to Physician, by prohibiting the production operation, see Figure 5.3.

- $\bar{\mathtt{T}} \in \mathcal{OP} \rightarrow$ *Non-disclosure*: the authorisee shall not transmit (disclose) to other actors any document that includes information in $\mathcal{I}$. For instance, Patient requires this in the authorisation over information personal data, medical history and present illness to Hospital, see Figure 5.3.

- *TrAuth* $= false \rightarrow$ *Not-reauthorisation*: the authorisee shall not redistribute the permissions to other actors. If the authorisee receives an authorisation that contains only prohibitions, then not-reauthorisation does not apply. An example is the authorisation from Alice to ModernLabs, which does not grant a transferrable authorisation, see Figure 5.3.

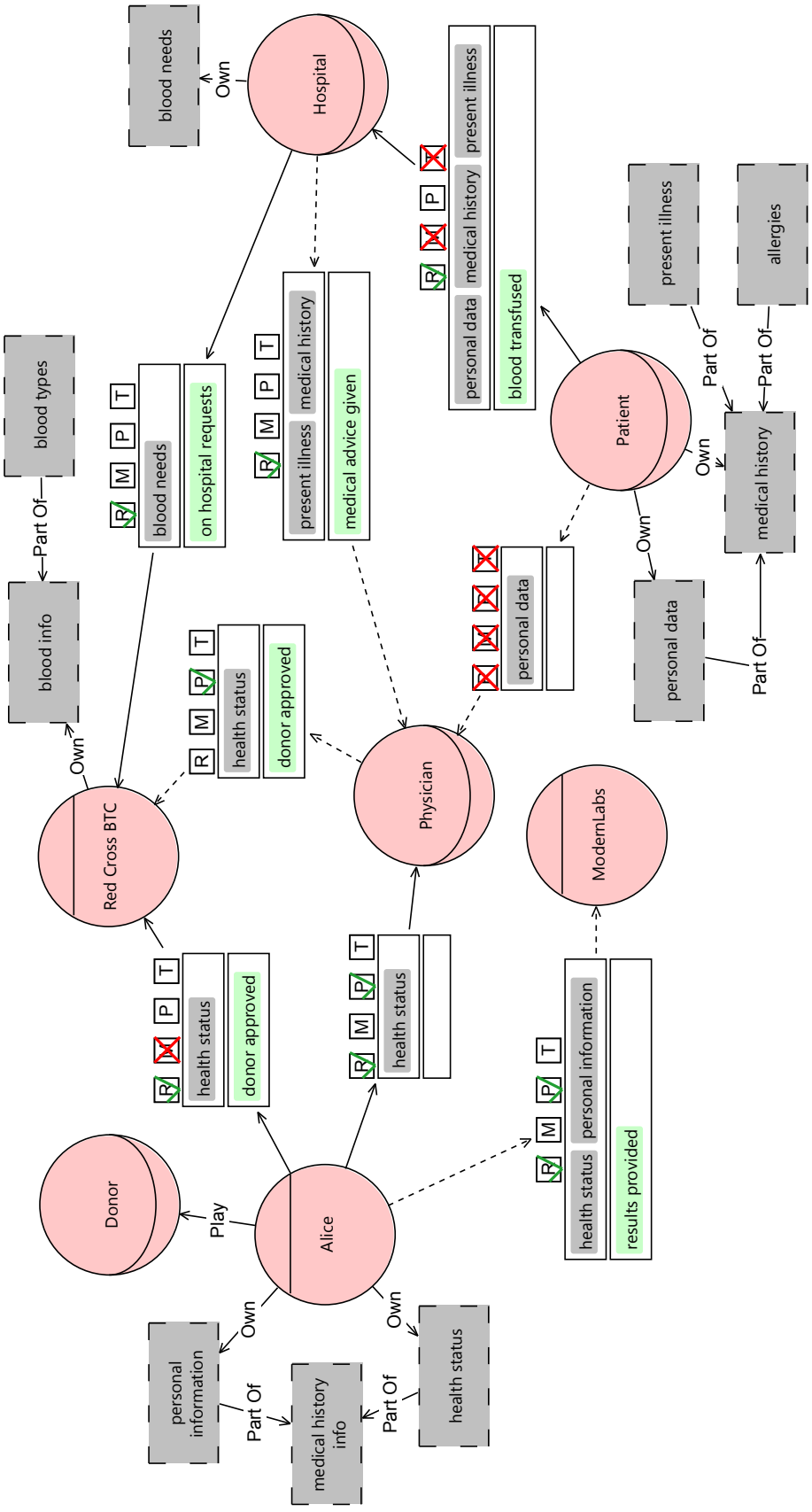


Figure 5.3: Partial STS-ml authorisation model of the healthcare scenario

5.5 Chapter summary

In this chapter we presented the three models supported by the STS-ml modelling language. The social model represents actors and their interactions, the information model represents actors as information owners and how their proprietary information is structured, while the authorisation model represents the permissions and prohibitions actors specify over their proprietary information to other actors in the system. Note that the information model serves as a bridge between the social and authorisation models, since in the first actors manipulate and exchange documents, while in the latter actors specify their authorisations over information. These distinction and the refinement of information supports expressing a richer set of security requirements, in particular over information.

Through the multi-view modelling approach STS aims at tackling the visual (cognitive) scalability problem of goal-models, and this separation of concerns is a first step towards that. The supporting toolset, as we will see in Chapter 7, allows the creation of STS-ml models within projects, where more STS-ml diagrams (overall model composed of the three models: social, information, and authorisation) of the same project can be created and maintained. Ideally, this modular approach should allow all the various diagrams to be integrated into one, while ensuring consistency, to create the complete model of a system at hand. This would promote collaboration of more requirements analysts and security engineers that create different diagrams of different parts of the system to be, instead of exchanging the same diagram. As of now, unfortunately, the underlying platform of STS-Tool does not support inter-diagrams consistency, only inter-model consistency within the same diagram is supported. This is a challenge for future developments of STS methodology and releases of STS-Tool.

Chapter 6

Automated analysis support

STS-ml models are actor- and goal-oriented, and they represent the business policies (how they intend to achieve their root goals) of the participants, their security requirements over information (informational assets), goals (intentional assets) and responsibility uptake. Being specified independently by different actors, actors' business policies and security requirements are likely to clash, thus leading to inconsistent specifications that cannot be satisfied by an implemented STS (at least one requirement would be violated).

In this chapter, we propose a framework for managing conflicts in STS-ml requirements. The framework suggests to iteratively (i) create STS-ml models for the domain at-hand, (ii) identify conflicts through automated reasoning techniques, and (iii) discuss possible conflicts' resolution alternatives. We first present the formal framework of STS-ml that defines the semantics of the language unambiguously (Section 6.1); then, we propose a number of different analysis techniques over STS-ml models supported by the STS methodology (Section 6.2).

Acknowledgement. Section 6.1 builds on top of [Paja et al., 2013b], while Section 6.2.2 builds on top of [Meland et al., 2014].

6.1 Formal framework

We introduce the formal framework for STS-ml models that enables automated reasoning, and illustrate it on the motivating scenario modelled in Chapter 4. Automated reasoning requires that the graphical models are translated (mapped) to formal specification to be used as input for the automated analysis techniques. Therefore, we define the elements and relationships composing an STS-ml model in order to allow its translation to formal specification.

We employ the following notation: atomic variables are strings in italic with a leading capital letter (e.g., G, I); sets are strings in the calligraphic font for mathematical expressions (e.g., $\mathcal{G}$, $\mathcal{I}$); relation names are in sans-serif with a leading non-capital letter (e.g., wants, possesses); constants are in typewriter style with a leading non-capital letter (e.g., `and`, `or`). The atomic

concepts of an STS-ml model (e.g., goal, document, event, threaten) are represented by the predicates in Tables 5.1–5.5 (e.g., goal, document, event, threaten), so we do not redefine them here.

We define first the informational knowledge base, which represents the relationships among elements of the information model. We define these relationships first given that the information model bridges the social and authorisation models, and as such serve as a basis for the other definitions.

Definition 1 (Informational knowledge base) *A tuple IKB* $= \langle \mathcal{I}, \mathcal{D}, \mathcal{IDR} \rangle$*, where* $\mathcal{I}$ *is a set of information elements,* $\mathcal{D}$ *is a set of documents, and* $\mathcal{IDR}$ *is a set of relationships over information in* $\mathcal{I}$ *and documents in* $\mathcal{D}$*:*

a) part-of-i(I_1, I_2)*: information* I_1 *is part of information* I_2*;*

b) part-of-d(D_1, D_2)*: document* D_1 *is part of document* D_2*;*

c) makes-tangible(I, D)*: document* D *materialises information* I*.* □

The Definition 1 formalises the relationships among elements of the information model, that is, how information and documents in the information model are interrelated: (*a*) part-of-i can be only among information entities, while part-of-i can be only among documents (*b*), therefore we cannot have any part-of relationship among an information entity and a document; the relationship tangible by, on the other hand, relates information entities with documents (*c*).

The information model in Figure 5.2 includes, e.g., relationships makes-tangible(personal data, medical record), part-of-i(present illness, medical history), and part-of-d(report, health record).

Next, we define intentional relationships, which characterise an actor in the socio-technical system, establishing how it pursues its goals.

Definition 2 (Intentional relationship) *A relationship within the scope of an individual actor* A*, which, thus, has no social meaning:*

- wants(A, G)*: actor* A *intends to achieve* G*;*
- decomposes$(A, G, \mathcal{S}, DecT)$*:* A *decomposes goal* G *into sub-goals from* $\mathcal{S}$*, where* $\mathcal{S}=\{G_1, \ldots, G_n\}$ *and* $|\mathcal{S}| \geq 2$ *and the decomposition is of type DecT (*`and` *or* `or`*);*
- possesses(A, D)*: actor* A *possesses document* D *(no other actor transmits it to* A*);*
- reads(A, G, D)*: actor* A *uses document* D *while achieving* G*;*
- modifies(A, G, D)*: actor* A *modifies document* D *while achieving* G*;*
- produces(A, G, D)*: actor* A *produces document* D *while achieving* G*.* □

Definition 2 enumerates the intentional relationships supported by STS-ml. Intentional relationships characterise the rationale of a single actor, i.e., how its goals are interrelated, and which documents the actor possesses and/or manipulates to fulfil its goals. For instance, consider actor Donor in Figure 5.1. The rationale of this actor (graphically represented by the oval shape around the actor) includes the intentional relationship reads(Donor, blood donated regularly, donor certificate).

The notion of an *actor model* formalises the contents of the oval balloon that is associated with a role or an agent shape in the social model. The goals and documents of a specific actor in a socio-technical system relate one to another. There are important relationships that need to be captured, and that enable defining the rationale of an actor, i.e., how it aims to attain its goals. This is the purpose of defining *actor model*, for it consists of an actor's intended goals, its possessed documents, and the relationships between these elements, aka its intentional relationships. For instance, the actor model of Red Cross BTC in Figure 5.1 includes: Red Cross BTC wants to fulfil goal blood distributed, for which it has to fulfil goals blood collected, blood consumption estimated, and blood transported; the fulfilment of blood consumption estimated requires modifying the document blood bank, and so on. Definition 3 ensures that all the relationships in an actor model are among elements of the same actor.

Definition 3 (Actor model) *An actor model AM is a tuple $\langle A, \mathcal{G}, \mathcal{D}, \mathcal{IRL}, T\rangle$ where A is an actor, $\mathcal{G}$ is a set of goals, $\mathcal{D}$ is a set of documents, $\mathcal{IRL}$ is a set of intentional relationships (Table 5.1) over goals in $\mathcal{G}$ and documents in $\mathcal{D}$, and T is an actor type (*`role` *or* `agent`*).*

Given an intentional relationship IRL in $\mathcal{IRL}$:

- *if $IRL = \mathsf{decomposes}(A', G, \mathcal{S}, \mathit{DecT})$, then $A' = A$, and both G and all goals in $\mathcal{S}$ are in $\mathcal{G}$;*
- *if $IRL = \mathsf{reads/modifies/produces}(A', G, D)$, then $A' = A$, G is in $\mathcal{G}$, and D is in $\mathcal{D}$*

We denote the set of actor models as $\mathcal{AM}$. □

Definition 3 states that an actor model is defined by the role or agent (A, defined by the type T), the goals it wants to achieve $\mathcal{G}$, the documents $\mathcal{D}$, together with goal decompositions, and the goal-document relationships determining whether the actor possesses the document or needs/modifies/produces it (in $\mathcal{IRL}$). Given the intentional relationships in an actor model, Definition 3 limits their creation only within an actor's scope: (i) given that all goals are in the set of goals of the actor, decompositions, also, are among these goals; and (ii) given goal-document relationships (reads/modifies/produces), their goals are in the set of the goals of the actor ($\mathcal{G}$) and documents are in the set of documents of the actor ($\mathcal{D}$).

The social model for the healthcare motivating scenario in Figure 5.1 includes multiple actor models, one per each modelled actor. An excerpt of an actor model, specifically for the Red Cross BTC is as follows:

$A =$ Red Cross BTC,
$\mathcal{G}$ includes blood distributed, blood collected, blood consumption estimated, etc.,
$\mathcal{IRL}$ includes decomposes(Red Cross BTC, blood distributed, {blood collected, blood consumption estimated, blood transported}, and),
modifies(Red Cross BTC, blood consumption estimated, blood bank), and
$T =$ agent.

After defining actors and their internal intentional structure, we define actor interactions, which are supported by social relationships in STS-ml.

Definition 4 (Social relationship) *A relationship that has a social meaning, i.e., it specifies how one or more actors are related in the STS:*

a) delegates(A_1, A_2, G)*: actor* A_1 *delegates goal* G *to actor* A_2*;*

b) transmits(A_1, A_2, D)*: actor* A_1 *transmits document* D *to actor* A_2*;*

c) authorises$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, \mathit{TrAuth})$*: actor* A_1 *authorises actor* A_2 *to perform operations* $\mathcal{OP}$ *on the information in* $\mathcal{I}$*, in the scope of the goals in* $\mathcal{G}$*, and allows (prohibits)* A_2 *to transfer the authorisation to others if TrAuth is true (false);*

d) plays(Ag_1, R_2)*: agent* Ag_1 *plays role* R_2*;*

e) owns(A_1, I_2)*: actor* A_1 *is the legitimate owner of information* I_2*.* □

Definition 4 enumerates and formally defines the social relationships supported by the STS-ml modelling language. STS-ml social relationships are modelled in the social and authorisation model. They define the social structure among the actors, i.e., relationships with validity in the modelled socio-technical system. Therefore, goal delegations, document transmission and authorisations are among actors in the socio-technical system for the achievement of a goal (*a*), transfer of a document (*b*) and delegation of authority (*c*) respectively; play relationships, on the other hand, relate an agent with a role (*d*), so in this case we do want to be explicit in specifying that this relationship stands between an agent and a role, and is not applicable to the generic actor term; finally, the social relationship for information ownership, owns, connects an actor with the information entity it owns (*e*).

We now define the security requirements derived from the specification of security needs over actors' interactions, aka their social relationships.

Definition 5 (Sequrity requirements) *A property that is required either (i) by an actor to another actor over the social relationship between them (either a goal delegation or a document transmission or an authorisation), or (ii) by the STS—here, intended as the legal context—any participating actor to comply with.*

Sequrity requirements over goal delegations. *If* $\texttt{Del} = \mathsf{delegates}(A_1, A_2, G)$*:*

- R_1. $\mathsf{non\text{-}repudiation\text{-}del}(A_2, A_1, \texttt{Del})$*: A_2 (the delegatee) requires the delegator A_1 not to repudiate the delegation* Del*;*
- R_2. $\mathsf{non\text{-}repudiation\text{-}acc}(A_1, A_2, \texttt{Del})$*: the delegator$A_1$ requires the delegatee A_2 not to repudiate the acceptance of the delegation* Del*;*
- R_3. $\mathsf{true\text{-}single\text{-}red}(A_1, A_2, G)$*: the delegator A_1 requires the delegatee A_2 to deploy concurrent redundant means for G;*
- R_4. $\mathsf{fback\text{-}single\text{-}red}(A_1, A_2, G)$*: the delegator A_1 requires the delegatee A_2 that, if the first strategy for G by A_2 fails, A_2 will deploy another strategy;*
- R_5. $\mathsf{true\text{-}multi\text{-}red}(A_1, A_2, G)$*: the delegator A_1 requires the delegatee A_2 to deploy concurrent redundant means for G involving at least another actor;*
- R_6. $\mathsf{fback\text{-}multi\text{-}red}(A_1, A_2, G)$*: the delegator A_1 requires the delegatee A_2 that, if the first strategy for G by A_2 (another actor A_3) fails, A_3 (A_2) will deploy another strategy;*
- R_7. $\mathsf{no\text{-}redelegation}(A_1, A_2, G)$*: the delegator A_1 requires the delegatee A_2 to not redelegate G;*
- R_8. $\mathsf{trustworthiness}(A_1, A_2, G)$*: the delegator A_1 requires the delegatee A_2 to be trustworthy in order to delegate it the fulfillment of G;*
- R_9. $\mathsf{goal\text{-}availability}(A_2, A_1, G)$*: the delegator A_1 wants the delegatee A_2 to guarantee a minimum availability level expressed in percentage for goal G;*
- R_{10}. $\mathsf{delegator\text{-}auth}(A_2, A_1, \texttt{Del})$*: the delegatee A_2 needs that the delegator A_1 authenticates herself;*
- R_{11}. $\mathsf{delegatee\text{-}auth}(A_1, A_2, \texttt{Del})$*: the delegator A_1 needs that the delegatee A_2 authenticates herself.*

Sequrity requirements over document transmissions. *If* $\texttt{Tx} = \mathsf{transmits}(A_1, A_2, \texttt{D})$*, then:*

- R_{12}. $\mathsf{non\text{-}repudiation\text{-}tx}(A_2, A_1, \texttt{Tx})$*: the receiver A_2 requires the sender A_1 not to repudiate the transmission* Tx*;*

R_{13}. non-repudiation-acc$(A_1, A_2, \mathtt{Tx})$*: the sender* A_1 *requires the receiver* A_2 *not to repudiate the acceptance of the transmission* Tx*;*

R_{14}. sender-integrity$(A_2, A_1, \mathtt{Tx})$*: the receiver* A_2 *requires the sender* A_1 *to ensure the integrity of transmission for the document* D *being transmitted;*

R_{15}. receiver-integrity$(A_1, A_2, \mathtt{Tx})$*: the sender* A_1 *requires the receiver* A_2 *to ensure the integrity of transmission for the document* D *being transmitted;*

R_{16}. system-integrity$(\mathtt{STS}_{,}\ \mathtt{Tx})$*: the system shall ensure that the integrity of transmission of the document* D *in transmission is preserved;*

R_{17}. doc-availability(A_2, A_1, D)*: the receiver* A_2 *requires the sender* A_1 *to guarantee an availability level expressed in percentage for the transmitted document* D*;*

R_{18}. sender-auth$(A_2, A_1, \mathtt{Tx})$*: the receiver* A_2 *needs that the sender* A_1 *authenticates herself to transfer the document* D*;*

R_{19}. receiver-auth$(A_1, A_2, \mathtt{Tx})$*: the sender* A_1 *needs that the receiver* A_2 *authenticates herself to have the document* D*;*

R_{20}. sender-conf$(A_2, A_1, \mathtt{Tx})$*: the receiver* A_2 *requires the sender* A_1 *to ensure the confidentiality of the document* D *being transmited;*

R_{21}. receiver-conf$(A_1, A_2, \mathtt{Tx})$*: the sender* A_1 *requires the receiver* A_2 *to ensure the confidentiality of the document* D *being transmited;*

R_{22}. system-conf$(\mathtt{STS}_{,}\ \mathtt{Tx})$*: the system shall ensure that the confidentiality of transmission of a document* D *in transfer is preserved.*

Sequrity requirements over responsibility uptake. *A property that the STS—here, intended as the legal context—requires any participating actor to comply with:*

R_{23}. role-sod$(\mathtt{STS}, Ag, R_1, R_2)$*: no agent* Ag *can play both roles* R_1 *and* R_2*;*

R_{24}. goal-sod$(\mathtt{STS}, Ag, G_1, G_2)$*: every agent* Ag *must not pursue both goals* G_1 *and* G_2*;*

R_{25}. role-cod$(\mathtt{STS}, Ag, R_1, R_2)$*: every agent* Ag *playing role* R_1 *(*R_2*), must also play* R_2 *(*R_1*);*

R_{26}. goal-cod$(\mathtt{STS}, Ag, G_1, G_2)$*: an agent* Ag *pursuing goal* G_1 *(*G_2*), should also pursue* G_2 *(*G_1*) too.*

Sequrity requirements over authorisations. *If Auth* = authorises$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, \mathit{TrAuth}) \in \Delta_{\mathcal{SR}}$*, where* A_1*,* A_2 *are actors,* $\mathcal{I}$ *is a set of information,* $\mathcal{G}$ *is a set of goals,* $\mathcal{OP}$ *is the set of allowed and prohibited operations* $\{\mathtt{R}, \mathtt{M}, \mathtt{P}, \mathtt{T}\} \cup \{\overline{\mathtt{R}}, \overline{\mathtt{M}}, \overline{\mathtt{P}}, \overline{\mathtt{T}}\}$*, and TrAuth is a boolean value determining transferability, then:*

R_{27}. need-to-know$(A_1, A_2, \mathcal{I}, \mathcal{G})$*: the authoriser actor* A_1 *requires the authorisee* A_2 *not to perform any operation (use/modify/produce) on documents that make some information in* $\mathcal{I}$ *tangible, for any goals not included in* $\mathcal{G}$*;*

R_{28}. non-reading$(A_1, A_2, \mathcal{I})$*: the authoriser actor* A_1 *requires the authorisee* A_2 *not to read documents representing information in* $\mathcal{I}$*;*

R_{29}. non-modification$(A_1, A_2, \mathcal{I})$*: the authoriser actor* A_1 *requires the authorisee* A_2 *not to modify documents that include information in* $\mathcal{I}$*;*

R_{30}. non-production$(A_1, A_2, \mathcal{I})$*: the authoriser actor* A_1 *requires the authorisee* A_2 *not to produce any documents that include information in* $\mathcal{I}$*;*

R_{31}. non-disclosure$(A_1, A_2, \mathcal{I})$*: the authoriser actor* A_1 *requires the authorisee* A_2 *not to transmit (disclose) to other actors any document that includes information in* $\mathcal{I}$*;*

R_{32}. not-reauthorised$(A_1, A_2, \mathcal{I}, \mathcal{G}, \{\mathtt{R}, \mathtt{M}, \mathtt{P}, \mathtt{T}\})$*: the authoriser actor (*A_1*) requires the authorisee (*A_2*) not to redistribute the permissions to other actor.*

We denote the set of security requirements supported by STS-ml as $\mathcal{SRQ}$*, which is the set of security requirements of the types* R_1–R_{32}. □

Following Definition 5, security requirements are security expectations that are expressed either by actors on social relationships or the STS itself, imposing security requirements on all participating actors. The definition of security requirements follows the way these requirements are captured in STS-ml. We provide an example of each.

In Figure 5.1, $\mathtt{Del}_1 =$ delegates(Patient, Hospital, patient registered) has a security requirement non-repudiation-acc(Patient, Hospital, $\mathtt{Del}_1$); the transmission of document donor certificate from Red Cross BTC to Donor requires receiver authentication as specified by the Red Cross BTC; a combination of duties is specified among goals transfusion performed and medical advice given of Physician; and finally, in Figure 5.3, the authorisation from Patient to Hospital implies, for instance, requirements about need-to-know (due to the goal scope blood transfused), non-reading and non-disclosure (for those operations are prohibited, R and T respectively).

We tie together all the elements in the social, information, and authorisation models to define an STS-ml model, as specified in Definition 6.

Definition 6 (STS-ml model) *An STS-ml model* M *is a tuple* $\langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{SRQ} \rangle$ *where* $\mathcal{AM}$ *is a set of actor models,* $\mathcal{SR}$ *is a set of social relationships, IKB is an informational knowledge base,* $\mathcal{SRQ}$ *is a set of security requirements of the categories* R_1–R_{32}. □

An STS-ml model is composed of all the actor models ($\mathcal{AM}$) of the identified actors, the social relationships among them ($\mathcal{SR}$), the information model (IM) tying together actors' information and documents, and the security requirements expressed by actors ($\mathcal{SRQ}$) over social relationships. The STS-ml model for the healthcare motivating scenario is composed of all the actor models of the actors drawn in the social model (Figure 5.1), the social relationships drawn in the social as well as the authorisation model (Figure 5.1 and 5.3), the information model (Figure 5.2), and the set of security requirements R_1–R_{32}.

When tying together the different elements in a STS-ml model, we need to ensure its well-formedness. Definition 7 lays down the constraints on the well-formedness of an STS-ml model.

Definition 7 (Well-formed STS-ml model) *An STS-ml model $M = \langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{SRQ} \rangle$ is well-formed if and only if:*

1. *social relationships are only over actors with models in $\mathcal{AM}$, and:*
2. *only leaf goals are delegated: for each $\mathsf{delegates}(A, A', G)$ in $\mathcal{SR}$, there is an actor model $\langle A, \mathcal{G}, \mathcal{D}, \mathcal{IRL}, T \rangle$ in $\mathcal{AM}$ such that $G \in \mathcal{G}$ and there is no decomposition of G in $\mathcal{IRL}$;*
3. *delegated goals appear in the delegatee's actor model: for each $\mathsf{delegates}(A', A, G)$ in $\mathcal{SR}$, there is an actor model $\langle A, \mathcal{G}, \mathcal{D}, \mathcal{IRL}, T \rangle$ in $\mathcal{AM}$ such that $G \in \mathcal{G}$;*
4. *the transmitter must possess the transmitted document for the document transmittion to take place: for each $\mathsf{transmits}(A, A', D)$ in $\mathcal{SR}$, there is an actor model $\langle A, \mathcal{G}, \mathcal{D}, \mathcal{IRL}, T \rangle \in \mathcal{AM}$ such that $D \in \mathcal{D}$. An actor possesses a document if:*
   (a) *it has the document since scratch, in which case $\mathsf{possesses}(A, D) \in \mathcal{IRL}$, or*
   (b) *it creates the document, i.e., $\exists\, G \in \mathcal{G}.\ \mathsf{wants}(A, G) \wedge \mathsf{produces}(A, G, D) \in \mathcal{IRL}$, or*
   (c) *there is an actor A'' such that $\mathsf{transmits}(A'', A, D) \in \mathcal{SR}$.*
5. *transmitted documents appear in the receiver's actor model: for each $\mathsf{transmits}(A', A, D)$ in $\mathcal{SR}$, there is an actor model $\langle A, \mathcal{G}, \mathcal{D}, \mathcal{IRL}, T \rangle$ in $\mathcal{AM}$ such that $D \in \mathcal{D}$;*
6. *authorisations are syntactically well-formed: for each $\mathsf{authorises}(A, A', \mathcal{I}, \mathcal{G}, \mathcal{OP}, TrAuth) \in \mathcal{SR}$, there must be at least one information entity specified ($|\mathcal{I}| \geq 1$), and at least one prohibition or permission is specified;*
7. *all security requirements in $\mathcal{SRQ}$ are over social relationships in $\mathcal{SR}$, actors with models in $\mathcal{AM}$, or over their goals;*
8. *the model M complies with the following constraints:*

(a) delegations have no cycles: for each delegates(A, A', G) *in* $\mathcal{SR}$*, there is no* A'' *such that* delegates(A'', A, G) *in* $\mathcal{SR}$ *or* delegates(A'', A, G_i) *in* $\mathcal{SR}$*, where* G_i *is a descendant of* G *in the goal tree of* A''*;*

(b) part-of *relationships (either over information or documents) have no cycles.* □

Following Definition 7 in the social model in Figure 5.1 we cannot have, for instance, a delegation of goal treatment received of Patient, rather its subgoals patient registered and transfusion needed are delegated (complying with 2); Patient's goal transfusion needed delegated to Physician appears in Physician's actor model (following 3); the Hospital would not be able to transfer document blood usage listings to the Red Cross BTC if it did not produce this document when pursuing goal blood transfused (complying with 4, specifically with 4b in this case); Hospital has document blood bank since it was transmitted from Red Cross BTC (following 5); in Figure 5.3 the authorisation from Physician to Red Cross BTC would be not well-formed, if the right to produce would not be specified (following 6);

The imposed constraints (see 8) are necessary as they determine exceptions that would not allow us to reason over STS-ml models. For instance, delegation cycles would result in an STS-ml model in which it is not clear which actor is responsible for fulfilling the delegated goal. For instance, in Figure 5.1 there cannot be a delegation from Physician back to Hospital of goals transfusion performed via specialist (following 8a). Cycles of part-of relationships, on the other hand, would result in ambiguities over information ownership (when specified over information entities) or ambiguities in reasoning over violations of security requirements (more specifically confidentiality requirements, such as non-reading, non-disclosure, etc.) when over documents. For instance, in Figure 5.2 there cannot be a part-of relationship from information medical history to information personal data (following 8b). As we will see in Chapter 7, the well-formedness of an STS-ml model can be verified with STS-Tool, which integrates this checks.

We define *authorisation closure* in order to determine actors' final rights and the security requirements they have to comply with. Authorisation closure formalises the intuition that, if an actor A_2 has no incoming authorisation for an information I, then A_2 has a prohibition for I. Such prohibition is an authorisation from the information owner that prohibits all operations as well as the right to transfer authorisations. Formally, it is defined in Definition 8.

Definition 8 (Authorisation closure) *Let* $M = \langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{SRQ} \rangle$ *be a well-formed STS-ml model. The authorisation closure of* $\mathcal{SR}$*, denoted as* $\Delta_{\mathcal{SR}}$*, is a superset of* $\mathcal{SR}$ *that makes prohibitions explicit, when no authorisation is granted by any actor.*
Formally, $\forall A, A'$ *with an actor model in* $\mathcal{AM}$*,*
$\forall \mathsf{owns}(A, I) \in \mathcal{SR}.\ \nexists A''.\mathsf{authorises}(A'', A', \mathcal{I}, \mathcal{G}, \mathcal{OP}, \texttt{TrAuth}) \in \mathcal{SR} \wedge I \in \mathcal{I} \rightarrow$
$\mathsf{authorises}(A, A'', I, \mathcal{G}', \overline{\mathcal{OP}}, \texttt{false}) \in \Delta_{\mathcal{SR}}$*, where* $\mathcal{G}'$ *is the set of goals of* A'*.* □

As described in Chapter 4, Section 4.3 and 4.5.1, security requirements over authorisations are generated over explicit prohibitions. But, since authorisations are summed up, it is not always clear from the models, what operations actors are ultimately granted over a given information. Whenever there are no incoming authorisation (no single authorisation relation has been drawn towards the actor), or no incoming authorisations grant a given operation from $\mathcal{OP}$ over I ($\nexists A''.\mathsf{authorises}(A'', A', \mathcal{I}, \mathcal{G}, \mathcal{OP}, \text{TrAuth}) \in \mathcal{SR}$), we generate an authorisation from the information owner that prohibits all operations, for which no permissions were granted, and sets transferability to false ($\mathsf{authorises}(A, A'', I, \mathcal{G}', \overline{\mathcal{OP}}, \texttt{false}) \in \Delta_{\mathcal{SR}}$, where A is the information owner). We do this to reason over possible unauthorised operations over the given information. For instance, in Figure 5.3, the lack of incoming authorisations towards the Red Cross BTC for information personal information, implies $\mathsf{authorises}(\mathsf{Alice}, \mathsf{Red\ Cross\ BTC}, \mathsf{personal\ information}, \mathcal{G}, \overline{\mathcal{OP}}, \texttt{false}) \in \Delta_{\mathcal{SR}}$, where $\mathcal{G}$ is the set of goals of the Red Cross BTC, and Alice is the owner of information personal information, see Figure 5.2 or 5.3, in which the details about information ownership are preserved.

We emphasise the key difference between the explicit prohibitions and the implicit (derived) prohibitions from authorisation closure. In the former case, an actor wants to ensure that another cannot do specific operations on some information. In the latter case, the lack of permissions can be a temporary situation, which could be changed by any actor having permission and authority to transfer such permission, and using this authority by specifying an authorisation that grants the said permission.

Indeed, we apply Definition 8 when executing the automated reasoning, at the end of the modelling process. This ensures that changes over the model are taken into account, and a recalculation of the authorisation closure takes place.

6.2 STS Automated Reasoning

The formal framework in Section 6.1 allows us to build the formal specification of a given STS-ml model, over which we can perform automated analysis. We support two main types of analysis: (i) verifying that all security requirements can be satisfied, as well as (ii) verifying the impact of events threatening actors' assets. The first analysis is known as *security analysis*, while the latter as *threat analysis*. Both analysis types are integrated in STS-Tool, as we will also see in Chapter 7, and they are executed upon a well-formed STS-ml model following Definition 7. We describe each in the following sections, Section 6.2.1 and Section 6.2.2 respectively.

6.2.1 Security Analysis

The purpose of this analysis is to verify whether the created STS-ml model allows the satisfaction of the specified security requirements. For all security requirements expressed by

stakeholders, we check in the model whether there is any possibility for the security requirement to be violated. We do this at two levels: (i) identifying possible conflicts among security requirements, i.e., two or more requirements cannot be all fulfilled by the same system, and (ii) identifying conflicts between actors own business policies and the security requirements imposed on them.

Conflicts among authorisations

We check if the stakeholders have expressed conflicting authorisations. This is non-trivial, for STS-ml models contain multiple authorisations over the same information, and every authorisation expresses both permissions and prohibition. For instance, the authorisation from Municipality to InfoTN allows reading (tick symbol over R, Fig. 5.3) information personal info, residential address and tax contributions, but prohibits the production (cross symbol over P) of the given information, and specifies nothing on modification and transmission. The authorisation is limited to the scope of goal system maintained. If we had another authorisation towards InfoTN for the same information which is not restricted to a goal scope, then we would have an authorisation conflict. Similarly, if we had more authorisations towards InfoTN prohibiting the granted operation (in this case R) or granting the right to perform any of the prohibited operations (in this case P) over any of the specified information entities (personal info, residential address or tax contributions) or parts of these information entities, in the scope of the same goal (i.e. including its subgoals), then we would have an authorisation conflict. This is formalised in Definition 9.

Definition 9 (Authorisation conflict) *Two authorisations* $\mathsf{authorises}(A_1, A_2, \mathcal{I}_1, \mathcal{G}_1, \mathcal{OP}_1, \mathit{TrAuth}_1)$, *and* $\mathsf{authorises}(A_3, A_2, \mathcal{I}_2, \mathcal{G}_2, \mathcal{OP}_2, \mathtt{TrAuth}_2)$ *are conflicting (*$\mathsf{a\text{-}conflict}(\mathit{Auth}_1, \mathit{Auth}_2)$*) if and only if they both regulate the same information (*$\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$*), and either:*

1. $\mathcal{G}_1 \neq \emptyset \ \wedge \ \mathcal{G}_2 = \emptyset$ *, or vice versa; or,*
2. $\mathcal{G}_1 \cap \mathcal{G}_2 \neq \emptyset$*, and either (i)* $\exists \mathtt{op}.\ \mathtt{op} \in \mathcal{OP}_1 \wedge \overline{\mathtt{op}} \in \mathcal{OP}_2$*, or vice versa (where* op *is one of* $\{\mathtt{R}, \mathtt{M}, \mathtt{P}, \mathtt{T}\}$*); or (ii)* $\mathtt{TrAuth}_1 \neq \mathtt{TrAuth}_2$. □

Definition 9 states that an authorisation conflict occurs if both authorisations apply to the same information, and either (1) one authorisation restricts the permission to a goal scope ($\mathcal{G}_1 \neq \emptyset$), while the other does not ($\mathcal{G}_2 = \emptyset$), that is, one implies a need-to-know requirement, while the other grants rights for any purpose; or, (2) the scopes are intersecting ($\mathcal{G}_1 \cap \mathcal{G}_2 \neq \emptyset$), and contradictory permissions are granted (on operations op, or authority to transfer TrAuth).

In Figure 5.3, there are two authorisations towards Physician on personal data: (i) that from Hospital, which grants the right to read (R) and specifies nothing on modification (M), production

(P) or transmission (T) of information personal data, present illness, and medical history; (ii) that from Patient, which prohibits all operations R, M, P, and T on information personal data. The first authorisation specifies a need-to-know security requirement in the scope of goal medical advice given, while the second authorisation does not specify anything, since imposes only prohibitions over the information personal data. Thus, following Definition 9, the two authorisations have intersection scopes (the scope of the first authorisation) and they are both specified over information personal data. Confronting the granted and prohibited operations from each, we can deduce that they are conflicting with respect to the read (R) operation for information personal data.

Definition 10 (Authorisation consistent STS-ml model) *An STS-ml model is authorisation-consistent when no authorisation conflicts exist.* □

Following Definition 10, in order to have an authorisation-consistent model for the healthcare scenario we need to resolve the identified conflict. Resolving the conflict requires that one of the actors changes its specified authorisations, either the Patient decides to grant the Physician the right to read information personal data or the Hospital specifies a non-reading security requirement too, prohibiting the right to read.

Conflicts between business policies and security requirements

Each actor model defines a specific actor's business policy, i.e., its goals and the alternative strategies to achieve these goals. Given an authorisation-consistent STS-ml model, we verify if any security requirement is violated by the business policy of any actor in the model. For instance, a requirement such as non-reading(Alice, Red Cross BTC, {personal information}) conflicts with a business policy for Red Cross BTC that includes the relationship reads(Red Cross BTC, donor approved, personal information).

An actor's *business policy* defines alternative strategies for an actor to achieve its root goals. It is a sub-model of the social model that includes all the goals and documents in the scope of that actor in the social model, the relationships (and/or-decompose, reads, modifies, and produces) among those goals and documents, as well as goal delegations and document transmissions that start from that actor. For instance, the business policy of Hospital includes goals health service provided, which is and-decomposed in goals health records maintained and blood transfused, the first is further and-decomposed into goals patient registered which produces document high quality data and medical history obtained, while the latter (goal search module built) reads document blood bank, produces document blood usage listings and is or-decomposed into transfusion performed via specialist and transfusion performed in surgery (see Figure 5.1). The or-subgoals denote alternative strategies: the actor can choose either of them

to achieve the upper level goal. Hence, Hospital may choose to perform transfusion procedures either with the help of a specialist or during surgery.

Definition 11 (Actor strategy) *Given a business policy for an actor A (denoted as P_A), an actor strategy S_{P_A} is a sub-model of P_A obtained by pruning P_A as follows:*

- *for every or-decomposition, only one subgoal is kept. All other subgoals are pruned, along with the elements that are reachable from the pruned subgoals only (via and/or-decomposes, reads/produces/modifies, transmits, and delegates relationships);*
- *for every root goal G that is delegated to A, G can optionally be pruned.* □

Definition 11 formalises the intuition that alternative strategies are introduced by: (i) choosing one subgoal in an or-decomposition; and (ii) deciding whether to pursue root goals that are delegated from other actors. In Figure 5.1, the business policy for Hospital includes one root goal and one delegated goal patient registered, which is not a root goal. Therefore, one strategy for the Hospital involves all goals (they are all and-decomposed) till goal blood transfused. This goal is or-decomposed; by Definition 11, one subgoal is retained in the strategy (e.g., transfusion performed via specialist), while other goals are pruned (in this case, transfusion performed in surgery). The reads relationship to document blood bank is retained, as well as the document itself. The same applies to the produces relationship and document blood usage listings. An alternative strategy could, however, involve not performing the transfusion via specialist, but in surgery. However, even in this alternative the rest of the elements in the actor model remains the same (given that these are leaf level goals and do not have any relationship to documents).

We define the notion of a *variant* to: (i) consistently combine actors' strategies (each actor fulfils the root goals in its strategy), by requiring that delegated goals are in the delegatee's strategy; and to (ii) include the authorises relationships in the STS-ml model. This enables us to identify conflicts that occur only when the actors choose certain strategies, see Definition 12.

Definition 12 (Variant) *Let M be an authorisation-consistent STS-ml model, $P_{A_1}, \ldots, P_{A_n}$ be the business policies for all actors in M. A variant for M (denoted as $\mathcal{V}_M$) consists of:*

- *a set of strategies $\{S_{P_{A_1}}, \ldots, S_{P_{A_n}}\}$ such that, for each A', A'', G, if* delegates(A', A'', G) *is in $S_{P_{A'}}$, then G is in $S_{P_{A''}}$, and*
- *all the* authorises *relationships from M.* □

Variants constrain the strategies of the actors. In Figure 5.1, every variant includes ModernLab pursuing goal tests taken, for this is a root goal delegated to ModernLab from Alice and Alice's root goal blood donated is not delegated to her by others (thus, it has to be in her strategy), and the only possible strategy involves and-decomposing blood donated into tests taken and

neg results received, and delegating goal tests taken to ModernLab. Thus, there exists no variant where the latter actor does not pursue the goal tests taken.

An STS-ml model may have multiple variants depending on the existance of alternative strategies (corresponding to or-decompositions). For example, if ModernLabs has to achieve tests taken by having in its strategy all goals infec. disease tested, drug test performed, and results provided, since these are and-subgoals of its root goal, however ModernLabs can choose to achieve results provided through results retrieved in person or results sent via courier, see Figure 5.1 to verify these alternatives within ModernLabs's actor model.

Variants enable detecting conflicts between actors' business policies and security requirements they have to comply with. The latter define (dis)allowed relationships for the actors' business policies, and at the interplay of the two we can identify conflicts, see Definition 13.

Definition 13 (Bus-Sec conflict) *Given a variant $\mathcal{V}_M$, a conflict between business policies and security requirements (Bus-Sec conflict) exists if and only if there is an actor A such that:*

- *the strategy of A in $\mathcal{V}_M$ contains one or more relationships (as denoted by its business policy) that are prohibited by a security requirement requested (prescribed) by another actor A' to A;*
- *the strategy of A in $\mathcal{V}_M$ does not contain any relationships required by some requirement requested by another actor A' to A.* □

Tables 6.1–6.6 describe semi-formally how these conflicts are verified at design time for the different types of security requirements that STS-ml supports. Below, we provide some more details.

We discuss the requirements for each category (see Figure 4.18 in Chapter 4) and give insights on the verification of runtime requirements.

Accountability requirements. Non-repudiation requirements ($\mathsf{R}_1, \mathsf{R}_2$, and $\mathsf{R}_{12}, \mathsf{R}_{13}$) expressed over goal delegations and document transmissions respectively, cannot be verified at design time for they require actions such as proof of fulfilment for the goal in case of non-repudiation of acceptance.

Not-redelegation (R_7) is verified if there is no delegation relationship having G or its subgoals as delegatum, from A_2 to other actors in the variant.

Role-based separation and combination of duties (R_{23} and R_{25} respectively) require all Ag *not to* or *to* play two roles through play relationships, respectively. Goal-based separation of duties (R_{24}) is verified if no Ag pursues both G_1 and G_2 or their subgoals. Finally, goal-based combination of duties (R_{26}) is verified in a similar way, but Ag should be the final performer (i.e., does not delegate) of both goals.

Table 6.1: Accountability security requirements: design-time verification against a variant $\mathcal{V}_M$

No.	Requirement	Verification at design-time
R_1	non-repudiation-del$(A_2, A_1, \texttt{Del})$	No
R_2	non-repudiation-acc$(A_1, A_2, \texttt{Del})$	No
R_{12}	non-repudiation-tx$(A_2, A_1, \texttt{Tx})$	No
R_{13}	non-repudiation-acc$(A_1, A_2, \texttt{Tx})$	No
R_7	non-redelegation(A_1, A_2, G)	$\nexists$delegates$(A_2, A_3, G') \in \mathcal{V}_M$. $G' = G$ or G' is a subgoal of G
R_{23}	role-sod$(\texttt{STS}, Ag, R_1, R_2)$	$\{\mathsf{plays}(Ag, R_1), \mathsf{plays}(Ag, R_2)\} \nsubseteq \mathcal{V}_M$
R_{24}	goal-sod$(\texttt{STS}, Ag, G_1, G_2)$	Ag should not pursue both G_1 and G_2 or their subgoals
R_{25}	role-cod$(\texttt{STS}, Ag, R_1, R_2)$	$\{\mathsf{plays}(Ag, R_1), \mathsf{plays}(Ag, R_2)\} \subseteq \mathcal{V}_M$
R_{26}	goal-cod$(\texttt{STS}, Ag, G_1, G_2)$	Ag should pursue both G_1 and G_2 or their subgoals

Reliability requirements. Redundancy requirements (R_3 to R_6) can be partially checked at design time. The existence of redundant alternatives can be verified, but a variant does not tell how alternatives are interleaved, i.e., if they provide true or fallback redundancy. As a result, true and fallback redundancy are checked the same way. Single-actor redundancy (R_3 and R_4) is fulfilled if A_2 has at least two disjoint alternatives (via or-decompositions) for G. Multi-actor redundancy (R_5 and R_6) requires that at least one alternative involves another actor A_3.

Trustworthiness requirements (R_8) cannot be verified at design time, since they require the delegatee to provide a proof of trustworthiness, e.g., issued by a certification authority.

Table 6.2: Reliability security requirements: design-time verification against a variant $\mathcal{V}_M$

No.	Requirement	Verification at design-time
R_3	true-single-red(A_1, A_2, G)	Partial. A_2 pursues goals in $\mathcal{V}_M$ that define at
R_4	fback-single-red(A_1, A_2, G)	least two disjoint ways to support G
R_5	true-multi-red(A_1, A_2, G)	Partial. Both A_2 and another actor A_3 support
R_6	fback-multi-red(A_1, A_2, G)	G, each in a different way
R_8	trustworthiness(A_1, A_2, G)	No

Authenticity requirements. Delegator/sender authentication ($\mathsf{R}_{10}, \mathsf{R}_{18}$) is typically imple-

mented in electronic commerce website, wherein a certification authority guarantees the authenticity of the sellers' website. Therefore, the verification of such requirement involves actions and mechanisms that can be verified only at runtime.

Delegatee/receiver authentication ($\mathsf{R}_{11}, \mathsf{R}_{19}$), on the other hand, is one we encounter everyday when browsing the web and using our credentials (username/password) to access web information such as our email. The fulfilment of these type of requirements cannot be verified at design time.

Table 6.3: Authenticity security requirements: design-time verification against a variant $\mathcal{V}_M$

No.	Requirement	Verification at design-time
R_{10}	delegator-auth$(A_2, A_1, \texttt{Del})$	No
R_{11}	delegatee-auth$(A_1, A_2, \texttt{Del})$	No
R_{18}	sender-auth$(A_2, A_1, \texttt{Tx})$	No
R_{19}	receiver-auth$(A_1, A_2, \texttt{Tx})$	No

Availability requirements. Verifying availability requirements (both goal and document availability—$\mathsf{R}_9, \mathsf{R}_{17}$) calls for a way to measure availability level. Notice that goal availability is highly related to the notion of service availability, where a provider specifies an uptime level for the service. In service-oriented settings, availability levels often become integral part of service-level agreements between providers and consumers. These requirements cannot be verified at design time.

Table 6.4: Availability security requirements: design-time verification against a variant $\mathcal{V}_M$

No.	Requirement	Verification at design-time
R_9	r-g-avail-ensured(A_2, A_1, G)	No
R_{17}	r-d-avail-ensured(A_2, A_1, D)	No

Integrity requirements. Non-modification (R_{29}) requires that A_2's strategy in the variant includes no modifies relationship on documents that make tangible part of $I \in \mathcal{I}$.

Integrity of transmission requirements ($\mathsf{R}_{14}, \mathsf{R}_{15}, \mathsf{R}_{16}$) prescribe that the integrity of transmission of the document is preserved, and this can only be verified at runtime.

Confidentiality requirements. Confidentiality requirements expressed through different types of authorisations prescribe the set of relationships that shall not be present in A_2's strategy in the variant. Need-to-know (R_{27}) is verified by the absence of reads, modifies, or produces

Table 6.5: Integrity security requirements: design-time verification against a variant $\mathcal{V}_M$

No.	Requirement	Verification at design-time
R_{29}	non-modification$(A_1, A_2, \mathcal{I})$	$\nexists$modifies$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{14}	sender-integrity$(A_2, A_1, \texttt{Tx})$	No
R_{15}	receiver-integrity$(A_1, A_2, \texttt{Tx})$	No
R_{16}	system-integrity$(STS_{,}, \texttt{Tx})$	No

relationships on documents that make tangible some information in $\mathcal{I}$ for some goal G' that is not in $\mathcal{G}$ or in descendants of some goal in $\mathcal{G}$. Requirements R_{28} and R_{30} are verified if A_2's strategy in the variant includes no read or produces relationships on documents that make tangible part of $I \in \mathcal{I}$, respectively. Non-disclosure (R_{31}) does a similar check but looking at document transmissions, i.e., transmits relationships. Non-reauthorisation (R_{32}) is fulfilled if there is no authorises relationship from A_2 to others on any operation in $\mathcal{OP}$ over $\mathcal{I}$ in the scope of $\mathcal{G}$.

Confidentiality of transmission ($\mathsf{R}_{20}, \mathsf{R}_{21}, \mathsf{R}_{22}$) can only be verified at runtime for their verification involves mechanisms for checking that the confidentiality of the transmitted document is preserved.

Table 6.6: Confidentiality security requirements: design-time verification against a variant $\mathcal{V}_M$

No.	Requirement	Verification at design-time
R_{27}	need-to-know$(A_1, A_2, \mathcal{I}, \mathcal{G})$	$\nexists$reads/modifies/produces$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$ and $G \notin \mathcal{G}$
R_{28}	non-reading$(A_1, A_2, \mathcal{I})$	$\nexists$reads$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{30}	non-production$(A_1, A_2, \mathcal{I})$	$\nexists$produces$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{31}	non-disclosure$(A_1, A_2, \mathcal{I})$	$\nexists$transmits$(A_2, A_3, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{32}	not-reauthorised$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP})$	$\nexists$authorises$(A_2, A_3, \mathcal{I}, \mathcal{G}', \mathcal{OP}') \in \mathcal{V}_M$. $\mathcal{G}' \subseteq \mathcal{G} \wedge \mathcal{OP}' \subseteq \mathcal{OP}$
R_{20}	sender-conf$(A_2, A_1, \texttt{Tx})$	No
R_{21}	receiver-conf$(A_1, A_2, \texttt{Tx})$	No
R_{22}	system-conf$(STS_{,}, \texttt{Tx})$	No

6.2.2 Threat Analysis

The purpose of this analysis is to assess the impact of events threatening actors' assets. As introduced in Chapter 4, Section 4.4, events may threaten actors' supporting assets (subgoals and documents), to exploit their primary assets (root goals and information). Threat analysis answers the question: *how does an event threatening any actor's supporting asset affect the rest of an STS-ml model?*

To answer this question, we calculate (Algorithm 1) how the impact of the event propagates over goal decompositions, intentional relationships internal to the actor (reads, modifies, and produces), as well as the social relationships involving goals and documents (delegates and transmits).

The analysis starts with the known events, identifying the elements (supporting assets) they threaten in a STS-ml model (line 1) and propagates their impact over goal decomposition trees, documents and social relationships. Results are kept in a list (line 2). The newly discovered elements are treated as threatened elements or objects. The analysis ends when all elements have been visited and no new elements are found (lines 3 – 5). The propagation rules are the following:

1. If an event threatens a goal (lines 6 – 14), then the threat is propagated to:

ALGORITHM 1: THREAT PROPAGATION

Input: Event e

Output: List of threatened assets.

```
1  Set ⟨Element⟩ tObj ⟵ e.GETELEMENTSTHREATENEDBY();
2  Set ⟨Element⟩ result;
3  while tObj contains elements that are not visited do
4      obj = tObj.GETNEXTUNVISITEDELEMENT();
5      obj.SETVISITED();
6      if obj.GETTYPE()=Goal then
7          foreach parent in obj.GETGOALPARENTS() do
8              if parent.GETTYPE()=AND then
9                  tObj += {parent};
10                 tObj += obj.GETDELEGATEDFROM();
11                 tObj += obj.GETDOCUMENTSPRODUCED();
12             end
13         end
14     end
15     if obj.GETTYPE() = Document then
16         tObj += obj.GETGOALSMODIFYING() + obj.GETGOALSREADING() +
                   obj.GETTRANSMISSIONSTO();
17     end
18 end
19 return result;
```

- the parent goal if the goal is an AND-subgoal (lines 8 and 9);
- the goal of the delegator if the goal is being delegated (line 10);
- documents being produced by the goal if any (line 11).

2. If an event threatens a document (lines 15 – 17), then the threat is propagated to the documents read or modified by the goal, as well as to the receiver's document (in case of a transmission).

6.3 Chapter Summary

Modelling languages are useful means to represent knowledge, but as they grow in size, the chance that they become inconsistent increases. STS-ml models are no exception to the rule. The detection and handling of conflicts between requirements is a hard task Finkelstein et al. [1994] due to the increasing size and complexity of models. This is true in STS-ml too: the gained expressiveness and the richer set of supported security requirements come with a price, conflict identification cannot be performed by looking at the models.

Handling these conflicts is crucial to avoid going to the design phase having an inconsistent specification of security requirements that cannot be satisfied by any system. Automated reasoning techniques come to help in identifying inconsistencies and eventual errors in the models.

In this chapter we presented the formal framework which defines the various elements of an STS-ml model in order to build the formal specification of a given STS-ml model, to be used as input to automated reasoning techniques. We introduced the automated reasoning techniques supported by STS methodology, spanning identification of security requirements conflicts, conficts among actors business policies and the security requirements they have to comply with, and determining the threat trace of events threatening actors' assets. These activities are important to avoid developing a system that violates some security requirements. Knowledge about threats and their impact over stakeholders' assets being spread across the socio-technical system is crucial to anticipate and ideally avoid potential problems in a running system.

The details of the implementation of these techniques will be provided in Chapter 7, while the results of the findings of the automated reasoning techniques applied to the healthcare motivating scenario will be shown in Chapter 9.

Chapter 7

Tool supported security requirements engineering: STS-Tool

STS-Tool is the modelling and analysis support tool for the STS methodology. As such, the modelling and analysis activities in STS-Tool are guided by the STS process. The tool supports modelling a socio-technical system using STS-ml's constructs and the supported set of security requirements types (described in Chapter 4) to derive the list of security requirements once the modelling is done. It also supports analysing the created STS-ml models in terms of (i) well-formedness, (ii) violation of security requirements, and (iii) threats impact over actors' assets following the formal framework presented in Chapter 6.

We present the architecture of STS-Tool together with its main features and provide technical details of the modelling and analysis capabilities. STS-Tool's modular architecture is presented in Section 7.1; Section 7.2 provides details on the installation, while the tool's main features are presented in Section 7.3.

Acknowledgment. This chapter revises and extends [Paja et al., 2012a,b,c, 2013d,e].

7.1 STS-Tool Architecture

STS-Tool is a standalone application written in Java, and its core is based on the Eclipse RCP (Rich Client Platform) Framework. The tool is distributed as a compressed archive for multiple platforms (Windows 32 and 64 bits, Mac OS X, Linux), and is freely available for download from the dedicated STS-Tool website [1].

STS-Tool has been developed using the Java programming language, and it has been build on top of different frameworks produced by the Eclipse community, see STS-Tool's architecture in Figure 7.1. As shown in this figure, the architecture of STS-Tool is composed of three macro

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

blocks. STS-Tool's architecture is presented in a layered architectural style, indicating that the layers above make use (rely) on the layers below.

Starting from the bottom, at the first layer (see Figure 7.1), we find the *System Component* block that contains the underlying operating system (Windows, Linux or OsX) and the Java virtual machine that executes the Java code.

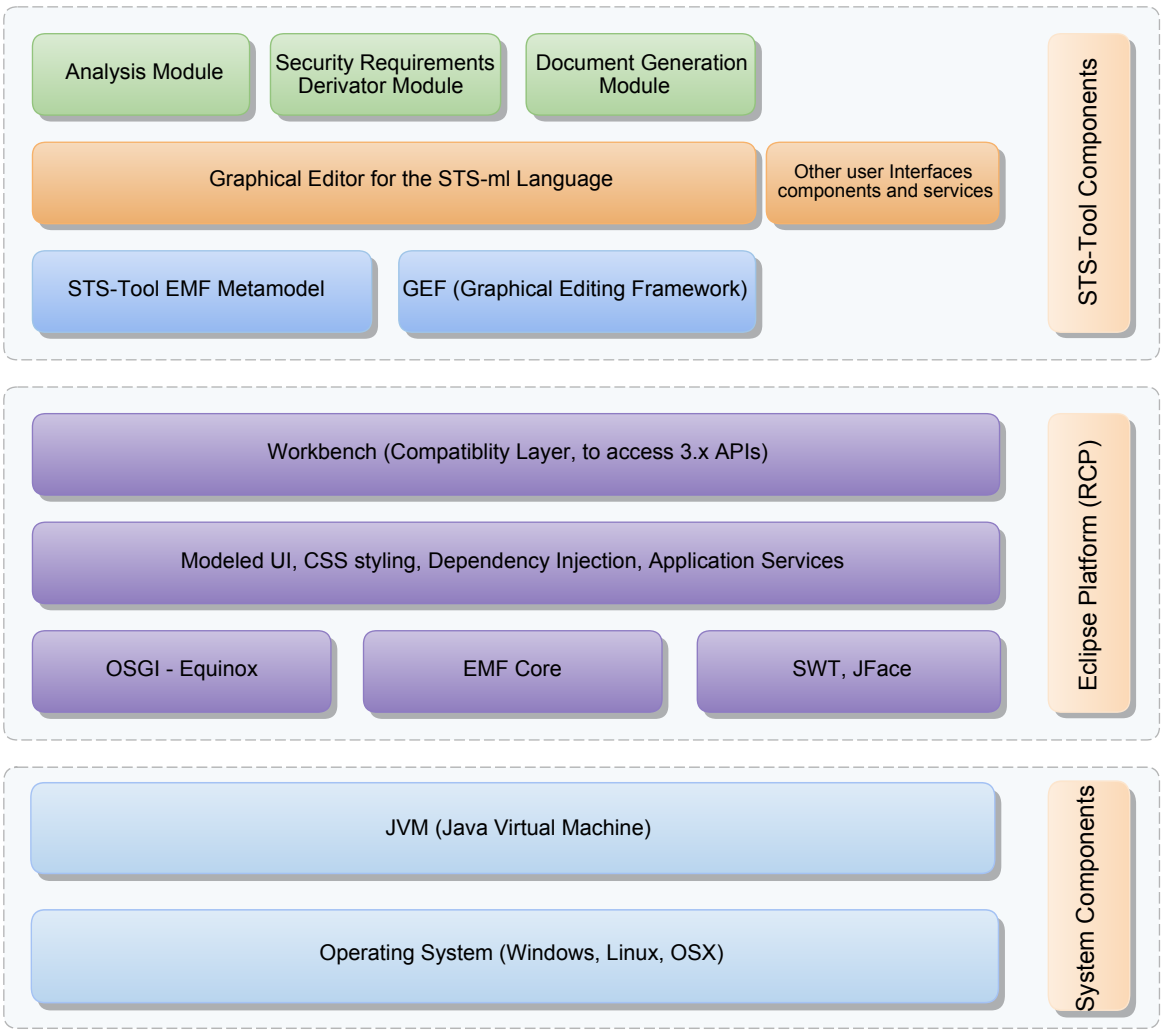


Figure 7.1: STS-Tool Architecture: Modules

At the second layer, we find the *Eclipse Platform*, also known as Eclipse Rich Client Platform (RCP). But, why was Eclipse RCP chosen to develop STS-Tool? Eclipse RCP is developed and maintained by the Eclipse community[2] and is a powerful framework for building multi-platform standalone applications. An Eclipse application consists of individual software

[2] `https://www.eclipse.org/`

components. One of the major advantages of this platform is *modularity*. To achieve this, Eclipse uses plugins. Each plugin is an independent module that provides a specific functionality inside the application, and can be easily added or replaced. Moreover, every plugin can define or consume *extensions points*; this feature allows other plugins to contribute functionality to the defined plugin. Due to the high modularity of the system it is possible to add new features with little effort and to maintain code in an easier way. Eclipse provides the SWT [Northover and Wilson, 2004] graphical library, which allows building efficient and portable applications that directly access the user-interface facilities of the operating systems it is implemented on. This revolutionary technology makes it possible to create Java-based applications that are indistinguishable from a platform's native applications. Last but not least, the Eclipse community develops a lot of parallel projects for various purposes that can be integrated to the Eclipse Platform, making the entire system more powerful. These are some of the reasons that Eclipse was chosen as the underlying platform for developing the STS-Tool.

Finally, at the third layer, we find the *STS-Tool Components*. The STS-Tool allows the user to create and modify STS-ml models (aka diagrams) described using a specific language, namely STS-ml. To support the particular specification of STS-ml, a graphical editor has been implemented using the GEF Framework [Foundation, 2014]. The STS-ml metamodel, based on EMF, is incorporated to ensure that diagrams follow the syntax of STS-ml. The rest of STS-Tool components correspond to the features it supports, such as analysis, security requirements derivation, and security requirements document generation.

We focus on the *STS-Tool Components* and provide more details over its underlying modules, in particular how the STS-Tool supports modelling (Section 7.1.1), security requirements derivation (Section 7.1.2), analysis (Section 7.1.3), and security requirements document generation (Section 7.1.4). Technical details of the implementation of STS-Tool are provided in [Paja et al., 2014c].

7.1.1 Modelling with STS-Tool: Graphical Editor for the STS-ml Language

To implement the graphical editor for the STS-ml language, the GEF Framework Foundation [2014] (see Figure 7.1 and 7.2) was chosen. The GEF Framework is an interactive Model-View-Controller (MVC) framework, which fosters the implementation of SWT–based tree editors Vogel [2013], and Draw2d–based Foundation [2014] graphical editors for the Eclipse Workbench UI Xenos [2005]. One of the challenges faced in the development of the graphical editor is related to the fact that the GEF framework is a single view editor, while the STS-Tool editor had to be a multi-view editor in order to support the multi-view modelling of STS-ml models. The problem was solved by implementing a custom multi-view editor (details in [Paja et al., 2014c]), namely the *Graphical Editor for the STS-ml Language* (see Figure 7.1), which non only supports the construction of the three models supported by STS (social, information, and

authorisation) through three corresponding views, but also considers insertion (addition) of new views in the future (should there be any).

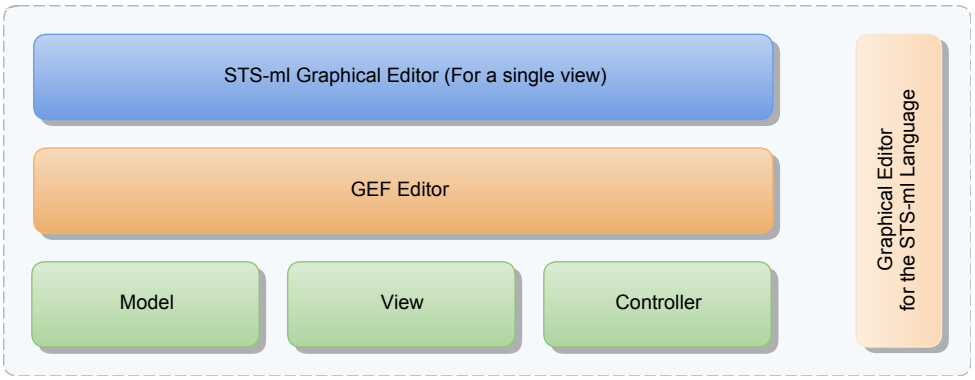


Figure 7.2: STS-Tool Graphical Editor

Apart from offering the multi-view feature, STS-Tool supports the fundamental modelling features (within each view) of the editor extending a GEF Editor. Since GEF is a MVC Framework, three main concepts should be provided for each, namely the Model, the View and the Controller (see Figure 7.2).

STS-Tool supports modelling with STS-ml, using its concepts and relationships by integrating the *STS-ml Metamodel*, see Figure 7.1. The *STS-ml Metamodel* is a custom Domain Specific Language metamodel to allow the creation of STS-ml models.

7.1.2 Security Requirements Derivator Module

The *Security Requirements Derivator* module supports the automatic generation of the security requirements as derived by the given STS-ml model. Each Element in the STS-ml Metamodel derives from the STSElement class which defines a containment relation (0..*) with an object of type ElementNeed, which keeps track of the security need specified over the element. Therefore, each Element in the model is related to its element need.

In STS-Tool, security requirements are a specialisation of the ElementNeed. They are defined in a separate plugin unitn.disi.ststool. securityrequirements, which defines the Elements Need used in the STS-ml language, and also provides a security requirement generator and a view (the security requirements tab, as in Figure 7.8) to display the evaluated and generated security requirements. Importantly, the security requirements are generated on-the-fly, while models are being drawn, as such the plugin evaluates the STS-ml model from the active editor to generate the new security requirements and update the list (also considering deletion of existing security requirements).

7.1.3 Analysis Module

STS-Tool supports analysis activities through a dedicated analysis module, namely the *Analysis Framework*, see Figure 7.3. Similarly to the other STS-Tool supported features, the analysis module has been developed and integrated through specific plugins.

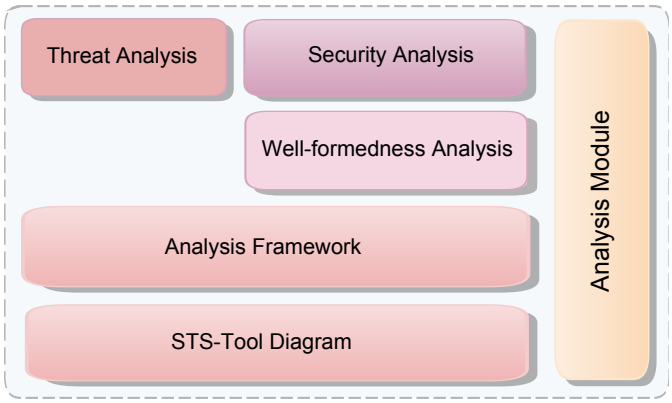


Figure 7.3: STS-Tool Automated Analysis

Security Analysis and *Threat Analysis* provide automated reasoning capabilities to be run over the created STS-ml models. Security analysis detects violations of the generated security requirements. This analysis relies on the well-formedness of STS-ml models, and thus, the *Well-formedness Analysis* module has been integrated to perform syntactical checks and verify semantic constraints over the given STS-ml models.

Since the security analysis is implemented in disjunctive Datalog (see Figure 7.4), it requires the use of Datalog program and engine (DLV Engine). But, the Datalog program is released only in native OS executables, and thus, a Java wrapper had to be implemented. To make it reusable, this wrapper was developed in a separate plugin.

The *Analysis Framework* is composed of a set of classes and interfaces that are used to standardise the execution of automated analyses. They define how the automated analysis are to be executed over the STS-ml models, the dialogs displayed to the user, the visualisation of results over the models, and so on. In this way, handling more analysis in the future requires no extra efforts with respect to this standard procedure to detect and render findings.

The execution of the security analysis is managed through a series of tasks, the main one being Violation Analysis. This task configures the Datalog Engine and is responsible of parsing results. Some other classes have been added to support this analysis:

- Predicate: this class object represent a Datalog predicate; it has a name and contains parameters that are mapped to a model object.
- Violation: a wrapper for a predicate. This class, apart from containing the predicate, con-

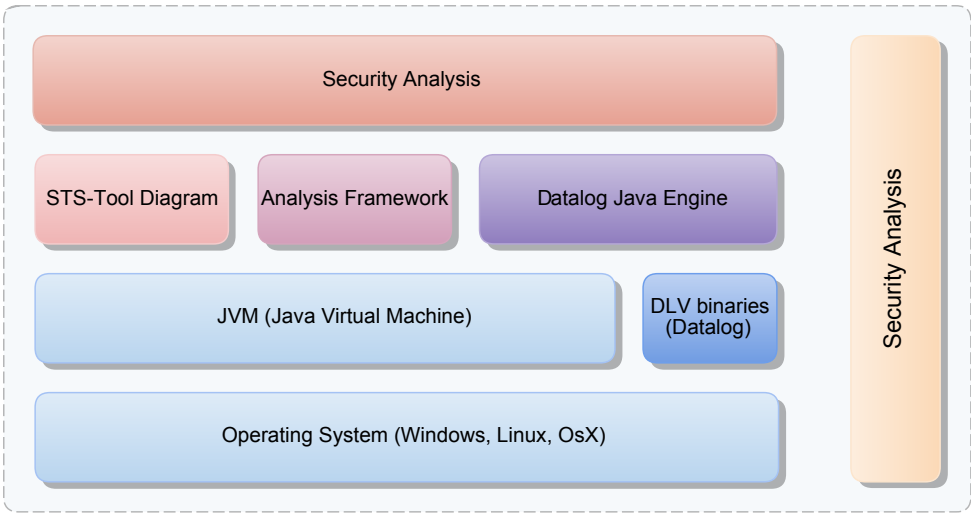


Figure 7.4: STS-Tool Security Analysis

tains also other values derived from the analysis, such as the total number of occurrences of the predicate, and the total number of models generated by the DLV Engine.

- ViolationDefinition: While Violation is used to wrap the result of the analysis, the ViolationDefinition class serves the purpose of containing the required values to discover a Violation. In particular, this class contains two attributes: (i) a predefined list of Datalog predicates that will compose the final Datalog program code, and (ii) the name of the predicate that will be generated by the Datalog program execution when a violation is found.

The schema presented in Figure 7.5 summarizes the process guiding the security analysis and the components involved in the process, which are integrated in STS-Tool. STS-ml model (*STSDiagram*, see Figure 7.5) are drawn through the STS-Tool, then automatically translated into Datalog textual files (*Diagram Parser* generates *List of Predicates* iterating over the entire model). The generated predicates are transformed one by one into String (rules for mapping each element of the model to a Datalog predicate have been specified) and the predicate values are mapped in a temporary Map (*Object Mapper*). This allows us to be able to reconstruct the predicates when the DLV output will be parsed. The DLV reasoner takes in input the generated STS-ml model file (*Datalog Input Program*) together with the Datalog rules specifying the checks performed by the security analysis (from *ViolationDefinition*) in order to get the results. The *ViolationDefinition* is set as a filter to the DLVEngine. Setting the output filter drastically reduces the output produced by the DLV program and consequently improves memory footprint and efficiency. The output parser reads the output produced by the engine, interprets it and creates instances of Predicates. When the DLVEngine completes its execution, all the generated predicates are transformed in Violations and returned to the specific Security analysis task

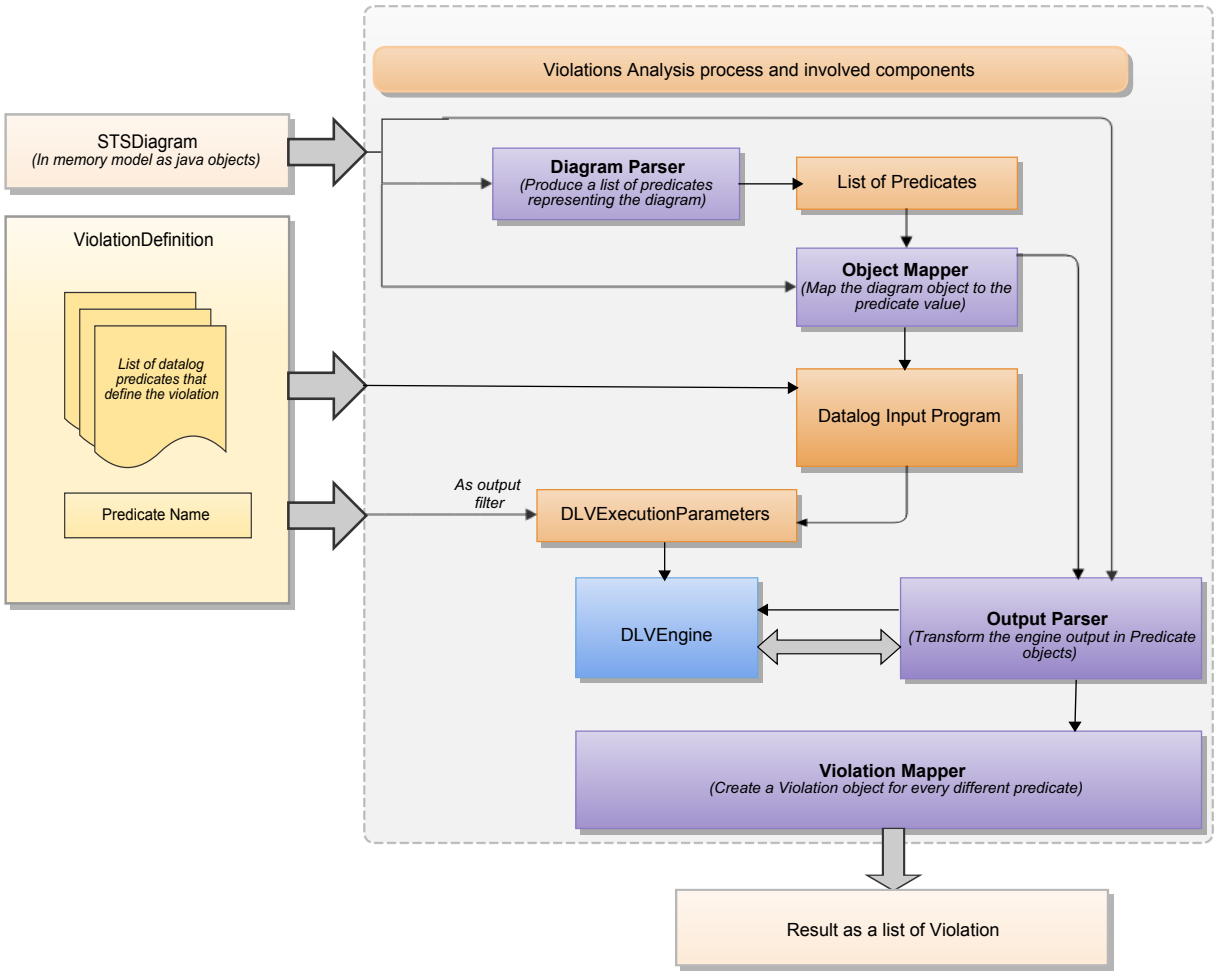


Figure 7.5: STS-Tool Security Analysis: Verification of Security Requirements Violations

that requested them, and are used to produce the analysis results. The results are then parsed (*Output Parser*) and a mapping back to STS-ml models (*Violation Mapper*) is performed in order to visualise the identified violations over the model (*Results as a list of Violation*). Results are collected, stored, and displayed into the analysis tab, as in Figures 7.10 and 7.11. Every results object has some properties: a *text* and a *description* that are used in the user interfaces to describe the result, the gravity of the result (OK,WARNING,ERROR), and a list of elements (or model object) that have to be highlighted when the user wants to display them over the STS-ml model (diagram). After the results are displayed in the analysis tab, the user can select them in order to visualise them over the diagram. Once a result is selected, this action causes a ResultManager retrieve from the selected results, which returns the list of objects that need to be highlighted and modified over the diagram (by setting a graphical property to each of this object graphical constraint), so the editor can change the displayed colour of the element to highlight it on the diagram (STS-ml model).

7.1.4 Document Generation Module

The STS-Tool supports the generation of the security requirements document through the *document generation module*, see Figure 7.6. Similarly to the analysis module, the document generation module has been distributed into multiple plugins.

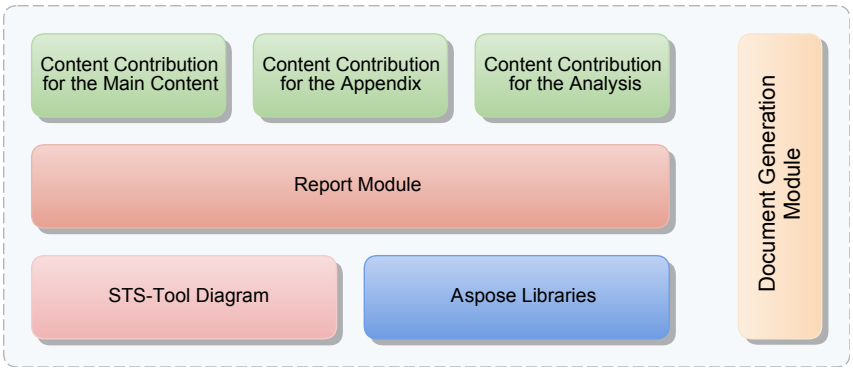


Figure 7.6: STS-Tool Security Requirements Document Generation

The Java *Aspose Libraries* allow editing and creating text documents and presentation documents in multiple formats, as well as some UI classes and a set of classes that generate the final documents. The *Report Module* defines standard document generation procedure and dialog windows to customise the document. As such, it has the same function the *Analysis Framework* has for the security analysis. While this plugin is involved in managing the creation of the documents, the content of the final document is obtained (contributed) via an extension point it.unitn.disi.ststool.documents. report.contribution that other plugins can use to add their own content. This extension point accepts multiple children of type contribution. A contribution must have a unique id, a priority value (a number) to make contributions sortable and a class that will perform the contribution (add the content of the document). Moreover the contribution type must have at least one child of type part. The part object is used in the graphical selection of the parts that can be generated, to allow customisation of the security requirements document by the analyst. This object has a unique id that can be used later in the code to retrieve the information about the selection, a name that is used in the UI, and some Boolean properties. part can also have part children to allow multi-level part-selection. While the it.unitn.disi.ststool.documents plugin contributes only to the report generation functionality, the it.unitn.disi.ststool.documents.report.analysis.security, it.unitn.disi.ststool.documents. report.analysis.wellformedness , it.unitn.disi.ststool.documents.report.securityrequirements, and it.unitn. disi.ststool.documents.report.view contribution plugins contribute to the content providing it through the extension points previously described. The content is generated analysing the STSDiagram; static code retrieves from the model the required information, caches it and at the end generates the content that has to be inserted into the document.

7.2 Installation details

The STS-Tool is distributed as a compressed archive for multiple platforms and it is free to download from the STS-Tool web site [3]. The tool is available in both source and binary form, and the license is APGL (Affere General Public License). As prerequisite, you need at least version 7 of the Java Virtual Machine. Previous versions of the tool are also available online in an Archive [4].

In order to install the STS-Tool, it is enough to download the version suitable for your machine and operating system, and then extract the content of the archive, containing the tool, to a folder on your computer. Finally, run the launcher file – no setup is needed.

The STS-Tool comes with *Online Help*. Help is produced by the Eclipse project, we provide only the content of the help. To obtain updates of the STS-Tool, one does not need to download the latest version from the website, rather an *update system* is already integrated in the tool. The update system is a customization of the Eclipse P2 (Eclipse update system). A public web site has been expressly created in order to update the new versions of the tool automatically. The STS-Tool checks for updates and if any are found, it asks the user to install them. However, the user has the choice of activating this feature (configuring updates from the menu: Windows – Preferences – Updates) or get updates manually.

7.3 STS-Tool features

STS-Tool has the following features:

1. **Modelling features**

   (a) *Specification of projects*: STS-ml models are created within the scope of project containers. A project refers to a certain scenario, and contains a set of models. Typical operations on projects are supported: create, rename, import, and export. See Figure 7.7 showing a screenshot of STS-Tool.

   (b) *Project Explorer*: an interesting feature of STS-Tool developed as a customization of the Eclipse RNF (Resource Navigator Framework). The project explorer allows the user to manage files better, organising them into folders and projects, see Figure 7.7.

   (c) *Diagrammatic modelling*: the tool enables the creation (drawing) of diagrams (models), see Figure 7.7. Diagrams are created only within a project and typical create/modify/save/load/ undo/redo operations are supported.

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

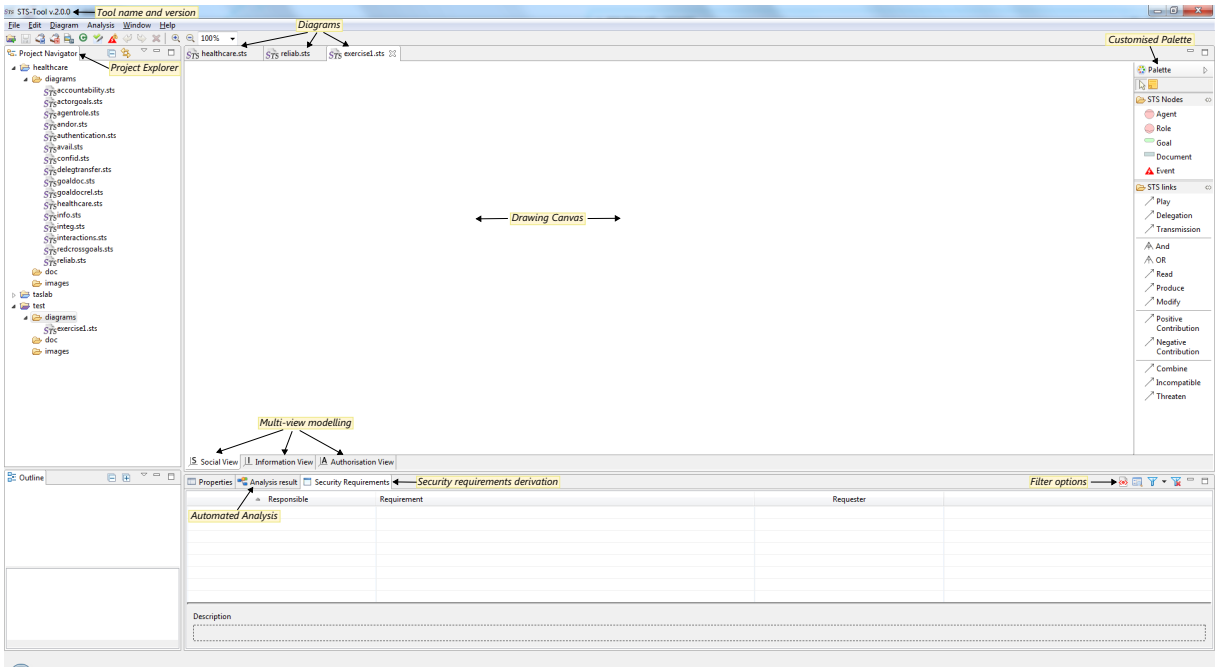


Figure 7.7: STS-Tool: Security Requirements Derivation

(d) *Multi-view modelling*: a particular feature of STS-Tool, which provides different tabs (views) in the tool to allow modelling the various models of a socio-technical system diagram, namely *social*, *information*, and *authorisation model*, following STS-ml's multi-view feature. Each tab (referred to as view) shows specific elements and hides others, while keeping always elements that serve as connection points between the models (e.g. roles and agents). STS-Tool supports separation of concerns through its multi-view modelling feature. Moving from one view to another is simple and quick. The palette of concepts and relationships is tailored (customised) for each view, grouping the concepts and relationships one needs at that particular view, see Figure 7.7. Inter-model consistency is ensured by for instance propagating insertion (deletion) of certain elements to all models (social, information, and authorisation) composing the overall STS-ml model. However, STS-Tool allows to hide specific elements from one or more of the models, should they not have specific details in that model. For instance, an actor that does not own any information and does not have any documents, nor does it manipulate any document, might be hidden (omitted) from the information view, for it does not add any details to the information view. This helps keep the models neat.

(e) *Visual scalability*: to leverage the burden of visual scalability and complexity of STS-ml models, STS-Tool allows to represent actors either collapsed (only the role/agent

notation is shown) or expanded (the whole actor model is shown, i.e., the actor together with its goals, documents and intentional relationships). It is possible to zoom in specific elements (e.g. actors). We are working towards further improving this feature, to support collapsing more STS-ml model elements.

(f) *Export diagram to different file formats*: STS-ml models (or parts of models, i.e., specific elements) can be exported to various formats, such as jpg, png, pdf, svg, etc.

2. **Security Requirements**

(a) *Derivation of security requirements*: the tool allows the automatic derivation of security requirements as relationships between a *requester* and a *responsible* actor for the satisfaction of a *requirement*. STS-Tool visually represents security requirements in a tabular form, in which they are listed and can be sorted or filtered according to their different attributes (*responsible*, *requirement*, and *requester* respectively), see Figure 7.8). For instance, filtering the requirements with respect to the responsible

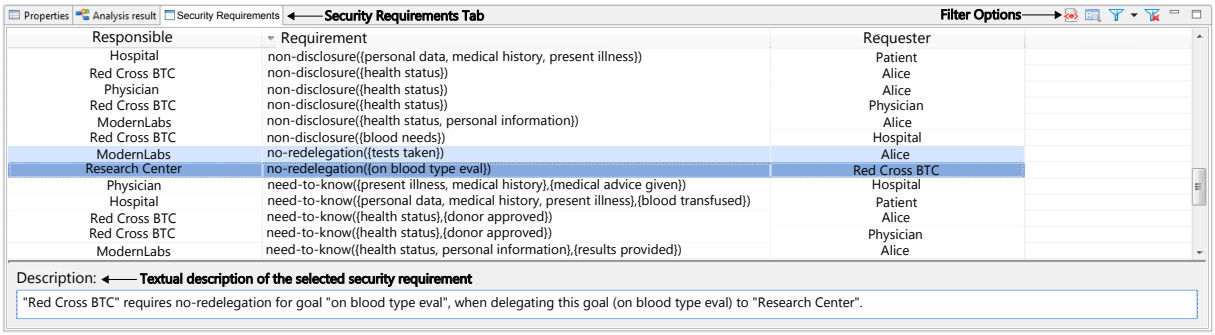
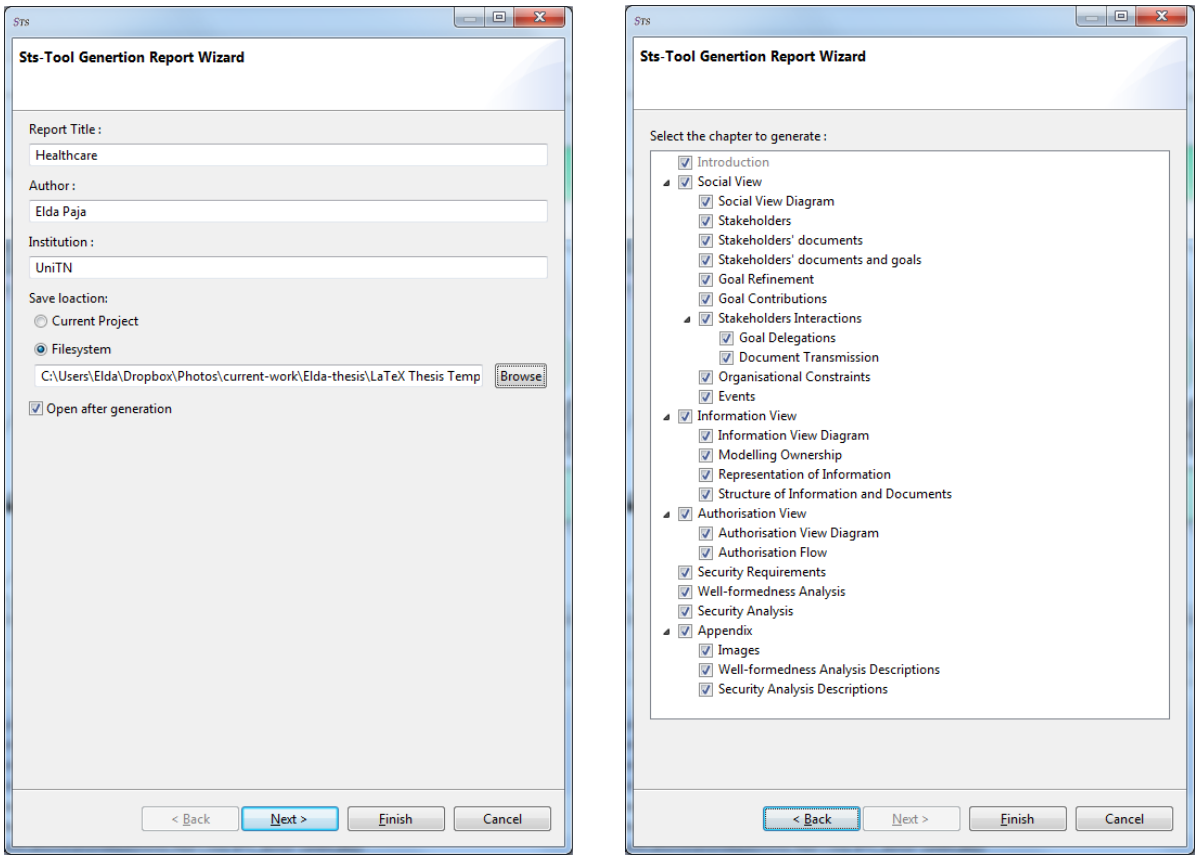


Figure 7.8: STS-Tool: Security Requirements Derivation

actor, gives an idea of who are the actors responsible to fulfil the security requirements, and what security requirements each has to satisfy. On the other hand, filtering security requirements according to their requirement type (as we have done in Figure 7.8), groups together security requirements that should be satisfied to fulfil a certain security requirement type. Finally, a textual *description* is provided for every *security requirement*, which is displayed below the list of security requirements when selected.

(b) *Generation of requirements documents*: the tool allows the automatic generation of a security requirements document that contains the list of security requirements derived from the model. This document contains information describing the models, information that is customisable by the designer (by choosing which model features to include, such as for instance including only a subset of the actors, focusing on one

view only, selecting only concepts or relations the involved parties want more information about, and so on). Anyhow, the overall document provides a short description of STS-ml and Tool and communicates security requirements by providing details of each STS-ml view, together with their elements. The diagrams are explained in detail providing textual and tabular descriptions of the models.



(a) (b)

Figure 7.9: Customising the security requirements document

In generating the document, the user may change the title it wants to give to the document, the author and organisation by editing the fields in Figure 7.9a. The document is organised in sections, which the designer can decide to include or not in the generated document (see Figure 7.9b). It is good practice to generate the requirements document at the end of the modelling, and after refining the models in order to fix eventual errors detected by the automated analyses. However, this document could be generated at any point of the modelling process, to improve the models based on the extra detail this document provides to the security requirements engineers.

The security requirements document is helpful especially when communicating with stakeholders, for it provides details about the different elements of the diagram, the discovered violations, for which negotiation with stakeholders might be necessary (in order to fix the identified problems).

3. **Analysis support**

   (a) *Automated reasoning*: STS-Tool integrates and supports the two automated reasoning techniques (*security analysis* and *threat analysis*) presented in Chapter 6. Security analysis is based on the formal framework presented in Section 6.1 and verifies the conflicts presented in Section 6.2.1, which are implemented in Datalog (the detailed rules are presented in Appendix B). Threat analysis, on the other hand, implements Algorithm 1 introduced in Section 6.2.2. Note that the execution of automated reasoning is to be performed over well-formed models. We verify well-formedness following Definition 7. The verification is practically performed in two steps, depending on the complexity of the check:

      i. *Online well-formedness checking*: the tool checks the models for syntactic well-formedness on-the-fly (*online*) while it is being drawn. For example, the tool does not allow drawing certain relationships, such as for instance a delegations leading to delegation cycles, part-of cycles, etc.;

      ii. *Offline well-formedness checking*: some validity rules are either too computationally expensive for online verification, or their continuous verification would limit the flexibility of the modelling activities. Thus, some checks about validity are performed upon explicit user request (e.g., authorisation duplicates). These checks are embedded within STS-Tool's *well-formedness analysis*.

   Security analysis and *threat analysis* are performed upon request of the end-user (security requirements engineer).

   (b) *Visualisation of analyses' results*: the tool visualises the results of the analyses (per each analysis) and provides details of the findings. Results are enumerated in a tabular form below the diagram (Analysis tab, Figure 7.10) and rendered visible on the STS-ml model itself when selected. A textual description provides details on the selected analysis result.

      i. *Security analysis* returns the list of security requirements conflicts and violations of security requirements. In Figure 7.10, we have executed security analysis over the healthcare case study and show some of the detected violations of security requirements. One of the violations regards not-redelegation. As the description of the selected violation explains, this violation occurs because *Alice* has expressed a not-redelegation security requirement over the delegation of goal *tests*

taken to *ModernLabs*, and the latter delegates goal *drug test performed* to *Drug Tests Inc*, goal that is the and-subgoal of *tests taken*.

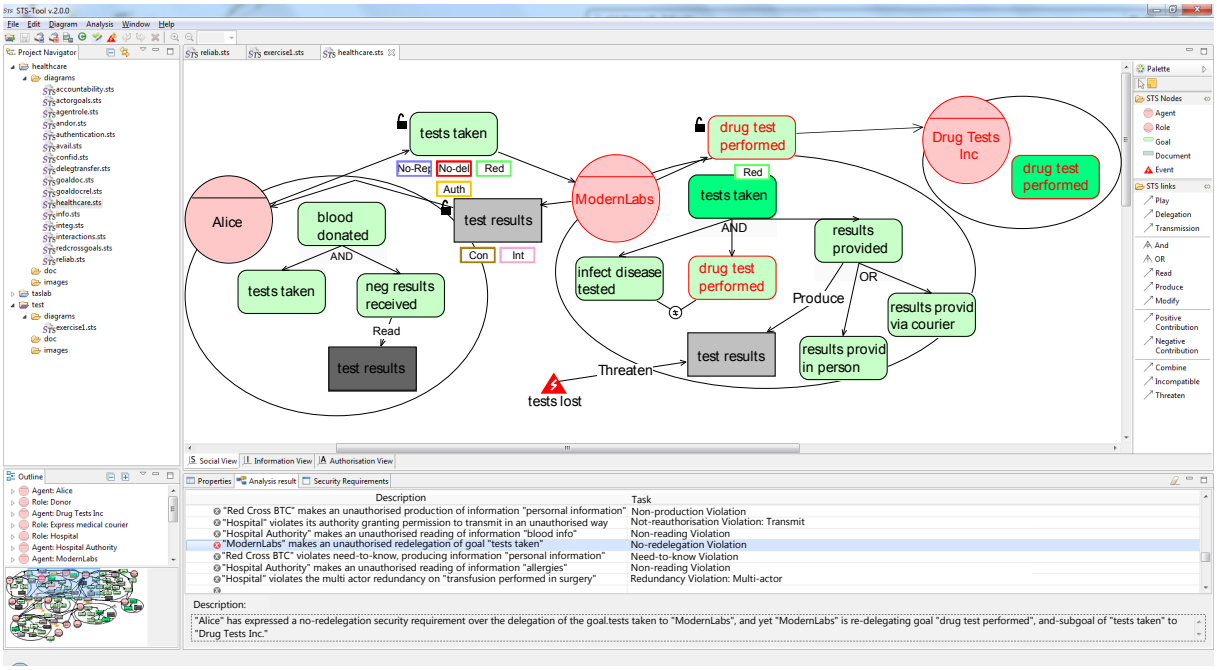


Figure 7.10: Executing security analysis: visualisation of results

ii. *Threat analysis* returns the trace of the impact the event, threatening an actor's asset, has throughout the socio-technical system. The trace contains all concepts and relationships affected by the threatening event, and is visualised as a whole when selected. In Figure 7.11, we show the impact of event *tests lost*, threatening *ModernLabs*' document *test results*, has on *Alice*, which is one of the actors interacting with *ModernLabs*. The latter transmits to *Alice* document *test results*, and since this document is threatened, then *Alice*'s goal *neg results received* is threatened too, since it needs to read *test results*. Similarly, goal *blood donated* is threatened since its and-subgoal (*neg results received*) is threatened.

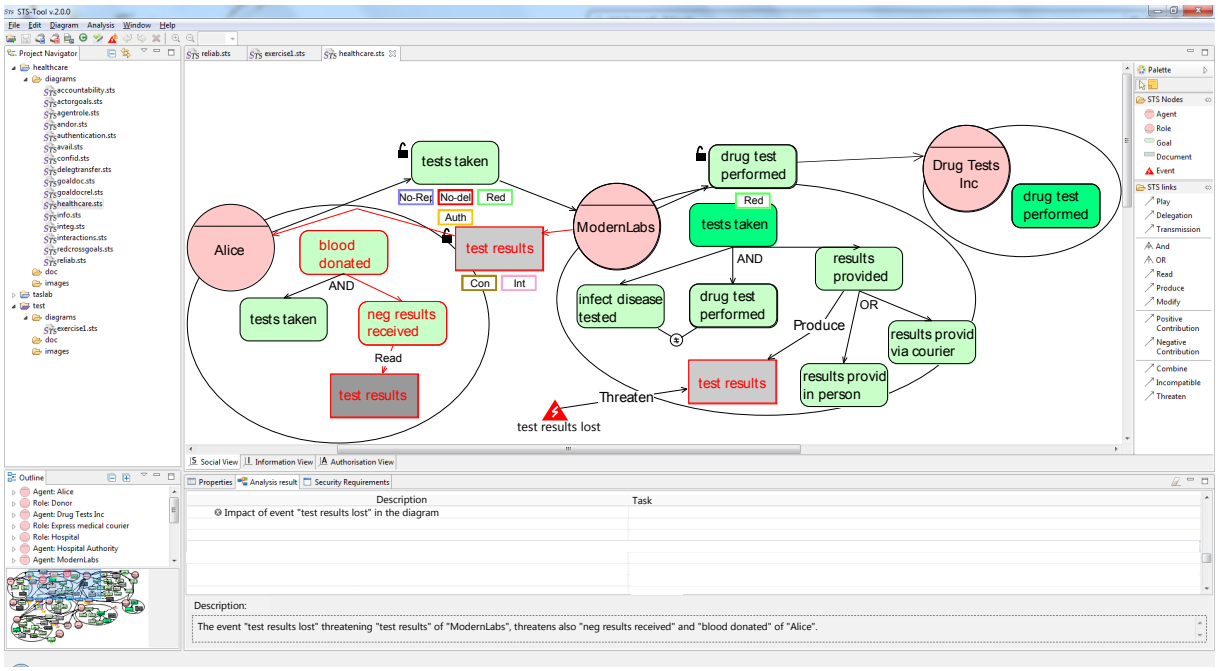


Figure 7.11: Executing threat analysis: visualisation of results

7.4 Chapter Summary

The STS-Tool is quite a stable graphical requirements engineering modelling tool. It is based on the Eclipse RCP/EMF/GMF frameworks and supports modelling and reasoning over the created models. The tool was developed as part of the FP7 EU-funded project Aniketos, and therefore, has involved many parties. However, the actual implementation was done by the team of University of Trento, having a dedicated programmer working on the development. My role in the development of the tool, apart from the proposal of the modelling language and methodology, has been that of designing the tool and developing the automated reasoning techniques in Disjunctive Datalog.

The latest version of the tool is the result of an iterative development process: we will present in Chapter 8 how STS-Tool was proved suitable to model and reason over models of a large size from different domains [Paja et al., 2013b], such as eGovernment and Air Traffic Management Control; while in Chapter 9 we will present results of the evaluation activities conducted with practitioners [Trösterer et al., 2012] in the scope of the EU FP7 Project Aniketos, evaluations that have guided the development and improvement of STS-Tool.

The tool has been presented in many events, not only by us, but also by our industrial partners, and has received very good comments. To this day, we amount to two thousand downloads of the tool, and have received feedback from many researchers in academia, apart from industrial partners.

Chapter 8

Application Scenario and Case Studies

This chapter presents the use of STS methodology in practice over an application scenario (via self-evaluation), as well as over two case studies from different domains. The objective is that of assessing the adequacy of STS methodology and the STS-ml modelling language (its primitives) to model scenarios from different domains while capturing security requirements for the considered socio-technical systems. Self-evaluation serves the purpose of assessing the *effectiveness* of the automated reasoning capabilities for discovering authorisation conflicts and security requirements violations. Therefore, in this chapter, we describe a self-evaluation activity conducted with the help on an application scenario in Section 8.1. The scenario is used also to perform a scalability study: running experiments (analysis) to demonstrate the *efficiency* of the automated reasoning techniques supported by STS (Section 8.2).

We describe the results and obtained feedback from practitioners who have modelled two case studies (Section 8.3.1 and Section 8.3.3) following STS methodology, applying it step by step in modelling and reasoning about security requirements. We present the outcomes of the modelling and reasoning activities for each, in Sections 8.1.1, 8.3.2 and 8.3.4 respectively.

Acknowledgement. Section 8.3.1 builds on top of [Paja et al., 2014a,b].

8.1 Self-evaluation Case Study: Trentino as a Lab

Trentino as a Lab (TasLab) [1] is an online collaborative platform to foster ICT innovation in the Trentino province [Shvaiko et al., 2010]. It aims at creating a community of research institutions, universities, enterprises and public administration, which collaborate in research-intensive ICT projects. TasLab provides information on local innovation trends, events, investment opportunities. It also offers an area where users can match innovation demand (from local government and municipalities) with innovation supply (by enterprises and research insti-

[1] http://www.taslab.eu

tutions), and they can collaboratively write project proposals.

We focus on a TasLab collaborative project about tax collection. This scenario is based on the project *Taxpayer's Knowledge Base* [2] and has been developed through discussions and interviews with Informatica Trentina.

The innovation demand comes from the Autonomous Province of Trento (*PAT*) and the Trentino *Tax Agency*, which require a system that verifies if correct revenues are gathered from *Citizens* and *Organisations*, provides a complete profile of taxpayers, generates reports, and enables online tax payments.

This is an example of a socio-technical system in which multiple actors interact with one another and with technical systems: citizens and organisations pay taxes online; municipalities (*Municipality*) furnish information about citizens, their addresses, and tax payments; Informatica Trentina (*InfoTN*) is the system contractor; other IT companies develop specific functionalities (e.g., data polishing, search modules); the Tax Agency is the system end user; and PAT withholds the land register in its database (information about buildings and lots).

These actors exchange confidential information and interact for processing such information. Each actor has its own business policy, i.e., goals achieved through processes that manipulate information, and expects others to comply with its security requirements, e.g., about integrity and confidentiality. For example, citizens require that their income tax data is treated confidentially by the municipality and it is not disclosed to third parties. The tax collection system is not monolithic: its operation depends by the successful interaction among taxpayers, the municipality, InfoTN, the Tax Agency, and the TasLab website.

Citizens are concerned with the protection of the confidentiality of their income declaration (information asset). PAT is concerned with ensuring that the Tax Agency successfully handles the tax verification process, i.e., fulfills goal *tax verified* (intentional asset). Our analysis should highlight participants' relevant assets, being these informational or intentional assets. Every participant/stakeholder has its own security needs, which constrain the way they would like others to behave when it comes to the assets they want to protect. For example, citizens allow the municipality to use their data for tax verification purposes, but they do not want the involvement of third parties.

8.1.1 Applying STS methodology to the TasLab application scenario

We apply STS methodology to the TasLab case study, following its modelling, automated analysis and specification phases. The activities are supported by STS-Tool, and therefore, we emphasise how the tool facilitates these activities while constructing and analysing the STS-ml models for the given scenario.

[2] `http://www.openlivinglabs.eu/livinglab/trentino-lab`

The modelling and analysis activities have been directly conducted by myself, however, they were performed through iterations and interactions with a representative of Informatica Trentina. In particular some prototype modelling took place in collaboration with the representative during the discussion, before the actual modelling with STS-Tool was performed. This was done to avoid inaccuracies in the model in representing the given application scenario.

Phase 1. Social modelling

Following the STS methodology, we start modelling activities with social modelling (refer to Figure 3.2, Chapter 3). To build the social model, we consider answering these questions: "*Which are the stakeholders we can identify from the tax collection scenario?*", "*How can they be represented in terms of roles and agents?*", "*What are the goals they have and how are these goals achieved?*", "*What are the documents actors have and manipulate (read, modify, produce)?*", "*What are the social interactions actors participate?*", "*What are their security expectations (needs)?*", "*Are there any events threatening stakeholders' assets?*", and so on.

Step 1.1. Identify stakeholders. As described in Chapter 4, stakeholders in STS-ml are represented via agents and roles. Therefore, we identify stakeholders in the TasLab case study and represent them in terms of agents and roles. For instance, in Figure 8.1, we have modelled *Informatica Trentina* (InfoTN) as an agent, knowing already at design time that it will be part of the system, while we modelled TN Company Selector as a role, not knowing yet which company will take over this responsibility, but knowing only the responsibilities encapsulated in this role. Following the same logic, we model PAT, BP Engineering Srl, and Okkam Srl as agents, while we represent Tax Agency as a role, see Figure 8.1.

Step 1.2. Identify assets and interactions. We identify for each modelled actor its *intentional assets*, aka its goals, as well as its *informational assets*, aka its documents (since we are in the social model). This step is important also to built actor models for each identified actor. For this we need to identify actors' intentional relationships. For instance, InfoTN wants to achieve goal online system built, for which it has to achieve goals search module built, navigation module built, and system maintained (*and-decomposes* relationship among these goals); the achievement of the latter goal requires the achievement of goals data completeness ensured and data files stored. InfoTN possesses document local copy of data, it reads document high quality data to have data completeness ensured, while it modifies personal records to achieve goal data refined, see Figure 8.1.

Tax Agency has the goal revenue system maintained, which it and-decomposes into goals tax verification performed, tax details verified, data collected, and consultancy offered; goal tax verification performed is further and-decomposed into historic maintained, payers record created,

due taxes calculated, and data completeness ensured; goal data collected requires reading document tax payers′ knowledge base; goal payers record created produces document payers record and it is or-decomposed into goals corporate records created and citizens′ records created; finally goal due taxes calculated requires reading document high quality data.

Similarly, we construct the actor models for the other actors represented in Figure 8.1.

STS-Tool support: When first created, roles and agents come together with their rationale (open compartment), so that we can specify the goals or documents (assets) they have. The rationales can be hidden or expanded, to give the designer the possibility to focus on some role or agent at a time. We place actor goals within their rationale. STS-Tool facilitates a correct modelling of goal trees, by not allowing goal cycles. Several checks are performed live by the tool for this purpose, such as not permitting the designer to draw a decomposition link from a subgoal to a higher level goal in a goal tree. Moreover, the tool helps the designer by allowing goal-document intentional relationships to be drawn only starting from the goal to the document, not vice-versa.

Notice that both InfoTN and the Tax Agency do not have from the beginning all the documents they need to read or modify in order to achieve their goals. For this they have to rely on other actors. Additionally, not all the goals of InfoTN are its intended goals, some have been delegated to InfoTN from other actors. Thus, let us now consider actors' social interactions, supported in STS-ml via social relationships.

In the TasLab case study, we do not have any knowledge at design time about agents adopting the identified roles, i.e., there are no examples of *play* in Figure 8.1. An example of *goal delegation* is that of Tax Agency delegating goal data completeness ensured to InfoTN . Moreover, InfoTN delegates goal search module built to TN Company Selector, while it delegates goal cadastre data verified to PAT .

STS-Tool support: When drawing a delegation, the tool makes sure that the actor does have a goal it wants to pursue, before allowing the designer to draw the goal delegation relationship starting from the given actor. It is worth emphasising that STS-ml allows only the delegation of leaf goals, delegation of upper level goals is forbidden, and STS-Tool support this. If a leaf goal is delegated and the designer decides to further decompose this goal within the delegator, the tool will prompt him with a message and not allow the further decomposition. Once the goal delegation relationship is drawn, the delegated goal is automatically created within the compartment of the delegatee. This goal is represented in a darker color than the original goal, in order to clearly distinguish for each actor its own goals from the goals delegated to it. Additionally, the tool does not allow the goal to be deleted from the delegatee's compartment unless the delegation is deleted. Importantly, delegation cycles are not permitted by the tool.

An example of *document transmission*, on the other hand, is that of InfoTN transmitting the document high quality data, which it received from TN Company Selector, to Tax Agency, see Figure 8.1. Moreover, PAT transmits document cadastre registry to InfoTN.

Step 1.3. Express security needs. STS-ml allows actors to express security needs over their interactions, in this step we analyze these interactions (goal delegations and document transmissions) actors participate in to elicit their needs with regard to security. Additionally, security needs derived from organisational constraints are modelled too.

STS-Tool support: To specify security needs using STS-Tool, the designer needs to right-click on the delegated goal, to have a drop down list of security needs and select the desired ones. The selection of at least one security need, shows a padlock on the goal or document. The selected security need can be shown explicitly by clicking on the padlock, which shows small boxes below the delegated goal or provided document; each box has a different colour and a different label (see Figure 8.1).

Over goal delegations. Let us consider InfoTN and analyse the security needs it expresses over the goal delegations it participates: in Figure 8.1, InfoTN wants Tax Agency not to repudiate the delegation of goal data completeness ensured; it requires TN Company Selector to ensure true redundancy single for goal data refined; it requires TN Company Selector not to redelegate goal search module built; it imposes a trustworthiness requirement when delegating tax contributions obtained to Municipality; and it requires the Municipality to guarantee an availability level of 95% for the goal tax contributions obtained. The delegation of goal data refined from InfoTN to TN Company Selector includes a delegator authentication requirement; the delegation of goal corporate data verified from InfoTN to PAT, on the other hand, includes a delegatee authentication requirement: InfoTN wants to ensure that the verification is performed by the same person that maintains the data record.

Over document transmissions. Okkam Srl requires Tax Agency not to repudiate the acceptance of transmission of document tax payers knowledge base, see Figure 8.1; Tax Agency requires InfoTN (the sender) to guarantee the transmission integrity of high quality data; InfoTN (receiver) requires TN Company Selector an availability level of 70% for the document high quality data; InfoTN expresses the need that the transmission of document high quality data necessitates the receiver's authentication, while Okkam Srl expresses the requirement that the transmission of document tax payers knowledge base necessitates the sender's authentication. An example of sender confidentiality of transmission is specified from InfoTN to Municipality: Municipality shall ensure the confidentiality of transmission of document tax contributions file is preserved when transmitting it to InfoTN.

Over responsibility uptake. An example of goal-sod is that between goals corporate records created and citizens′ records created of Tax Agency, see Figure 8.1, while an example of goal-cod is that between goals semantic search built and enterprise search built of TN Company Selector, see Figure 8.1. There are no examples of separation of duty or combination of duty between roles in this scenario.

Step 1.4. Modelling threats. In the TasLab tax collection scenario we have identified two events threatening stakeholders assets: the event auditor sick threatens goal corporate data verified, while the event data lost threatens document high quality data of OkkamSrl (see Figure 8.1).

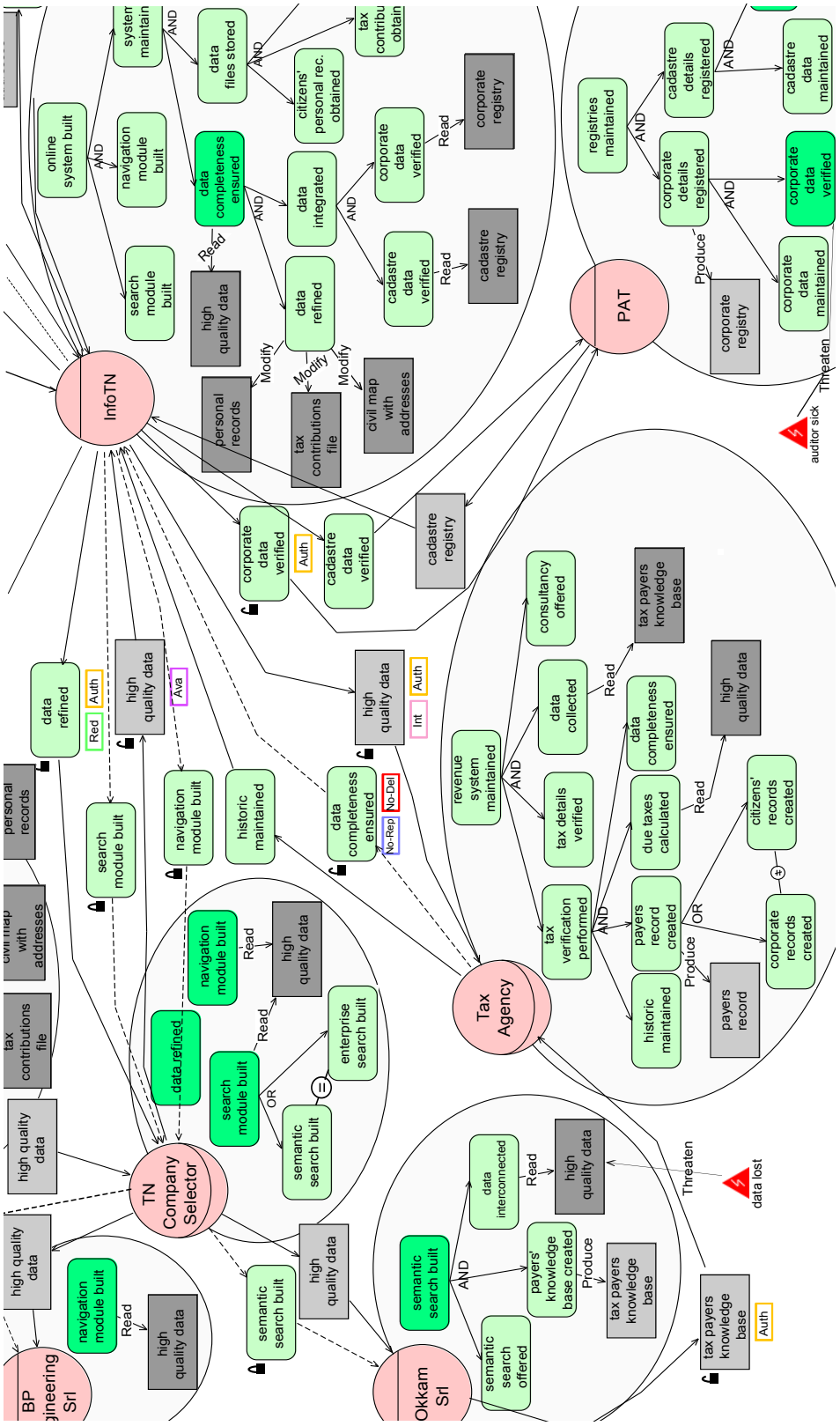


Figure 8.1: Partial STS-ml social model of the tax collection scenario

Phase 2. Information modelling

We continue modelling activities with the second phase, information modelling (see Figure 3.2, Chapter 3). To build the information model, we consider answering these questions: "*What is the informational content of the documents represented in the social view?*", "*Who are the owners of this information?*", "*What is the structure of information?*", "*Is there a structure of documents?*".

Step 2.1. Identify information/ownership. We consider the modelled actors and identify their informational assets, connecting the latter to the actors that own them [3]. For instance, Citizen is the owner of information personal info, while land details, location and fiscal code are information entities owned by PAT (see Figure 8.2). Municipality owns information residential address and tax contributions.

Information is represented via documents. For instance, information personal info is made tangible by Citizen's personal data, while information location is made tangible by document residential buildings (see Figure 8.2).

Information can be represented by one or more documents (through multiple Tangible By relationships). For instance, information personal info is made tangible not only by document personal data, but also by document local copy of data of InfoTN, and document personal records of Municipality.

On the other hand, one or more information entities can be made tangible by the same document. For instance, information fiscal code together with tax contributions is made tangible by document corporate registry, see Figure 8.2.

STS-Tool support: STS-Tool allows the relation owns to be drawn starting from the role or agent towards the information it owns, and the relationship Tangible By to be drawn only starting from information to documents.

Step 2.2. Structure information. Another feature of the information model is to support composite information (documents). The structuring of information and documents is done via Part Of relationships, allowing designers to build a hierarchy of information entities and documents, respectively. For instance, this allows representing that information location is part of the information residential address, while document land lots is part of document cadastre registry in Figure 8.2.

[3] Recall that STS-ml supports multiple (shared) ownership, as such there can be multiple owners for the same information.

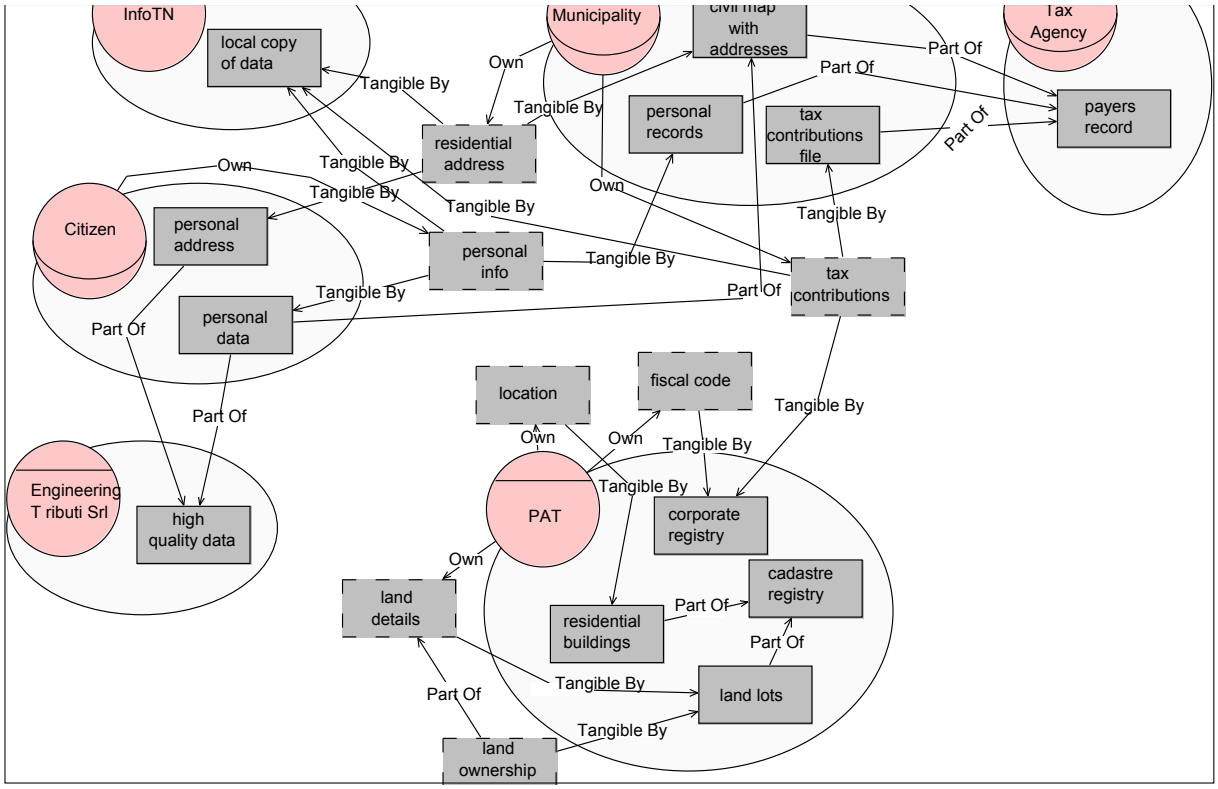


Figure 8.2: Partial STS-ml information model of the tax collection scenario

STS-Tool support: The tool helps the designer in building this structure by allowing the Part Of relations to be drawn only between information or documents respectively. Additionally, cycles of Part Of are not allowed by the tool.

Phase 3. Authorisation modelling

We continue modelling activities with the third phase, authorisation modelling (see Figure 3.2, Chapter 3). To build the authorisation model, we consider answering these questions: "*Are there any authorisations granted from the information owners?*", "*Is authority to transfer authorisations granted?*", "*Which are the information for which authorisation is granted?*", "*Are there any limitations of authority?*".

Step 3.1. Model authorisations. In the TasLab case study, Figure 8.3, Municipality authorises InfoTN to read information personal info, tax contributions, and residential address, but it prohibits any modification of such information, in the scope of goal system maintained, while granting a transferable authorisation.

Authorisation modelling supports the specification of security requirements. The authorisa-

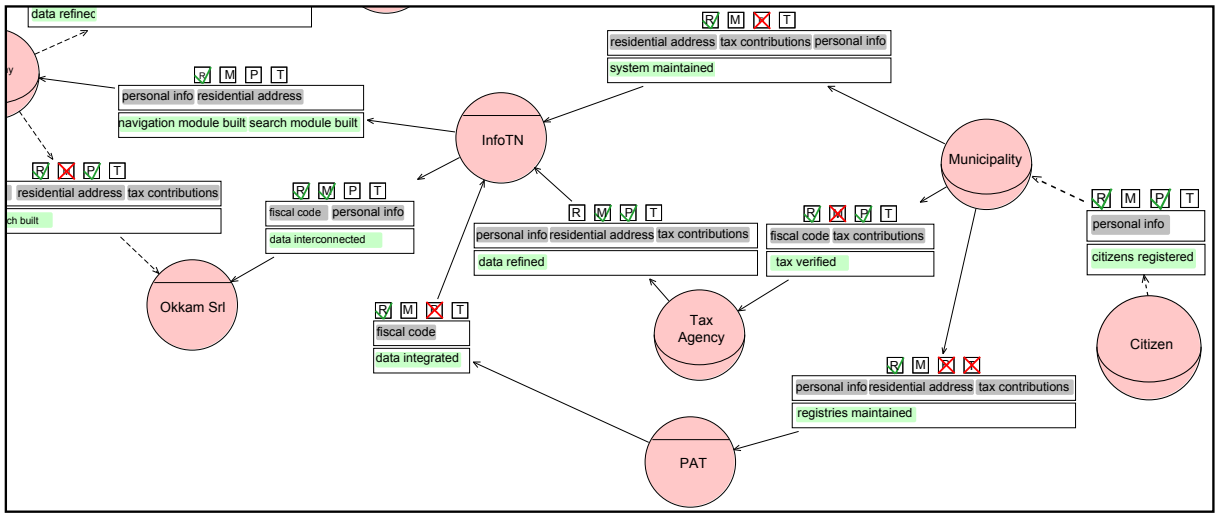


Figure 8.3: Partial STS-ml authorisation model of the tax collection scenario

tion from Tax Agency to InfoTN is an example of need-to-know: personal info, residential address and tax contributions shall be read only for goal data refined, see Figure 8.3. There is no example of non-reading in the TasLab case study, for authority to read has not been explicitly prohibited to any actor. Municipality requires the Tax Agency not to modify fiscal code and tax contributions, see Figure 8.3 where authority to modify is prohibited. PAT expresses a non-production requirement on fiscal codes to InfoTN, by prohibiting the production operation, see Figure 8.3. Non-disclosure is, for instance, required by Municipality to PAT when authorising the latter to read information personal info, residential address and tax contributions, but prohibiting the right to produce and to transmit. An example of explicit non-reauthorisation is the authorisation from Citizen to Municipality, see Figure 8.3, since the authorisation is non-transferrable. while an example of implicit non-reauthorisation is, for instance, the case of Tax Agency, which has no incoming authorisation over information personal info.

STS-Tool support: The tool helps the designer in specifying authorisations, once the relationship has been drawn between two actors, via tooltips on specifying permissions and prohibitions, as well as inputing the information by double-clicking in the second slot.

Phase 4. Automated Analysis

We run the automated analyses supported by STS-Tool over the created models to assess (i) the ability of verifying the well-formedness of the STS-ml model, (ii) the ability of the reasoning mechanisms to discover authorisation conflicts and security requirements violations rising as a result of conflicts between business policies and security requirements (*Bus-Sec conflict*), as well as (iii) the ability to calculate the impact of events threatening stakeholders' assets.

Step 4.1. Wellformedness Analysis. The well-formedness analysis has found no errors over the STS-ml model for the TasLab case study. We should emphasise, however, that STS-Tool helps the construction of well-formed models, and the dimensions of the modelled scenario are not as big as to go unnoticed by the security requirements engineer.

Step 4.2. Security Analysis. We provide evidence by presenting the findings from the application of the STS-Tool in the modelling and analysis of the TasLab case study. Based on the models that we created with the stakeholders (Figures 8.1, 8.2, 8.3), the analysis returned a number of conflicts that we had not identified during the modelling, including:

- *On authority to produce*: Tax Agency authorises InfoTN to produce documents with information personal info, residential address and tax contributions to obtain refined data, whereas Municipality authorises reading only, and requires not production of the same information, see Figure 8.4 visualising this conflict.
- *On authority to modify*: InfoTN grants Okkam Srl the authority to modify documents with information personal info to obtain interconnected data, whereas TN Company Selector requires no document representing this information is modified.

These conflicts, which went unnoticed while modelling, originate from the different authorisation policies of the stakeholders. Conflict resolution activities can be used to ultimately reach a consistent model. The former conflict can be resolved by negotiating the provision of adequate rights with the Municipality, while the latter can be fixed by revoking the authorisation, given that Okkam Srl does not need it (from the social view).

After fixing authorisation conflicts, we used the tool's capabilities to identify *Bus-Sec conflicts*. This activity provided us with further useful insights:

- no-redelegation: InfoTN relies on TN Company Selector to refine the data obtained (delegation of data refined). On the other hand, Tax Agency relies on InfoTN to ensure data completeness (delegation of data completeness ensured) and requires it not to redelegate this goal. This security requirement is in conflict with the business policy on delegating data refined, since the later is a subgoal of data completeness ensured, for which no-redelegation is required.

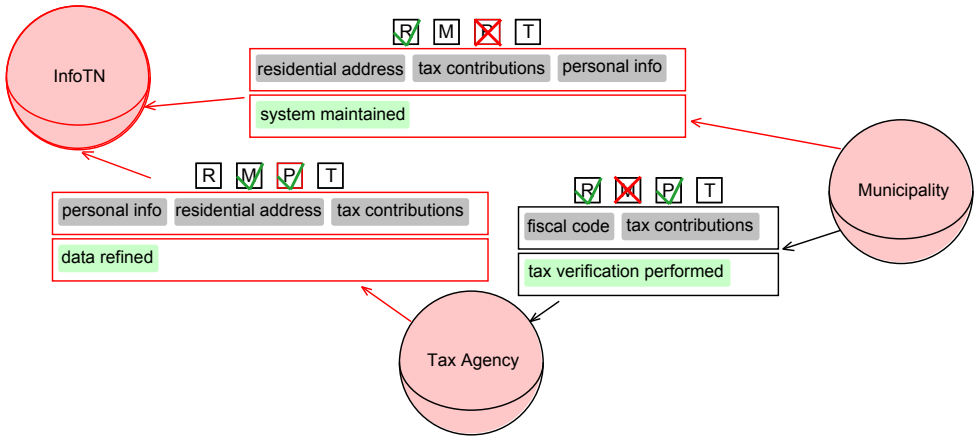


Figure 8.4: Authorisation conflict towards InfoTN on authority to produce

- true-multi-red: TN Company Selector has not employed more strategies on its own to fulfil the goal data refined, for which InfoTN has required multi actor true redundancy.
- non-production: PAT makes an unauthorised production of tax contributions, for this information is owned by the Municipality and the authorisation to produce is prohibited to PAT.
- not-reauthorised: Tax Agency has no authority to modify information location, however Tax Agency further authorises InfoTN to modify this information, see Figure 8.4, which capture this conflict too.
- goal-cod: goals semantic search built and enterprise search built should be pursued by the same actor, since a combination of duties is specified between these goals. A conflict occurs because TN Company Selector is not the final performer for both goals (semantic search built is delegated to Okkam Srl).

These conflicts are due to the different policies of the companies. They can be resolved through trade-offs Elahi and Yu [2007] between business policies and security requirements. Notice that relaxation is often not an option, especially if a requirement derives from norms in the legal context.

Step 4.3. Threat Analysis. We execute the threat analysis over the two events modelled in Step1.4. We provide evidence by presenting the findings from the application of the STS-Tool threat analysis of the TasLab case study. The finding of the analysis are as follows:

- The event data lost threatening document high quality data of Okkam Srl, threatens goal data interconnected that requires reading this document; as a result, it threatens goal

semantic search built for which data interconnected is an and-subgoal; semantic search built has been delegated to Okkam Srl by Tn Company Selector, and thus Tn Company Selector's goal semantic search built is threatened too.

- The event auditor sick threatening goal corporate data verified of PAT, threatens also this goal's father corporate details registered, which, in turn, affects the document corporate registry and its father goal registries maintained; goal corporate data verified is delegated from InfoTN, and therefore InfoTN's goal corporate data verified is threatened; this latter goal is a leaf goal in the goal tree (all and-decompositions) of InfoTN, as such it affects goals data integrated, data completeness ensured, system maintained, and online system built; document corporate registry produced by PAT is transmitted to InfoTN, and therefore it affects InfoTN's document corporate registry too; InfoTN's goal data completeness ensured has been delegated to InfoTN by the Tax Agency, and thus, Tax Agency's goal is threatened too; the effects of this are that goal tax verification performed and revenue system maintained are threatened too.

In a nutshell, looking at these examples we can see how the impact in one end is propagated throughout all parts of the system.

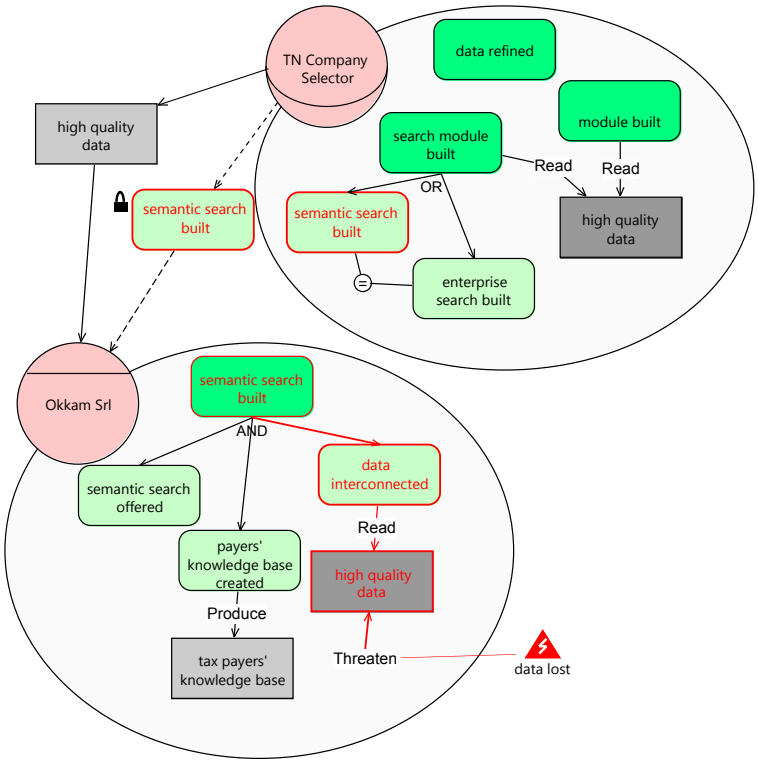
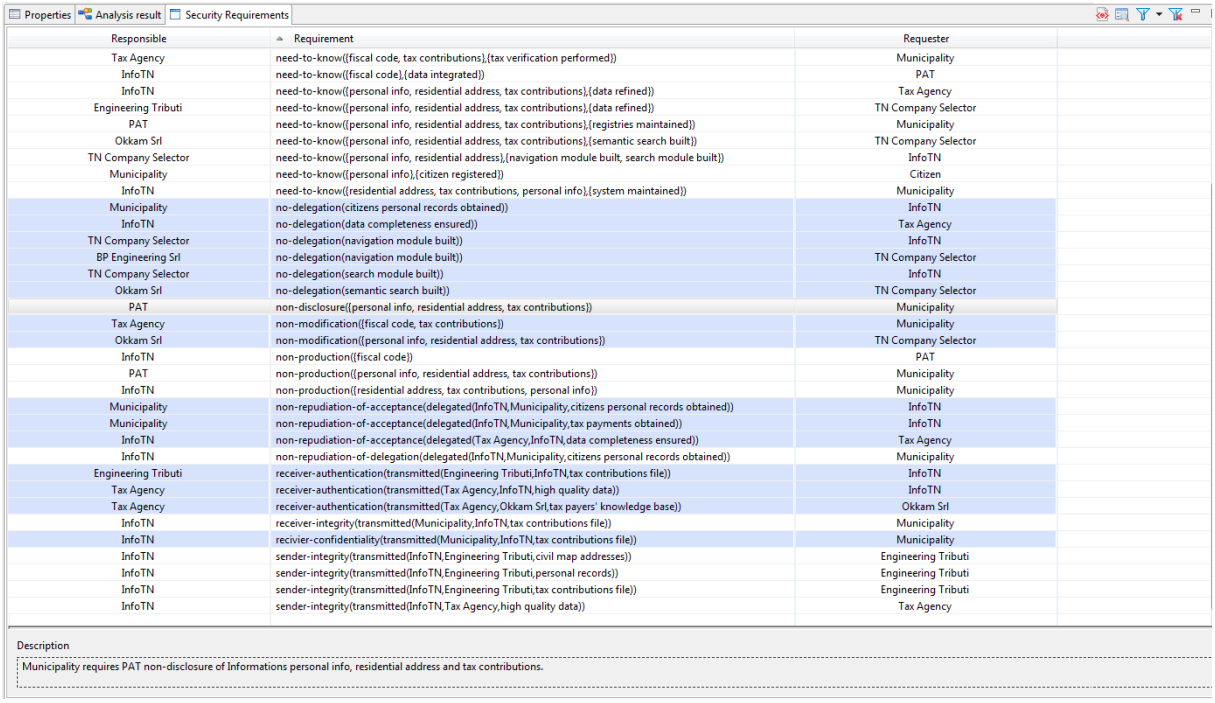


Figure 8.5: TasLab case study—Threat analysis results for the event data lost

For simplicity, we show only the visualisation of the results of the first threat impact, see Figure 8.5.

Phase 5. Specification

The security requirements specification is generated with the help of STS-Tool, as described in Chapter 7. We report the security requirements for the online tax collection and verification scenario in Figure 8.6.


Properties | Analysis result | Security Requirements

Responsible	Requirement	Requester
Tax Agency	need-to-know({fiscal code, tax contributions},{tax verification performed})	Municipality
InfoTN	need-to-know({fiscal code},{data integrated})	PAT
InfoTN	need-to-know({personal info, residential address, tax contributions},{data refined})	Tax Agency
Engineering Tributi	need-to-know({personal info, residential address, tax contributions},{data refined})	TN Company Selector
PAT	need-to-know({personal info, residential address, tax contributions},{registries maintained})	Municipality
Okkam Srl	need-to-know({personal info, residential address, tax contributions},{semantic search built})	TN Company Selector
TN Company Selector	need-to-know({personal info, residential address},{navigation module built, search module built})	InfoTN
Municipality	need-to-know({personal info},{citizen registered})	Citizen
InfoTN	need-to-know({residential address, tax contributions, personal info},{system maintained})	Municipality
Municipality	no-delegation(citizens personal records obtained))	InfoTN
InfoTN	no-delegation(data completeness ensured))	Tax Agency
TN Company Selector	no-delegation(navigation module built))	InfoTN
BP Engineering Srl	no-delegation(navigation module built))	TN Company Selector
TN Company Selector	no-delegation(search module built))	InfoTN
Okkam Srl	no-delegation(semantic search built))	TN Company Selector
PAT	non-disclosure({personal info, residential address, tax contributions})	Municipality
Tax Agency	non-modification({fiscal code, tax contributions})	Municipality
Okkam Srl	non-modification({personal info, residential address, tax contributions})	TN Company Selector
InfoTN	non-production({fiscal code})	PAT
PAT	non-production({personal info, residential address, tax contributions})	Municipality
InfoTN	non-production({residential address, tax contributions, personal info})	Municipality
Municipality	non-repudiation-of-acceptance(delegated(InfoTN,Municipality,citizens personal records obtained))	InfoTN
Municipality	non-repudiation-of-acceptance(delegated(InfoTN,Municipality,tax payments obtained))	InfoTN
InfoTN	non-repudiation-of-acceptance(delegated(Tax Agency,InfoTN,data completeness ensured))	Tax Agency
InfoTN	non-repudiation-of-delegation(delegated(InfoTN,Municipality,citizens personal records obtained))	Municipality
Engineering Tributi	receiver-authentication(transmitted(Engineering Tributi,InfoTN,tax contributions file))	InfoTN
Tax Agency	receiver-authentication(transmitted(Tax Agency,InfoTN,high quality data))	InfoTN
Tax Agency	receiver-authentication(transmitted(Tax Agency,Okkam Srl,tax payers' knowledge base))	Okkam Srl
InfoTN	receiver-integrity(transmitted(Municipality,InfoTN,tax contributions file))	Municipality
InfoTN	recivier-confidentiality(transmitted(Municipality,InfoTN,tax contributions file))	Municipality
InfoTN	sender-integrity(transmitted(InfoTN,Engineering Tributi,civil map addresses))	Engineering Tributi
InfoTN	sender-integrity(transmitted(InfoTN,Engineering Tributi,personal records))	Engineering Tributi
InfoTN	sender-integrity(transmitted(InfoTN,Engineering Tributi,tax contributions file))	Engineering Tributi
InfoTN	sender-integrity(transmitted(InfoTN,Tax Agency,high quality data))	Tax Agency

Description

Municipality requires PAT non-disclosure of Informations personal info, residential address and tax contributions.

Figure 8.6: List of security requirements for the TasLab case study

This outcome is useful for the security requirements engineer to identify security requirements for each category (type), in order to determine what requirements should be fulfilled to, for instance, comply with non-disclosure security requirements. For this, we have ordered the derived security requirements with respect to the *requirement* attribute, and identified one security requirement of the type non-disclosure, for which PAT is responsible: *Municipality requires PAT non-disclosure of Informations personal info, residential address and tax contributions.*, see Figure 8.6 highlighting this security requirement. Similarly, we can check requirements for need-to-know, authentication, and so on, for all the types of derived security requirements.

8.2 Scalability study

We report a scalability study to asses how well the automated reasoning techniques cope with large STS-ml models. Specifically, we study how the execution time is affected by the model size.

8.2.1 Design of experiments

We select a model to serve as a basic building block, and clone it to obtain larger models. For the conducted experiments, we chose the model of the TasLab case study. However, any of the other models could be used, for the other models are about the same size.

We increase the size of a model in two ways: first, we augment the *number of elements* (nodes and relationships) in the model; second, we increase the *number of variants* in the model. The latter strategy is motivated by our reasoning techniques, which rely upon the generation of STS-ml model variants (Definition 12, Chapter 6) for identifying possible conflicts between actors' business policies and security requirements in the model.

To obtain larger versions of an STS-ml model M, we performed the following steps:

1. create an identical copy (clone) M' of the given STS-ml model M;
2. add a fictitious leaf goal G to a randomly chosen actor A in M;
3. delegate G to the clone of A (actor A') in M';
4. decompose (see below for the decomposition type) the goal G of A' into (a) the root goal of its existing goal model, and (b) a fictitious goal.

This process increases the number of variants, for the initial model M already contains variability. We repeat these steps to obtain larger and larger models.

In order to independently test the effect of increasing the number of elements and the number of variants, we adopted three different customisations of the procedure above:

- *no-variability*: all decompositions in M and M' are treated as AND-decomposition, thus leading to one variant;
- *medium-variability*: goal G of A' is AND-decomposed, all other decomposition are not modified;
- *high-variability*: goal G of A' is OR-decomposed, all other decomposition are not modified.

The first strategy (no-variability) enables assessing scalability with respect to the number of elements, while the latter two strategies (medium- and high-variability) are meant to assess scalability with respect to the number of variants.

We conduct three sets of tests with models of increasing size generated using each of the strategies above (no-, medium-, and high-variability). For each cloned model, we run the analysis 7 times.

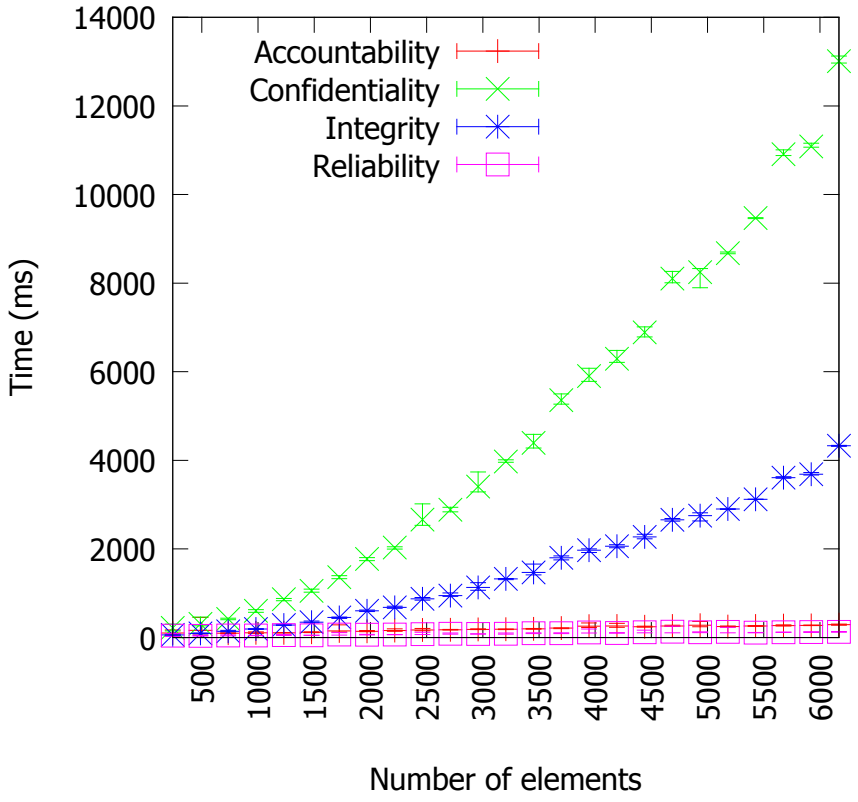


Figure 8.7: Scalability analysis for *no-variability*: y-axis in linear scale

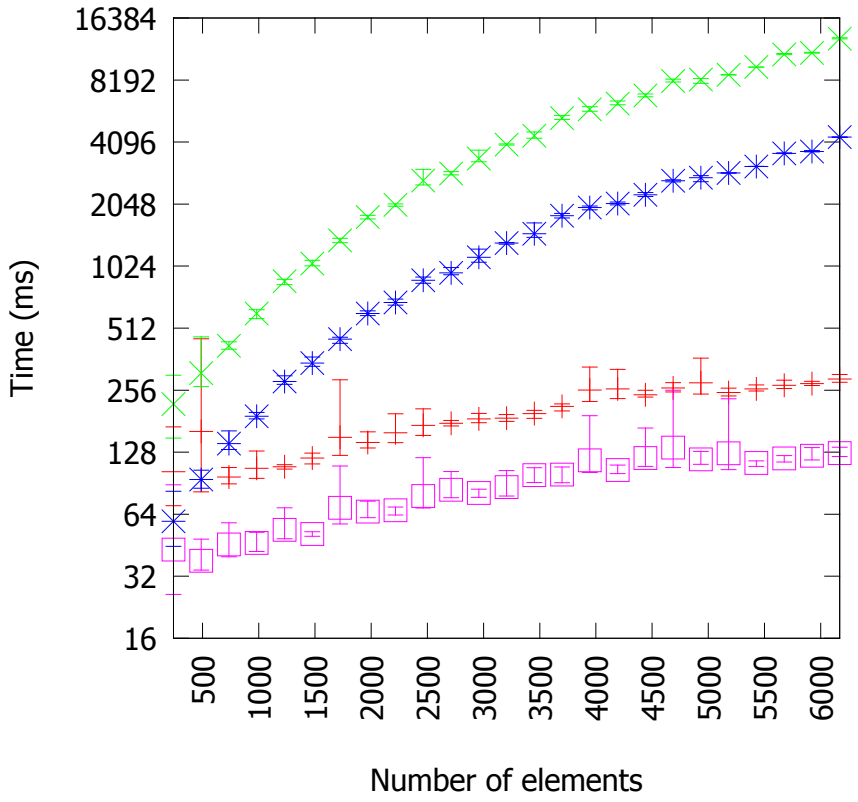


Figure 8.8: Scalability analysis for *no-variability*: y-axis in logarithmic scale

8.2.2 Results

We have conducted experiments on a DELL Optiplex 780 machine, Pentium(R) Dual-Core CPU E5500 2.80GHz, 4Gb DDR3 399, powered by Windows 7. Figure 8.7–8.10 summarise the results of our scalability experiments. Below, we detail and discuss the results for each set of tests.

No-variability

The scalability graphs are shown in Figure 8.7 and 8.8, and are split according to the time spent to analyse requirements of the different categories in Tables 6.1–6.6 that can be verified at requirements time, aka security requirements pertaining to accountability, confidentiality, integrity, and reliability. We present the y-axis both in linear scale and logarithmic scale; the latter visualisation serves to take a closer look at reliability and accountability, which take considerably less time than the other types of requirements. As noticeable by the plot, all techniques scale very well (linear or quasi-linear growth). Interestingly, the tool is able to reason about extra-large models (>6000 elements) in about twelve seconds. We conclude that the number of

elements does not constitute an obstacle to the scalability of our techniques.

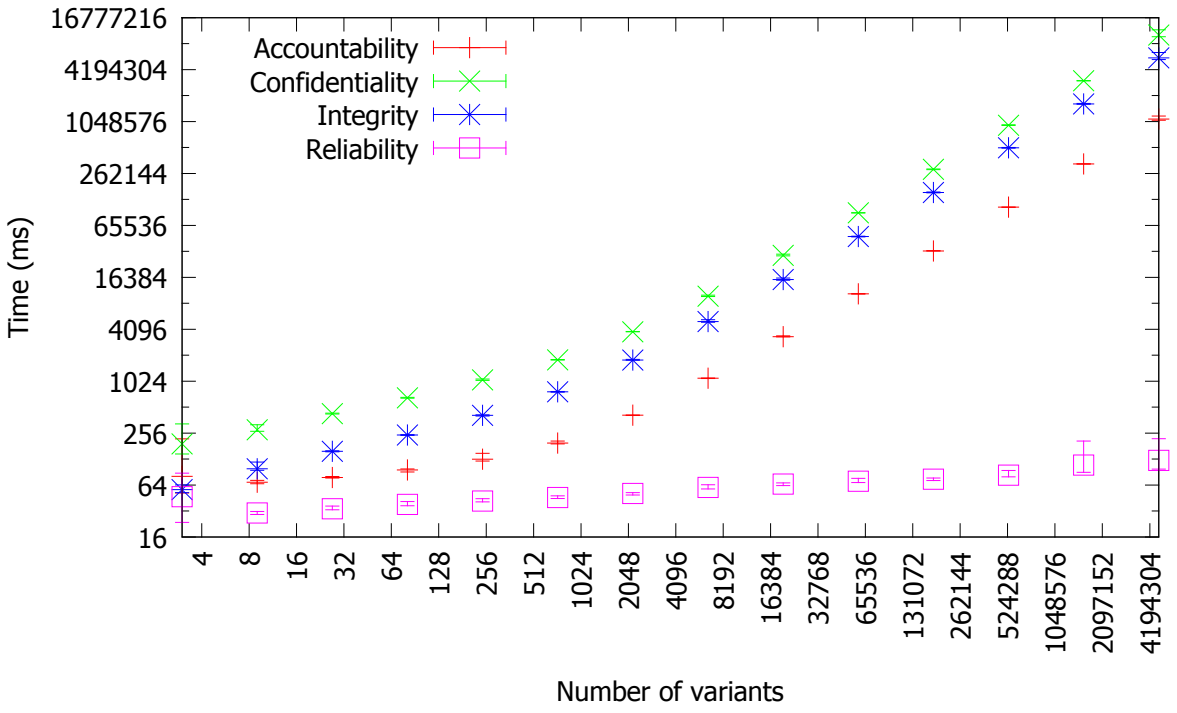


Figure 8.9: Scalability analysis for *medium-variability*

Medium- and high-variability

The graphs in Figure 8.9 and Figure 8.10 present (in $\log_2$ scale) the results of the scalability analysis with respect to the number of variants. This dimension affects the execution time of our techniques the most. The analysis of reliability requirements is not affected by the increasing number of variants: the reason for this is that our tool processes this type of requirements before increasing the number of variants. As far as the other types of requirements are concerned, the growth is still linear in the number of variants, but it is exponential in the number of elements (the model with 1,048,576 variants in Figure 8.9 consists of 2,500 elements). Notice that the tool deals with dozens of thousands of variants in less than a minute. In this case, we conclude that the tool is adequate to deal with large models, but needs to be improved to deal with extra-large models.

In general, the results are promising: the number of variants in all the modelled real world scenarios from the case studies was significantly smaller than in the models obtained via cloning.

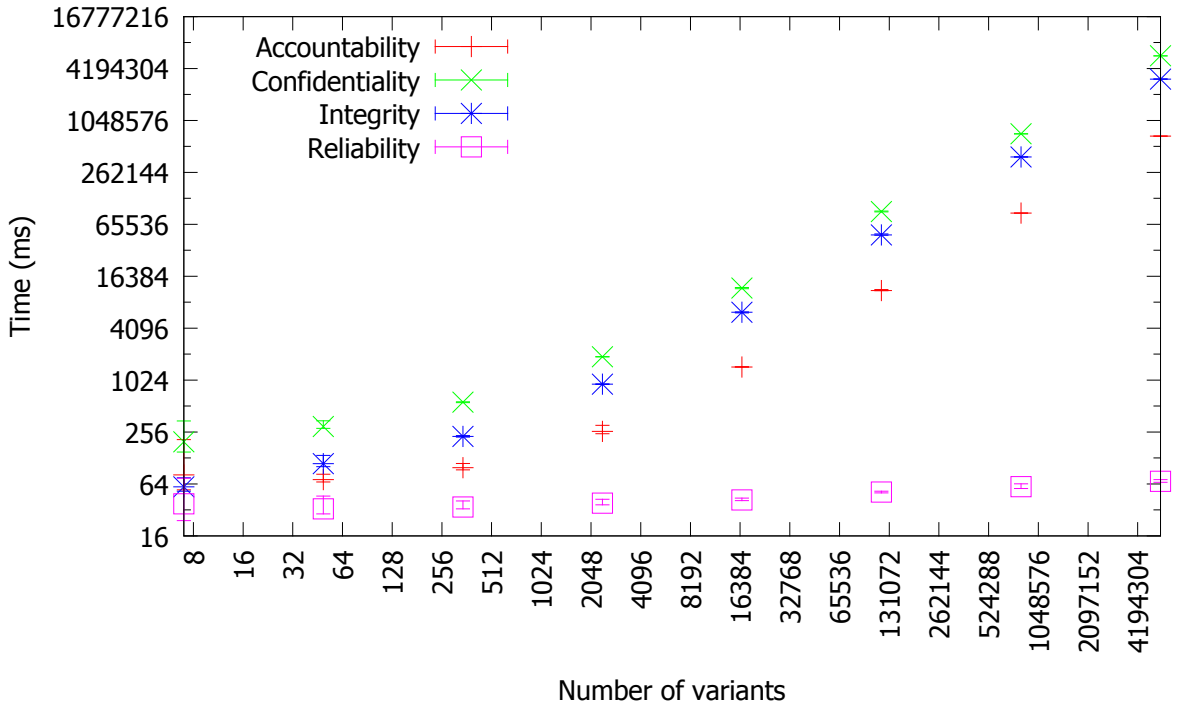


Figure 8.10: Scalability analysis for *high-variability*

8.3 Case Studies

We present here two case studies that have been developed by the industrial partners of the FP7 EU Funded Project Aniketos [4].

8.3.1 Case Study 1: eGovernment

We consider here a scenario from the eGovernment case study, which is a variant of the "*Land-buying and e-Governance*" case study of the Aniketos project. This case study is reported in Deliverable 6.4 of the Aniketos project.

The Department of Urban Planning (DoUP) wants to build an application which integrates the existing back-office system with the available commercial services to facilitate the interaction of involved parties when searching for a lot. The *Lot Owner* wants to sell the lot, he defines the lot location and may rely on a Real Estate Agency (*REA*) to sell the lot. *REA* then creates the lot record with all the lot details, and has the responsibility to publish the lot record together with additional legal information arising from the current Legal Framework. *Ministry of Law* publishes the accompanying law on building terms for the lot. The *Interested Party* is searching for a lot and: (i) accesses the DoUP application to invoke services offered by the various REAs; (ii) defines a trustworthiness requirement to allow only trusted REAs to contact him; (iii) sets a criteria to search and select a *Solicitor* and a *Civil Engineer* (CE) to asses the conditions of the lot; (iv) assigns solicitor and CE to act on his behalf so that the lot information is available for evaluation; and (v) populates the lot selection for the chosen CE and Solicitor. *Aggregated REA* defines the list of trusted sources to be used to search candidate lots, it collects candidate lots from trusted sources, and ranks them to visualize to the user. *The Chambers* provide the list of creditable professionals (CE and Solicitors).

8.3.2 Applying STS methodology to the eGovernment case study

We have explained in detail the application of the steps of the STS methodology for Case Study 1, here we limit ourselves to provide the results of the modelling and automated analysis supported by the STS methodology.

The reported models were created by industrial partners from DAEM [5], who are domain experts, and then revised by us, method designers, reading the supporting documentation and talking to domain experts. Thus, the presented models have undergone a two iterations, one conducted by pracitioners, and the other conducted by method designers. More details on this eperience and the collected feedback are provided in Chapter 9.

[4] `http://www.aniketos.eu`
[5] `http://www.daem.gr/`

Phase 1. Social modelling

Step 1.1. Identify stakeholders. The identified roles are Lot Owner, Real Estate Agency, Map Service Provider, Interested Party, Solicitor, CE Chambers, and Solicitor Chambers, while the represented agents are: DoUP Application, Aggregated REA, and Ministry of Law, see Figure 8.11. The reason for this is that *roles* refer to general actors that are instantiated at run time, while *agents* refer to concrete entities already known at design time. That is, we do not know who Lot Owner or Interested Party is going to be, but we consider that there is only one Aggregated REA and one Ministry of Law in this scenario, which are known already at design time.

Step 1.2. Identify assets and interactions. In Figure 8.11, Lot Owner has goal lot sold. He could sell the lot either privately or through an agency. Therefore, Lot Owner or-decomposes lot sold into lot sold privately and lot sold via agency. In the Lot searching scenario, we consider that the Lot Owner interacts with a real estate agency (Real Estate Agency), hence we further refine how this is achieved. To sell the lot through an agency: a lot record should be created, lot information needs to be provided, the lot location needs to be defined and finally the lot price needs to be approved. Thus, this is represented through the and-decomposition of goal lot sold via agency into goals lot record created, lot info provided, lot price approved, and lot location defined.

To actually have the lot sold via agency, Lot Owner delegates goal lot record created to the Real Estate Agency. Now, we consider how the Real Estate Agency achieves its goals. Real Estate Agency reads lot info to achieve goal lot record created (the owners personal data are necessary to create the lot record). This document (lot info) is produced by the Lot Owner while providing lot information (goal lot info provided). Actors can transmit documents to others only if they possess the required document. For instance, in Figure 8.11, Lot Owner is the creator of lot info (i.e., possesses the document) and he transmits this document to Real Estate Agency.

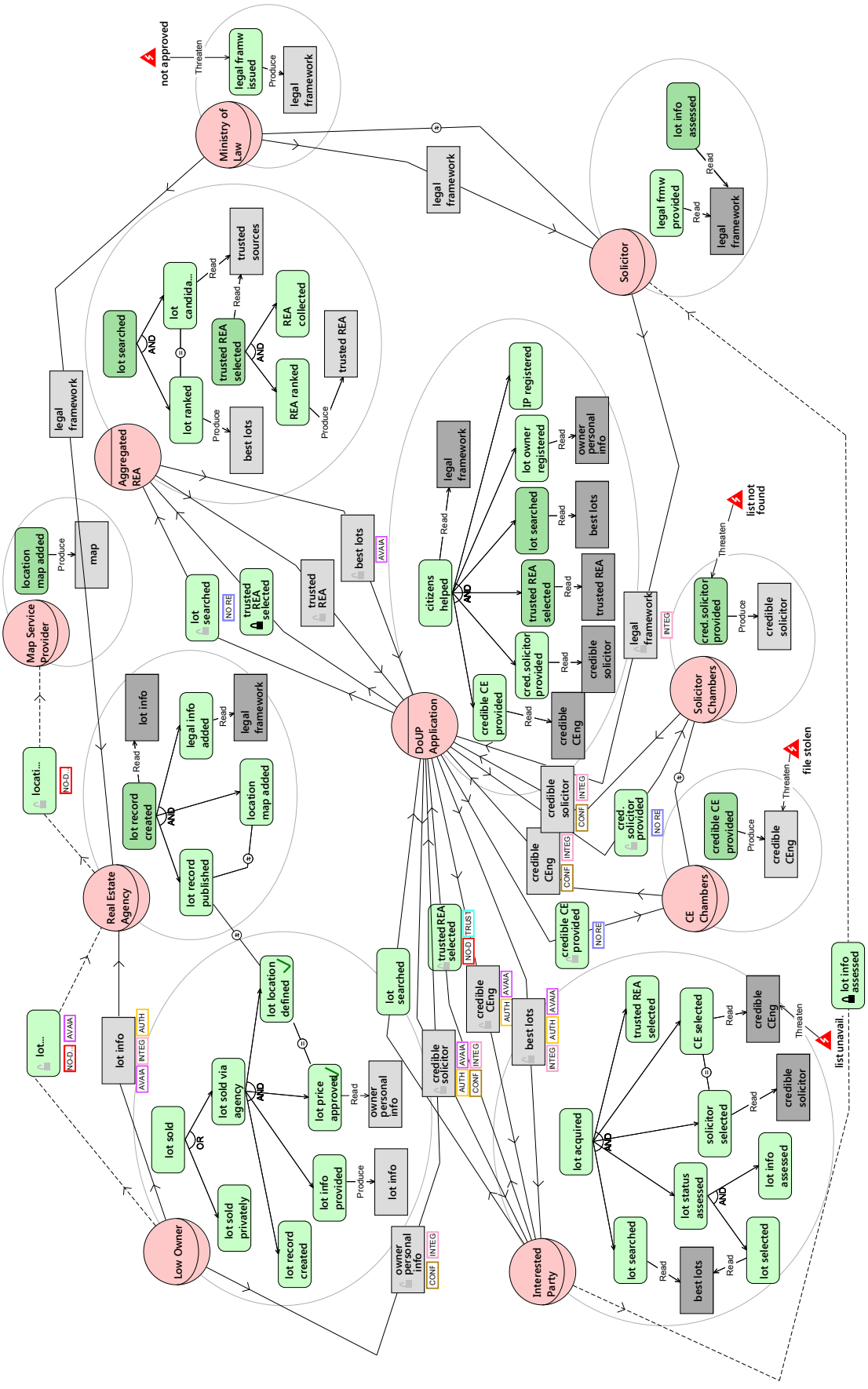


Figure 8.11: eGov Land Selling scenario—Social Model

Step 1.2. Express security needs.

Over goal delegations. In Figure 8.12 Lot Owner requires the Real Estate Agency not to re-delegate the goal lot record created; Interested Party imposes on the DoUP Application a true redundancy single security need for the achievement of goal trusted REA selected; the delegation of goal trusted REA selected from Interested Party to DoUP Application will take place only towards trustworthy application providers; Lot Owner requires the Real Estate Agency 90% availability for goal lot record created. DoUP Application requires CE Chambers non-repudiation of the acceptance of goal credible CE provided, see Figure 8.11.

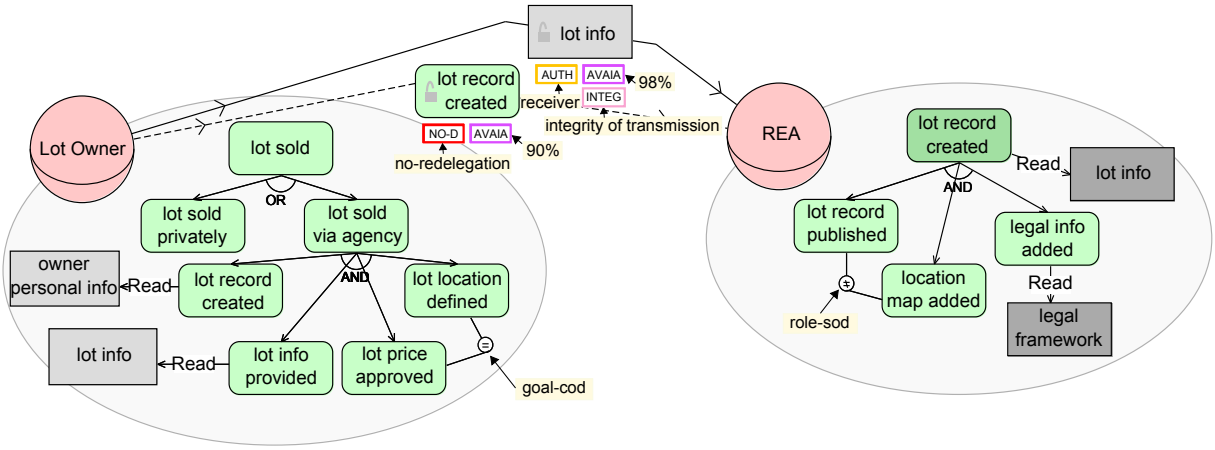


Figure 8.12: Expressing security needs: REA

Over document transmissions. In Figure 8.13, DoUP Application shall ensure sender integrity on the transmission of document best lots to Interested Party; DoUP Application shall ensure sender confidentiality on the transmission of document credible solicitor to Interested Party; DoUP Application shall ensure an availability level of 94% for the document best lots and an availability level of 90% for the document credible solicitor, when transmitting both these documents to Interested Party.

Over responsibility uptake. As far as separation of duties is concerned, the goals lot record published and location map added are defined as incompatible (unequals sign, see Figure 8.12). An example of role-sod is that among roles CE Chambers and Solicitor Chambers, see Figure 8.11. With respect to combination of duties, in Figure 8.12, there is a goal-cod expressed among goals solicitor selected and CE selected of Interested Party.

Step 1.4. Modelling threats. Figure 8.11 represents the events identified to threaten actors' assets. For instance, event file stolen threatens document credible CE of CE Chambers (see Figure 8.14 which zooms over Figure 8.11).

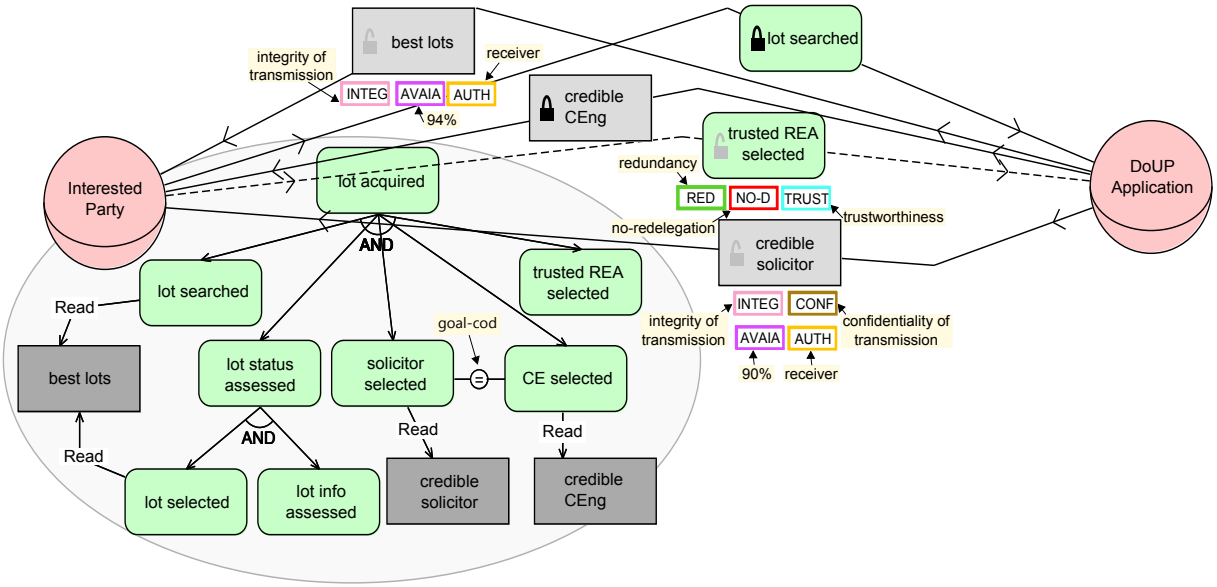


Figure 8.13: Expressing security needs for Interested Party

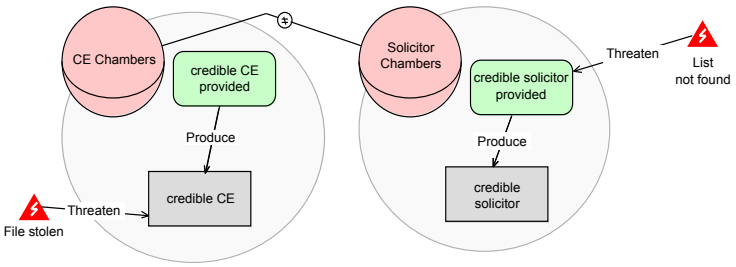


Figure 8.14: Modelling threats

Phase 2. Information modelling

Step 2.1. Identify information/ownership. Lot Owner provides information about the lot, we identify information lot info details, which is owned by the Lot Owner himself and is represented (made tangible) by document lot info (see Figure 8.15). Additionally, Lot Owner owns information VAT number, ID Card number and lot geo location.

Aggregated REA owns information list of credible REA, the CE Chambers owns information list of credible CE, the Solicitor Chambers owns information list of credible sol, while Ministry of Law owns information legal info.

Information VAT number and ID Card number are made tangible by document Owner Personal Info, while information lot geo location is made tangible by map. Similarly, the rest of information entities is represented via documents, see Figure 8.15.

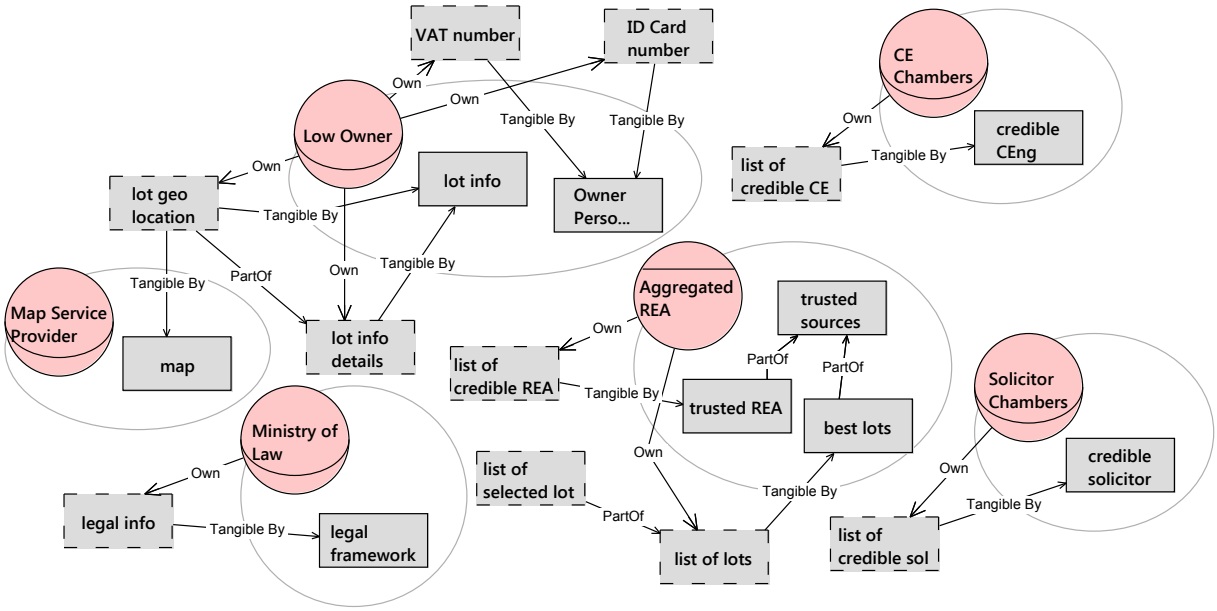


Figure 8.15: eGov Land Selling scenario—Information Model

Step 2.1. Structure information. We identify the Part Of relationships among information and documents respectively. For instance, in Figure 8.15, information lot geo location is part of information lot info details, while documents trusted REA and best lots are part of document trusted sources.

Phase 3. Authorisation modelling

Step 3.1 Model authorisations. For instance, in Figure 8.16, the Lot Owner authorises Real Estate Agency to read, produce, and transmit information lot info details and lot geo location. No prohibitions are specified through this authorisation relationship. Instead a prohibition on modifying information legal info is expressed from the Ministry of Law toReal Estate Agency.

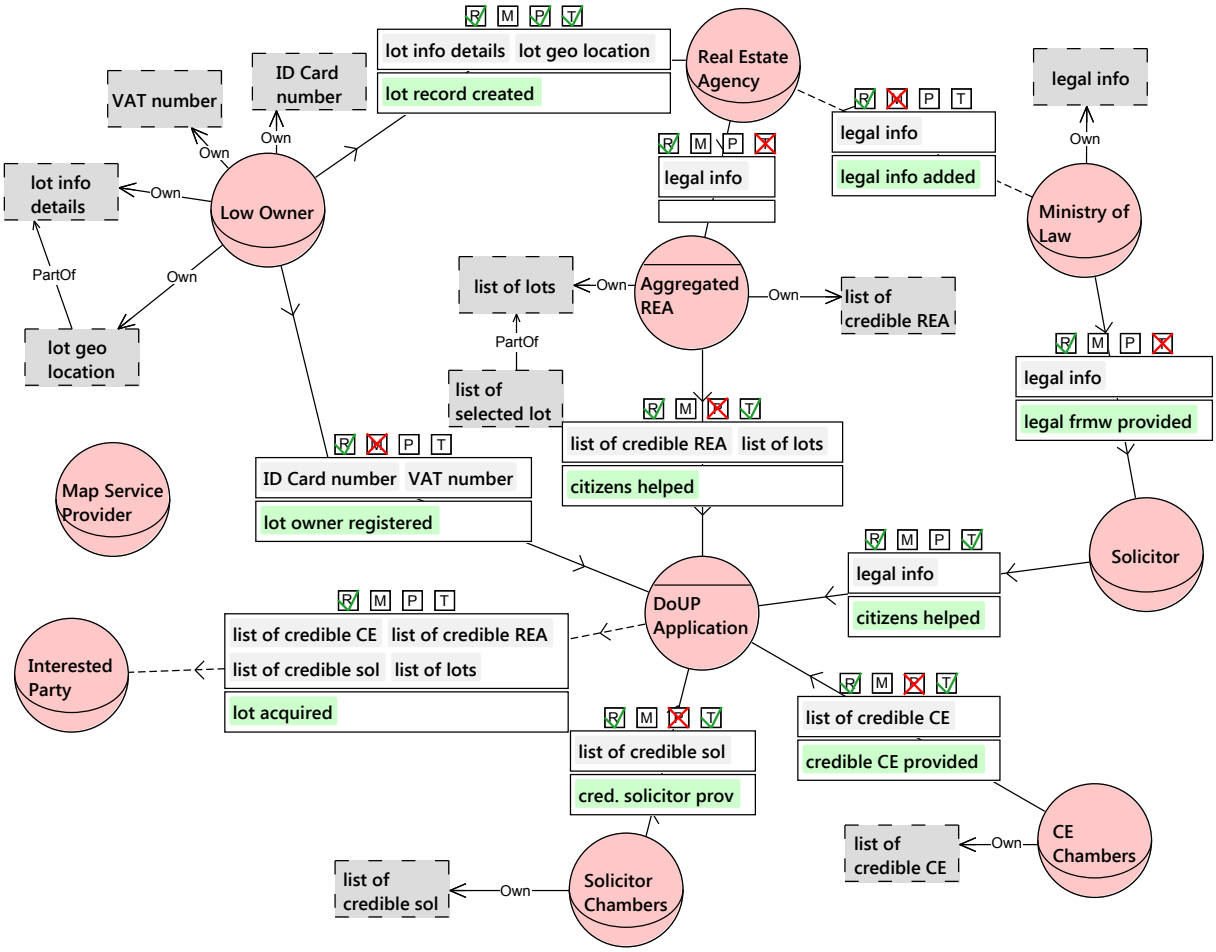


Figure 8.16: eGov Land Selling scenario—Authorisation Model

Moreover, see how the Lot Owner authorises Real Estate Agency in the scope of goal lot record created, not for every goal of Real Estate Agency. The authorisation from Lot Owner to Real Estate Agency is a transferable authorisation (continuous arrow line), while the one from DoUP Application to the Interested Party granting the authority to read information list of credible CE, list of credible REA, list of credible sol and list of lots for goal lot acquired, is a non-transferrable authorisation (dashed arrow line).

Security requirements over authorisations are captured implicitly by prohibiting certain operations and limiting the scope of the authorisation: for instance, Lot Owner authorises

DoUP Application to read information ID Card number and VAT number only for the purpose of being registered (goal lot owner registered), expressing a need-to-know security requirement to DoUP Application, on reading this information only for lot owner registered, see Figure 8.16. There are no examples of non-reading in our eGov scenario. DoUP Application shall not modify documents representing informationID Card number and VAT number, for the authorisation from Lot Owner grants the right to read information ID Card number and VAT number, but prohibits the right to modify these information entities, see Figure 8.16. DoUP Application shall not produce documents that represent information list of credible solicitors or information list of credible CE, given that the authorisations from Solicitor Chambers and CE Chambers prohibit the operation to produce the respective information entities. Solicitor shall not transmit documents representing information legalinfo, see Figure 8.16. An example of *non-reauthorisation* is expressed towards Real Estate Agency, see Figure 8.16; Real Estate Agency shall not further authorise other actors on legal info, for the authorisation coming from Ministry of Law on this information is non-transferable (dashed arrow line).

Phase 4. Automated Analysis

Well-formedness Analysis. This analysis did not find any errors when executed over the models.

Security Analysis. The security analysis found several violations of the specified security needs (identifying errors), such as for instance the violation of non-production by the Map Service Provider. As it can be seen by the diagram in Figure 8.16 showing authorisation relations, there is no authorisation relationship towards Map Service Provider on information lot geo location, which following the semantics of STS-ml is translated into an authorisation from the owner of this information, namely Lot Owner, prohibiting all operations over this information. This means that the Map Service Provider is required all security requirements derived from an authorisation relationship over the given information (i.e., non-reading, non-modification, non-production, non-disclosure, not-reauthorisation). But, from Figure 8.11, we see that Map Service Provider can produce lot geo location since there is a produce relationship from its goal location map added towards document map representing (making tangible) information lot geo location, owned by the Lot Owner who requires non-production of this information. Thus, we identify a conflict that results in the violation of the non-production security requirement.

Similarly, there is a possible violation of a combination (binding) of duties between the goals lot price approved and lot location defined of Lot Owner. A combination of duties requires that the same actor pursues both goals, but there is no single actor achieving both these goals, see Figure 8.17. However, this could change in runtime, and is to be verified through monitoring techniques. At the design level, we verify throughout the models whether any strategies are

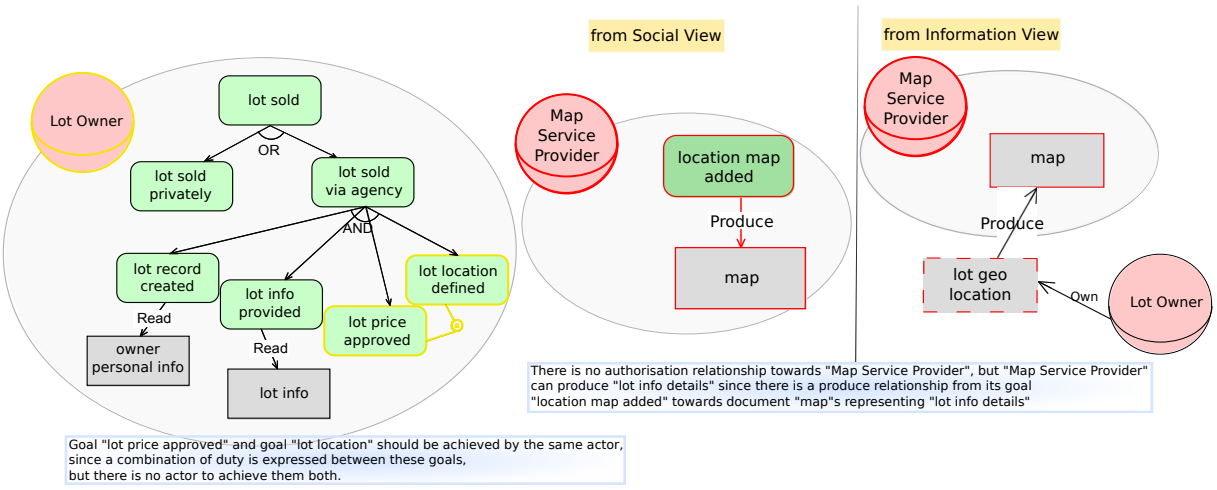


Figure 8.17: Executing security analysis: visualisation of results

undertaken to fulfil the imposed security requirement. Therefore, this conflict is considered a warning, differently from the previous one which is considered an error, and needs to be resolved before implementation. Resolution techniques might, however, require negotiation among service consumers and providers and trade-off analysis Elahi and Yu [2007].

Threat Analysis. We calculate the propagation events threatening actors' assets. We consider the threats shown in Figure 8.14 and calculate their impact. We present the results of this analysis for the event list not found threatening goal credible solicitor provided in Figure 8.18).

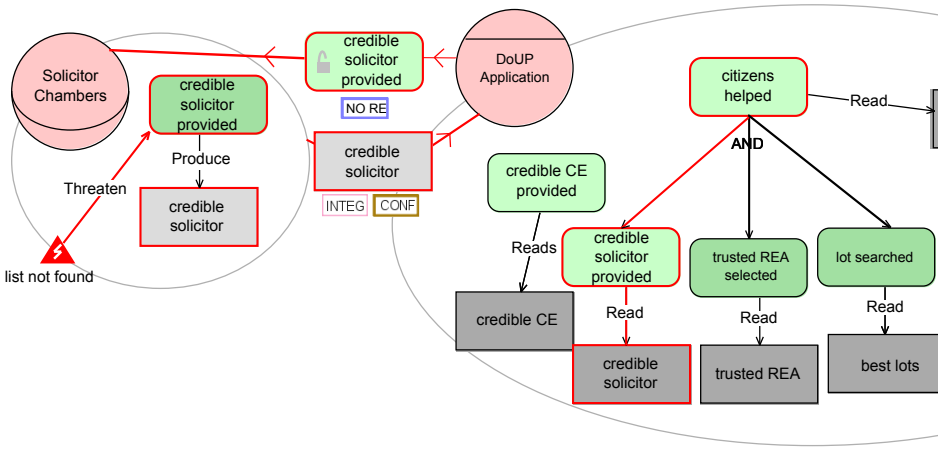


Figure 8.18: Executing threat analysis

Given the results of the threat analysis, we may need to consider alternatives to ensuring the provision of credible solicitors, such that do not account on Solicitor Chambers. However,

this may not always be possible, it depends on the resources available in the socio-technical systems.

Phase 5. Specification

The security requirements specification is generated with the help of STS-Tool, as described in Chapter 7. We report the security requirements for the Lot searching scenario in Figure 8.19. The importance of automatically deriving security requirements and generating the security requirements document stands in that it facilitates identifying what are the security requirements to be satisfied in order to fulfill a given security requirement type, as well as identify security requirements which are still violated (reported in the document).

In this scenario, we could ask questions such as: "*What are the security requirements for non-modification in the Lot searching scenario?*", or " *What about security requirements for non-disclosure of legal information?*".

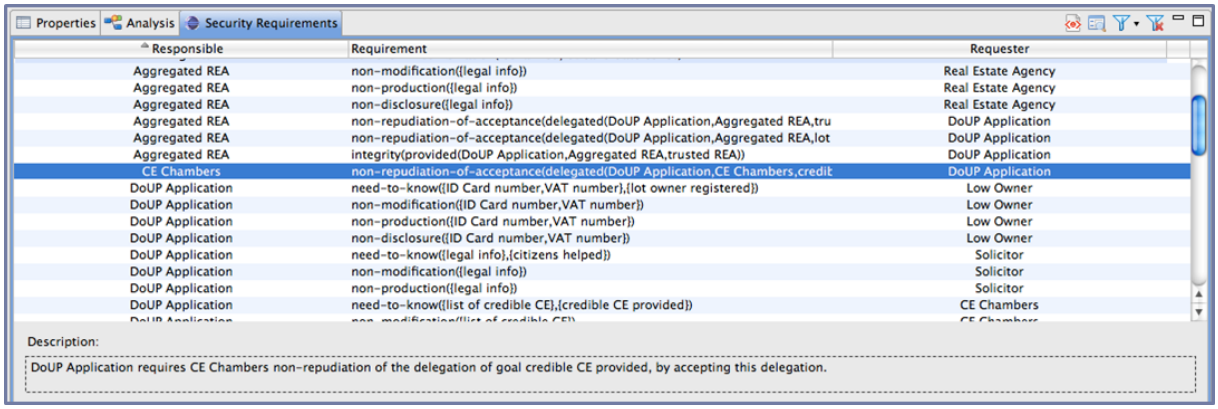


Figure 8.19: Security requirements for the Lot searching scenario

To identify security requirements for non-modification, we consider the security requirements for the Lot searching scenario (see Figure 8.19) and order them with respect to the requirement, so to group together requirements on non-modification. Similarly, we identify security requirements on non-disclosure, and identify only one on legal information, to be satisfied by Aggregated REA (no violation was identified by the security analysis).

We generate the security requirements document, which allows to have more information on the constructed view, explains in detail the findings of the analysis, and enumerates the list of security requirements, facilitating the work of the security requirements engineer in preparing this documentation.

8.3.3 Case Study 2: Air Traffic Management

We consider here a scenario from the Air Traffic Management (ATM) case study, which is a variant of the case study "*The emerging European Air Traffic Management systems*" of the project Aniketos.

ATM systems are increasingly complex, pervasive and important in our everyday lives, rising stakeholders' concerns about security risks accordingly. Over the course of last years, the European Air Navigation Services and their supporting systems are going through subtantial changes due to the introduction of the System Wide Information Management (SWIM) [Eurocontrol, 2013], which will support the collaboration and communications (including data exchange) of the various parties involved in the system. Importantly, SWIM will allow the participation of numerous external parties. These will access a common virtual pool of information, having data stored at different locations. However, this paradigm shift, makes ATM systems open (interconnecting parties from all over Europe), in which newcomers (even untrusted players) might also participate. This opens up many security issues, for the services offered by ATM through the SWIM vary from safety critical data to catering and lost luggage, and security breaches could happen at any point of communication.

Traditionally, ATM was quite a closed system: it allowed only well-defined and established trusted participants (actors) and ad-hoc point-to-point communication of technical systems with other trusted technical systems. However, recently, pilots are starting to use iPads or Windows Surface 2 tables as Electronic Flight Bags, connecting these general purpose (untrusted) equipment to other (trusted) ATM systems. The results are expected to maximase the efficiency of the airspace, however, at the expense of increased security threats. The lack of industrial experience and background in security engineering in the ATM domain and the tremendous complexity of all the trades involved in the ATM system of systems make security need elicitation a hot topic amongst ATM stakeholders today [Meland et al., 2014].

The scenario we focus on, namely *Meteo Datalink*, is as follows. A pilot in an aircraft makes a data link request for weather information via SWIM. SWIM routes the request to the meteo data provider, obtains the response and returns the response to the pilot. A/G SWIM can offer information on weather and turbulence forecasts, as well as current weather situation for areas beyond the range of aircraft sensors and on-board weather radar. Data may include graphical shape of a large hazardous area, or textual information (METAR, TAF) on weather situation at alternative diversion airports.

8.3.4 Results of the Application of the STS methodology to the ATM case study

Differently from the previous case study, the application of the methodology to the ATM case study, has been conducted entirely by practitioners, modellers and engineers, who work closely

with domain experts. The reported models are part of Deliverable 6.4 of the Aniketos project, and have been created by industrial partners from Thales Research [6], Deep Blue Srl [7], and Sintef [8]. Our role in this process has been that of providing support and feedback throughout the use of STS-ml and STS-Tool in creating the models. The relevance of this process is mentioned in Chapter 9, where we demonstrate how this interaction has helped improve the STS methodology, STS-ml modelling language, and STS-Tool.

In this section, we report on the constructed models (describing the outcomes of each phase, without detailing each step) and highlight how the modelling and automated analysis supported by STS helped identify potential security violations and threats to the well-functioning of ATM via the SWIM infrastructure. The modelling of this scenarios was performed in two STS-ml models (diagrams) to simplify modelling and to allow different modellers to work on parts of the scenario. We have merged the two diagrams into one to represent the complete scenario. The reported feedback is adapted from Deliverable D6.4 of the Aniketos project.

Phase 1. Social modelling

The Meteo data link scenario considers a situation in which an aircraft flying its en-route Business Trajectory and weather conditions are changing. Therefore, the pilots make a data-link request for Meteo Information provision via SWIM, performing a manual request for an update of MET information, due to changing weather conditions, through the ACARS aircraft system. The Aircraft SWIM Node manages information exchanges of the airborne applications. These airborne applications are consumers or providers of information placed on board the aircraft. The aircraft SWIM Node interfaces directly with the A/G SWIM Access Point (SAP) to which it is associated at a given moment. The A/G SWIM Access Point represents the A/G Gateway on the ground making the link between the Airborne systems and the ground ATM systems. It implements the A/G SWIM Interface on its top and the G/G SWIM Interface on the bottom, controlling and managing all the SWIM information exchanges so they comply to the required behaviour of these exchanges. As such, it has the goals of routing the meteorological information request and routing the meteorological information itself, see Figure 8.20. SWIM MET Node is related to the ground ATM systems MET service provider. The ATM Systems interested in communicating via SWIM with the aircraft implements on the SWIM Nodes a G/G SWIM Interface. Meteo service provider is an ATM system willing to participate in the SWIM information exchange with an aircraft, providing MET information through SWIM services. The MET service providers update and publish the new MET information, while the SWIM TI messaging provides the updated MET information to the aircraft. The aircraft Airborne Broker

[6] `https://www.thalesgroup.com/en`
[7] `http://www.dblue.it/`
[8] `http://www.sintef.no/home/`

calls an intermediary Ground Broker. The Ground Broker enriches the information (e.g. aircraft request for meteorological data with flight id, a ground system transforms flight id with its route) and provides the most updated Meteo Information to the Aircraft Airborne Broker. This updated Meteo Information is timely stored in the SWIM by a Meteo Service Provider to be exploited by all the Meteo Service subscribers. These interactions are all modelled in Figure 8.20.

As far as the representation of security needs is concerned, a number of availability security needs have been specified over both goal delegations and document provisions, see Figure 8.20. For instance, a 99.9% availability level is required over from SAP to Meteo Service Provider on Meteorological Information.

An integrity of transmission is specified over the transmission of Meteorological Information between the Meteo Service Provider and SWIM in Figure 8.20.

A separation of duties is specified over the various Aircraft message Brokers, ACD Broker, PIESD Broker and AISD Broker.

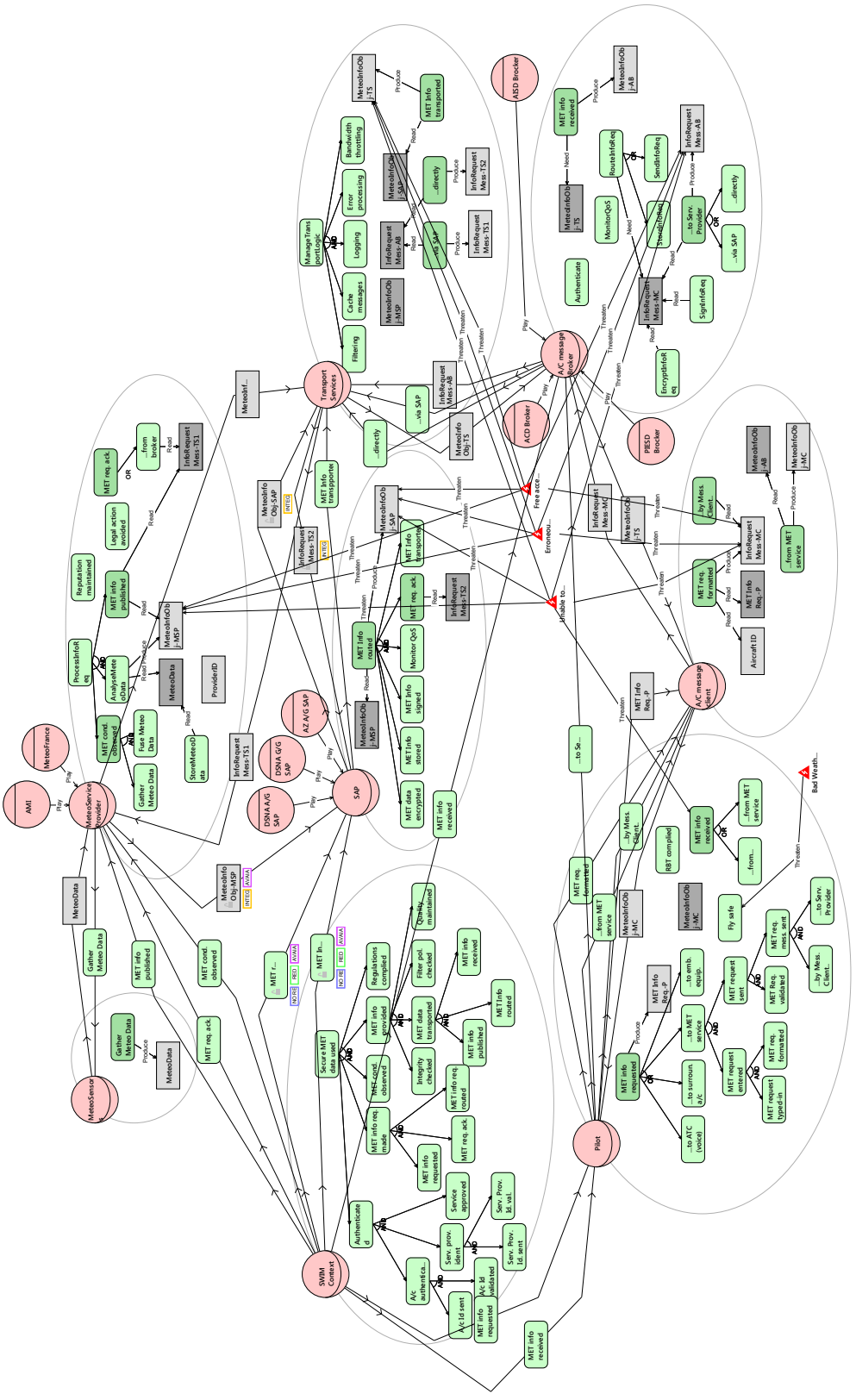


Figure 8.20: ATM Meteo data link scenario—Social model

Phase 2. Information modelling

The following informational assets were identified for the Swim Access Point (SAP): meteorological information request (MET Info Request), owned by Pilot; Meteorological Information and meteorological server identification (Provider ID Number), which are owned by Meteo Service Provider; aircraft identification (Aircraft ID Number), owned by A/C Message Client, see Figure 8.23.

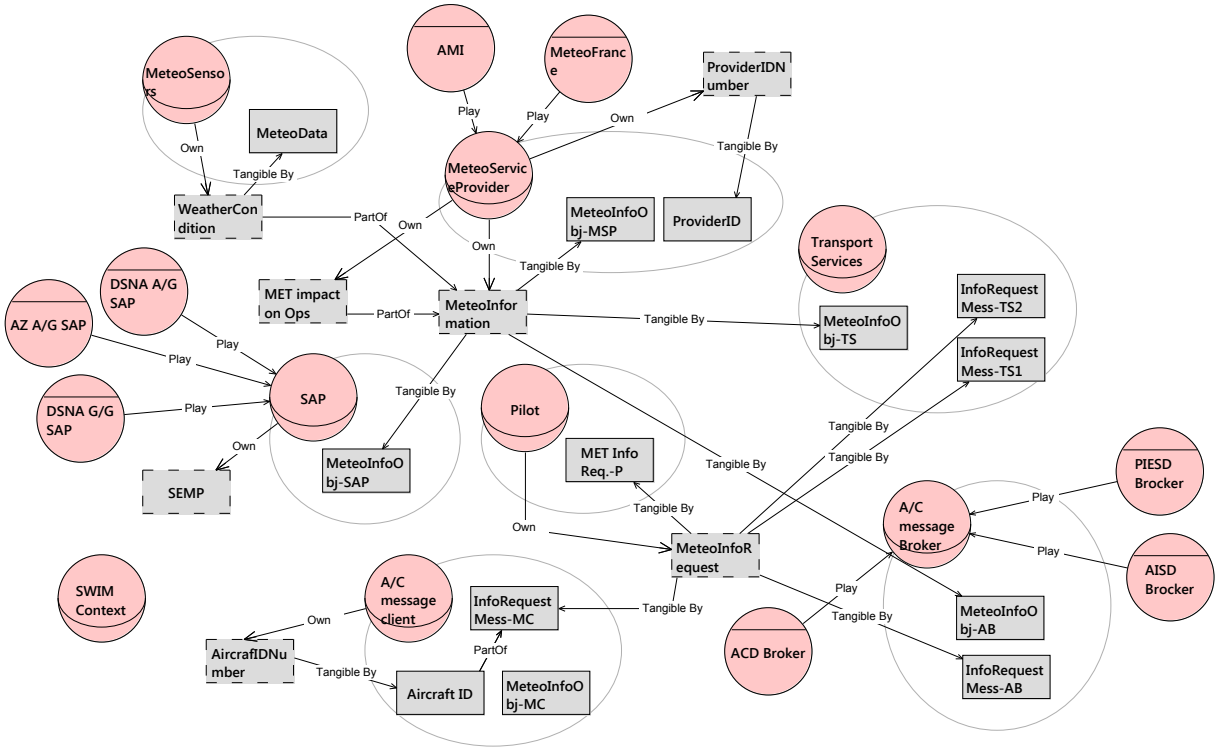


Figure 8.21: ATM Meteo data link scenario—Information model

All identified information entities are represented via documents which are local copies of the information. As such, we have: local copies of the meteorological information request, local copies of the meteorological information, local copies of the aircraft identification, and local copies of the meteorological server identification.

Information forecasted meteorological impacts on operations (MET Impact on Ops) and Weather Conditions were both identified to be parts of Meteorological Information. These were the only identified decomposition of information, no composite documents were identified.

Phase 3. Authorisation modelling

MeteoServiceProvider only authorises SWIM to (transmit) distribute Meteorological Information, i.e. the SWIM is not allowed to read, modify or produce meteorological information. This au-

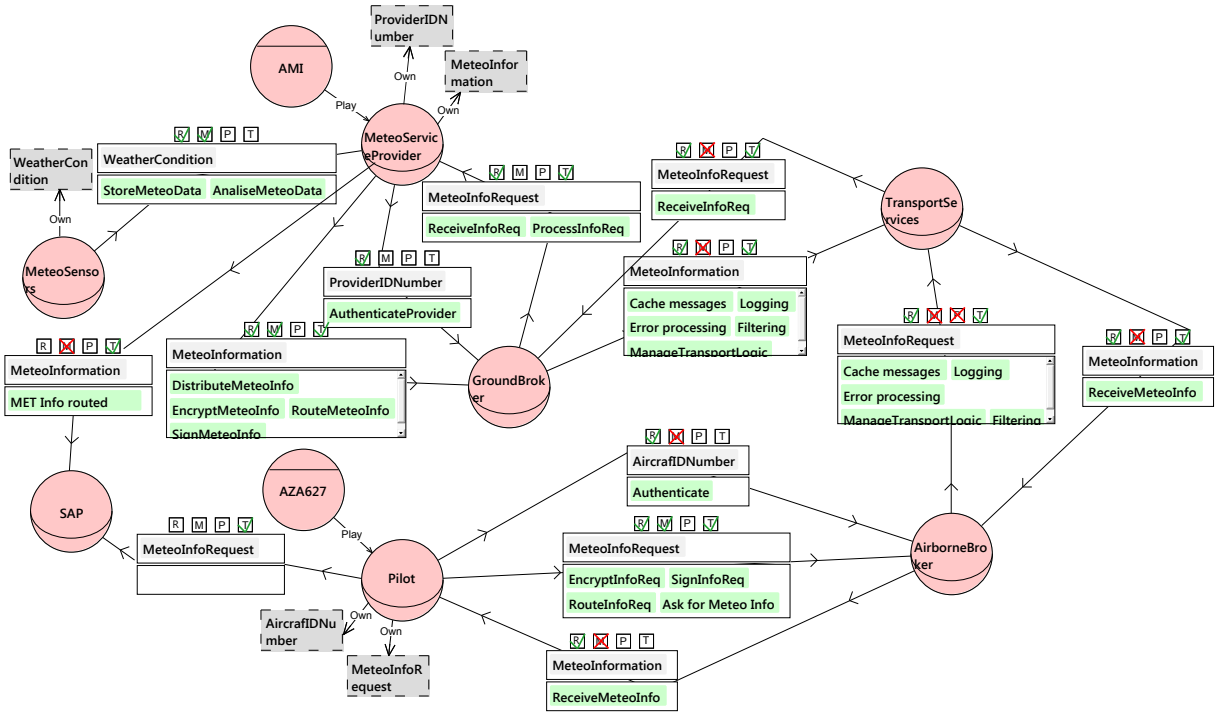


Figure 8.22: ATM Meteo data link scenario—Authorisation model

thorisation intends to express a security requirement on non-modification, with the purpose of requiring some level of integrity.

SWIM cannot read the Meteorological Information, and it cannot produce its own Meteorological Information, see Figure 8.22.

Phase 4. Automated Analysis

Well-formedness Analysis. The modellers report that, after finalising their model, they verified well-formedness by running the corresponding analysis with STS-Tool. The analysis detected only two warnings, both relating to sub-goals of the meteorological service provider that are redelegated back (by the context) and part of the decomposition of the "process information request" goal. Modellers conclude that the identified warning does not create problems with the models, and therefore, the model remain as is. However, they consider it useful to have the tool point out and raising the warning to attract attention on unusual constructs or problems that could go unnoticed.

Security Analysis. The modellers report on 50 violations detected by the STS-Tool on security requirements. The tool allows classifying the identified violations, and in this case, 4 main types of security requirements were violated:

1. fback-single-red: the analysis detected two errors on the routing of the Meteorological Information(request and answer) by the SWIM Access Point (SAP); in Chapter 6, we have explained that the verification performed by STS is partial, the real verification can only be done at run-time. Indeed, the modellers note that there is no explicit redundancy at STS level; redundancy will be made available at technical level, through a full-meshed network. Therefore, the modellers have dropped this violation.

2. non-disclosure: the analysis detected ten violations of this security requirements. The modellers state that the errors are all related to actors that are outside the perimeter of the study. However, this concept is strictly related to the application domain, and as such does not converge with the logic implemented in STS, in which all actors are peers and permissions/prohibitions need to be made explicit to capture the possible violations of authority in manipulating information. The violations were solved by assigning the designated rights to the actors within the perimeter of the study.

3. non-reading: the analysis detected twenty four violations of this security requirements, twenty one of which, again, relate to actors that are outside the perimeter of the study; the other three remaining errors concern SAP reading the Meteorological Information Request, the Meteorological Information, and the Meteorological Impact on Ops without having been authorised to do so. The modellers identify the issue to rise because of an inappropriate modelling of the reading relationship here, for they consider reading to only be necessary for the distribution of the said information. In their view, distribution should encapsulate reading too. However, this issue is again domain specific. Ideally, a carrier serving as a transmission point needs to have only distribution permissions, not necessarily any about reading what is being transferred.

4. non-production: security analysis detected fourteen errors, of which ten relate to actors that are outside the perimeter of the study. Again, these were treated as the other two aforementioned cases. The other four errors concern the SWIM Access Point (SAP) making copies of information, in the form of documents, in order to route and log the information; information is not produced, it is duplicated. STS, however, considers production to cover copying information as well, as such this operation needs to be granted whenever information is being represented via new documents, other than the original ones containing it.

Phase 5. Specification

Modellers have interpreted the findings and discussed about the resolution of conflicts. At last, they produced seventeen security requirements by applying the methodology and using STS-ml

and STS-Tool. The identified security requirements were confronted with SESAR's imposed security requirements, and it was found that STS helped capture these requirements, and in addition, the modelling activities helped identify new security requirements (deemed useful by the modellers) as well.

Properties | Analysis | Security Requirements

Responsible	Requirement	Requester
MeteoServiceProvider	integrity(provided(SAP,MeteoServiceProvider,MeteoInfoObj-MSP))	SAP
SAP	availability(MeteoInfoObj-MSP, 0%)	MeteoServiceProvider
SAP	need-to-know({MeteoInformation},{MET Info routed})	MeteoServiceProvider
SAP	non-modification({MeteoInformation})	MeteoServiceProvider
SAP	non-production({MeteoInformation})	MeteoServiceProvider
SAP	non-repudiation-of-acceptance(delegated(SWIM Context,SAP,MET info req. routed))	SWIM Context
SAP	single-actor-fallback-redundancy(MET info req. routed))	SWIM Context
SAP	availability(MET info req. routed, 0%)	SWIM Context
SAP	non-repudiation-of-acceptance(delegated(SWIM Context,SAP,MET Info routed))	SWIM Context
SAP	single-actor-fallback-redundancy(MET Info routed))	SWIM Context
SAP	availability(MET Info routed, 0%)	SWIM Context
SAP	need-to-know({MeteoInfoRequest},{MET info req. routed})	Pilot
SAP	non-modification({MeteoInfoRequest})	Pilot
SAP	non-production({MeteoInfoRequest})	Pilot
Transport Services	integrity(provided(SAP,Transport Services,InfoRequestMess-TS2))	SAP

Figure 8.23: ATM Meteo data link scenario—Security requirements

Modellers have acknowledged that the new identified requirements correctly reflect their modelling purposes. Examples include the security requirement on need-to-know from Meteo Service Provider to SAP on information Meteorological Information, in the scope of goal MET Info routed, the non-reading security requirement, as well as security requirements on non-modification and non-production required again by Meteo Service Provider to SAP on information Meteorological Information.

8.4 Chapter Summary

In this chapter we have evaluated our approach on an application scenario and two case studies. We applied STS methodology in an application scenario, namely TasLab, for which we created STS-ml models (via the corresponding social, information, and authorisation models), while capturing security requirements through the modelling of security needs, and finally, we analysed the outputs of each activity to identify security violations and the impact of threats in the overall socio-technical system. This evaluation activity was useful also to demonstrate the efficiency of the reasoning techniques.

As far as the case studies are concerned, we report on the results of the application of STS methodology, as performed by industrial partners of the European project Aniketos. The results of the ATM case study are particularly useful, as they show modelling as conducted by practitioners only, their findings, and above all their interpretations to the identified problems. This

activity has resulted quite useful, not only in terms of validating the methodology, language and tool with industrial case studies, but also to obtain feedback for their improvement throughout their developement. We will discuss more on the development of STS methodology, the different improvements on STS-ml and the various releases of STS-Tool in Chapter 9.

The contribution of the chapter is twofold. Apart from showing the adequacy of our modelling primitives to represent realistic scenarios characterised by security issues over interactions and threats over stakeholders' assets, in particular information, we have performed scalability experiments to determine if the proposed analysis techniques implemented in STS-Tool scale well when considering larger settings (i.e., models).

Chapter 9

User-Oriented Empirical Evaluation

We have evaluated our approach with end-users (modellers) of STS methodology, STS-ml and STS-Tool. We have conducted a set of experiments with modellers—expert practitioners and M.Sc. students—to assess the adequacy of the methodology for supporting security requirements engineering in different domains. In this chapter, we report on evaluation activities describing the evaluation process in Section 9.1, as well as presenting the details of each evaluation activity in Sections 9.2–9.4. Finally, in Section 9.5, we discuss the results, present threats to validity and draw our conclusions.

Acknowledgement. Section 9.2 builds on top of [Trösterer et al., 2012].

9.1 Evaluating STS methodology: the process

STS methodology, as presented in this thesis, is the result of a *continuous evaluation process* that started since the early design phases and has led to refinements of the primitives of the language, the introduction of additional security needs, reasoning capabilities, on the basis of the feedback from end-users. We adopted an iterative development process: new language and tool releases have been applied to real settings and evaluated by practitioners. Specifically, we interacted with industrial partners from the FP7 EU-funded research project Aniketos [1]: security engineers and analysts from Thales, Sintef, DeepBlue, domain experts from DAEM, ATOS Research, and Selex Elsag. Moreover, additional feedback was obtained through two experiments with M.Sc. students from the University of Trento. Carrying out appropriate evaluation throughout the development of a comprehensive framework is crucial in order to reach a high quality outcome, given the complexity of the task [Trösterer et al., 2012]. With this in mind, we have started evaluation since the early design stages, actively involving end-users.

As a result, the presented version of STS-ml supports expressing a set of security require-

[1] Ensuring Trustworthiness and Security in Service Composition: `http://aniketos.eu/`

ments types that addresses the needs of the practitioners in expressing security concerns for their given domain of discourse, in addition to adhering to international security standards (Figure 4.18).

Notice that the purpose of this section is not to provide detailed description for each experiment, but rather to give an overview of the evaluation process we followed and its impact on the methodology. We were supported in the organisation of workshops and design of online questionnaires by specialists from the HCI Usability Unit and Christian Doppler Laboratory at the University of Salzburg.

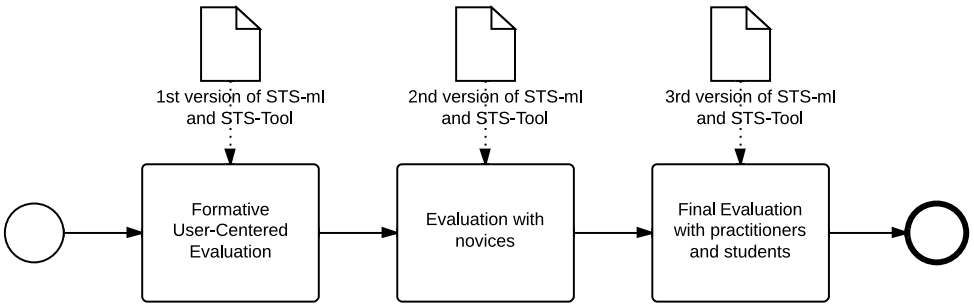


Figure 9.1: Evaluation process followed throughout the development of the STS methodology

In the following sections we discuss in detail the evaluation activities depicted in Figure 9.1. For each activity, we detail the research questions, experiment design, results, and conclusions. Moreover, we outline how the methodology has evolved from one evaluation activity to another.

9.2 Formative User-Centred Evaluation

The first evaluation was conducted at an early design stage on initial version of STS-ml and STS-Tool. At that stage, the tool supported a limited set of security requirements, and allowed only modelling (no automated reasoning). We employed a formative user-centred evaluation [Rosson and Carroll, 2002], whose aim was to identify problematic aspects in the design artefact—here, language and tool—when the development process is not completed. A comprehensive report of this study is in [Trösterer et al., 2012], while details about this workshop are provided in [2].

Research questions. We were particularly interested in the usability of the language and tool. To such extent, we devised the following research questions (E_x-RQ_y stands for the y-th research question of the x-th evaluation activity):

E_1-RQ_1 How usable are the modelling language and its support tool?

[2] http://disi.unitn.it/~paja/pdf/WorkshopTrentoJuly2011_FinalReport.pdf

E_1-RQ$_2$ What are the missing concepts that would be essential to model security aspects?
E_1-RQ$_3$ How adequate/easy to understand is the graphical notation?
E_1-RQ$_4$ What are the concepts whose semantics is unclear/underspecified?
E_1-RQ$_5$ What are the technical issues that limit the usability of the tool?

9.2.1 Experiment Design

The evaluation was organised as a two day workshop (see Figure 9.2) and included seven application domain experts from three companies: Thales, DeepBlue, and DAEM, participating as end-users. These participants were all to interact with the modelling language and tool, but had no knowledge of these artefacts (this was the first time they were introduced to the language and tool). However, practitioners were domain experts, providing rich knowledge and experience, each for its domain of expertise.

The first day was dedicated to *training* activities, where STS-ml and STS-Tool were presented by method designers. At the end of the training day, two questionnaires were provided to the participants on the quality of the training, and the gained knowledge on the language and tool respectively.

The second day consisted of the actual experiment. The day included *modelling* activities in the morning, and *post modelling* discussions in the afternoon to gather detailed feedback. During the modelling participants were asked to model a scenario from their area of expertise—corresponding to their expertise (air traffic management and e-Government). Participants were provided reference sheets for language and tool, as well as a set of cards to write down comments, problems during modelling, ratings about concepts and graphical representations of concepts and relationships, missing concepts/relationships/security needs, etc. A third questionnaire on concepts and graphical representations was required to be filled out by the participants at the end of the modelling.

The participants were observed—paying attention to avoid intrusiveness—in their discussions by the usability and user acceptance specialists from the University of Salzburg, who also helped design the experiment and served as moderators during the interviews. Structured interviews and group discussions were used to gain a deeper understanding of participants' experience with the language and tool, as well as to collect, categorise, discuss and prioritise the identified problems.

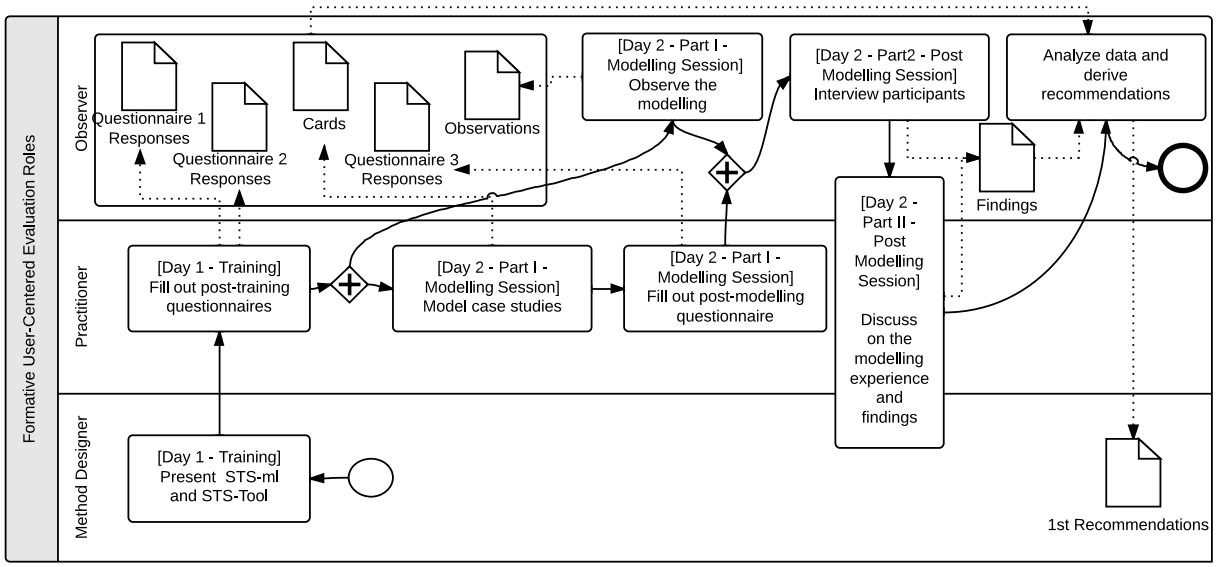


Figure 9.2: Experiment Design for the formative user-centred evaluation

9.2.2 Results

Different methods were applied to evaluate STS-ml and STS-Tool—questionnaires, cards, observation, interviews, group discussion—adopting then method triangulation [Golafshani, 2003] to obtain accurate outcomes. Quantitative data were obtained from the questionnaires, while qualitative data were obtained from observation, interviews complemented with cards' notes, and group discussions. An excerpt of the results for each research question are listed in Table 9.1. We identified several tool usability issues, some concerns about the graphical notation (size, color, etc.), and were suggested a number of missing concepts.

Concerning the *language*, most participants suggested to include threats to model risk perception. The semantics of transferable authorisations was unclear to some participants. Multiple ownership of information was not supported, and considered crucial in some domains. Concerning the *tool*, the participants highlighted the need for autosaving files, automated diagram layout, reduction of the GUI complexity depending on current work, online help, etc. Model scalability was an issue related to supporting big real case studies, spanning hundreds of actors.

RQ	**Findings**
E_1-RQ_1: Usability	adaptive GUI; online help; scalability
E_1-RQ_2: Missing concepts	threats and risks; multiple ownership; trustworthiness
E_1-RQ_3: Graphical representation	adapt size, color
E_1-RQ_4: Semantics	transferable authorisation unclear
E_1-RQ_5: Tool	autosave; automated diagram layout

Table 9.1: Formative user-centred evaluation: findings for each research question

9.2.3 Conclusions and outlook

Based on the evaluation results, we discussed all those findings, and released a new version of the methodology that addressed relevant problems. For instance, the next version of STS-ml provided support for representing events threatening actors' assets. Moreover, it clarified semantical misconceptions, providing a clear-cut definition of resources in terms of information and documents. Additionally, a richer set of security needs was supported, including refinements of *non-repudiation*, *redundancy*, *trustworthiness*, *integrity of transmission*, *non-reading*, *non-modification*, *non-production*, *non-disclosure*, *separation of duties*, and *binding of duties*.

A number of tool bugs was fixed, and a bug-tracking system was created. To improve modelling scalability, we introduced the feature of hiding all goals related to an actor, and provided zoom in and zoom out functions. The complexity of the GUI was reduced by providing a customised palette in each model (social, information, authorisation). Also, we included was improved to include the automatic derivation of the security requirements document, as well as analysis capabilities in terms of validity verification and security analysis (Chapter 6).

The formative evaluation emphasised the need for and the importance of training in supporting users with different backgrounds and skills. As a result, following this first evaluation, we improved the available material on the language and tool, providing extensive documentation on each, while setting up a dedicated website with video tutorials, templates, examples, and the different versions of the language and corresponding tool versions available for download.

9.3 Evaluation with novices

The second evaluation workshop we organised (Evaluation with novices in Figure 9.1) involved improved versions of the modelling language and tool. Additionally, we had devised an initial method that guided users through the modelling and reasoning activities. Again the workshop

was co-organised with the experts from the University of Salzburg.

Research Questions. The objective of the evaluation was to asses the adequacy of the methodology for teaching security requirements engineering. This lead us to the following research questions:

E_2-RQ_1 How do first time users of STS-ml and STS-Tool come along with it?

E_2-RQ_2 How long does it take to learn using STS-ml and STS-Tool?

E_2-RQ_3 How usable are STS-ml and STS-Tool?

E_2-RQ_4 What scenarios cannot be expressed?

E_2-RQ_5 What are the missing concepts?

E_2-RQ_6 Which are the most difficult to understand/less useful concepts?

These questions were mapped to three evaluation criteria: (i) *Learnability*: how easy is it for first-time users to accomplish basic tasks?; (ii) *Usability of STS-ml*; and (iii) *Usability of STS-Tool*.

9.3.1 Experiment Design

In accordance with the objective of this evaluation, we recruited as participants students attending the M.Sc. courses in *Requirements Engineering* and *Organisational Information Systems* at the University of Trento. Most of the participants had already some experience in security modelling and goal-orientation, having adopted in the past languages and tools like SI* [Giorgini et al., 2005a, 2006]. However, they were all method ignorant, i.e., none of them had previous knowledge of STS-ml or STS-Tool, and for this we refer to them as novices. As far as domain knowledge is concerned, no predefined scenario was imposed on the students in order to avoid learning effects, rather each student could choose a desired scenario (for which they had previous knowledge, in order to maximize expertise about the application domain).

The evaluation workshop was organised in three parts: (i) *training*—two lectures of 90 minutes each took place to present STS-ml and STS-Tool, while having a hands-on experience with the tool; (ii) *modelling session* of about 3 hours, during which groups of two students were instructed to model their chosen scenario; and (iii) *interviews* with the lecturers on the overall experience. The design is outlined in Figure 9.3, while details about this workshop are provided in [3].

The training was video recorded so to provide online training material to other prospective users. At the end of the training, participants were asked to fill out an online questionnaire [4], which assessed their gained knowledge, their previous experience with modelling, their estimation on the quality of the training, among others. This questionnaire was adopted from the

[3] http://disi.unitn.it/~paja/pdf/WorkshopTrento2_FinalReport.pdf

[4] http://disi.unitn.it/~paja/pdf/Post-tutorial-questionnaire-May2012.pdf

questionnaire used during the formative user-centred evaluation and adjusted for the needs of this second evaluation.

During the modelling phase, students were provided with supporting material: slides and tutorials on STS-ml and STS-Tool, and a cheat sheet summarising all concepts, relationships and security needs. The experiment involved 36 participants, which were split in three rooms to perform the modelling in a quieter environment. Participants worked in groups of two mainly, with the exception of some students working on their own. They were provided with blank papers to note any eventual problems during modelling as well as missing concepts or relationships.

One observer and one method designer (lecturer) were present in each classroom. The observer played an active role and took notes of the questions raised by the participants, their behaviour (discussions, notes, use of training material, etc.), and the behaviour of the method designers (reaction to questions, answering individually or towards all present participants, and other general impressions). The lecturers, on the other hand, played a passive role so to avoid influencing their modelling: they answered only to general questions on the assigned task and on the features of the tool, and the responses were given to all the students (not only to the student who made the question). No solution to the raised problems or doubts was provided.

After the modelling session students were asked to fill out a second online questionnaire [5], which comprised the standardised Subjective Usability Scale (SUS) [Bangor et al., 2009] provided separately for the modelling language and the tool. The questionnaire included also open questions regarding concepts students considered as difficult to understand or useless, as well as questions on their overall impression of STS-ml and STS-Tool respectively.

The interview with the lecturers was organised following the results of the formative user-centred evaluation on the importance of training, with the purpose of understanding the impressions of the lectures along those of the participants (provided through the first online questionnaire). Lecturers were asked about their impressions of the training and modelling sessions, whether students seemed motivated, whether they thought students could easily follow the lectures or do the modelling task, which were the most critical problems following the generated models and according to the questions that arouse. Finally, they were asked if there were any things they would change and do differently in the future.

[5] `http://disi.unitn.it/~paja/pdf/Post-exercise-questionnaire-May2012.pdf`

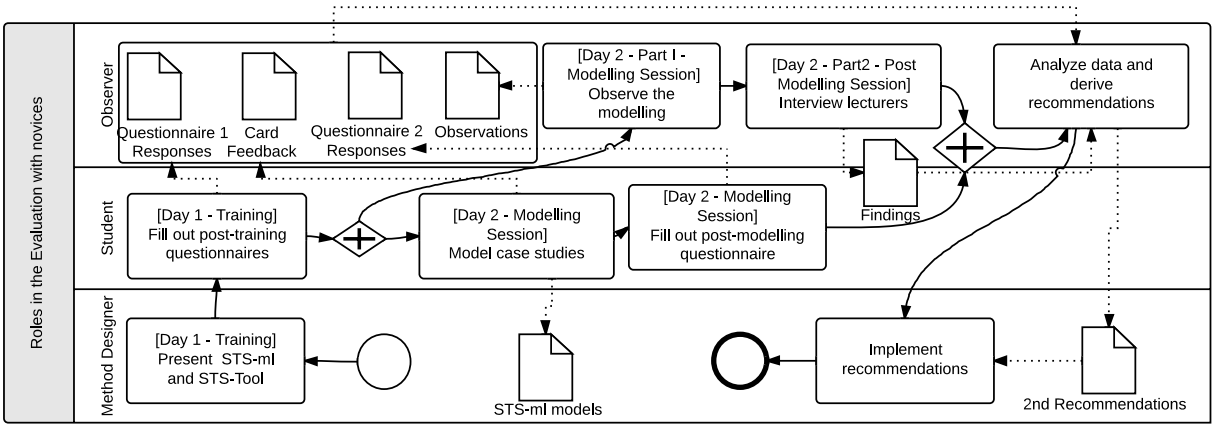


Figure 9.3: Experiment Design for the evaluation with novices

9.3.2 Results

Different evaluation methods were employed, including questionnaires, cards, observation, and interviews with method designers to obtain results. We provide the results for each research question:

(E_2-RQ_1) In the post-training questionnaire, more than 75% of the students provided correct judgements for 8 of the 10 sentences and were sure about their decision. Observation showed that the students immediately started to model, and that they mostly worked with their laptops, i.e. no drawing or writing on paper (except for the evaluation). Students appeared to be very concentrated, fine with their modelling, *not desperate, confused or lost*. Most of the students used the provided tutorial material (slides, cheat sheet) at least once during the modelling session.

(E_1-RQ_2) We cannot give a quantitative answer to this question. However, we can say that after two lectures of 90 minutes each over STS-ml and STS-Tool respectively, students were able to understand the basic concepts of STS-ml and STS-Tool.

Usability of STS-ml, (E_2-RQ_3), was measured using the SUS-Score, which was 64 for STS-ml, indicating that the usability of the language was ok to good [Bangor et al., 2009; Sauro, 2011]. Participants perceived the language as simple to use, easy to learn, with intuitive concepts, good expressiveness, and "powerful enough to describe complex relationships". The results of the questionnaire, observation and the interviews support this outcome. However, participants also explicitly stated aspects (concepts) that are missing in STS-ml, such as softgoals. As far as the understandability of the STS-ml concepts is concerned, participants encountered difficulties in clearly understanding some concepts and security needs. For instance, the distinction between information and documents was not clear in a couple of cases. Moreover, the security needs of non-repudiation and redundancy also caused some difficulties in understand-

ing, particularly the difference between true redundancy and fallback redundancy.

Usability of STS-Tool, (E_2-RQ_3), was also measured using the SUS-Score, which was 68, indicating that the usability of the tool is ok to good [Bangor et al., 2009] or good [Sauro, 2011]. Participants indicated that the amount of features supported by the tool is good, in particular the inter-model consistency which automatically generates related content in the other models. Furthermore, it was pointed out that using the tool is very intuitive and simple. However, there are some missing features that could improve usability, such as keyboard shortcuts for certain operations, and undo/redo/copy/paste functionalities on individual concepts/relationships in a diagram. It was pointed out also that the label naming was too short and should be extended. These were already known to us designers, but were limited in the face of constraints imposed by the platform upon which STS-Tool builds.

(E_2-RQ_4) Participants explicitly stated aspects (concepts) that are missing in STS-ml, such as softgoals and partial contributions.

(E_2-RQ_5) The results have not indicated any cases of scenarios that could not be modelled. Only one respondent expressed concerns about the adequacy of the language for modelling machine-automated processes; however, this is clearly out of the scope of STS-ml.

(E_2-RQ_6) Participants encountered difficulties in clearly understanding some concepts and security needs. For instance, the distinction between information and documents was not clear in two cases. Moreover, the redundancy security need also caused some difficulties in understanding, particularly the difference between true redundancy (R_3, R_5) and fallback redundancy (R_4, R_6).

9.3.3 Conclusions and outlook

The outcomes of this second evaluation helped us improve usability issues in STS-ml and STS-Tool, and provided useful feedback on the use of our approach as a didactic tool in higher education.

Learnability is influenced by the amount of previous knowledge users have and by the quality of the training they receive. So, we consider previous experience as an important factor. Most of the students had experience with organisational modelling and goal-oriented modelling, while some were even experienced in security modelling. However, we identified also a negative effect on STS-ml, for many participants tended to compare it to the previous approaches. Hence, we discarded all recommendations that aimed to complete the language with concepts coming from these existing approaches that we deemed not useful, and for which no strong justification was provided on their necessity. We however took this into account to improve our training materials.

As far as training quality is concerned, the results of the first questionnaire showed that the students were able to understand the basic concepts of STS-ml and STS-Tool. The participants

rated the quality of the training as good to very good, and the pace of the training as normal. However, it was pointed out that the training could be improved by extending the timing of the lecture on STS-ml, as well as by having a more interactive session (which was limited by the video-recording).

Meanwhile, after the first workshop, we kept receiving feedback on new releases from the practitioners involved in the first evaluation workshop. Their feedback was obtained either through direct interactions in meetings or by providing us feedback via the bug-tracking system. As a result, the successive improvements of STS-ml and STS-Tool took into consideration both sources of feedback.

The problems encountered with regard to the *usability of STS-ml and Tool* were in a way foreseen, especially those related to security needs. The problems that the students encountered were about the security requirements that cannot be verified at design time (or can be partially verified). Practitioners, on the contrary, did not encounter any trouble; our interpretation is that the practitioners had significantly more expertise in security.

9.4 Final Evaluation

The final evaluation involved different types of subjects: M.Sc. and Ph.D. students at the University of Trento, as well as experienced domain experts (most of them were first presented with STS-ml and STS-Tool during the formative user-centred evaluation and had used it ever since). The objective was to provide an assessment of the adequacy of our methodology for conducting security requirements engineering, after almost three years of development.

Research questions. We assess the adequacy of the methodology via the following questions:

- **E_3-RQ$_1$** How effective is the methodology in identifying security requirements?
- **E_3-RQ$_2$** How efficient is the methodology for modelling and analysing real-world scenarios?
- **E_3-RQ$_3$** How easy to use is the methodology?
- **E_3-RQ$_4$** How useful is the methodology in identifying security requirements?
- **E_3-RQ$_5$** How willing are the participants to adopt the methodology?

9.4.1 Experiment Design

Participants of the experiment were composed of two family of users, namely students and practitioners. The first participants were M.Sc. or PhD students attending the Organisational Systems Engineering course at the University of Trento. The students had no previous knowledge of the methodology, however they were experienced in other goal-oriented methodologies.

As far as domain knowledge is concerned, again students chose a scenario from a domain they knew quite well (this was done beforehand, not on the spot during the evaluation activity). The group of practitioners consisted of the participants playing end-users in the first evaluation, and some other colleagues that learned the methodology as part of the Aniketos project. As such, a comprehensive overall training was not conducted for practitioners, instead updated material and a presentation showing the updates on the current version of the methodology was made.

Practitioners had autonomously, and as part of their daily activities used the methodology to model scenarios/case studies from their domain of expertise (see Figure 9.4). We had practitioners from the three domains of air traffic control management, telecommunications, and eGovernement, who had used the different versions of the methodology, language and tool over the past three years. Thus, we can say that practitioners had both good method knowledge and good domain knowledge.

In total, 30 students and 14 practitioners participated.

As far as the protocol followed with the students is concerned (see Figure 9.4), it included extensive training as suggested by the outcomes of the second evaluation: (i) *training* was organised so to provide more time to present the language, and give students time to understand the newly introduced concepts; discussion took place to clarify doubts and determine their acquired knowledge; (ii) a hands-on tutorial of STS-Tool followed, and students were asked to use the tool on small examples within a few days; (iii) a session was organised to discuss doubts and identified problems after their small modelling exercises. Most students had performed the modelling before of this session, others continued modelling in class too; (iv) the actual modelling session took place.

In the end, we asked both the students and practitioners to answer an online questionnaire [6] (see Figure 9.4) comprising questions for each and every aspect of the methodology, its perceived ease of use, perceived usefulness, and intention to adopt. The questionnaire was designed following the Method Evaluation Model [Moody, 2003] to assess all of these aspects.

[6] `http://disi.unitn.it/~paja/pdf/STSEvaluationQuestionnaireMay2013.pdf`

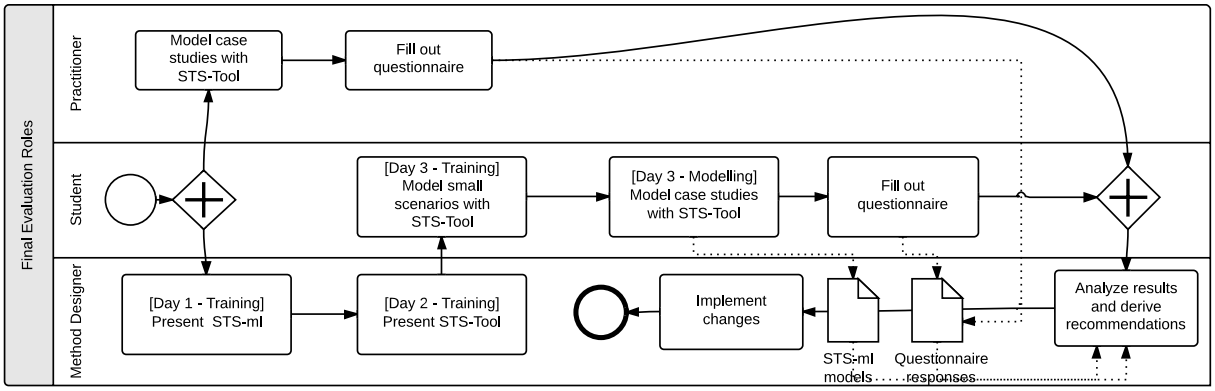


Figure 9.4: Experiment design for the final evaluation with students and experienced practitioners

9.4.2 Results

We analysed the produced models and the feedback provided through the questionnaire to derive results for each research question:

(E_3-RQ_1) All users report the methodology—particularly the method and language—to be *effective* in capturing all important security requirements for their given setting. While students confirm the adequacy/exhaustiveness of the supported security requirements, domain experts suggest a number of extensions. Their suggestions appeared useful for domain-specific variants of the methodology, but not generalizable to all domains.

(E_3-RQ_2) The methodology is considered *efficient* in ensuring model validity and reducing the effort of modelling and analysis large real-world scenarios, supported by STS methodology and STS-Tool.

(E_3-RQ_3) Almost all practitioners find the methodology *easy to use*, while 20% of the students did not take a stance.

(E_3-RQ_4) In general, the methodology was considered *useful* by practitioners (in particular the method, the reasoning capabilities of the tool and information provided to the users).

(E_3-RQ_5) All but one practitioner (who was not sure about adoption) expressed their intention to adopt the methodology for security requirements engineering. The results from students' answers show more reluctance in their decision, with about 50% choosing "difficult to say". However, when they were asked to confirm their intention for *not* adopting the methodology, 80% of them disagreed.

9.4.3 Conclusions and outlook

The experience in modelling heterogeneous case studies has proven useful in developing and improving our methodology. Following users feedback (the ones adopting the methodology),

the STS methodology facilitates the modelling process. Most users find the reasoning quite useful and informative to then improve STS-ml models. The visualisation of errors and the security requirements document facilitates the communication with stakeholders.

The feedback has been quite positive in terms of the adequacy of the methodology to model the case studies and application domains. The features supported by the methodology and tool, such as multi-view modelling (having three different sub models of the same model: social, information, and authorisation) appear to ease the modelling process.

However there is still place for improvement especially with respect to usability issues, reported also in the previous evaluation activity, which could not be addressed in the current platform. These are mostly issues related to the toolset, not the conceptual work behind the methodology which is quite stable by now. Nevertheless, the new version of STS-Tool has considerably improved usability issues. The current version of the tool, 2.0, builds on top of EMF as opposed to the previous one (v1.3.3), which was built on top of GMF framework, and supports already undo/redo/copy/paste shortcuts that facilitate the work of the modellers.

9.5 Discussion and threats to validity

Apart from the general feedback on the methodology, each experiment had a different purpose, and the heterogeneity of the participants provided evidence concerning both the applicability of the methodology in practice, and its usage for teaching.

We have identified interesting outcomes and have received quite useful feedback to iteratively improve the methodology. The current version of the methodology is indeed the result of such an iterative and incremental evaluation process. However, some issues are still to be addressed.

First, practitioners have reported that *modelling scalability*—supporting models with hundreds of actors—is to be improved in our methodology. As a matter of fact, this is a common limitation of many model-based approaches, especially goal-oriented methods. We have considerably improved scalability by separating concerns (building three different models—social, information, and authorisation), offering the possibility to hide details within an actors' rationale and zoom in on others. However, more work needs to be done to offer the flexibility that practitioners demand. One way to tackle this, would be perhaps to have a many-to-many interconnection besides the one-to-one interconnection among the sub models of an STS-ml model, in order to support several scenarios of the same case study at once.

Second, practitioners consider time or sequencing information an important factor in reading/interpreting the models. Considering the procedural perspective goes beyond the purpose of our methodology, which lies at a higher level of abstraction, and aims to understand "why" [Yu and Mylopoulos, 1994] security requirements exist. However, the raised concern has to be

addressed in the future, so to better link the language with process modelling languages.

The presented evaluation activities suffer from some threats to validity.

Internal validity: the selection process involved mainly subjects that were in our professional network; moreover, some of the subjects gained knowledge of the methodology over time, and their opinions and experience are thus affected by this factor.

Construct validity: our evaluation suffers from the so-called mono-method bias, i.e., the subjects were treated only with our method: further work involves conducting controlled experiments that compare our methodology to others.

External validity: due to time limitations, the modellers have mainly used small- or medium-sized examples. However, the practitioners have gained experience with large-scale models outside the context of the experiment.

Finally, the sample of subjects is not a perfect representative of the population, the results cannot be easily generalized.

One of the evidences in the reported feedback is that users need to be trained through either workshops or online training materials. Otherwise, their experience shows a tendency of learn-by-doing due to lack of time in consulting the accompanying extensive documentation.

9.6 Chapter Summary

In this chapter, we presented the evaluation activities conducted with end-users over time, since the first proposal of the methodology, modelling language and tool till now. Each activity has provided useful insights and feedback to improve these three components of this thesis throughout their development.

One of the major challenges we faced and addressed throughout this process has been that of presenting the language to an audience with different backgrounds, mainly non-professional modellers, and getting feedback from them. Additionally, the shift to goal-orientation for practitioners already proficient with other modelling practices such as UML, BPMN, etc., appears to be a real shift of mind. Nonetheless, throughout the various interactions we have achieved the goal of making the distinction clear, and explaining for what STS methodology is meant.

Chapter 10

Discussion, Conclusions and Future work

10.1 Fulfillment of success criteria

We discuss how the proposed methodology and evaluation activities presented in the previous chapters have contributed to the fulfillment of the success criteria presented in Section 1.3.

Table 10.1: Fulfilment of success criteria via evaluation activities

	Methodology			Language			Analysis		
	SC1.1	SC1.2	SC1.3	SC2.1	SC2.2	SC2.3	SC3.1	SC3.2	SC3.3
(E_1) *Self-evaluation study*	done	done		done	done				
(E_2) *Case study*	done	done		done	done				
(E_3) *Empirical study*	partially	done	done		partially				
(E_4) *Formal semantics*	done					done	partially	done	
(E_5) *Scalability study*									done

• SC1.1 Able to conduct modelling and analysis activities in few steps • SC1.2 Applicable to different domains, and different socio-technical systems • SC1.3 Usable by both researchers and practitioners	• SC2.1 Able to capture stakeholders' security requirements (starting from security needs) along system and stakeholders' requirements • SC2.2 In line with the terminology in international security standards • SC2.3 Equipped with a formal semantics	• SC3.1 The definition of a set of properties to be verified over security requirements models • SC3.2 Automated analysis for conflict identification (with tool support) • SC3.3 Analysis results are provided in acceptable time

Table 10.1 summarizes the fulfillment of the defined success criteria by the evaluation activities discussed in Section 1.3. In this table, done and partially mean that the evaluation activity has fully or partially met the success criteria, respectively.

In the following, we discuss in more detail how each and every artefact produced by this work, namely methodology, language, and analysis, addresses the research questions that have guided this research work. Each artefact corresponds to one research question, the fulfillment of which is verified with the fulfillment of the corresponding success criteria.

STS Methodology. Addressing RQ_1:

$SC1.1$ *Able to conduct modelling and analysis activities in few steps*. We have verified the fulfillment this success criteria by carrying out two evaluation activities, namely *Self-evaluation study* (E_1), and *Case study* (E_2). Moreover, the definition of formal semantics (E_4) *Formal semantics* has contributed to the fulfillment of this success criteria with respect to conducting automated analysis, while (E_3) *Empirical study* has partially fulfilled $SC1.1$. More specifically:

- We have performed a self-evaluation study conducting modelling and analysis activities to build an STS-ml model following the steps of the STS process for a collaborative project in Trentino, focusing on a particular scenario, namely tax collection (details in Section 8.1). Modelling and analysis activities were conducted in few steps (with a maximum of two rounds of iterations), however, we do acknowledge that this might be related to the size of the considered scenario.
- We have analysed two case studies developed by practitioners, who have built STS-ml models from two domains, namely eGovernment (Section 8.3.1) and Air Traffic Control Management (Section 8.3.3) respectively, showing the application of the modelling and analysis steps, while presenting the interpretations of the results by practitioners (domain experts). Notably, this evaluation activity is considered to fulfil $SC1.1$, for the involved practitioners were happy with the constructed models and the obtained results.
- We have ascribed formal semantics (Section 6.1) to the constructs of STS-ml to support the automated analysis activities of the STS methodology. Automated analysis is mainly supported by the STS-Tool, which allows security requirements engineers to run automated analysis leaving transparent all technical details, while providing useful information for all identified breaches and potential problems.
- We have continuously conducted empirical studies with both students and practitioners, as reported in Chapter 9. We say that this activity partially fulfills $SC1.1$, given that in the process we have worked with bigger case studies, not necessarily

focusing on particular scenarios. Dealing with the overall case study highlights the need (already considered by the STS methodology) for an iterative and incremental approach to completing the modelling and analysis activities. As such, these activities are conducted in few steps for given scenarios, but need to be followed over various rounds (iterations of the STS process) for the overall case study (comprising more scenarios). Thus, we determine that this evaluation activity partially fulfills this success criteria.

$SC1.2$ *Applicable to different domains, and different socio-technical systems*. This success criteria is important to prove the generality of the proposed methodology, which should support the modelling of various socio-technical systems, and capturing different security requirements types, while offering good coverage for different domains.

- The conducted self-evaluation, (E_1) *Self-evaluation study*, and the case study approach, (E_2) *Case study*, show the applicability of the methodology to different domains, namely tax collection, eGovernment and land buying, and Air Traffic Management control. Details about the various scenarios and case studies, which are from different domains, are discussed in Chapter 8.
- We have conducted a series of empirical studies, (E_3) *Empirical study*, which fulfil this criteria by having end-users (practitioners, and students playing end-users) follow the process supported by the STS methodology on a variety of domains. In particular, leaving the students explore domains of their choice opens up to the application of the methodology to a myriad of socio-technical systems and domains, see Chapter 9.

$SC1.3$ *Usable by both researchers and practitioners*. It is important that the methodology finds applicability not only among researchers, but also among practitioners. This adoption is important to understand the limitations of the methodology when applied to real settings, to then guide further improvements. We have fulfilled this success criteria by running a series of empirical studies with both practitioners (actual end-users) and M.Sc. and PhD students (playing end-users) as part of (E_3) *Empirical study*. The analysis and results reported in Chapter 9, and especially in Section 9.4, demonstrate not only the application of the methodology by both types of end-users, but also their intention to adopt the methodology to conduct security requirements engineering.

STS-ml Modelling Language. Addressing RQ_2:

$SC2.1$ *Able to capture stakeholders' security requirements (starting from security needs) along system and stakeholders' requirements*. This criteria is fulfilled through the self-evaluation

(E_1), case study approach (E_2) and the empirical studies (E_3), through which we apply the STS-ml language constructs for the modelling of different socio-technical systems from various domains, while capturing security and stakeholders' requirements. We refer to system and stakeholder requirements as actor's business policy (see Section 6.2.1), while the supported security requirements are derived by capturing stakeholders' needs with respect to security (details reported in Chapter 4).

$SC2.2$ *In line with the terminology in international security standards*. We have proposed a taxonomy of security requirements types inspired by international security standards, namely ISO-27002 [ISO/IEC, 2005], see Section 4.5. The evaluation activities E_1, E_2, and E_3, have each contributed to fulfilling this success criteria, for they have allowed us to interact with domain experts and practitioners, who have provided feedback on the need for expressing certain security requirements, while being in line with international standards for security. Details about this feedback are provided in Chapter 8 and Chapter 9. However, we consider this success criteria to be partially satisfied by the empirical studies, because we consider that a full satisfaction/fulfillment of the criteria can be reached when a compliance process is defined and/or certification or standardisation efforts are made. For now this is left as future work.

$SC2.3$ *Equipped with a formal semantics*. We have provided a formal semantics for the constructs of STS-ml in Chapter 6, as part of (E_4) *Formal semantics*.

Automated Analysis. Addressing RQ_3:

$SC3.1$ *The definition of a set of properties to be verified over security requirements models*. We provide a formal semantics for STS-ml in Section 6.1, where we present the formal framework that allows the verification of a set of properties over STS-ml requirements models. Such verification is guided by the following questions: (i) *Is the model well-formed?*, (ii) *Are there any security requirements conflicts?* (Section 6.2.1), (iii) *Are there any violations of security requirements?* (Section 6.2.1), and (iv) *What is the impact of events threatening stakeholders' assets?* (Section 6.2.2). The formal framework allows to perform the verification of properties at design time, however, several STS-ml supported security requirements can be verified only at run-time, when the system is operating. Therefore, we consider this success criteria to be partially satisfied. Run-time verification and monitoring, which requires a different type of reasoning approach, is left as future work.

$SC3.2$ *Automated analysis for conflict identification (with tool support)*. We have integrated and implemented the formal framework, proposed in Section 6.1, into STS-Tool (see Chap-

ter 7). The formal framework supports among others, conflict identification, described in detail in Section 6.2.1.

$SC3.3$ *Analysis results are provided in acceptable time*. We have fulfilled this success criteria by conducting a scalability study as part of E_5, see Section 8.2. The reported results show the efficiency of the automated analysis, especially with respect to the model size.

10.2 Conclusions

Security has been a hot topic in software engineering and system development for over a decade [Devanbu and Stubblebine, 2000; Giorgini et al., 2003]. Dealing with security issues is crucial, since security violations may result in legal infringement, monetary sanctions, and loss of reputation, resulting in loss of market value of threatened organisations. The socio-technical perspective opens up new challenges for software engineers when dealing with security. The threats analysts need to consider are not only technical, but also social. Analysts must consider the underlying business policies of an organisation. Security is often an afterthought in the development of systems [Mouratidis and Giorgini, 2007b; Anderson, 2008; Johnstone, 2009], resulting in expensive and challenging after-the-fact fixes, when security incidents occur, given that solutions need to be integrated into the architecture of already running systems.

We have proposed a security requirements engineering methodology for the design of secure socio-technical systems, namely STS, to specify secure socio-technical systems. Key features of STS are: (i) analysing the problem domain in terms of both social and technical aspects, and (ii) relating security requirements to the interactions among the actors in the socio-technical system.

In the following, we summarise the contributions made by this thesis.

STS Methodology. The methodology integrates with existing security requirements engineering methodologies, refer to Chapter 2 for an overview. Most mainstream approaches to security requirements engineering do not take into account social aspects, focusing strictly on technical mechanisms. Goal-oriented approaches, on the other hand, recognize the importance of considering security from a social and organisational perspective. Despite their advantages, goal-oriented approaches have several limitations when applied to security requirements engineering: (i) security requirements are expressed at a very high level of abstraction making them difficult to operationalise to technical requirements for the system-to-be, (ii) information is not treated as a first-class citizen, and (iii) their underlying ontologies are not expressive enough to effectively represent real-world security requirements. STS takes advantage of social and organisational approaches, modelling a socio-technical system in terms of goal-oriented actors that play different organisational roles and interact to fulfil their objectives. Specifically, STS addresses

the limitations of previous approaches by: (i) relating security to interaction and recognising the importance of considering social and organisational threats, not strictly technical ones; (ii) giving information its rightful place, as a first-class citizen, when dealing with information security; and (iii) separating responsibility uptake from information and permission/prohibition flow, which yields to a rich set of security requirements types from six categories, namely confidentiality, integrity, availability, reliability, authentication, and accountability.

The STS methodology covers the entire security requirements engineering phase, starting from the elicitation of stakeholders' needs, their representation in models, their analysis, until the specification of the security requirements for the system to-be. It supports over thirty security requirement types that address a variety of security aspects (see Figure 4.18).

This high expressiveness is ensured by clearly distinguishing actors intentional and informational assets, keeping track of information ownership and the needs actors impose on the usage and manipulation of their information (via authorisations), apart from considering security issues over actors interactions.

The iterative and incremental nature of the STS methodology makes it appropriate for the specification of secure socio-technical systems from initial design, but also for the analysis of existing systems. In the latter case, the methodology could be applied to reverse engineer existing systems, in order to understand the underlying security policies. This is particularly useful in scenarios such as the migration of data or the merging/fusion of companies and their information systems.

Modelling language. The STS methodology starts with the elicitation of stakeholders needs with respect to security to then finally derive the security requirements specification for the system-to-be. To do so, STS relies on the creation of security requirements models on the STS-ml language [Dalpiaz et al., 2011]. STS-ml models represent the stakeholders in the socio-technical system together with their assets (to be protected), their expressed security needs concerning assets, and the threats affecting these assets.

STS-ml was designed following a set of principles that, firstly, reflect the challenges faced in specifying secure socio-technical systems, and secondly, respect properties for general modelling languages. Modelling with STS-ml is based upon high-level representation of assets, modelling actors' goals, information, and documents. STS-ml clearly distinguishes information from its representation (in terms of documents), and keeps track of information structure and permissions granted or prohibitions specified over the use of information. This allows STS-ml to consider information as a first-class citizen (goals and documents are considered relevant from a security point of view because of the information they use and represent, respectively) when dealing with information security, and at the same time support a rich set of security requirements types.

Formal framework supporting automated analysis. We have proposed a descriptive formal framework that presents the semantics of STS-ml and enables automated analysis. We presented three analysis techniques: (i) *validity checking*, to determine model validity, (ii) *security analysis*, to identify conflicts among security requirements and among staheloders' business policies and security requirements, and (iii) *threat analysis*, to identify the impact of events on stakeholders' assets. In particular, we have used Disjunctive Datalog to define the formal semantics of the STS-ml modelling language and the supported security requirements types to identify potential violations of security requirements (as a result of conflicts).

Requirements modelling and analysis support framework, STS-Tool. The modelling and analysis activities of requirements engineers and analysts are supported by the STS-Tool. The STS-Tool provides a graphical interface for creating STS-ml models through three different views: social, information, and authorisation view. Multi-view modelling leads to the construction of three corresponding (sub)models of the said STS-ml model, namely social model, information model, and authorisation model. The tool supports inter-model consistency by allowing for well-formedness checks on the fly, ensuring the models comply with the syntax of the modelling language.

The STS-Tool provides analysis capabilities by integrating the formal framework for STS-ml. Security analysis is implemented in Disjunctive Datalog and the DLV Engine is integrated in the STS-Tool to support this analysis. The created STS-ml models are automatically mapped to formal specifications in order to allow reasoning. Once the results are produced, the opposite mapping is performed to visualise the findings of the analysis over the models and provide detailed analysis results to the security requirements engineer.

Self-evaluation and case study evaluation. We have shown the applicability of STS methodology to different domains with the help of an application scenario, namely TasLab, and two distinct case studies, namely eGovernment and Air Traffic Management. In particular, the methodology has been extensively used in the context of the European project Aniketos. Apart from our own validation (self-evaluation) conducting modelling and analysis activities with the application scenario, we have reported on the experience of practitioners in applying STS methodology on two the case studies. From our validation and that of the practitioners, the methodology has proved useful in modelling and reasoning over security requirements in the various application domains. Finally, with the help of a scalability study, we demonstrated the efficiency of our reasoning techniques in identifying security requirements conflicts. We used the TasLab case study for the scalability experiments, however, any of the case studies could have been used.

User-oriented empirical evaluation of the approach. We have evaluated the STS methodology, STS-ml modelling language and the modelling and analysis tool, STS-Tool, through a series of user-oriented empirical studies. STS is the result of ongoing work and an iterative development process that included several evaluation activities involving both practitioners and M.Sc. and PhD students. Working with different groups of users, especially with practitioners, has given us the chance to investigate how modellers use STS-ml and STS-Tool in real settings. The advantage of the STS methodology is that it provides a guideline for the requirements analyst and the security engineer for building the different models and capturing the applicable security requirements. The iterative nature of the methodology reflects the way practitioners work, which is decidedly iterative, with several iterations taking place, particularly during modelling. The number of revisions, however, varies from practitioner to practitioner.

We have observed a tendency of *learning by doing*, i.e., jumping into modelling phases without first knowing STS-ml and STS-Tool. This tendency has influenced some of the results presented in Chapter 9, given that the use of the STS methodology while modelling and reasoning with STS-ml and STS-Tool, requires extensive training. Nonetheless, this is not to be considered as a limitation of the methodology, rather it emphasizes the need for training, particularly for those new to goal-orientation. It is therefore important to help users understand the importance of either learning (given the extensive documentation on the STS website) or training activities, along with efforts to reduce their learning struggles. This suggests an important direction for future research, that of reducing the learning curve for STS-ml, e.g., by introducing self-learning mechanisms or step-by-step guidelines to acquire knowledge and skills for particular activities on-demand.

10.3 Ongoing and future work

While the STS methodology, STS-ml and STS-Tool are quite mature, several future directions remain open.

On the interplay with trust. Trust and security are closely related. In fact, a comprehensive security requirements engineering process requires a consideration of trust properties along with security properties that the system-to-be should satisfy. In [Chopra et al., 2011; Paja et al., 2013a] we have proposed a conceptual model for architectural trust that specifies a socio-technical system in terms of trust relationships. The model considers trust relationships to be essential in establishing interactions between the participants of a socio-technical system. The proposed approach to designing trustworthy socio-technical systems is commitment-based [Singh, 1999]. Commitments are used in the model as a warranty to create a more robust architecture of the system-to-be with respect to trust. Exploiting this concept, we have pro-

posed to iteratively specify a more trustworthy socio-technical system through a series of trust supporting mechanisms. The definition of trust supporting mechanisms relies on patterns of trust [Singh et al., 2009; Chopra and Singh, 2011], such as *mutualTrust*, *renegotiate*, *compensate*, and *revert*. The intuition behind introducing the trust supporting mechanisms (through a *supports* relationship) is exception-handling. If there is a supporting trust relation for a particular trust relationship, then exceptions pertaining to the latter can potentially be handled within the system itself. In the absence of a *supports* relationship, the exception would have to be dealt with outside the system. In [Paja et al., 2013a], we have presented a methodology to guide the specification of a socio-technical system in terms of trust relationships starting from a set of stakeholder requirements. Besides the requirements themselves, the methodology relies on domain knowledge in order to come up with the *supports* relationships.

The establishment of trust relationships has an effect on the security needs that stakeholders have when participating in interactions, in particular when exchanging information. Therefore, we want to enrich STS-ml with more specific trust requirements, and perform the mapping of the security and trust requirements from STS-ml to the architectural trust model. As such, the next step will be to define this mapping, together with some simulations to evaluate eventual designs that meet the given security and trust requirements. We will provide a clear semantics for the mechanisms, and perform reasoning to identify trust characteristics over models, evaluating the impact of applying more mechanisms and possible conflicts. Mapping security and trust specifications to trust supporting mechanisms is necessary, in order to take these specifications into account before the design is defined. The designer might complement stakeholders' requirements, adding missing information, which might be domain-specific.

Cost Estimation and relation to Cognitive Trust: building a trustworthy socio-technical system by introducing a number of trust supporting mechanisms can be expensive. We need to evaluate the cost of introducing these mechanisms, and when the benefits outweigh the costs. Perhaps, taking into account *cognitive trust*—referring to actors' mental models of each other [Castelfranchi and Tan, 2002]— will help in the process. Ideally, if two parties trust each other, they might not need trust supporting mechanisms to enter interactions, lowering the cost of an interaction. We want to explore how *cognitive trust* may influence the architectural design, and vice versa, how a good architectural trust model may influence *cognitive trust*.

Assessing privacy and analysing privacy requirements. STS supports privacy as confidentiality, by expressing security requirements concerning the use and manipulation of information by other parties. Although expressive in terms of security requirements it supports, our approach does not support other types of privacy requirements, such as *privacy as control* or *privacy as practice* [Gürses et al., 2010; Berendt, 2012]. As such, an interesting direction would be to extend STS-ml with other types of privacy requirements.

In light of these ideas, we have initiated work in collaboration with Professor Travis Breaux at Carnegie Mellon University, on using the STS methodology and extending STS-ml to capture stakeholders' privacy requirements when using mobile applications and participating in social networks. Specifically, we have conduced a user study in order to identify stakeholders' privacy requirements when interacting within a system, including how privacy requirements change with respect to time and space, as well as considering how stakeholders' needs with respect to privacy evolve when provided more information about the implications of their actions on privacy.

Conducting comparative evaluation study. We reported on our experience about the evaluation of the STS methodology with practitioners and M.Sc. and PhD students. However, we have not yet conducted a comparative study to evaluate the effectiveness of the STS methodology with respect to other security requirements engineering methodologies, such as, for instance, Secure Tropos. We intend to conduct a controlled experiment in the near future to help us determine the effectiveness and perception of the STS methodology as in [Labunets et al., 2013].

Informing later phases. Another interesting future direction involves exploring how STS-ml can inform later phases for the design of secure socio-technical systems, such as, for instance, the definition of access control policies. This will require mapping the security requirements specification derived from the process followed by the STS methodology to specification policy languages, e.g., XACML. We are currently working, in collaboration with Nicola Zannone, towards extending STS-ml with other organisational relationships among actors which introduce hierarchies and authority. These new relationships are necessary to apply policy-combining algorithms supported by XACML to help eliminate some of the conflicts detected by our current automated analysis techniques.

Optimising reasoning techniques. We intend to devise further analysis techniques for more sophisticated reasoning. In particular, efforts will be dedicated to further optimizing reasoning techniques by: (i) checking whether there is a variant in which there is no conflict, and checking whether there is one variant in which there is a conflict for a given security requirement, in order to avoid generating all possible variants; and (ii) setting priorities among security requirements according to their severity in order to eliminate some of the possible conflicts [Liaskos et al., 2010, 2011]. Last, but not least, we will explore techniques for conflict resolution [van Lamsweerde et al., 1998; Elahi and Yu, 2007].

Improving STS-Tool. The future developments envisaged in the previous paragraphs will have an impact on STS-Tool. Ongoing work includes: (i) the development of a new version of STS-

Tool with improved usability, and (ii) the possibility to have multiple models of the same type, e.g., three social models drawn by different analysts, in order to further promote separation of concerns and modularity during the modelling process. Most importantly, a future development of STS-Tool includes getting certified according to international security standards, such as, for instance, ISO27002 or ISO15408 (Common Criteria). Such a process requires a comprehensible documentation of the software product, including a detailed threat analysis [Beckers et al., 2013]. This step is important for the industrial uptake of STS-Tool from companies other than those we collaborated within the context of EU Funded Projects.

10.4 Future Lines of Research

Apart from ongoing and future work that aims to make STS more comprehensive and complete, the work reported in this thesis opens up new lines of research.

Secure evolution and adaptation. Socio-technical systems are continuously evolving due to various possible changes: (i) system and subsystems changes, (ii) organizational and domain changes, (iii) normative and regulations changes, and (iii) assets and requirements changes. Any of these changes might affect the ability of the socio-technical system to satisfy its intended requirements, and in particular the specified security requirements. Considering this evolving nature of socio-technical systems and the importance of dealing with security already at requirements time, we envisage, as part of this line of research, to:

- Advance current security requirements engineering techniques in order to manage the effects of evolution of socio-technical systems to maintain compliance with security requirements.

- Exploit and explore secure adaptation as a means of preserving compliance with security requirements. Threats to security requirements or properties will be considered as a trigger to adaptation. One interesting direction of research is that of investigating domain variability into STS-ml models through the use of awareness requirements [Silva Souza et al., 2011] and context-based mechanisms [Ali et al., 2009; Dalpiaz et al., 2013b].

Reverse engineering. Security requirements engineering methodologies are typically used to analyse a system-to-be, to then derive security requirements for the to-be system, assuming that there is no system already operating or that there is going to be a new version of the existing system. However, often it is the case that socio-technical systems are already working, but they might undergo changes or evolution. For instance, an emerging new technology such as the

cloud, has led many organizations to migrate their data to the cloud. On the other hand, organizational changes, such as fusion of companies and their information systems, might have an impact on security, which remains a desired property of the system. But, if we are dealing with an already running system, "*how do we establish what were the initial security requirements for the said socio-technical system?*" in order to maintain compliance with existing requirements.

Improving the learning curve for STS-ml. The proposed modelling language suffers from well-known goal-modelling limitations, such as scalability and the learning curve. In light of the results of the conducted evaluation activities, an interesting line of research is related to making efforts for dealing with scalability and reducing the learning curve for goal-oriented modelling languages. The idea would be that of starting with STS-ml, and making efforts about improving (reducing) the cognitive scalability, providing step by step guidelines, as well as pattern-based modelling to foster reuse. Such efforts should ideally have an impact not only on STS-ml, but also on other goal-oriented and model-driven approaches. Most importantly, they should contribute to improving the industrial uptake of goal-oriented modelling languages.

Serious games. Many areas of software development do not necessarily follow software engineering practices. One such area is game production, and in particular serious games, which might benefit from requirements engineering techniques to better capture the functional, non-functional, and other desired requirements a given game should satisfy. A variant of STS could be used for this purpose, for which we envisage the extension of STS, in particular STS-ml, with other requirements such as emotional requirements, which are essential to the assessment of a produced game. The STS approach would be applicable to such domain since a running game can be seen as a socio-technical system consisting of the game itself, its users, and the interactions among them. Following a requirements engineering approach will help the development of games that reflect not only functional requirements, but also users' needs resulting in other types of requirements.

Bibliography

Common Criteria for Information Technology Security Evaluation. Technical report, COMMON CRITERIA, September 2012.

COBIT 5 for Information Security. Technical report, ISACA, 2012.

Albert, Cecilia and Dorofee, Audrey J. Octave criteria, version 2.0. 2001.

Ali, Raian; Dalpiaz, Fabiano, and Giorgini, Paolo. A goal modeling framework for self-contextualizable software. In *Enterprise, Business-Process and Information Systems Modeling*, pages 326–338. Springer, 2009.

Anderson, Ross. *Security engineering: A guide to building dependable distributed systems*. Wiley, April 2008. ISBN 978-0-470-06852-6.

Appari, Ajit and Johnson, M Eric. Information security and privacy in healthcare: current state of research. *International journal of Internet and enterprise management*, 6(4):279–314, 2010.

Ashley, Paul; Hada, Satoshi; Karjoth, Günter; Powers, Calvin, and Schunter, Matthias. Enterprise privacy authorization language (epal 1.2). *Submission to W3C*, 2003.

Asnar, Yudistira; Paja, Elda, and Mylopoulos, John. Modeling design patterns with description logics: a case study. In *Proceedings of the 23rd International Conference on Advanced Information Systems Engineering*, volume 6741 of *LNCS*, pages 169–183, 2011.

Bangor, Aaron; Kortum, Philip, and Miller, James. Determining what individual sus scores mean: adding an adjective rating scale. *Journal of Usability Studies*, 4(3):114–123, May 2009.

Basin, David; Clavel, Manuel, and Egea, Marina. A decade of model-driven security. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 1–10. ACM, 2011.

Baxter, Gordon and Sommerville, Ian. Socio-technical systems: From design methods to systems engineering. *Interacting with Computers*, 23(1):4–17, 2011.

Becker, Moritz Y; Fournet, Cédric, and Gordon, Andrew D. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

Beckers, Kristian. Comparing privacy requirements engineering approaches. In *2012 Seventh International Conference on Availability, Reliability and Security (ARES)*, pages 574–581. IEEE, 2012.

Beckers, Kristian; Hatebur, Denis, and Heisel, Maritta. A problem-based threat analysis in compliance with common criteria. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 111–120. IEEE, 2013.

Berendt, Bettina. More than modelling and hiding: towards a comprehensive view of web mining and privacy. *Data Mining and Knowledge Discovery*, 24(3):697–737, 2012.

Breaux, Travis D. and Antón, Annie I. Analyzing Regulatory Rules for Privacy and Security Requirements. *IEEE Transactions on Software Engineering*, 34(1):5 –20, 2008. ISSN 0098-5589. doi: 10.1109/TSE.2007.70746.

Bresciani, Paolo; Perini, Anna; Giorgini, Paolo; Giunchiglia, Fausto, and Mylopoulos, John. Tropos: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8 (3):203–236, May 2004.

Brucker, Achim D.; Doser, Jürgen, and Wolff, Burkhart. A model transformation semantics and analysis methodology for secureuml. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems*, volume 4199 of *LNCS*, pages 306–320, 2006.

Butch, Suzanne H. Computerization in the transfusion service. *Vox sanguinis*, 83(s1):105–110, 2002.

Cardoso, Evellin; Almeida, João Paulo A.; Guizzardi, Renata S.S., and Guizzardi, Giancarlo. A method for eliciting goals for business process models based on non-functional requirements catalogues. *International Journal of Information System Modeling and Design*, 2(2):1–18, 2011.

Castelfranchi, Cristiano and Tan, Yao-Hua. The role of trust and deception in virtual societies. *International Journal of Electronic Commerce*, 6(3):55–70, 2002.

Chopra, Amit K. and Singh, Munindar P. Specifying and applying commitment-based business patterns. In *Proceedings of the 10th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 475–482. IFAAMAS, 2011.

Chopra, Amit K.; Paja, Elda, and Giorgini, Paolo. Sociotechnical trust: An architectural approach. In *Proceedings of 30th International Conference on Conceptual Modeling*, pages 104–117, 2011.

Colwill, Carl. Human factors in information security: The insider threat–who can you trust these days? *Information security technical report*, 14(4):186–196, 2009.

Constante, Elisa; Paci, Federica, and Zannone, Nicola. Privacy-aware web service composition and ranking. In *20th International Conference on Web Services*, pages 131–138. IEEE, 2013.

Cranor, Lorrie; Langheinrich, Marc; Marchiori, Massimo; Presler-Marshall, Martin, and Reagle, Joseph. The platform for privacy preferences 1.0 (p3p1.0) specification. URL `http://www.w3.org/TR/P3P/`.

Dalpiaz, Fabiano; Ali, Raian; Asnar, Yudistira; Bryl, Volha, and Giorgini, Paolo. Applying tropos to socio-technical system design and runtime configuration. In *Proceedings of the 9th Workshop on Objects and Agents*, pages 101–107, 2008.

Dalpiaz, Fabiano; Chopra, Amit K; Giorgini, Paolo, and Mylopoulos, John. Adaptation in open systems: giving interaction its rightful place. In *Proceedings of the 29th International Conference on Conceptual Modeling*, volume 6412 of *LNCS*, pages 31–45, 2010.

Dalpiaz, Fabiano; Paja, Elda, and Giorgini, Paolo. Security requirements engineering via commitments. In *Proceedings of the First Workshop on Socio-Technical Aspects in Security and Trust*, pages 1–8, 2011.

Dalpiaz, Fabiano; Giorgini, Paolo, and Mylopoulos, John. Adaptive socio-technical systems: a requirements-driven approach. *Requirements Engineering*, 18(4):1–24, 2013a. ISSN 0947-3602. URL `http://dx.doi.org/10.1007/s00766-010-0110-z`.

Dalpiaz, Fabiano; Giorgini, Paolo, and Mylopoulos, John. Adaptive socio-technical systems: a requirements-based approach. *Requirements engineering*, 18(1):1–24, 2013b.

Dalpiaz, Fabiano; Paja, Elda, and Giorgini, Paolo. *Security Requirements Engineering: Designing Secure Socio-Technical Systems*. Accepted for publication, MIT Press, 2014.

Danezis, George and Gürses, Seda. A critical review of 10 years of privacy technology. *Proceedings of Surveillance Cultures: A Global Surveillance Society*, 2010.

Dardenne, Anne; van Lamsweerde, Axel, and Fickas, Stephen. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, April 1993.

De Landtsheer, Renaud and Van Lamsweerde, Axel. Reasoning about confidentiality at requirements engineering time. In *Proceedings of the 13th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 41–49, 2005.

Deng, Mina; Wuyts, Kim; Scandariato, Riccardo; Preneel, Bart, and Joosen, Wouter. A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32, 2011.

Devanbu, Premkumar T. and Stubblebine, Stuart. Software Engineering for Security: a Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (FOSE 2000)*, pages 227–239, 2000. ISBN 1581132530.

Dubois, Eric and Mouratidis, Haralambos. Guest Editorial: Security Requirements Engineering: Past, Present and Future. *Requirements Engineering*, 15(1):1–5, 2010. ISSN 0947-3602.

Easterbrook, Steve. An introduction to formal modeling in requirements engineering. In *Proceedings of the 10th Joint International Requirements Engineering Conference, Essen, Germany*, 2002.

Elahi, Golnaz and Yu, Eric. A goal oriented approach for modeling and analyzing security trade-offs. In *Proceedings of the 26th International Conference on Conceptual Modeling*, volume 4801 of *LNCS*, pages 375–390, 2007.

Emery, Fred E. Characteristics of socio-technical systems. Technical Report 527, London: Tavistock Institute, 1959.

Ernst, Neil A; Mylopoulos, John, and Wang, Yiqiao. Requirements evolution and what (research) to do about it. In *Design Requirements Engineering: A Ten-Year Perspective*, pages 186–214. Springer, 2009.

Ernst, Neil A.; Borgida, Alex; Mylopoulos, John, and Jureta, Ivan J. Agile requirements evolution via paraconsistent reasoning. In *Proceedings of 24th International Conference on Advanced Information Systems Engineering*, pages 382–397. 2012.

Eurocontrol, (Producer). System wide information management (swim), April 2013. URL `http://www.eurocontrol.int/services/system-wide-information-management-swim`.

Fabian, Benjamin; Gürses, Seda; Heisel, Maritta; Santen, Thomas, and Schmidt, Holger. A comparison of security requirements engineering methods. *Requirements engineering*, 15(1):7–40, 2010.

Finkelstein, Anthony; Gabbay, Dov; Hunter, Anthony; Kramer, Jeff, and Nuseibeh, Bashar. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8): 569–578, 1994.

Firesmith, Donald G. Security use cases. *Journal of Object Technology*, 2(3):53–64, May–June 2003.

Foundation, The Eclipse. Gef (mvc), 2014. URL `http://www.eclipse.org/gef/gef_mvc/index.php`. Lastchecked: March, 2014.

Fuxman, Ariel; Pistore, Marco; Mylopoulos, John, and Traverso, Paolo. Model checking early requirements specifications in tropos. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 174–181, 2001.

Giorgini, Paolo; Mylopoulos, John; Nicchiarelli, Eleonora, and Sebastiani, Roberto. Reasoning with goal models. In *Proceedings of the 21st International Conference on Conceptual Modeling*, volume 2503 of *LNCS*, pages 167–181, 2002.

Giorgini, Paolo; Massacci, Fabio, and Mylopoulos, John. 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*, volume 2813 of *LNCS*, pages 263–276, 2003.

Giorgini, Paolo; Massacci, Fabio; Mylopoulos, John, and Zannone, Nicola. Modeling security requirements through ownership, permission and delegation. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 167–176, 2005a.

Giorgini, Paolo; Mylopoulos, John, and Sebastiani, Roberto. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005b.

Giorgini, Paolo; Massacci, Fabio; Mylopoulos, John, and Zannone, Nicola. Requirements engineering for trust management: model, methodology, and reasoning. *International Journal of Information Security*, 5:257–274, October 2006.

Golafshani, Nahid. Understanding reliability and validity in qualitative research. *The Qualitative Report*, 8(4):597–607, December 2003.

Gollmann, Dieter. *Computer security*. John Wiley & Sons, 3 edition, 2011.

Guizzardi, Giancarlo. Agent roles, qua individuals and the counting problem. In *Software Engineering for Multi-Agent Systems IV*, pages 143–160. Springer, 2006.

Gürses, Seda; HMDB, CS; COSIC, ESAT, and Leuven, KU. *Multilateral Privacy Requirements Analysis in Online Social Networks*. PhD thesis, Ph. D. Thesis, HMDB, Department of Computer Science, KU Leuven, Belgium, May, 2010.

Haley, Charles B.; Laney, Robin R.; Moffett, Jonathan D., and Nuseibeh, Bashar. Security requirements engineering: a framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, January–February 2008.

Hernan, Shawn; Lambert, Scott; Ostwald, Tomasz, and Shostack, Adam. Threat modeling-uncover security design flaws using the stride approach. *MSDN Magazine*, pages 68–75, November 2006.

Horkoff, Jennifer and Yu, Eric. Evaluating goal achievement in enterprise modeling–an interactive procedure and experiences. In *The Practice of Enterprise Modeling*, pages 145–160. Springer, 2009.

Horkoff, Jennifer and Yu, Eric. Finding solutions in goal models: an interactive backward reasoning approach. In *Proceedings of the 29th International Conference on Conceptual Modeling*, volume 6412 of *LNCS*, pages 59–75, 2010.

Horkoff, Jennifer and Yu, Eric. Analyzing goal models: different approaches and how to choose among them. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 675–682. ACM, 2011.

Houmb, Siv Hilde; Islam, Shareeful; Knauss, Eric; Jürjens, Jan, and Schneider, Kurt. Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec. *Requirements Engineering*, 15(1):63–93, March 2010.

ISO/IEC, . BS ISO/IEC 27002:2005. Technical report, 2005.

Jackson, Michael. *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

Johnstone, Michael N. Security requirements engineering-the reluctant oxymoron. In *Australian Information Security Management Conference*, page 5, 2009.

Jureta, Ivan; Borgida, Alexander; Ernst, Neil A, and Mylopoulos, John. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Proceeding of the 18th IEEE International Requirements Engineering Conference*, pages 115–124, 2010.

Jürjens, Jan. Umlsec: Extending uml for secure systems development. In *Proceedings of the 5th International Conference on Model Engineering, Concepts, and Tools*, volume 2460 of *LNCS*, pages 412–425, 2002.

Jürjens, Jan; Marchal, Loïc; Ochoa, Martín, and Schmidt, Holger. Incremental security verification for evolving umlsec models. In *Proceedings of the 7th European Conference on Modelling Foundations and Applications*, volume 6698 of *LNCS*, pages 52–68, 2011.

Kalloniatis, Christos; Kavakli, Evangelia, and Gritzalis, Stefanos. Addressing privacy requirements in system design: the pris method. *Requirements Engineering*, 13(3):241–255, 2008.

Kissel, Richard. Glossary of key information security terms. Technical Report IR 7298 Rev 1, NIST, 2011.

Kissel, Richard. Glossary of key information security terms. Technical Report IR 7298 Revision 2, NIST, May 2013.

Labunets, Katsiaryna; Massacci, Fabio; Paci, Federica, and Tran, Le Minh Sang. An experimental comparison of two risk-based security methods. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 163–172. IEEE, 2013.

Lapouchnian, Alexei. Goal-oriented requirements engineering: An overview of the current research. Technical Report Technical report, University of Toronto, Canada (available online: `http://www.cs.toronto.edu/~alexei/pub/Lapouchnian-Depth.pdf`), 2005.

Lewis, Nicole. Health it managers say tablets can cause problems, 2012. URL `http://www.informationweek.com/regulations/health-it-managers-say-tablets-can-cause-problems/d/d-id/1102641?`

Liaskos, Sotirios; McIlraith, Sheila A; Sohrabi, Shirin, and Mylopoulos, John. Integrating preferences into goal models for requirements engineering. In *18th IEEE International Requirements Engineering Conference*, pages 135–144. IEEE, 2010.

Liaskos, Sotirios; McIlraith, Sheila A; Sohrabi, Shirin, and Mylopoulos, John. Representing and reasoning about preferences in requirements engineering. *Requirements Engineering*, 16(3):227–249, 2011.

Liu, Lin; Yu, Eric, and Mylopoulos, John. Analyzing security requirements as relationships among strategic actors. In *Symposium on Requirements Engineering for Information Security*, 2002.

Liu, Lin; Yu, Eric, and Mylopoulos, John. Security and privacy requirements analysis within a social setting. In *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, pages 151–161, 2003.

Lodderstedt, Torsten; Basin, David, and Doser, Jürgen. Secureuml: A uml-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on Model Engineering, Concepts, and Tools*, volume 2460 of *LNCS*, pages 426–441, 2002.

Loucopoulos, Pericles and Karakostas, Vassilios. *System requirements engineering*. McGraw-Hill, Inc., 1995.

Lund, Mass Soldal; Solhaug, Bjørnar, and Stølen, Ketil. *Model-driven risk analysis: the CORAS approach*. Springer, 2010.

Masolo, Claudio; Vieu, Laure; Bottazzi, Emanuele; Catenacci, Carola; Ferrario, Roberta; Gangemi, Aldo, and Guarino, Nicola. Social roles and their descriptions. In *9th International Conference on the Principles of Knowledge Representation and Reasoning*, pages 267–277, 2004.

Massacci, Fabio and Zannone, Nicola. Detecting conflicts between functional and security requirements with secure tropos: John rusnak and the allied irish bank. *Social modeling for requirements engineering. MIT Press, Cambridge*, 2008.

McDermott, John and Fox, Chris. Using abuse case models for security requirements analysis. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 55–64, 1999.

Mead, Nancy R.; Hough, Eric D., and Stehney II, Theodore R. Security quality requirements engineering (square) methodology. Technical Report CMU/SEI-2005-009, 2005.

Meland, Per Håkon; Paja, Elda; Gjære, Erlend Andreas; Paul, Stéphane; Dalpiaz, Fabiano, and Giorgini, Paolo. Threat analysis in goal-oriented security requirements modelling. *International Journal of Secure Software Engineering*, 2014.

Mellado, Daniel; Fernández-Medina, Eduardo, and Piattini, Mario. A common criteria based security requirements engineering process for the development of secure information systems. *Computer standards & interfaces*, 29(2):244–253, February 2007.

Mellado, Daniel; Blanco, Carlos; Sánchez, Luis E., and Fernández-Medina, Eduardo. A systematic review of security requirements engineering. *Computer Standards & Interfaces*, 32(4):153–165, jun 2010.

Menzel, Michael; Thomas, Ivonne, and Meinel, Cristoph. Security requirements specification in service-oriented business process management. In *2009 International Conference on Availability, Reliability and Security*, pages 41–48. IEEE, 2009.

Mohammed, Noman; Fung, Benjamin; Hung, Patrick CK, and Lee, Cheuk-kwong. Anonymizing healthcare data: a case study on the blood transfusion service. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1285–1294. ACM, 2009.

Moody, Daniel L. The method evaluation model: a theoretical model for validating information systems design methods. In *Proceedings of the 11th European Conference on Information Systems*, pages 1327–1336, 2003.

Moody, Daniel L. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.

Moody, Daniel L.; Heymans, Patrick, and Matulevicius, Raimundas. Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax. In *17th IEEE International Requirements Engineering Conference*, pages 171–180. IEEE, 2009.

Mouratidis, Haralambos and Giorgini, Paolo. Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2): 285–309, April 2007a.

Mouratidis, Haralambos and Giorgini, Paolo. *Integrating security and software engineering: Advances and future visions*. Igi Global, 2007b.

Mouratidis, Haralambos; Giorgini, Paolo, and Manson, Gordon. Integrating security and systems engineering: Towards the modelling of secure information systems. In *Advanced Information Systems Engineering*, pages 63–78. Springer, 2003.

Mylopoulos, John; Chung, Lawrence, and Nixon, Brian. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, 1992.

Mylopoulos, John; Chung, Lawrence, and Yu, Eric. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.

Northover, Steve and Wilson, Mike. *Swt: the standard widget toolkit, volume 1*. Addison-Wesley Professional, 2004.

Northrop, Linda; Feiler, Peter; Gabriel, Richard P.; Goodenough, John; Linger, Rick; Kazman, Rick; Schmidt, Douglas; Sullivan, Kevin, and Wallnau, Kurt. Ultra-large-scale systems—the software challenge of the future. *Technical report Software Engineering Institute Carnegie Mellon University*, 2006.

OASIS, Standard. extensible access control markup language (xacml) version 3.0, January 2013. URL `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`.

Object Management Group, Standard. Business process model and notation 2.0. http://www.omg.org/spec/BPMN/2.0, Jan 2011.

Paja, Elda; Dalpiaz, Fabiano; Poggianella, Mauro; Roberti, Pierluigi, and Giorgini, Paolo. Modelling security requirements in socio-technical systems with STS-Tool. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering, CAiSE Forum*, volume 855 of *CEUR-WS*, pages 155–162, 2012a.

Paja, Elda; Dalpiaz, Fabiano; Poggianella, Mauro; Roberti, Pierluigi, and Giorgini, Paolo. STS-Tool: Using commitments to specify socio-technical security requirements. In *Proceedings of the 31st International Conference on Conceptual Modeling, ER Workshops*, pages 396–399, 2012b.

Paja, Elda; Dalpiaz, Fabiano; Poggianella, Mauro; Roberti, Pierluigi, and Giorgini, Paolo. STS-Tool: socio-technical security requirements through social commitments. In *Proceedings of the 20th IEEE International Conference on Requirements Engineering*, pages 331–332, 2012c.

Paja, Elda; Chopra, Amit K, and Giorgini, Paolo. Trust-based specification of sociotechnical systems. *Data & Knowledge Engineering*, 87:339–353, 2013a.

Paja, Elda; Dalpiaz, Fabiano, and Giorgini, Paolo. Managing security requirements conflicts in socio-technical systems. In *Proceedings of the 32nd International Conference on Conceptual Modeling*, volume 8217 of *LNCS*, pages 270–283, 2013b.

Paja, Elda; Dalpiaz, Fabiano, and Giorgini, Paolo. Designing secure socio-technical systems with STS-ml. In *6th International i* Workshop (iStar'13)*, 2013c.

Paja, Elda; Dalpiaz, Fabiano; Poggianella, Mauro; Roberti, Pierluigi, and Giorgini, Paolo. Specifying and reasoning over socio-technical security requirements with STS-Tool. In *Proceedings of the 32nd International Conference on Conceptual Modeling, ER Workshops*, pages 504–507, 2013d.

Paja, Elda; Dalpiaz, Fabiano; Poggianella, Mauro; Roberti, Pierluigi, and Giorgini, Paolo. STS-Tool: Specifying and reasoning over socio-technical security requirements. In *6th International i* Workshop (iStar'13)*, pages 79–84, 2013e.

Paja, Elda; Dalpiaz, Fabiano, and Giorgini, Paolo. The socio-technical security requirements modelling language for secure composite services. In *Aniketos book*. 2014a.

Paja, Elda; Dalpiaz, Fabiano, and Giorgini, Paolo. Sts-tool: Security requirements engineering for socio-technical systems. In *NESSOS book*. 2014b.

Paja, Elda; Poggianella, Mauro; Dalpiaz, Fabiano; Roberti, Pierluigi, and Giorgini, Paolo. Security requirements engineering with sts-tool. In *Aniketos book*. 2014c.

Pavlovski, Christopher J. and Zou, Joe. Non-functional requirements in business process modeling. In *Proceedings of the fifth Asia-Pacific conference on Conceptual Modelling-Volume 79*, pages 103–112. Australian Computer Society, Inc., 2008.

Pfleeger, Charles P. and Pfleeger, Shari L. *Analyzing computer security: a threat/vulnerability/countermeasure approach*. Prentice Hall, 2012.

PUB, FIPS. Standards for security categorization of federal information and information systems. 2004.

Rodríguez, Alfonso; Fernández-Medina, Eduardo, and Piattini, Mario. A bpmn extension for the modeling of security requirements in business processes. *IEICE transactions on information and systems*, 90(4):745–752, 2007.

Rosson, Mary B. and Carroll, John J. M. *Usability engineering: scenario-based development of human-computer interaction*. Morgan Kaufmann, 2002.

Roy Sarkar, Kuheli. Assessing insider threats to information security using technical, behavioural and organisational measures. *Information Security Technical Report*, 15(3):112–133, 2010.

Russell, Nick; Ter Hofstede, Arthur HM; Edmond, David, and van der Aalst, Wil MP. Workflow resource patterns. Technical report, BETA Working Paper Series, WP 127, Eindhoven University of Technology, 2004.

Sankar, Pamela; Mora, Susan; Merz, Jon F, and Jones, Nora L. Patient perspectives of medical confidentiality. *Journal of general internal medicine*, 18(8):659–669, 2003.

Sauro, Jeff. Measuring usability with the system usability scale (sus), February 2011. URL `http://www.measuringusability.com/sus.php`.

Schumacher, Markus; Fernandez-Buglioni, Eduardo; Hybertson, Duane; Buschmann, Frank, and Sommerlad, Peter. *Security patterns: integrating security and systems engineering*. John Wiley & Sons, 2005.

Shvaiko, Pavel; Mion, Luca; Dalpiaz, Fabiano, and Angelini, Giuseppe. The taslab portal for collaborative innovation. In *Proceedings of the 16th International Conference on Concurrent Enterprising*, 2010.

Silva Souza, Vítor E; Lapouchnian, Alexei; Robinson, William N, and Mylopoulos, John. Awareness requirements for adaptive systems. In *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*, pages 60–69. ACM, 2011.

Sindre, G. Mal-activity diagrams for capturing attacks on business processes. *Requirements Engineering: Foundation for Software Quality*, pages 355–366, 2007.

Sindre, Guttorm and Opdahl, Andreas L. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, January 2005.

Singh, Munindar P. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

Singh, Munindar P.; Chopra, Amit K., and Desai, Nirmit. Commitment-based service-oriented architecture. *IEEE Computer Society*, 42(11):72–79, 2009. ISSN 0018-9162.

Sommerville, Ian and Sawyer, Pete. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

Sommerville, Ian; Cliff, Dave; Calinescu, Radu; Keen, Justin; Kelly, Tim; Kwiatkowska, Marta; Mcdermid, John, and Paige, Richard. Large-scale complex IT systems. *Communications of the ACM*, 55(7): 71–77, July 2012.

Spiekermann, Sarah and Cranor, Lorrie Faith. Engineering privacy. *IEEE Transactions on Software Engineering*, 35(1):67–82, 2009.

Stallings, William and Brown, Lawrence V. *Computer Security*. Prentice Hall, 2008.

Tixier, Jerome; Dusserre, G; Salvi, O, and Gaston, D. Review of 62 risk analysis methodologies of industrial plants. *Journal of Loss Prevention in the Process Industries*, 15(4):291–303, 2002.

Trösterer, Sandra; Beck, Elke; Dalpiaz, Fabiano; Paja, Elda; Giorgini, Paolo, and Tscheligi, Manfred. Formative user-centered evaluation of security modeling: results from a case study. *International Journal of Secure Software Engineering*, 3(1):1–19, 2012.

van Lamsweerde, Alex. Goal-oriented Requirements Engineering: A Guided Tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 249–263, 2001.

van Lamsweerde, Axel. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th International Conference on Software Engineering*, pages 148–157, 2004.

van Lamsweerde, Axel. Requirements engineering: From system goals to uml models to software specifications, 2009.

van Lamsweerde, Axel; Darimont, Robert, and Letier, Emmanuel. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, November 1998.

Vogel, Lars. Building eclipse rcp applications based on eclipse 4, 2013. URL `http://www.vogella.com/tutorials/EclipseRCP/article.html`. Revision history: Revision 0.1 - 6.9 14.02.2009 - 04.07.2013.

Wolter, Christian; Menzel, Michael, and Meinel, Cristoph. Modelling security goals in business processes. *Modellierung 2008*, 127:201–216, 2008.

Xenos, Stefan. Inside the workbench a guide to the workbench internals, October 2005. URL `http://www.eclipse.org/articles/Article-UI-Workbench/workbench.html`. Lastchecked: March, 2014.

Yu, Eric. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, Canada, 1995.

Yu, Eric and Cysneiros, Luiz. Designing for privacy and other competing requirements. In *2nd Symposium on Requirements Engineering for Information Security, Raleigh, North Carolina*, pages 15–16, 2002.

Yu, Eric and Liu, Lin. Modelling trust for system design using the i* strategic actors framework. In *Trust in Cyber-societies*, pages 175–194. Springer, 2001.

Yu, Eric and Mylopoulos, John. Understanding "why" in software process modelling, analysis, and design. In *Proceedings of the 16th International Conference on Software Engineering*, pages 159–168, 1994.

Yu, Eric; Giorgini, Paolo; Maiden, Neil, and Mylopoulos, John. *Social Modeling for Requirements Engineering*. MIT Press, 2011.

Yu, Eric SK. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 226–235. IEEE, 1997.

Zannone, Nicola. *A requirements engineering methodology for trust, security, and privacy*. PhD thesis, Ph. D. dissertation, University of Trento, 2007.

Zhu, Yan-xin; Qian, Zheng; Ding, Yi-qiang; Zhao, Na, and Wang, Yu. Enforce medical risk management to improve medical service qualitity. *Journal of Medical Postgraduates*, 1:023, 2006.

Appendix A

Security Requirements in STS-ml

We list the security requirements supported by STS-ml, grouping them with respect to the social relationships that allow expressing them. We provide their syntax and description of the expressed security requirement.

Table A.1 presents security requirement types supported over goal delegation relationships, Table A.2 presents the types of security requirements that can be expressed over document transmission relationships, Table A.3 presents the types of security requirements constraining responsibility uptake, and last but not least, Table A.4 presents the types of security requirements implicitly captured through authorisation relationships.

Table A.1: Security requirement types over goal delegations

Requirement type		**Description**
R_1	non-repudiation-del$(A_2, A_1, \texttt{Del})$	the delegatee (A_2) requires the delegator (A_1) not to repudiate the delegation `Del`
R_2	non-repudiation-acc$(A_1, A_2, \texttt{Del})$	the delegator (A_1) requires the delegatee (A_2) not to repudiate the acceptance of the delegation `Del`
R_3	true-single-red(A_1, A_2, G)	the delegator (A_1) requires the delegatee (A_2) true redundancy single for the achievement of goal G
R_4	fback-single-red(A_1, A_2, G)	the delegator (A_1) requires the delegatee (A_2) fallback redundancy single for achieving goal G
R_5	true-multi-red(A_1, A_2, G)	the delegator (A_1) requires the delegatee (A_2) true redundancy multi for the achievement of goal G
R_6	fback-multi-red(A_1, A_2, G)	the delegator (A_1) requires the delegatee (A_2) fallback redundancy multi for achieving goal G
R_7	no-redelegation(A_1, A_2, G)	the delegator (A_1) wants the delegatee (A_2) not to further delegate fulfilment of goal G [1]
R_8	trustworthiness(A_1, A_2, G)	the delegator (A_1) requires the delegatee (A_2) to be trustworthy in order to delegate it the fulfillment of G
R_9	goal-availability(A_2, A_1, G)	the delegator (A_1) wants the delegatee (A_2) to guarantee a minimum availability level for goal G
R_{10}	delegator-auth$(A_2, A_1, \texttt{Del})$	the delegatee (A_2) needs that the delegator (A_1) authenticates herself
R_{11}	delegatee-auth$(A_1, A_2, \texttt{Del})$	the delegator (A_1) needs that the delegatee (A_2) authenticates herself

Table A.2: Security requirement types over document transmissions

Requirement type		**Description**
R_{12}	non-repudiation-tx$(A_2, A_1, \mathtt{Tx})$	the receiver (A_2) requires the sender (A_1) not to repudiate the transmission `Tx`
R_{13}	non-repudiation-acc$(A_1, A_2, \mathtt{Tx})$	the sender (A_1) requires the receiver (A_2) not to repudiate the acceptance of the transmission `Tx`
R_{14}	sender-integrity$(A_2, A_1, \mathtt{Tx})$	the receiver (A_2) requires the sender (A_1) to ensure the integrity of transmission for the document being transmitted
R_{15}	receiver-integrity$(A_1, A_2, \mathtt{Tx})$	the sender (A_1) requires the receiver (A_2) to ensure the integrity of transmission for the document being transmitted
R_{16}	system-integrity$(\mathtt{STS}_{,}, \mathtt{Tx})$	the system shall ensure that the integrity of transmission of the document in transmission is preserved
R_{17}	doc-availability(A_2, A_1, D)	the receiver (A_2) requires the sender (A_1) to guarantee an availability level expressed in percentage (x%) for the transmitted document (D)
R_{18}	sender-auth$(A_2, A_1, \mathtt{Tx})$	the receiver (A_2) needs that the sender (A_1) authenticates herself to transfer the document
R_{19}	receiver-auth$(A_1, A_2, \mathtt{Tx})$	the sender (A_1) needs that the receiver (A_2) authenticates herself to have the document
R_{20}	sender-conf$(A_2, A_1, \mathtt{Tx})$	the receiver (A_2) requires the sender (A_1) to ensure the confidentiality of the document being transmited
R_{21}	receiver-conf$(A_1, A_2, \mathtt{Tx})$	the sender (A_1) requires the receiver (A_2) to ensure the confidentiality of the document being transmited
R_{22}	system-conf$(\mathtt{STS}_{,}, \mathtt{Tx})$	the system shall ensure that the confidentiality of transmission of a document in transfer is preserved

Table A.3: Security requirement types over responsibility uptake

Requirement type		**Description**
R_{23}	role-sod($\mathtt{STS}, Ag, R_1, R_2$)	no agent Ag can play both roles R_1 and R_2
R_{24}	goal-sod($\mathtt{STS}, Ag, G_1, G_2$)	every agent Ag must not pursue both goals G_1 and G_2
R_{25}	role-cod($\mathtt{STS}, Ag, R_1, R_2$)	every agent Ag playing role R_1 (R_2), must also play R_2 (R_1)
R_{26}	goal-cod($\mathtt{STS}, Ag, G_1, G_2$)	an agent Ag pursuing goal G_1 (G_2), should also pursue G_2 (G_1) too

Table A.4: Security requirement types over authorisation relationships

Requirement type		**Description**
R_{27}	need-to-know($A_1, A_2, \mathcal{I}, \mathcal{G}$)	the authoriser actor (A_1) requires the authorisee (A_2) not to perform any operation (use/modify/produce) on documents that make some information in $\mathcal{I}$ tangible, for any goals not included in $\mathcal{G}$
R_{28}	non-reading($A_1, A_2, \mathcal{I}$)	the authoriser actor (A_1) requires the authorisee (A_2) not to read documents representing information in $\mathcal{I}$
R_{29}	non-modification($A_1, A_2, \mathcal{I}$)	the authoriser actor (A_1) requires the authorisee (A_2) not to modify documents that include information in $\mathcal{I}$
R_{30}	non-production($A_1, A_2, \mathcal{I}$)	the authoriser actor (A_1) requires the authorisee (A_2) not to produce any documents that include information in $\mathcal{I}$
R_{31}	non-disclosure($A_1, A_2, \mathcal{I}$)	the authoriser actor (A_1) requires the authorisee (A_2) not to transmit (disclose) to other actors any document that includes information in $\mathcal{I}$
R_{32}	non-reauthorisation($A_1, A_2, \mathcal{I}, \mathcal{G}$, $\{\mathtt{R}, \mathtt{M}, \mathtt{P}, \mathtt{T}\}$)	the authoriser actor (A_1) requires the authorisee (A_2) not to redistribute the permissions to other actors

Appendix B

Reasoning about conflicts in STS-ml using Datalog

We have implemented our framework using Datalog, to support the automatic identification of conflicting authorisations as well as that of violations of security requirements.

In the following, we present the Datalog rules for identifying conflicts, together with the general rules necessary for defining the propagation of properties, as well as for capturing actors' business policies.

B.1 Informational Knowledge Base

Table B.1 presents the rules for the model's informational knowledge base, which define when a given actor possesses a certain document (rules 1-4): an actor possess a document that is within his model (has-in-scope) (1), the actor is not producing the document and no other actor is providing this document to him (2), the actor has a goal that produces the document and possesses such document being the first actor to create the document (3), and finally an actor possesses a document if it is provided the document by some other actor (4). Additionally, the rules specify ownership propagation over parts of information (rule 5), that is, an actor that owns a given information, owns also its constituent pieces of information.

Table B.1: Informational Knowledge Base Rules

Id	Datalog Rules
1.	possesses(A, D) :- has_in_scope(A, D), $0 = \#$count$\{G$:- produce$(A, D, G)\}$, $0 = \#$count$\{A_1$: transmits$(A_1, A, D)\}$.
2.	possesses(A, D) :- produces(A, D, G), has(A, G).
3.	transmitted(A_1, A_2, D) :- possesses(A_1, D), transmits(A_1, A_2, D), $A_1 \neq A_2$.
4.	possesses(A_2, D) :- transmitted$(_, A_2, D)$.
5.	own(A, I_1) :- own(A, I), partOfl(I_1, I).

B.2 Verifying Security Requirements over Goal Delegations

Table B.2 and B.3 present the datalog rules for the verification of no-redelegation and redundancy respectively. These checks identify a conflict if there is a conflict in at least one variant of the considered STS-ml model. no-redelegation is violated if this security requirement is specified over the delegation, but the delegatee redelegates the delegatum or a subgoal of it.

Table B.2: Interaction Requirements Verification: No-redelegation

Id	Datalog Rules for no-redelegation
1.	violate-no-redelegation(A_2, A_1, G, G_i) :- delegated(A_1, A_2, G), no-redelegation(A_1, A_2, G, G_i), delegated$(A_2, _, G_i)$.
2.	no-redelegation(A_1, A_2, G, G_i) :- no-redelegation$(A_1, A_2, _, G)$, has(A_2, G), is-refined(A_2, G, G_i).
3.	has(A, G_i) :- has(A, G), and-dec(A, G), is-refined(A_2, G, G_i).
4.	has$(A, G_i) \vee -$has(A, G_i) :- has(A, G), or-dec(A, G), is-refined(A_2, G, G_i).
5.	$-$has(A, G_i):-or-dec(A, G),$0 = \#$count$\{G_i$:is-refined(A, G, G_i),has$(A, G_i)\}$.
6.	$-$has(A, G_i) :- or-dec(A, G),$1 < \#$count$\{G_i$:is-refined(A, G, G_i),has$(A, G_i)\}$.
7.	delegated(A_1, A_2, G_i) :- has(A_1, G), delegates(A_1, A_2, G_i).
8.	has(A_2, G_i) :- delegated$(_, A_2, G_i)$.
9.	subgoal(G_i, G, A) :- is-refined(A, G, G_i).
10.	subgoal(G_1, G_2, A) :- subgoal(G_1, G_3, A), subgoal(G_3, G_2, A).

The verification of redundancy considers goal tree as composed of and-decompositions (all decompositions, being them or-decompositions or and-decompositions are considered as and-decompositions by the framework), to be *pursued* by the actor. This is the only way (and a partial one) for us to reason over redundancy, for decisions on which alternative the actor has selected are verifiable only at runtime. This means that only one variant is generated, since we cannot verify redundancy in case only one alternative is selected to accomplish the desired goal.

Table B.3: Interaction Requirements Verification: Redundancy

Id	Datalog Rules for Redundancy
	Single actor redundancy: single-red(A_1, A_2, G)
1.	violate-single-red(A_2, A_1, G) :- delegated(A_1, A_2, G), single-red(A_1, A_2, G), $1 \geq$ #count$\{G_i$: or-dec(A_2, G), is-refined$(A_2, G, G_i)\}$.
2.	violate-single-red(A_2, A_1, G) :- delegated(A_1, A_2, G), single-red(A_1, A_2, G), or-dec(A_2, G), is-refined(A_2, G, G_i), delegated$(A_2, _, G_i)$.
3.	has(A, G_i) :- has(A, G), and-dec(A, G), is-refined(A, G, G_i).
4.	has(A, G_i) :- has(A, G), or-dec(A, G), is-refined(A, G, G_i).
5.	delegated(A_1, A_2, G_i) :- has(A_1, G), delegates(A_1, A_2, G_i).
6.	has(A_2, G_i) :- delegated$(_, A_2, G_i)$.
7.	subgoal(G_i, G, A) :- is-refined(A, G, G_i).
8.	subgoal(G_1, G_2, A) :- subgoal(G_1, G_3, A), subgoal(G_3, G_2, A).
	Multi actor redundancy: multi-red(A_1, A_2, G)
1.	violate-multi-red(A_2, A_1, G):-delegated(A_1, A_2, G), multi-red(A_1, A_2, G), $1 \geq$ #count$\{G_i$: or-dec(A_2, G), is-refined$(A_2, G, G_i)\}$.
2.	violate-multi-red(A_2, A_1, G):-delegated(A_1, A_2, G), multi-red(A_1, A_2, G), $0 =$ #count$\{A_3$: delegated(A_2, A_3, G_i), subgoal$(G_i, G, A_2)\}$.
3.	has(A, G_i) :- has(A, G), and-dec(A, G), is-refined(A, G, G_i).
4.	has(A, G_i) :- has(A, G), or-dec(A, G), is-refined(A, G, G_i).
5.	delegated(A_1, A_2, G_i) :- has(A_1, G), delegates(A_1, A_2, G_i).
6.	has(A_2, G_i) :- delegated$(_, A_2, G_i)$.
7.	subgoal(G_i, G, A) :- is-refined(A, G, G_i).
8.	subgoal(G_1, G_2, A) :- subgoal(G_1, G_3, A), subgoal(G_3, G_2, A).

B.3 Verifying Security Requirements over Authorisations

Table B.4 introduces the authorisation rules, which are necessary to capture the transfer of authorisations from actor to actor. Since authorisations in STS-ml capture permissions and prohibitions over the operations that actors may perform over information, we maintain three states over operations: 0 corresponds to prohibition, 1 corresponds to neither prohibition nor permission, while 2 corresponds to permission over the given information. Transferrability is binary, either true or false, which are captured through values 1 and 0 respectively.

Security requirements over authorisations are derived on the basis of prohibitions, therefore it is important to capture anytime a 0 is passed through the authorises relationship. Similarly, we need to know who are the actors authorised explicitly, which calls for keeping track of the permissions (2 through the authorises). As far as implicit prohibitions are concerned, they are derived from summing authorisations. Based on the authorisations an actor receives, we can determine what authority this actor has over the information it wants to manipulate. For this, we use the predicate hasAuthority.

The owner of an information has full authority (all operations are permitted—setting their values to 2, and the authorisation is transferrable—setting it to 1) over its information (rules 1), even when this actor has no goals (rule 2); whenever an actor authorises another to perform operations over information for the scope of some goal, it also authorises the actor to perform operations over information while achieving subgoals of the authorised goals (rule 3), similarly for parts of information (rule 4); whenever a given authorisation is granted the predicate hasAuthority keeps track of an actor's authority to perform operations over a given information, in the scope of some goal, having the authority to transfer authorisations or not (rule 5).

Rules 6 to 17 define when an actor could (because of their intentional or social relationships) read, modify, produce or transmit (disclose) a given information. In particular, rules 8, 11, 14, and 17 capture the fact that the actor has no authority to perform operations read, modify, produce and transmit respectively, as a result of summing authorisations (no incoming authorisation grants the actor permission to perform the operation).

The authorisation scope limiting an authorisation to a goal scope defines for which goals the actor has authority to perform operations on the granted information (rule 18). Rule 19 instead defines the goals that are outside an authorisation's scope. These rules lay the ground for the verification of authorisation requirements.

Table B.5 defines the rules for identifying authorisation conflicts. For all actors, the incoming authorisations are considered and for every pair an authorisation conflict is detected whenever one of the authorisations grants permission on performing an operation (authorise-reading, authorise-modification, authorise-production, and authorise-transmission), or grants the authority to further transfer authorisations (authorise-transferibility), whereas the other authorisation prohibits either performing the operations or transferring the authorisation (rules 11 to 15).

To lay down these rules, however, we first need to capture when permission is granted or prohibition

Table B.4: Authorisation Rules

Id	Datalog Rules for Authorisations Propagation
1.	hasAuthority$(A, 2, 2, 2, 2, I, G, 1)$:- own(A, I), has(A, G).
2.	hasAuthority$(A, 2, 2, 2, 2, I,$ all_goals, $1)$:-own(A, I),$0 =$#count$\{G$:has$(A, G)\}$.
3.	authorise$(A_1, A_2, I, G_1,$ R, M, P, T, TrAuth$)$:- authorises$(A_1, A_2, I, G,$ R, M, P, T, TrAuth$)$, subgoal(G_1, G, A_2).
4.	authorise$(A_1, A_2, I_1, G,$ R, M, P, T, TrAuth$)$:- authorises$(A_1, A_2, I, G,$ R, M, P, T, TrAuth$)$, partOfI(I_1, I).
5.	hasAuthority$(A_2,$ R, M, P, T, $I, G,$ TrAuth$)$:- authorises$(A_1, A_2, I, G,$ R, M, P, T, TrAuth$)$.
6.	can-read(A, I, D, G) :- has(A, G), reads(A, D, G), madeTangibleBy(I, D).
7.	$-$has-authority-to-read(A, I) :- role(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, 2, _, _, _, _)\}$.
7.	$-$has-authority-to-read(A, I) :- agent(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, 2, _, _, _, _)\}$.
9.	can-modify(A, I, D, G) :- has(A, G), modifies(A, D, G), madeTangibleBy(I, D).
10.	$-$has-authority-to-modify(A, I) :- role(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, _, 2, _, _, _)\}$.
11.	$-$has-authority-to-modify(A, I) :- agent(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, _, 2, _, _, _)\}$.
12.	can-produce(A, I, D, G):-has(A, G), produces(A, D, G), madeTangibleBy(I, D).
13.	$-$has-authority-to-produce(A, I) :- role(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, _, _, 2, _, _)\}$.
14.	$-$has-authority-to-produce(A, I) :- agent(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, _, _, 2, _, _)\}$.
15.	can-transmit(A, I, D) :- transmits$(A, _, D)$, madeTangibleBy(I, D).
16.	$-$has-authority-to-transmit(A, I) :- role(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, _, _, _, 2, _)\}$.
17.	$-$has-authority-to-transmit(A, I) :- agent(A), information(I), not own(A, I), $0 =$ #count$\{A_2$: authorises$(A_2, A, I, G, _, _, _, 2, _)\}$.
18.	scope-g(A, I, G) :- hasAuthority$(A, _, _, _, _, I, G, _)$.
19.	$-$scope-g(A, I, G) :- hasAuthority$(A, _, _, _, _, I, G_1, _)$, has$(A, G)$, has$(A, G_1)$, $G \neq G_1, 0 =$ #count$\{G_2$: hasAuthority$(A, _, _, _, _, I, G_2, _), G_2 = G\}$.
20.	$-$has-authority-to-authorise(A, I) :- hasAuthority$(A, _, _, _, _, I, _, 0)$.

is specified; we do this for all operations, as well as for the transferability dimention (rules 1 to 10).

Table B.5: Authorisation Conflicts Verification

Id	Datalog Rules for Authorisations Conflicts
1.	authorise-reading(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, 2, _, _, _, _)$.
2.	$-$authorise-reading(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, 0, _, _, _, _)$.
3.	authorise-modification(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, 2, _, _, _)$.
4.	$-$authorise-modification(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, 0, _, _, _)$.
5.	authorise-production(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, _, 2, _, _)$.
6.	$-$authorise-production(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, _, 0, _, _)$.
7.	authorise-trasnsmission(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, _, _, 2, _)$.
8.	$-$authorise-transmission(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, _, _, 0, _)$.
9.	authorise-transferability(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, _, _, _, 2)$.
10.	$-$authorise-transferability(A_1, A_2, I) :- authorise$(A_1, A_2, I, _, _, _, _, _, 0)$.
11.	authorisation-conflict(A_2, I):-authorise-reading(A_1, A_2, I), $-$authorise-reading(A_3, A_2, I).
12.	authorisation-conflict(A_2, I):-authorise-modification(A_1, A_2, I), $-$authorise-modification(A_3, A_2, I).
13.	authorisation-conflict(A_2, I):-authorise-production(A_1, A_2, I), $-$authorise-production(A_3, A_2, I).
14.	authorisation-conflict(A_2, I):-authorise-transmission(A_1, A_2, I), $-$authorise-transmission(A_3, A_2, I).
15.	authorisation-conflict(A_2, I):-authorise-transferibility(A_1, A_2, I), $-$authorise-transferibility(A_3, A_2, I).

After detecting authorisation conflicts, the analysis verifies if there are any conflicts among business policies and security requirements derived from authorisation. Tables B.6 to B.11 present the rules for identifying these conflicts, per each security requirement derived from authorisations. All the violations are propagated through the information structure (following the part of relationships).

Thus, Table B.6 identifies violations of need-to-know. Note that this requirement is verified only when permissions on operations are granted; we cannot talk about need-to-know in case of prohibitions. Moreover, this requirement is related to goal-document operations, therefore transmission is not one of

Table B.6: Authorisation Requirements Verification: Need to Know

Id	**Datalog Rules for**: need-to-know(A_1, A_2, I, G)
1.	violate-ntk(A_2, I, G) :- $-$scope-g(A_2, I, G), read(A_2, I, G), not violate-non-reading(A_2, I, G).
2.	violate-ntk(A_2, I, G) :- $-$scope-g(A_2, I, G), modified(A_2, I, G), not violate-non-modification(A_2, I, G).
3.	violate-ntk(A_2, I, G) :- $-$scope-g(A_2, I, G), produced(A_2, I, G), not violate-non-production(A_2, I, G).
4.	violate-ntk(A_2, I_1, G) :- violate-ntk(A_2, I, G), partOfl(I_1, I).
5.	violate-ntk(A_2, I, G) :- violate-ntk(A_2, I_1, G), partOfl(I_1, I).

the operations that affects need-to-know.

Table B.7: Authorisation Requirements Verification: Non reading

Id	**Datalog Rules for**: non-reading(A_1, A_2, I)
1.	violate-non-reading(A_2, I, G) :- $-$has-authority-to-read(A_2, I),read(A_2, I, G).
2.	read(A_2, I, G) :- possesses(A_2, D), can-read(A_2, I, D, G).
3.	violate-non-reading(A_2, I_1, G) :- violate-non-reading(A_2, I, G), partOfl(I_1, I).
4.	violate-non-reading(A_2, I, G) :- violate-non-reading(A_2, I_1, G), partOfl(I_1, I).

Table B.7 identifies violations of non-reading.

Table B.8 identifies violations of non-modification.

Table B.9 identifies violations of non-production.

Table B.10 identifies violations of non-disclosure.

Finally, Table B.11 enumerates the rules for identifying all actors which violate their authorities, while reauthorising other actors: (i) without having the right to tranfer authorisations (`TrAuth` $= 0$); (ii) authorising others on operations they do not have permission (or are prohibited) to perform themselves.

Table B.8: Authorisation Requirements Verification: Non modification

Id	**Datalog Rules for**: non-modification(A_1, A_2, I)
1.	violate-non-modification(A_2, I, G) :- $-$has-authority-to-modify(A_2, I), modified(A_2, I, G).
2.	modified(A_2, I, G) :- possesses(A_2, D), can-modify(A_2, I, D, G).
3.	violate-non-modification(A_2, I_1, G) :- violate-non-modification(A_2, I, G), partOfI(I_1, I).
4.	violate-non-modification(A_2, I, G) :- violate-non-modification(A_2, I_1, G), partOfI(I_1, I).

Table B.9: Authorisation Requirements Verification: Non production

Id	**Datalog Rules for**: non-production(A_1, A_2, I)
1.	violate-non-production(A_2, I, G) :- $-$has-authority-to-produce(A_2, I), produced(A_2, I, G).
2.	produced(A_2, I, G) :- can-produce(A_2, I, D, G).
3.	violate-non-production(A_2, I_1, G) :- violate-non-production(A_2, I, G), partOfI(I_1, I).
4.	violate-non-production(A_2, I, G) :- violate-non-production(A_2, I_1, G), partOfI(I_1, I).

Table B.10: Authorisation Requirements Verification: Non disclosure

Id	**Datalog Rules for**: non-disclosure(A_1, A_2, I)
1.	violate-non-disclosure(A_2, I, D) :- $-$has-authority-to-transmit(A_2, I), transmitted(A_2, I, D).
2.	transmitted(A_2, I, D) :- possesses(A_2, D), can-transmit(A_2, I, D).
3.	violate-non-disclosure(A_2, I_1, D) :- violate-non-disclosure(A_2, I, D), partOfI(I_1, I).
4.	violate-non-disclosure(A_2, I, D) :- violate-non-disclosure(A_2, I_1, D), partOfI(I_1, I).

Table B.11: Identifying unauthorised transfer of authorisations

Id	**Datalog Rules for** non-reauthorisation(A_1, A_2, I, G, Op)
1.	violate-non-reauthorisation(A_1, A_2, I) :- $-$has-authority-to-authorise(A_1, I), authorise-reading(A_1, A_2, I).
2.	violate-non-reauthorisation(A_1, A_2, I) :- $-$has-authority-to-authorise(A_1, I), authorise-modification(A_1, A_2, I).
3.	violate-non-reauthorisation(A_1, A_2, I) :- $-$has-authority-to-authorise(A_1, I), authorise-production(A_1, A_2, I).
4.	violate-non-reauthorisation(A_1, A_2, I) :- $-$has-authority-to-authorise(A_1, I), authorise-transmission(A_1, A_2, I).
5.	non-reauthorisation-reading(A_1, A_2, I) :- not has-authority-to-read(A_1, I), authorise-reading(A_1, A_2, I), not violate-non-reauthorisation(A_1, A_2, I).
6.	non-reauthorisation-modification(A_1, A_2, I) :- not has-authority-to-modify(A_1, I), authorise-modification(A_1, A_2, I), not violate-non-reauthorisation(A_1, A_2, I).
7.	non-reauthorisation-production(A_1, A_2, I) :- not has-authority-to-produce(A_1, I), authorise-production(A_1, A_2, I), not violate-non-reauthorisation(A_1, A_2, I).
8.	non-reauthorisation-transmission(A_1, A_2, I) :- not has-authority-to-transmit(A_1, I), authorise-transmission(A_1, A_2, I), not violate-non-reauthorisation(A_1, A_2, I).

B.4 Verifying Security Requirements over Responsibility Uptake

As far as security requirements over responsibility uptake (imposed as organisational constraints) are concerned, security analysis verifies whether the specification of role-sod, role-cod, goal-sod, and goal-cod brings up conflicts with actors business policies. The analysis defines a final performer actor, and propagates the normative requirements over an actor's model and over social relationships it has with others, to identity conflicts (see Tables B.12 and B.13).

Table B.12: Security Requirements over Responsibility Uptake: Goal Rules

Id	Goal Rules for SoD and CoD
1.	$\mathsf{has}(A, G_i)$:- $\mathsf{has}(A, G)$, $\mathsf{and\text{-}dec}(A, G)$, $\mathsf{is\text{-}refined}(A, G, G_i)$.
2.	$\mathsf{has}(A, G_i)$:- $\mathsf{has}(A, G)$, $\mathsf{or\text{-}dec}(A, G)$, $\mathsf{is\text{-}refined}(A, G, G_i)$.
3.	$\mathsf{delegated}(A_1, A_2, G_i)$:- $\mathsf{has}(A_1, G)$, $\mathsf{delegates}(A_1, A_2, G_i)$.
4.	$\mathsf{has}(A_2, G_i)$:- $\mathsf{delegated}(\_, A_2, G_i)$.
5.	$\mathsf{subgoal}(G_i, G, A)$:- $\mathsf{is\text{-}refined}(A, G, G_i)$.
6.	$\mathsf{subgoal}(G_1, G_2, A)$:- $\mathsf{subgoal}(G_1, G_3, A)$, $\mathsf{subgoal}(G_3, G_2, A)$.
7.	$\mathsf{final\text{-}performer}(R, G)$:- $\mathsf{has}(R, G)$, $0 = \#\mathsf{count}\{R_1{:}\mathsf{delegates}(R, R_1, G)\}$.

Table B.13: Verification of Security Requirements over Responsibility Uptake

Id	Datalog Rules for Separation and Combination of Duty
	Separation of Duty: role-sod(STS, Ag, R_1, R_2), goal-sod(STS, A, R_1, R_2)
1.	violate-sod-role(A, R_1, R_2) :- sod-role(R_1, R_2), plays(A, R_1), plays(A, R_2).
2.	violate-sod-goal(A, R_1, G_1, R_2, G_2) :- sod-goal(G_1, G_2), final-performer(R_1, G_1), final-performer(R_2, G_2), play(A, R_1), plays(A, R_2).
3.	violate-sod-goal(R, R, G_1, R, G_2) :- sod-goal(G_1, G_2), final-performer(R_1, G_1), final-performer(R_2, G_2), $0 = \#\text{count}\{A : \text{plays}(A, R)\}$.
4.	violate-sod-goal(A, A, G_1, R, G_2) :- sod-goal(G_1, G_2), final-performer(A, G_1), final-performer(R, G_2), agent(A), role(R), plays(A, R).
5.	sod-goal(G_a, G_2) :- sod-goal(G_1, G_2), or-dec(R, G_1), is-refined(R, G_1, G_a), final-performer(R, G_a).
6.	sod-goal(G_1, G_a) :- sod-goal(G_1, G_2), or-dec(R, G_2), is-refined(R, G_2, G_a), final-performer(R, G_a).
	Combination of Duty: role-cod(STS, Ag, R_1, R_2), goal-cod(STS, A, R_1, R_2)
1.	violate-cod-role(A, R_1, R_2) :- cod-role(R_1, R_2), not plays(A, R_1), plays(A, R_2).
2.	violate-cod-role(A, R_1, R_2) :- cod-role(R_1, R_2), plays(A, R_1), not plays(A, R_2).
3.	violate-cod-goal(A, R_1, G_1, R_2, G_2) :- cod-goal(G_1, G_2), final-performer(R_1, G_1), final-performer(R_2, G_2), agent(A), role(R_1), role(R_2), plays(A, R_2), not plays(A, R_1).
4.	violate-cod-goal(A, R_1, G_1, R_2, G_2) :- cod-goal(G_1, G_2), final-performer(R_1, G_1), final-performer(R_2, G_2), agent(A), role(R_1), role(R_2), plays(A, R_1), not plays(A, R_2).
5.	violate-cod-goal(R_1, R_1, G_1, R_2, G_2) :- cod-goal(G_1, G_2), final-performer(R_1, G_1), final-performer(R_2, G_2), $0 = \#\text{count}\{A : \text{agent}(A)\}$.
6.	violate-cod-goal(R_1, R_1, G_1, R_2, G_2) :- cod-goal(G_1, G_2), final-performer(R_1, G_1), final-performer(R_2, G_2), agent(A), not plays(A, R_1), not plays(A, R_2).
7.	violate-cod-goal(A, A, G_1, R, G_2) :- cod-goal(G_1, G_2), final-performer(A, G_1), final-performer(R, G_2), agent(A), role(R), not plays(A, R).
8.	cod-goal(G_a, G_2) :- cod-goal(G_1, G_2), or-dec(R, G_1), is-refined(R, G_1, G_a), final-performer(R, G_a).
9.	cod-goal(G_1, G_a) :- cod-goal(G_1, G_2), or-dec(R, G_2), is-refined(R, G_2, G_a), final-performer(R, G_a).