Security Requirements Modeling Tool

Socio-Technical Security Modeling Language

(rev 1.0)

For STS-Tool Version 2.0 & 2.1

Contact: ststool@disi.unitn.it

Table of contents

1 Introduction

Socio-technical systems are an interplay of social (human and organizations) and technical subsystems, which interact with one another to reach their objectives, making a system a network of social relationships. Today's software systems are a complex interplay of different subsystems, which are not only technical, but also social. In principle, they are socio-technical systems. Each subsystem is a participant of the socio-technical system. Participants in a socio-technical system are autonomous, and the system is defined in terms of the interactions among participants, which may be: *social reliance*, i.e., participants rely on others to achieve their goals, and *information exchange*, i.e., participants exchange relevant information. In such systems, many security issues arise from the interaction among participants, and on how the exchanged information is manipulated. Social aspects are thus a main concern when analyzing the security of socio-technical systems.

Goal-oriented approaches to security requirements engineering [3][4][12][13] offer a suitable abstraction level for the design of secure socio-technical systems. They model socio-technical systems as a set of actors that are intentional – they have objectives, and social – they interact with others to achieve their objectives. Unfortunately, their underlying ontology is too abstract to effectively represent real-world information security requirements, which include fine-grained and contradictory authorizations over information entities [13][14].

To overcome this limitation, we have proposed the Socio-Technical Security modeling language (STS-ml) for socio-technical systems 6[2]. STS-ml relies on a more expressive ontology and expresses security requirements as relationships between couples of socio-technical systems participants (also referred to as *actors*), where a requester actor requires a requestee actor to comply with a *security need*. STS-ml enables each participant to express its own requirements.

The Socio-Technical Security modeling language (STS-ml) is an actor– and goal–oriented security requirements modeling language. It captures system security needs and requirements at the organizational (business/operational) level. The language enables to represent and reason about organizational *assets*, *social dependencies*, and *trust* properties that are essential to capture and understand stakeholders' security needs.

To support the elicitation process, the language captures the operational relationships between actors. It allows for expressing the system security needs of the stakeholders. These needs are then transformed into the security requirements (with the aid of security risk assessment techniques) that will in turn drive the design and assembly of service compositions to support the goals of the involved actors.

A basic component of the language consists of the constructs for expressing interactions between actors. To successfully cope with the security threats and changes that unexpectedly may arise at runtime, adaptation may take place taking advantage of the alternatives captured by STS-ml.

STS-ml consists of a set of concepts that can be employed to conduct security requirements analysis for a wide range of applications. The language supports the representation of security needs that are later transformed into security requirements for the system to-be. We provide the intuition behind these concepts, their semantics, and their graphical representation. In the following subsections we present such concepts and the models that can be created by using the language. We present a method that facilitates modeling systems and capturing stakeholders security needs. We describe the analysis process, whose aim is to improve the overall model of the organization with respect to security concerns.

1.1 Overview

STS-ml offers the possibility to represent the interaction between different actors that are interacting to achieve their own objectives. We use the concept of *social commitment* (briefly, *commitment*) to model the interactions between participants in a socio-technical system. A commitment [6][7] is a quaternary relation *C(debtor, creditor, antecedent, consequent)* that stands for a promise made by the debtor to the creditor that, if the antecedent is brought about, the consequent will be brought about. For instance, the commitment *C (hotel, customer, prepayment done, room booked)* represents a commitment from *hotel* to the prospective *customer* that if a *prepayment is done* to book the room, then the reservation will be made (*room* will be *booked*). Some commitments are unconditional (i.e., their antecedent is true). Commitments have contractual validity within a socio-legal context: the debtor might incur in penalties if he does not keep his promises.

We use commitments as a mechanism of control, to check consistency between the promises participants (represented via actors) make and what they actually bring about. This mechanism helps to obtain a more robust system, by ensuring things work in compliance with organizational rules and regulations, and software restrictions (access control is usually specified in the software). Specifically, we use commitments to verify whether security requirements are met.

The STS method, guiding the modeling and analysis of STS-ml models (models constructed with STS-ml concepts and relationships), offers security requirements engineers the opportunity to model different perspectives of a considered setting separately. This solution has two advantages: first, it facilitates the work of engineers when dealing with non-trivial settings; second, it presents orthogonal perspectives separately, applying the separation-of-concerns principle. These different perspectives are *views* over the same model, and together form the complete model that one or more engineers construct. In order to guarantee consistency among the views, the supporting toolset is provided with automated reasoning mechanisms that detect inconsistencies and provides the requirements engineer with enough detail to fixing the identified problems. The STS method supports the specification of secure socio-technical systems by modeling the following views:

1. Social view: represents actor intentionality and sociality. Actors are intentional entities that aim to attain some state of affair, by manipulating the information to their disposal. Actors can be either social or technical in nature; for example, a web service that acts on behalf of a travel agency is technical, whereas a customer interacting with such web service is social. An important characteristic of actors is, however, their sociality, i.e., their need to interact with others to achieve their desired state of affairs.
2. Information view: represents the information in the considered organization/setting together with the documents that represent such information, as well as the relationships among these informational entities or documents respectively.
3. Authorization view: represents the authorizations granted by some actors to other actors concerning the exchange and manipulation of information for particular purposes.
4. *Security Requirements*: represents the list of security requirements expressed in terms of commitments that hold/should hold between actors to address the security needs expressed through the above three views. Figure 1 outlines STS-ml and its modus operandi. A central role in the schema is played by *security needs*, which are communicated by the stakeholders in the organization and modeled by the security requirements engineer of the said socio-technical system. Such security needs are *expressed in* the operational view that describes how the organization (system) operates. The operational view consists of the three views that we listed above: *social*, *information*, and *authorization*. Together, these views provide a comprehensive picture of the organization's key business/operational concerns as well as security aspects, eliciting security needs and discovering potential threats to security.

The socio-technical security modeling language goes beyond elicitation: it automatically derives security requirements as a set of commitments between actors. Such commitments shall be established—via security mechanisms—and continuously monitored for compliance. The derived security requirements, if satisfied, ensure security in the organization.

The modeling and analysis of STS-ml models is guided step by step by the *STS method*. *STS-Tool* [8], on the other hand, is the modeling and analysis support tool for STS-ml. As such it supports the modeling activities and the derivation of security requirements as proposed in STS-ml. At the end of the modeling process, the STS-Tool allows security requirements engineers to export models into various formats such as jpeg, gif, svg, and png to name a few, and to automatically generate a security requirements document (see Figure 1). This document provides a description of each STS-ml view and the corresponding elements, which is helpful while communicating with stakeholders. However, the document is customizable: designers can choose among a number of model features to include in each section. More details on how to use the modeling and reasoning capabilities of STS-Tool are provided in [10].

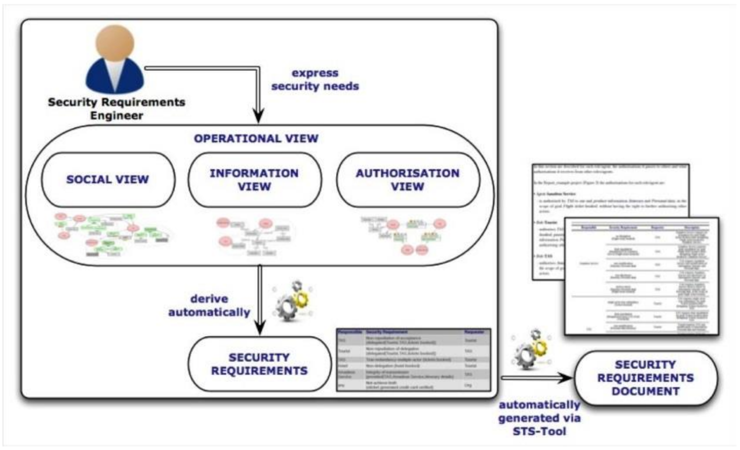


Figure 1: Multi-view modelling

In Section 2 we introduce and explain one by one the STS-ml concepts and relationships. We do so by going through the different views supported by the STS method, describing their purpose along with the introduction of the concepts that are used in each view. In Section 3 we describe the derivation of security requirements in STS-ml, enumerating the comprehensive list of security requirements supported by the language. Section 4 introduces and explains the STS method for security requirements engineering that supports the modeling and analysis of socio-technical systems using STS-ml. The method guides these activities step by step, and is illustrated with the help of the running example introduced in Section 1.2. Finally, Section 5 summarizes STS-ml constructs in the form of a cheat-sheet to support any potential user of STS-Tool.

1.2 Running example

We illustrate STS-ml using a running example concerning compliance with the data protection legislation [3]. The case study concentrates on the compliance of Italian public administration, such as universities, to Italian security and privacy legislation [4]. This law/act specifies requirements over the public administrations to devise internal regulations and policies based on the ISO-17799 standard. The focus is on personal data and data processing such as data usage, update, modification and production. The University of Trento (UniTN) has enforced the Data Protection Act since January 14th, 2002. UniTN offers several international programs that attract a large number of international students. We consider a scenario in which an international *student* needs a document from the *program coordinator*; such document has to be presented to the local immigration office to get his stay permit extended. The following roles are involved:

- *Student*: needs an official document to prove he is enrolled in the study program and his incomes are enough to afford the stay. He asks the program coordinator to issue the document. For this reason, he has to transmit his personal data, as well as financial information. His personal data is stored in the UniTN information system.
- *Program Coordinator*: issues the official document for the student. He might transfer responsibility for parts of this activity to his secretary.
- *Secretary*: retrieves student information (personal data and financial data) from the information system and drafts the document.
- *IS Manager*: manages the information about students stored in the UniTN information system in accordance with confidentiality restrictions.

2 Modeling security needs with STS-ml: the operational view

We detail the three sub-views that constitute the operational view of STS-ml. Together, these views enable modeling the security needs expressed by stakeholders along with their business policies – how they intend to achieve their objectives.

2.1 Social view

The social view—whose meta-model is shown in Figure 2—represents the perspectives of different intentional actors in the considered setting together with the interactions that take place among them.

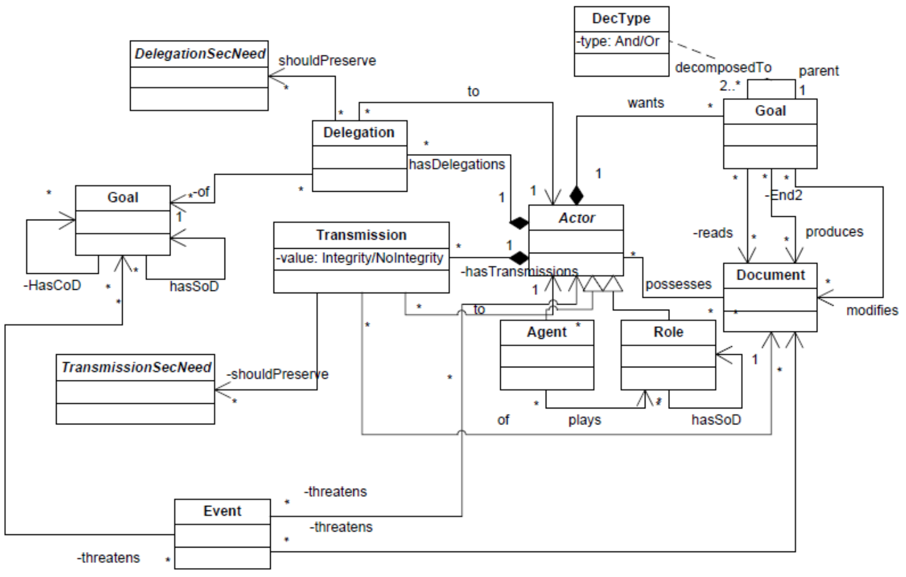


Figure 2: Multi-view modelling

We consider an abstract concept of *actor*[1], and refine it into two distinct concrete concepts: *role* and *agent*, used in order to represent socio-technical system participants. Agents refer to concrete known participants of the system, while roles are an abstract characterization of the behavior of an active entity in the system. Roles are an important concept in the modeling of a socio-technical system, because designers are often unaware of the actual participants at runtime. Specifying systems at the role level is a flexible approach that defines the requirements for an agent to play a role, as opposed to mandating the existence of a specific agent. At runtime the actual participants will adopt those roles. We capture this through the concept of *agent*, which is said to *play* a *role*, as shown by the meta-model in Figure 2. To exemplify the usage of roles and agents we represent a situation in which Anna is a secretary, by modeling *Secretary* as a role, and *Anna* as an agent playing that role (Figure 3). The reason behind such representation is that Anna is a known participant, while secretary could be played by different agents at runtime; therefore it is represented as a role.

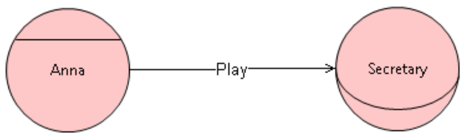


Figure 3: An agent playing a role

1 Note that actor is not a concept in STS-ml; rather it is used whenever we do not need to distinguish between the concepts of role and agent on properties and/or relationships that apply to both these concepts.

Some agents might be already known at requirements-time. For instance, the *Prefecture of Trento* is modeled as an agent, because there is only one prefecture and it is known that students should invariably interact with it to renew their stay permit.

Actors are *intentional* entities: they have strategic interests/objectives they want to pursue. These objectives are captured in STS-ml through the concept of *goal*. The goals actors want to achieve can be elicited by conducting organizational analysis and performing interviews with stakeholders. The goals that are assigned to roles represent expectations on the behavior of agents playing that role. However, due to their autonomy, some agents might behave differently at runtime (however, some others might be expressly designed to be compliant with the expectations expressed by the stakeholders).

The goals actors want to achieve are represented within the *scope* of the actor (graphically represented by a circle attached to an actor, see Figure 4). An actor may have one or more *goals* that it *wants* to *achieve*.

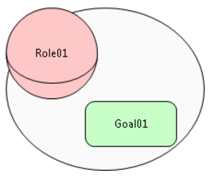


Figure 4: Actor scope

STS-ml offers the possibility to build models of the system that can be refined and extended incrementally. For this purpose, we analyze goals from the actor perspective and refine them, creating a *goal model*. Goals are analyzed and refined in STS-ml via AND/OR decompositions. The *AND/OR decomposition* relationship is used to refine goals into sub-goals, and combine them depending on the way the initial (top-level goal) can be achieved/fulfilled by answering *how* questions. As a result we obtain a goal model, which is an AND/OR tree whose nodes are goals (Figure 9). More specifically, AND-decomposition represents the process of achieving a goal (all sub-goals in the AND-decomposition need to be satisfied for the goal to be satisfied). OR-decomposition, on the other hand, represents alternative ways for achieving a goal (at least one of the sub-goals in the OR-decomposition needs to be satisfied for the goal to be satisfied). In Figure 9 the secretary wants to achieve goals *Write new doc.*, *Get Student Record.*, and so on. The goal *Get Student Record* is AND-decomposed to two sub-goals, namely *Get Student Pers. Data* and *Derive Financial Stat*.

The goal model ties together goals and documents. An actor possesses a set of documents. Possession is different from ownership – that will be introduced in Section 2– for it refers only to the disposal of a document.

- We tie together goals and documents, within an actor's scope, in various ways:
- an actor possesses (has) a set of documents;
- an actor reads one or more documents to fulfill a goal;
- an actor produces documents while fulfilling a goal;
- an actor modifies one or more documents while fulfilling a goal. A document is modified if, despite of the change or update, the document identity is unvaried. For example, the personal data file of a student can be modified if the student's address changed.

Thus, we distinguish between documents that are read to fulfill a goal from documents that are modified or produced while fulfilling a goal. This is modeled through relationships between goals and documents they respectively read, modify and produce. Figure 5 depicts the graphical representation of goals and documents, while Figure 6 and Figure 7 shows the read, modify and produce relations as arrows from the goal to the document. The arrows are labeled with read, modify and produce respectively and their direction reflects the actual direction of the actions.

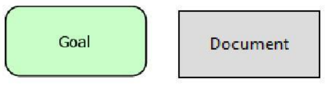


Figure 5: Graphical representations of goal and document

In Figure 9, the secretary's goal *Write new doc.* reads the document *Document Template* and modifies that document once the actual letter is written by filling the template. On the other hand, the program coordinator's goal *Write doc for I.O.* produces the document *Signed Official Doc*.

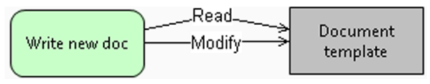


Figure 6: Read and modify relationships

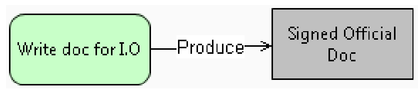


Figure 7: Produces relationship

The relation *possess* indicates that a role has a specific document, i.e. possesses the document and can furnish or transmit it to other roles. For instance, the secretary is in possession of a *Document template* (Figure 8) that is read to write a new document when requested by some student.

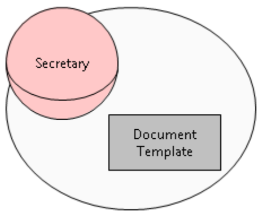


Figure 8: Example of Possession of documents from the running example

We consider social actors that collaborate to fulfill their own objectives. This reflects the way business is conducted in reality, where *social interaction* is crucial to enable actors to achieve goals that they are not capable to achieve themselves or that others can perform with higher quality or lower costs.

The purpose of the social view is to represent the social relationships between actors in the considered system. For this, the social view supports two types of social relationship, namely *goal delegation* and *document transmission*. Whereas the former captures the expectations that one actor has from others in terms of the goals that he can delegate, the latter enables to represent the information flow – how documents are transferred from one actor to another. The social view for the running example is depicted in Figure 9.

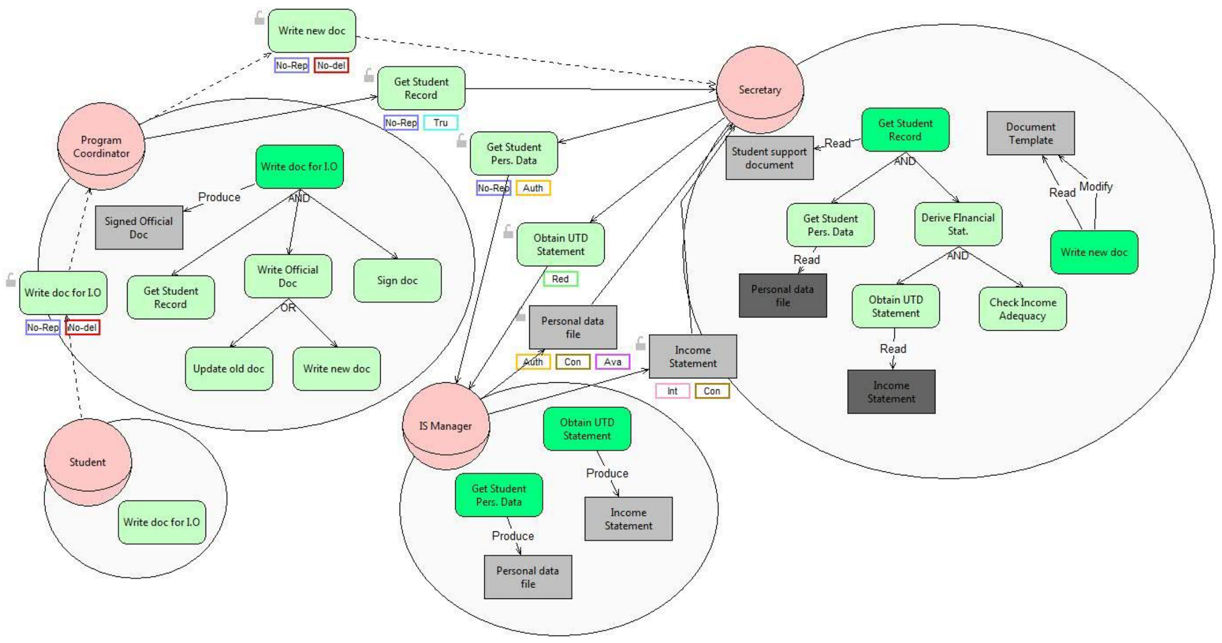


Figure 9: Social View

A key concept in the social view is that of *security need*. Given that in STS-ml security is related to actors' interactions, the term security need refers to the expectation concerning security that actors impose on the interactions (represented via social relationships) they participate in. Thus, we will analyze the two social relationships supported by the social view and detail the security needs that are can be expressed over these social relationships.

The first social relationship, *goal delegation*, models the transfer of responsibilities from one actor to another. It links two actors and a goal: a delegator actor delegates the fulfillment of a goal (delegatum) to a delegatee actor. A high-level classification of security needs related to goal delegation is shown in Figure 10, see DelegationSecNeed.

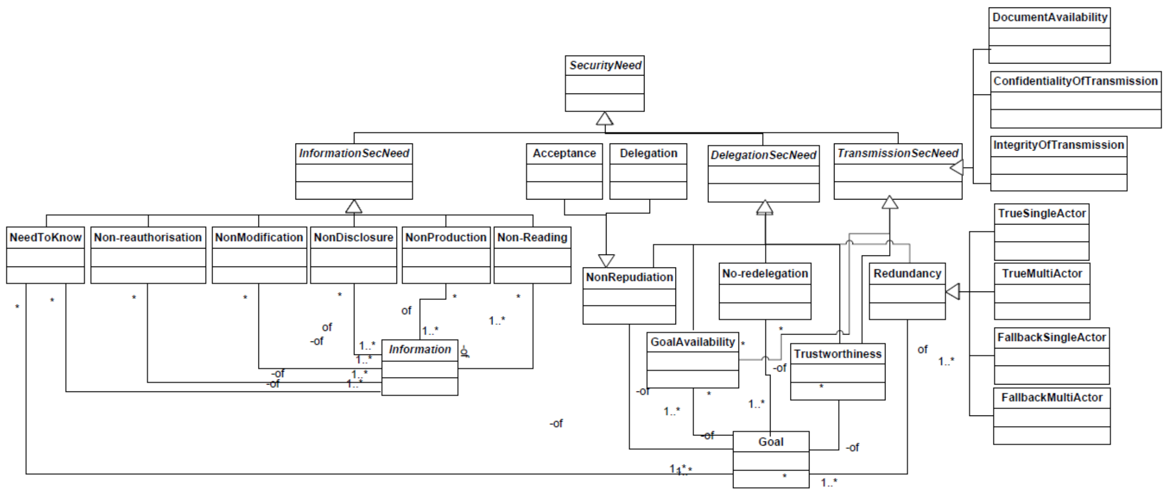


Figure 10: Security needs supported in STS-ml

In Figure 11, the student delegates the fulfillment of goal *Write doc for I.O.*, namely write document for immigration office, to the program coordinator.

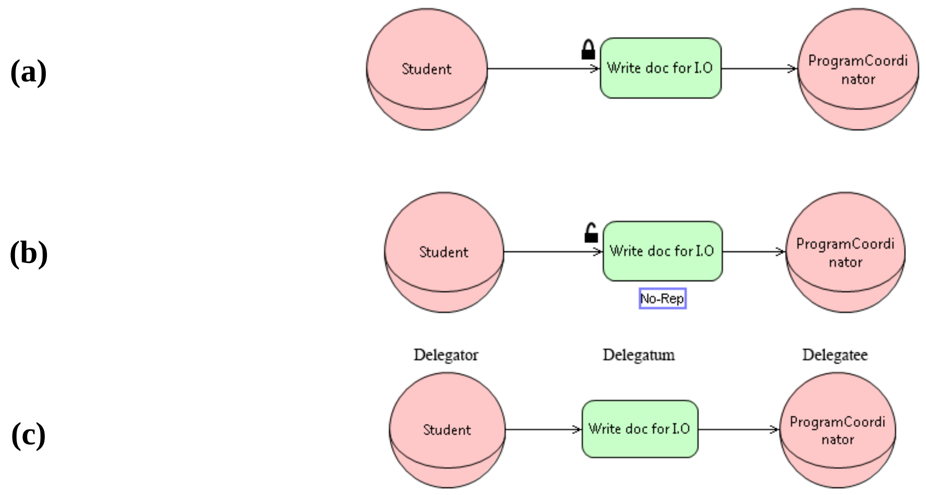


Figure 11: Goal delegation example

In order to specify security needs over goal delegations, delegations are annotated via security needs the interacting parties (being those agents or roles) want each other to comply with. Graphically security needs can be specified over goal delegations by right-clicking on the goal and selecting the desired security needs from a given list. Certain security needs cannot coexist, and as such cannot be selected together simultaneously over a goal delegation. The list allows for the selection of one security need per type. More details will be provided in the following.

The selection of at least one security need shows a black padlock on the goal (see Figure 11a). The selection of security needs is shown explicitly by clicking on the padlock, which shows small boxes below the delegated goal; each box has a distinguishing color and label to reflect the selected security needs; no-redelegation makes an exception, for apart from the representation of the colored and labeled box below the goal delegation, this security need is rendered differently through a dashed arrow line—see Figure 11a—the delegation arrow line from the delegator to the delegatee (solid—see Figure 11b—if such security need is not expressed). Figure 11c, instead, shows the case in which no security needs are specified over a goal delegation. Below we provide the list of security needs the STS-ml modeling language supports:

Non-repudiation (NonRep): the delegator actor wants the delegatee actor not to be able to challenge the validity of the goal delegation. A non-repudiation security need requires the adoption of security mechanisms that guarantee that the interacting parties cannot repudiate that the delegation has taken place. Graphically, the requirement is represented as an annotation for delegations with the label "No-Rep". We distinguish two types of non-repudiation:

Non-repudiation of Acceptance: required by the delegator, so that the delegatee cannot repudiate the delegation. As we will detail in Section 3, such a security solution consists of the establishment of a commitment — for the non-repudiation of that goal delegation — from the delegatee to the delegator. For instance, the program coordinator wants non-repudiation for the delegation of the goal *Write new doc* delegated to the secretary (Figure 9). A possible way to satisfy this security need is for the program coordinator to accept the delegation and provide proof of fulfillment to the student;

Non-repudiation of Delegation: required by the delegatee, so that the delegator cannot repudiate having delegated to the delegatee. This solution consists in the generation of a commitment — for the non-repudiation of the goal delegation — from the delegator to the delegatee. For instance, in Figure 9, the secretary might want the program coordinator not to repudiate having delegated to the former the goal *Write new doc*. A way to satisfy this security need is for the program coordinator to digitally signing a delegation statement.

Redundancy: the delegatee has to adopt redundant strategies for the achievement of the delegated goal. He can either use different internal capabilities, or can rely on multiple actors. Graphically, the requirement is represented as an annotation for delegations with the label "Red". We consider two types of redundancy:

Fallback redundancy: a primary strategy is selected to fulfill the goal, and at the same time a number of other strategies are considered and maintained as backup to fulfill the goal. None of the backup strategies is used as long as the first strategy successfully fulfils the goal.

True redundancy: at least two or more different strategies are considered to fulfill the goal, and they are executed simultaneously to ensure goal fulfillment.

Within these two categories of redundancy, two sub-cases exist: (i) only one actor employs different strategies to ensure redundancy: *single actor* redundancy; and (ii) multiple actors employ different strategies to ensure redundancy: *multi actor* redundancy.

In total, we can distinguish four types of redundancy, which are all mutually exclusive, so we can consider them as four different security needs, namely (i) *fallback redundancy single* (fback_rs), (ii) *fallback redundancy multi* (fback_rm), (iii) *true redundancy single* (true_rs), and (iv) *true redundancy multi* (true_rm). To ensure redundancy the delegator actor should make a commitment to the delegatee actor that it will achieve the goal by adopting redundant strategies in compliance with the type of redundancy required by the corresponding security need. An example of redundancy is that of secretary requiring the IS manager *true redundancy single* when delegating the goal *Obtain UTD Statement*, see Figure 9.

No-redelegation: this requirement is expressed over goal delegations, and it is the delegator's request for the delegatee to take full responsibility for 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 the label "No-del", which stands for "no-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. An example of such security need is that of the program coordinator that wants the secretary not to redelegate goal *Write new doc*, see Figure 9.

Trustworthiness: this security need specifies a requirement to potential actors playing the delegatee role. Only delegatee actors that are trustworthy can play that role for the delegation to take place. In Figure 9, the delegation between the program coordinator and the secretary will take place only if the secretary is trustworthy.

Availability: the delegator wants the delegatee to guarantee a minimum availability level concerning the provision of the delegated goal. If one conceives the delegatee as a service provider (where the service is to satisfy the goal), availability corresponds to an uptime guarantee on service provision. For instance, the secretary requires the IS manager to ensure an availability level of 90% over the goal *Obtain UTD Statement*, see Figure 9.

Authentication: For the delegator, this requirement about authenticity indicates the delegatee's request that the delegator 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. For the delegatee, this type of the authenticity requirement expresses the delegator's need that the delegatee should authenticate. We encounter this kind of authentication every day when we browse the web and use our credentials (username/password) to access web information such as our email. In Figure 9 the delegation of the goal *Obtain UTD Statement* from secretary to the IS manager requires the authentication of both the delegator and the delegatee.

Actors want to achieve goals, which might imply the use of some information (document). We distinguish actors that have the document containing the needed information and can provide it to others, from actors that need the said information (documents). At a given moment of time, an actor can be in possession of the document and may transmit it to other actors. In STS-ml the *document transmission* relationship is used to capture the exchange of information between actors (Figure 12), where a sender actor transmits a document to a receiver actor. A document can be transmitted only by an actor that possesses it. Transmission refers strictly to the actual supply/delivery of the document.

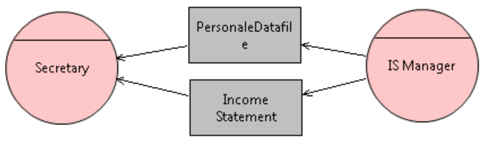


Figure 12: Document transmission

Information as is (e.g. ideas) cannot be transferred if not explicitly made concrete by a document (e.g. a paper, an e-mail). We will further elaborate on this distinction in Section 2.2.

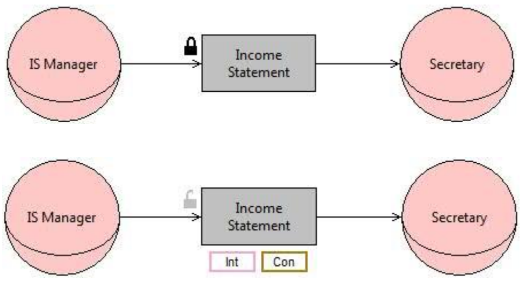


Figure 13: Expressing security needs over document transmissions

The transmission of documents can be subject to security needs as well (see Figure 13). Graphically the expression of security needs and their selection is similar to that of goal delegations, see Figure 13. STS-ml supports the following security needs over document transmissions:

- *Integrity of transmission*: requires the sender to guarantee the integrity of the given document while providing it. For instance, in our running example the secretary requires IS Manager to guarantee the integrity of the document *Income Statement* while transferring it to the secretary. A possible way for the IS Manager to satisfy this requirement is to digitally sign the document before emailing it. The specification of security needs over document transmission is similar to the one over goal delegations (see Figure 9).
- *Document Availability:* requires the sender to guarantee an availability level of *x*% for the transmission of the specified document (similar to availability in goal delegation). For instance, while providing the document *Income Statement*, IS Manager is required by the Secretary to ensure an availability level of 70% for the transmitted document, see Figure 9.
- *Confidentiality of transmission*, which requires the transmitted to guarantee the confidentiality of transmission of the given document while providing it. For instance, the IS Manager is responsible to ensure the confidentiality of transmission of the document *Income Statement*, while providing it to the Secretary.
- *Authentication:* this requirement about authenticity could be specified either by the sender or by the receiver to the other party during a document transmission. Sender authentication indicates the receiver's request that the 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. Receiver authentication expresses the sender's need that the receiver is authenticated. We encounter this kind of authentication every day when we browse the web and use our credentials (username/password) to access web information such as email.

Apart from these security needs, which are related to the actual transfer of documents, STS-ml supports another set of *security needs*, which restrict the way received documents can be read, modified, produced, and further propagated. STS-ml takes these needs into account by considering the *social* view together with the *information* and *authorization* views that are presented in Sections 2.2 and 2.3 respectively.

2.1.1 Capturing normative requirements through organizational constraints

Apart from the security needs specified by actors when entering into interactions, there are others which are dictated by the environment, or the underlying organization, dictating role-role relations as well as agent-role and goal-goal relations. We are currently enriching STS-ml to support *Separation of Duties*, and *Combination of Duties*. These security needs are generally derived (defined) by law, organizations, or business rules and regulations. Therefore, they are somewhat different from the rest of the security needs we have considered so far. As a result, the commitments that are generated from their specification also differ, as they are not made from an actor to another actor, but instead from the actors towards the organization/law/regulation. Therefore, we represent these commitments as having only the responsible actor, but leave a dash (-) for the requestor actor.

- *Separation of duties* (SoD): we distinguish two cases of separation of duties.
  - *Between roles*: defines incompatible roles, in the sense that when specified between the two roles, it does not allow the same agent to play both the roles.
  - *Between goals*: defines incompatible goals, in the sense that when SoD is specified between two goals an actor is not allowed to or should not achieve both.

Graphically we represent SoD constraints through the "*incompatible"* relationship, which is represented as a circle with the *unequals* sign within. The relation is symmetric, and as such it does not have any arrows pointing to the concepts it relates (these being either roles or goals). Figure 14 gives an example of SoD from the running example, in which program coordinator and secretary are considered as incompatible roles, i.e., no agent can play these two roles simultaneously.

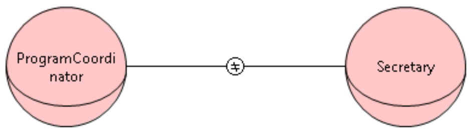


Figure 14: Organizational constraints – separation of duties among roles

- *Combination of duties* (CoD): we distinguish two cases of combination of duties.
  - *Between roles*: defines compatible roles, in the sense that when specified between two roles, CoD requires that the same agent adopts both the roles if it plays any of them.
  - *Between goals*: defines a relationship among two goals, so that a role (agent) achieving one of the goals has to achieve also the other goal.

Graphically we represent a CoD constraint through the "*combines"* relationship, which is represented as a circle with the *equals* sign within. The relation is symmetric, and as such it does not have any arrows pointing to the concepts it relates (being these roles or goals). Figure 15 shows that the goals *Derive Financial Stat* and *Fill tax Returns* should be combined, given that a CoD constraint is specified between them.

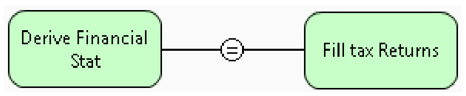


Figure 15: Organizational constraints – combination of duties among goals

Similarly to the specification of security needs over goal delegations and document transmissions, the specification of these organizational constraints, SoD and CoD, leads to the generation of commitments for their satisfaction.

2.1.2 Threats represented in STS-ml

Security analysis often considers the perspective of possible malicious users attacking the system by exploiting system vulnerabilities. In the same spirit, STS does not overlook threats, and supports the identification of social and organizational threats, which do not necessarily exploit technical vulnerabilities of a software system. STS-ml represents threats through events that exploit the vulnerabilities of actors' supporting assets (*subgoals* and *documents*) in order to undermine their primary assets (*root goals* and *information*). The primitives of STS-ml are the entity *Event* and the relationship *threatens*, which links an event to a document or a goal. Note that the STS method assumes 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, which is out of the scope of our work. STS offers stakeholders a way to verify how the identified events threaten the rest of their assets, while leaving them the choice among CORAS[2], OCTAVE[3], or any other risk analysis method that better suits their needs.

As shown in Figure 16 there are several constructs that could be threatened by an event, represented namely with associations towards roles, agents, goals, documents and delegations. Our primary focus is towards documents and goals, as important actor assets.

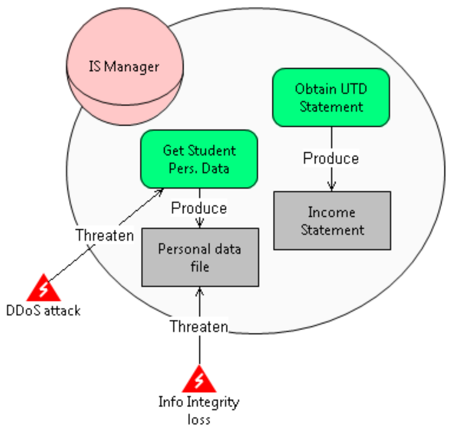


Figure 16: Threat events related to various model constructs

Figure 16 represents a simple example illustrating several different threatening events:

- DDos attack on service: since there is a goal for *IS Manager* to *Get Student Pers. Data*, an unwanted event is that some internal or external entity manages to deliberately block this.
- Information integrity loss: it would be a serious threat if an external malicious entity is able to make an unauthorized modification of student's personal data file.

[2] http://coras.sourceforge.net

[3] http://www.cert.org/resilience/products-services/octave/octave-method.cfm

2.2 Information view

Documents play a fundamental role in the social view: actors possess documents, as well as they read, modify, produce, and transmit them while fulfilling their goals. Especially when we consider security, it is very important to consider the documents that are exchanged among various actors, and define what information they represent. This is fundamental to keep track of the information flow, being this information electronic, physical or related to some specific knowledge.

The purpose of the information view is to introduce primitives and relationships to differentiate between (as well as relate) informational elements (information) and representation means (document). See the meta-model in Figure 17. We use the concepts of *information* and *document* to respectively identify and distinguish information from its representation. Resource is a generic term to represent documents and information.

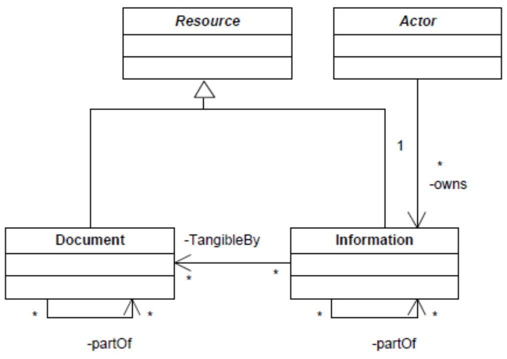


Figure 17: Information view metamodel

Information representation in terms of documents is represented through the *Tangible By* relationship. For instance, in Figure 18, the *Financial Status* of the student is an information entity (it exists irrespective of any document representing it). Such information can be transferred only if made tangible by some document. For example, when contained in a document *Income Statement*, see the Tangible By relationship from information *Financial Status* towards document *Income Statement* in Figure 18.

Information view represents who are the owners of the information given information, in order to make it clear who are the owners of the information represented by the documents exchanged in the social view. Ownership is denoted by an arrow starting from an actor towards the information owned by that actor with the label “Own”. For instance, in our running example, the student owns information *Personal Data* and *Financial Status* (Figure 18).

Note that ownership is different from possession. Actors might possess documents withholding important information, however when actors transmit information they own towards others via document transmissions, they do not give up the ownership of information. For instance, Secretary possesses document *Income Statement*, but does not own information *Financial Status*, instead the student is the owner of this information.

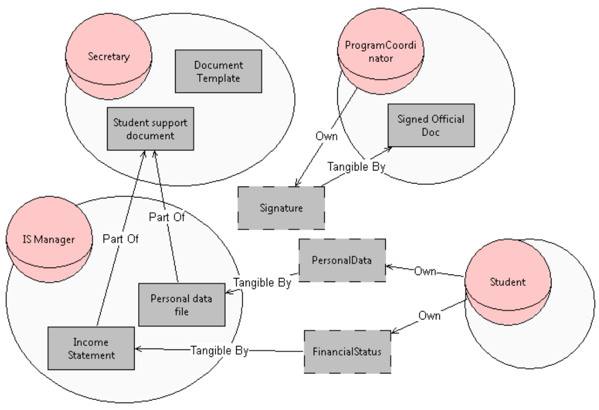


Figure 18: Information view for the running example

This distinction stands at the basis of security analysis, allowing security requirements engineers to capture possible violations of security needs, identify unauthorized manipulation of information, and so on.

Another feature of the information view is to support composite information (documents). Broadly, information contains other pieces of information, or documents can be created by putting together other documents. In the social view we keep a clean representation of documents. In the information view, instead, we want to know the pieces of information composing a given piece of information or which documents are contained within another document, so that we can reason about possible security issues at different levels of information or document structure enabling fine-grained specification of security needs. This structuring is supported by means of the *part Of* relationship, which is applied between information entities or between documents. For instance, this allows representing that a *Personal Data file* is part of the *Student Supporting Doc* the student should deliver.

The information view is flexible in representing information, documents, and the relations between them. Some examples:

Information can be made tangible by (more) different documents.

- A document can have no relevant information. This is the case, for instance, of the *Document Template,* which contains no relevant (read, confidential) information for the purpose of issuing the permit of stay.
- A document might be part of multiple documents. This might be the case of *Income Statement*, which might be also part of a scholarship application.
- Considering the kind of information that flows in an application, we do not specify vulnerabilities over documents, as it would apply in other settings, in which we have physical resources that could be exposed to certain vulnerabilities.

2.3 Authorization view

An adequate representation of authorizations is necessary to determine if information is exchanged and used in compliance with confidentiality restrictions. The information owner is the unique actor that can legitimately transfer rights to other actors. However, it might transfer full rights to another actor, so that the latter becomes entitled to transfer the same rights the owner can grant.

The authorization view presents the authorizations actors grant to others over information starting from the information owner or actors that have authority to do so. Figure 19 shows the meta-model of this view.

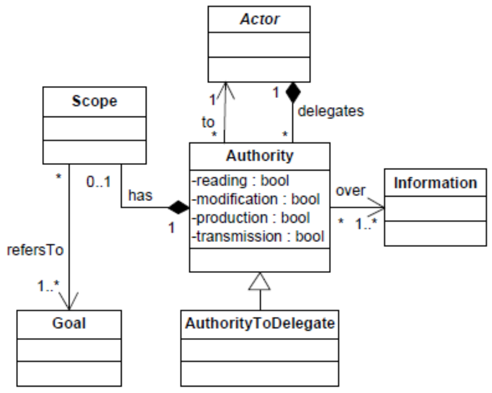


Figure 19: Authorisation view metamodel

STS-ml includes the authorization primitive, to capture two key concepts in security, namely permissions and prohibitions. The main idea behind this view, is that actors (typically information owners) may want to specify what they allow or prohibit others to do over their information. Following this intuition, the authorization relationship in STS-ml is specified over four dimensions:

- *Allowed/prohibited operations:* define whether the authorized actor is permitted (graphically green tick symbol) or prohibited (red cross symbol) to Read (R), Modify (M), Produce (P), and/or Transmit (T) any document that makes tangible the information (operations are graphically represented in four boxes with distinguishable labels, R, M, P, and T respectively) for which authorization is passed, see Figure 20.
- *Information*: authorization is granted over at least one information entity. Give the structuring of information entities in terms of part-of relationships, authorizing some actor over some information means that the actor is authorized over parts of information as well. This is because ownership of information propagates top-down through part-of relationships. The information entities over which authorization is passed is represented right below the allowed/prohibited operations, see Figure 20.
- *Scope of authorization*: authority over information can be limited to the scope of a certain goal. The delegator wants to ensure the information is manipulated for a specific purpose (achieving the specified goal). Our notion of goal scope adopts the definition in Dalpiaz *et al.* [5], 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 utilize 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.
- *Transferability of the permissions*: it specifies whether the actor that receives the authorization is in turn entitled to transfer the received permissions or specify prohibitions (concerning the received permissions) to other actors. Graphically, transferability of the authorization is allowed when the authorization arrow line connecting the two actors is solid, while it is not granted when it is dashed.

Figure 20 shows the authorization view for the running example. Facts about ownership are preserved from the information view. The graphical representation of the authorization relationship has changed from the initial version of STS-ml.

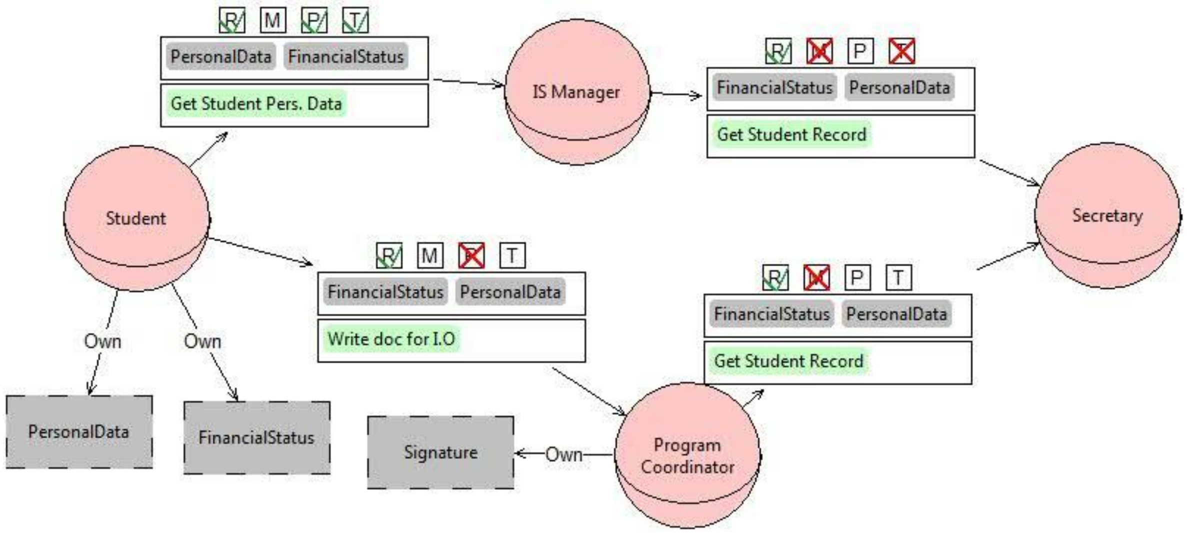


Figure 20: Authorization view for the stay permit scenario

Graphically authorization relationships reflect the four dimensions discussed above. An authorization box contains three slots: the upper defines the allowed/prohibited operations (from left to right: *Read*, *Modify*, *Produce, and Transmit*); the middle slot is the list of information over which authorization is delegated; and the lower slot is the goal scope. The fourth dimension, transferability, is captured through the authorization line: authorization to delegate is transferable (full line) or not (dashed line). For example, the student delegates the permission to read *Personal Data* and *Financial Status* to the program coordinator, in the scope of goal *Write doc for I.O*, granting a transferrable authority. Since authority to delegate is transferred, the program coordinator delegates authority to read *Personal Data* and *Financial Status* to the secretary in the scope of goal *Get Student Record* (which is a sub-goal of *Write doc for I.O.*), granting a non-transferrable authorization.

The authorization view expresses security needs about the use of information. In contrast to the social view, here security needs are expressed implicitly, through the authorizations actors pass to others. We have refined the supported security needs by considering all the different authorizations that are granted through the authorization view into the following, for each type of operation that can be performed and depending on whether authority is limited to a goal scope or not. As such, STS-ml supports the following security needs related to the use of information:

- *Non-reading*: prohibiting the read operation expresses a non-reading security requirement, which requires the information is not read in an unauthorized way; it implies that the authorizee should not read any documents making tangible the specified information. There are no examples of the non-reading security requirement in our running example.
- *Non-modification*: prohibiting the right to modify information expresses a security need about the non-modification of such information. The IS Manager expresses such security need on the delegation of authority over information *Personal data* and *Financial status* to the secretary, see Figure 20.
- *Non-production*: requires the information is not produced in a new document in an unauthorized way. For example, the student requires the program coordinator not to produce his *Personal Data* and *Financial Status*, see Figure 20 in which the operation P is prohibited being crossed over.
- *Non-disclosure*: requires that no document representing the specified information is transmitted to other actors by the authorizee. For example, the IS Manager expresses such security need in the authorization over information *Personal Data* and *Financial Status* granted to the secretary by prohibiting the right to transmit, operation T is crossed over in Figure 20.
- *Need-to-know*: when restricted to a goal scope, the authorization reflects a *need-to-know* security need. Indeed, the actor granting the authority enables the authorizee actor to perform operations or to further reauthorize others as long as the operations or the authorization are within the specified goal scope. The student's authorization to the IS Manager expresses a need-to-know security need: *Personal Data* and *Financial Status* should be read, produced or transmitted only in the scope of goal *Write doc for I.O.*

- *Non-reauthorization*: requires that the authorization is not transferrable, i.e., the authorizee does not further transfer rights for operations not granted to him or when transferability is set to false. This means that any *non-reading*, *non-modification*, *non-production* and *non-disclosure* security need implies a *not-reauthorize* security need for the operations that are prohibited.

3 Deriving security requirements

The operational view described in Section 2 models how the organization operates and helps analysts to capture and represent stakeholders' security needs. As shown in Section 2.3 however, security needs are often modeled implicitly. Thus, their identification should be facilitated via automated reasoning. The security requirements are *automatically derived* from the operational view and are expressed via social commitments that fulfill the required security needs. For each security need specified over an interaction, a commitment on the opposite direction is expected for the fulfillment of that security need. Security requirements are derived not only from security needs, but take into account also the organizational constraints expressed over an STS-ml model. Whenever an organizational constraint is expressed, be it an SoD or a CoD constraint, a commitment for its fulfillment is automatically generated via the supporting toolset, STS-Tool.

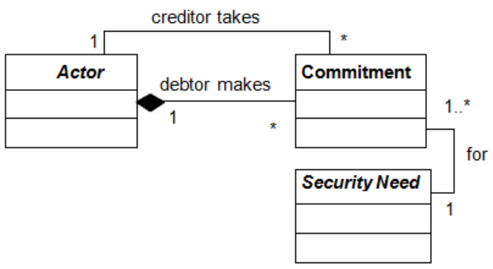


Figure 21: Security requirements metamodel

An important feature of STS-ml is that it relates security requirements to *interaction* between actors (interaction is understood in business terms). Interaction is captured by the social relations supported by STS-ml, with the intent to represent interactions among *service users* (represented by delegator, receiver, or authorizee) and service providers (delegatee, sender, or authorizer). On top of these interactions security needs are expressed.

At requirements time, commitments are expressed at the level of roles (with the exception of the agents that are already known and modeled) for the satisfaction of these needs. At runtime, these commitments shall be made by the involved agents (playing those roles). Since security needs are generally expressed by the service user towards the service provider, the latter shall make a commitment for the satisfaction of the expressed security need. After their identification, it is therefore crucial, during the architectural design phase, to link commitments to technical security mechanisms that offer guarantees for the satisfaction of security requirements. The commitments view, that helps capture security requirements, is graphically depicted in Figure 21. The notion of social commitment is specialized so that it can be exploited in the context of security requirements. A commitment is made by a debtor actor (*responsible*) to a creditor actor (*requester*) for the satisfaction of a security need (*requestee*). In turn, security needs are defined in terms of the concepts used in the operational view (as shown in the previous sections). In the case of organizational constraints, the commitment is required by the STS, representing the socio-technical system, to all actors participating in the given system, as already explained in Section 2.1.1.

The way commitments are implemented is highly dependent on whether the involved actors are agents or roles. If the debtor is a role, making that commitment becomes a necessary condition for any agent playing that role, since the debtor is the responsible actor for bringing about the security requirement to satisfy the security need being specified. The commitment becomes part of the description of the role. If the creditor is a role, that commitment is a security guarantee for any agent playing that role, since the creditor is the requester of the security need. If the debtor is an agent, the system-to-be should ensure that the specific agent makes those security commitments when interacting with others. If the creditor is an agent, such commitments become prerequisites for other agents interacting with it.

Table 1 presents the commitments' types supported by STS-ml to capture security requirements for each expressed security need. In these commitments *a* and *b* refer to actors, *R* refers to roles, *I* stands for set of information (as such could contain also only one single information entity), *G* a set of goals (single specific goals for commitments (w) – (z), following their semantics described below), *D* for documents, while *Ops* is the set of operations {*R*,*M*,*P*,*T*} actors can perform over information through goal-document relationships or document transmissions.

Id	Commitment type
(a)	*C(b, a, authorised(a,b,I,G,Ops,t), need-to-know(I, G, Ops))*
	Actor *b* commits to actor *a* that information in *I* will be read/modified/created/transmitted (as specified in Ops) *only* in the scope of the goals in *G*
(b)	*C(b, a, authorised(a,b,I,_,Ops\\{T},_),non-disclosure(I))*
	b commits to *a* that information in the set *I* will not be further transmitted
(c)	*C(B, A, authorised(a,b,I,_,Ops\\{P},_), non-production(I))*
	b commits to *a* that information in *I* will not be produced in an unauthorized way
(d)	*C(b, a, authorised(a,b,I,_,Ops\\{M},_), non-modification(I))*
	a commits to *b* that information in *I* will not be modified in an unauthorized way
(e)	*C(b, a,authorised(a,b,I,_Ops\\{R},_), non-reading(I))*
	b commits to *a* that information in *I* will not be used in an unauthorized way
(f)	*C(b, a, authorised(a,b,I,_,Ops,false),non-reauthorization(b,c,I,G,Ops))*
	b commits to *a* that *b* will not reauthorize any other actor *c* [2]
(g)	*C(b, a, non-repudiation-of-acceptance(isdelegated(b,a,G)))*
	b commits to *a* that *b* will not repudiate that it (*b*) has been delegated the goals in *G*
(h)	*C(a, b, non-repudiation-of-delegation(delegated(a,b,G)))*
	a commits to *b* that *a* will not repudiate having delegated *b* the goals in *G*
(i)	*C(a, b, fback_rs(G))*
	a commits to *b* that fallback redundant strategies involving a single actor will be adopted to fulfil the goals in *G*
(j)	*C(a, b, fback_rm(G))*
	a commits to *b* that fallback redundant strategies involving multiple actors will be adopted to fulfil the goals in *G*
(k)	*C(a, b, true_rs(G))*
	a commits to b that true redundant strategies involving a single actor will be adopted to fulfill the goals in G
(l)	*C(a, b, true_rm(G))*
	a commits to *b* that true redundant strategies involving multiple actors will be adopted to fulfill the goals in *G*
(m)	*C(a, b, no-redelegation:(G))*
	a commits to *b* that goal *G* will not be delegated to others
(n)	*C(a, b, availability(G,x%))*
	a commits to *b* that *G* will have a minimum availability level of at least *x*%
(o)	*C(a,_,delegatedTo(b,trustw(b))*
	a commits to delegate only to *b* that has a minimum trustworthiness level of *x*
(p)	*C(a, b, integrity(D))*
	a commits to *b* to guarantee the integrity of the transmitted document *D*
(q)	*C(a, b, availability(D,x%)*
	a commits to *b* that it will guarantee a minimum availability level of *x*% for document *D*
(r)	*C(a, b, confidentiality(D,x%)*
	a commits to *b* to guarantee the confidentiality of the transmitted document *D*

(s)	*C(a,b,delegator-authentication(delegated(a,b,G))*
	a commits to b to authenticate before delegating G to b
(t)	*C(b,a,delegatee-authentication(delegated(a,b,G))*
	b commits to a to authenticate for the delegation of goal G by a
(u)	*C(a,b,sender-authentication(transmitted(a,b,D))*
	a commits to b to authenticate for the transmission of document D
(v)	*C(b,a,receiver-authentication(transmitted(a,b,D))*
	b commits to a to authenticate for the transmission of document D by a
(w)	*C(a,STS,role-sod(R1,R2))*
	*a*commits *not to play both* roles *R1* and *R2*
(x)	*C(a,STS,goal-sod(G1,G2))*
	a commits *not to pursue both* goals *G1* and *G2*, i.e. not to be the final performer of both these goals
(y)	*C(a,STS,role-cod(R1,R2))*
	a commits to *play both* roles *R1* and *R2*
(z)	*C(a,STS,goal-cod(G1,G2))*
	a commits to *achieve both* goals *G1* and *G2*, i.e. to be the final performer of both these goals

Table 1: Commitment types to express security requirements

Table 2 lists the commitments for the stay permit scenario derived from the business view presented in the previous sections. With the help of examples in Table 2, the semantics of the various commitment types in Table 1 may be easily described as follows:

(a) A need-to-know commitment from *b* to *a* implies that a set of information *I* will be read / modified / produced / transmitted (in accordance with the operations specified in *Ops*) only within the scope of a set of goals *G*. In case the committed actor has the authority to delegate rights, other actors might in turn be authorized for the information. However, to guarantee the commitment made by *b*, each of other actors has to make a commitment to *a* for the need-to-know of the information. For example, in Table 2, the IS Manager commits (C_1) to the student for the need-to-know of *Personal Data* and *Financial Status* in the scope of goal *Get Student Pers. Data*. Allowed operations are *production* and *transmission*. In turn, this implies a commitment (C_{11}) from the secretary to the IS Manager for that information and operations in the scope of the sub-goal *Get Student Record*.

(b) A non-disclosure commitment guarantees that the debtor will not transmit information to other actors. This type of commitment protects delegations of authority that allow for transmitting the information. For example, the secretary commits (C_{18}) to the IS Manager for the non-disclosure of *Personal Data* and *Financial Status*.

(c) A non-production commitment for some information I implies that the information will not be produced through the creation of any new document. For example, the program coordinator (C_2) commits to student not to produce his *Personal Data* and *Financial Status*.

(d) A non-modification commitment for a set of information *I* implies that the information will not be modified. The debtor actor commits that not only will he not modify the information, but also that — if he transmits such information to other actors — each of these actors will commit for the non-modification of the information. For example, the program coordinator commits (C_5) to the student for the non-modification of *Personal Data* and *Financial Status*, since he gets no authority to modify such data. In turn, a similar commitment (C_{13}) is made from the secretary to the program coordinator.

(e) A non-reading commitment for a set of information *I* implies that the information will not be read. We do not have a non-reading security need expressed in our running example.

(f) A non-reauthorization commitment for a set of information *I* implies that no authorization for the information will be passed to other actors. For example, the program coordinator (C_8) commits to student not to pass authorization on producing his *Personal Data* and *Financial Status*.

(g) Commitments for non-repudiation required by the delegator are essential to guarantee accountability. We are concerned here with non-repudiation of goal delegations. The committed actor ensures that he will not repudiate that he was delegated the fulfillment of the goals in *G*, non-repudiation of acceptance. For example, the program coordinator commits (C_9) to the student for the non-repudiation of goal *Write doc for I.O.*

(h) A non-repudiation-of-delegation commitment requires the delegator not do deny (repudiate) having delegated the goal to the delegatee[4].

(i) A fallback redundancy single (fback_rs) commitment guarantees that the goals in *G* will be fulfilled by adopting fallback redundant strategies by the committed actor, and no external actors will be involved.

(j) A fallback redundancy multi (fback_rm) commitment guarantees that adopting fallback redundant strategies by the committed actor will fulfill the goals in G, and external actors might be involved as well.

(k) A true redundancy single (true_rs) commitment guarantees that the goals in *G* will be fulfilled by adopting true redundant strategies by the committed actor, and no external actors will be involved.

(l) A true redundancy multi (true_rm) commitment guarantees that adopting true redundant strategies by the committed actor will fulfill the goals in G, and external actors might be involved as well. For example, the IS Manager commits (C_{21}) for redundant fulfillment (more specifically using *true redundancy multi*) of goal *Get Student Pers. Data*. The IS Manager can fulfill it by either retrieving two statements from different databases, or delegating the task to two technicians.

(m) A no-redelegation commitment is a guarantee by the debtor actor that a goal will be fulfilled without delegating it to others. Such restriction applies to the descendants of the goal in the goal hierarchy. For example, the IS Manager Commits (C_{20}) to the secretary that he will not delegate goals *Get Student Pers. Data* to others, as the secretary trusts only the manager and not technician for such confidential activities.

(n) An availability commitment over goal delegations guarantees that a minimum availability level, as specified by the security need, will be ensured for the delegated goal.

(o) A trustworthiness commitment over goal delegations guarantees that the delegator will only delegate to delegatees with a minimum trustworthiness level, as specified in the security need.

(p) An integrity commitment over a document transmissions guarantees that the sender will ensure the integrity of transmission for the transmitted document. For example, IS Manager commits to Secretary that will assure integrity of transmission for document Income Statement (C_{23}).

(q) An availability commitment over document transmissions guarantees that a minimum availability level will be ensured for the transmitted document. For example, the Secretary commits to IS Manager that a 70% minimum availability will be assured for document *Income Statement*(C_{24}).

(r) A confidentiality commitment over document transmissions guaranties that the sender will ensure the confidentiality of transmission for the transmitted document. For example, IS Manager commits to Secretary that will assure confidentiality of transmission for document Income Statement (C_{25}).

(s) A delegator-authentication commitment over goal delegations guaranties that the delegator will authenticate for the delegation to take place.

(t) A delegatee-authentication commitment over goal delegations guaranties that the delegate will authenticate before taking action towards the achievement of the delegated goal. For example, the IS Manager commits to Secretary that it will authenticate for achieving the delegated goal Obtain UTD Statement.

(u) A sender-authentication commitment over document transmissions guaranties that the sender will authenticate for the transmission to take place.

(v) A receiver authentication commitment over document transmissions guaranties that the receiver will authenticate before taking any actions over the transmitted document. For instance, the Secretary makes a commitment to authenticate in order to have document Personal data file from the IS Manager.

4 The supporting toolset offers the opportunity to specify a non-repudiation-of-delegation-and-acceptance commitment that is equivalent to two commitments, one made by the delegator for not repudiating the delegation it has passed to the delegatee, and the second one made by the delegatee for not repudiating having been delegated the fulfillment of the delegated goal.

(w) A commitment for role-based separation of duties (not-play-both) guarantees that no agent will adopt roles among which separation of duties is expressed. For instance, all agents in the running example commit not to ever play both roles *Program Coordinator* and *Secretary*, if they play one of these roles (C_{26}).

(x) A commitment for goal-based separation of duties (not-achieve-both) guarantees that for every agent A, A will not achieve both goals G1 and G2, should A take actions to achieve one of the two (either G1 or G2).

(y) A commitment for role-based combination of duties (play-both) guarantees that if agent A will adopt role R1 (R2), it will adopt role R2 (R1) too.

(z) A commitment for goal-based combination of duties (achieve-both) guarantees that for every agent A, if A pursues goal G1 (G2), then A should pursue G2 (G1) too. For instance, any agent in our running example should achieve both goals *Derive Financial Stat* and *Fill tax returns*, if it decides to achieve one of them (C_{27}).

ID	Debtor	Creditor	Security Requirement
C_1	IS Manager	Student	need-to-know({PersonalData,FinancialStatus}, {Get Student Pers. Data}, {u,m})
C_2	IS Manager	Student	non-production({PersonalData,FinancialStatus})
C_3	IS Manager	Student	non-disclosure({PersonalData, FinancialStatus})
C_4	IS Manager	Student	need-to-know({PersonalData, FinancialStatus}, {Write doc for I.O.}, {u})
C_5	IS Manager	Student	non-modification({PersonalData, FinancialStatus})
C_6	Program Coordinator	Student	non-production({PersonalData, FinancialStatus})
C_7	Program Coordinator	Student	non-disclosure({PersonalData, FinancialStatus})
C_8	Program Coordinator	Student	non-reauthorization({PersonalData, FinancialStatus}, {p})
C_9	Program Coordinator	Student	non-repudiation({Write doc for I.O.})
C_{10}	Secretary	Program Coordinator	non-repudiation-of-delegation({Write new doc})
C_{11}	Secretary	Program Coordinator	non-repudiation-of-acceptance({Get Student Record})
C_{12}	Secretary	Program Coordinator	no-redelegation({Write new doc})
C_{13}	Secretary	Program Coordinator	need-to-know({PersonalData, FinancialStatus}, {Get Student Record}, {u})
C_{14}	Secretary	Program Coordinator	non-modification({PersonalData, FinancialStatus})
C_{15}	Secretary	Program Coordinator	non-production({PersonalData, FinancialStatus})
C_{16}	Secretary	Program Coordinator	non-disclosure({PersonalData, FinancialStatus})
C_{17}	Secretary	IS Manager	need-to-know({PersonalData, FinancialStatus}, {Get Student Record}, {p,d})
C_{18}	Secretary	IS Manager	non-production({PersonalData, FinancialStatus})
C_{19}	Secretary	IS Manager	non-disclosure({PersonalData, FinancialStatus})
C_{20}	IS Manager	Secretary	no-redelegation({GetStudentPers.Data})
C_{21}	IS Manager	Secretary	non-repudiation({GetStudentPers.Data})
C_{22}	IS Manager	Secretary	true_rm({GetStudentPers.Data})
C_{23}	IS Manager	Secretary	integrity({Income Statement})
C_{24}	IS Manager	Secretary	availability(Personal Data File, 86%)
C_{25}	IS Manager	Secretary	confidentiality({Income Statement})
C_{26}	All agents	STS	not-play-both(ProgramCoordinator, Secretary)
C_{27}	All agents	STS	pursue-both(DeriveFinancialStat, Fill tax returns)

Table 2: An excerpt of security requirements expressed via commitments

Operationalizing commitments: the security commitments specified in the commitments view are security requirements at the organizational level. However, their level of abstraction is closer to technical requirements than existing goal-oriented methods for security requirements engineering. We show some examples to justify the validity of our claim.

Commitment C_1 requires ensuring need-to-know. A possible technical requirement that operationalizes C_1 is to log access to the information system and require IS users to specify the purpose for which they access confidential data. The purpose might be derived from interaction. In our example, the system to-be can match whether the IS manager is utilizing *Personal Data* and *Financial Status* upon a request (e.g. by the secretary) to get student personal data.

Commitment C_5 is about non-modification. At least two technical options exist: preventively denying modification grants to the Program Coordinator, or monitoring its access to *Personal Data* and *Financial Status*.

C_9 is about non-repudiation of goal *Write doc for I.O*. To ensure C_9, an information system can be developed; students have to send their requests (delegations) through it. The Program Coordinator has to accept the task. The log of the information system is the proof that the delegation was accepted.

To guarantee commitments C_{19} and C_{20}, non-disclosure and no-redelegation, respectively, the information flow should be followed. While C_{19} directly refers to information, C_{20} does it indirectly, since the delegated goals produce documents that can be traced.

4 The STS method for security requirements engineering

The STS-method supports security requirements engineering considering security issues as early as in the requirements phase. It takes an interaction-oriented stance to security, allowing actors to express their expectations regarding security over interactions with other actors. We consider these expectations to be the desired security needs from which security requirements are derived. On the other hand, the method enables to capture the business rationale behind these security requirements, representing actors' business policies, that is, their goal models (how they intend to achieve their goals).

STS method builds on top of Tropos [9] and its security-oriented extension [4]. As such, it includes high-level organizational concepts such as actor, goal, delegation, etc. Security requirements are mapped to social commitments [7] — contracts among actors — that actors in the STS shall comply with at runtime.

STS method is based on the idea of building a model of the system that is incrementally and iteratively constructed and refined by focusing on different perspectives (views) at a time. That is, modeling consists of building three complementary views: *social*, *information*, and *authorization* view, so that different interactions among actors can be analyzed by working on a specific perspective (view) instead of building a single big model of the system-at-hand. This approach promotes modularity and helps designers separating concerns.

The STS method considers a socio-technical system as composed of actors that have objectives to achieve and interact with others to get things done. We consider social actors that collaborate to fulfill their own objectives. The *social view* represents actors as intentional and social entities. Actors are intentional as they have goals they want to achieve, and they are social, because they interact with others to get things done, mainly by delegating goals. Actors may possess documents, they may read, modify, or produce documents while achieving their goals, and they may transmit documents and exchange information through document transmissions to other actors.

The STS method distinguishes between information and its representation via documents. It is important to keep track of how information and documents are interconnected, to be able to identify which information actors manipulate, while using, modifying, producing, or transmitting documents for achieving their goals. This is the purpose of the *information view*, which shows the informational content of the different documents in the social view, and transmits a structured representation of the various information pieces and documents in the given setting as well.

An adequate representation of authorizations is necessary to determine if information is exchanged and used in compliance with confidentiality restrictions. The *authorization view* shows the permission flow from actor to actor, that is, the authorizations actors grant to others about information, specifying the operations actors can perform on the given information, namely read, modify, produce, and transmit. Apart from granting authority on performing operations, we consider also whether authority to further give authorizations is granted.

Following our intuition of relating security to interactions, we allow stakeholders to express their security requirements over goal delegations, document transmissions and authorizations regarding information. We consider the interplay between different requirements sources: the business policies of individual actors, their security expectations on other actors, and the normative requirements in the STS. For more details on the three views and the supported security requirements, see Section 2.

We present now the steps the security requirements engineer can follow in order to perform the modeling and analysis activities (see Figure 22), while illustrating them in building the model for the running example. Each view is built incrementally, considering the various stakeholders to be represented, their assets, interactions, and so on, until all necessary details are modeled. Thus, each step might be repeated several times, till the security requirements engineer considers the model to be complete. Moreover, for every view details are refined, so that changes in one view are reflected in the other, to maintain consistency of the overall model.

- ***Phase 1. Model the Social View***: STS modeling starts with the representation of the actors present in the scenario (*step 1.1. Identify stakeholders*). We consider actors to be either *agents*, to refer to concrete participants, or *roles*, to refer to abstract actors (abstracted from agents, used when the actual participant is unknown). In our running example, we represent for instance *Student* as a role. The reason is that Student can be instantiated by any international student studying at the University of Trento. From the description of the running example we focus on the actors highlighted in Section 1.2. If we considered a broader scope, say enter in the details of programs offered by UniTN and represent UniTN as an actor of the system, then UniTN would be an agent, because in this context this actor is invariably the same. Moreover, this is an actor we know already at design time when we create the models that are going to be part of the system. Intuitively, this step is *repeated* for as many times as there are actors present in the considered setting.

- *step 1.2. Identify stakeholders' assets and interactions*: after identifying stakeholders, we continue with the identification of their assets. We consider actors that have *goals* to achieve and that possess *documents* necessary for the achievement of their goals. Goals and documents are what we call actors' *assets*. Actors may have the capability to achieve some of their goals, for others they need to rely on other actors. Similarly, they might possess some of the necessary documents, for others they need to rely on other actors that can transmit the desired documents to them. This reflects the way business is conducted in reality, where social interaction is crucial to enable actors achieve goals that they are not capable of achieving themselves or that others can perform with higher quality or lower costs. The STS method supports the identification of actors' assets considering the various actors one by one, while refining their inner models, and exploring their interactions with other actors at the same time. In *step 1.2.*, we start with the identification of stakeholders' assets, namely their goals. Actors want to achieve one or more goals: for instance, Secretary has goal *Get Student Record*, for which it reads document *Student Supporting Doc* (see Figure 9 in Section 2.1). Goals are refined by AND/OR-decompositions obtaining goal trees: online system built is the root goal to be fulfilled. The root goal *Get Student Record* is and-decomposed into *Get Student Pers. Data* and *Derive Financial Stat*. The goal *Derive Financial Stat* is further decomposed through an and-decomposition into goals *Obtain UTD statement* and *Check Income Adequacy*. The refinement process continues for the other goals too. For simplicity, we will not enter the details of other actors' goal models.
- *Identify interactions (iteration of step 1*.2): for some goals, actors need to rely on others through goal delegation. We identify the interactions the actor needs to enter in order to achieve its goals (*step 1.2. Identify stakeholders' assets and interactions*). For instance, the Secretary relies on IS Manager to obtain student's personal data, by delegating to IS Manager the goal *Get Student Pers. Data.* Similarly, it delegates the goal Obtain UTD statement, again to IS Manager.
- *Are there any Security Requirements? (step 1.3. Express security requirements)*: this is a particular feature of the STS method that allows stakeholders to express their concerns regarding security over the interactions they take part. Following this, the security requirements engineer analyses goal delegations, to see if any of the supported security needs applies over the identified goal delegations. In Figure 9, the Secretary wants the IS Manager not to further delegate goal *Get Student Pers. Data*. A more complete list of supported security requirements is provided in Section 2.
- *Refining the Social View*: to achieve their goals, actors might read, modify, or produce documents (iteration of step 1.2.). So, in this iteration of step 1.2., we identify the rest of stakeholders' assets, namely their documents. For instance, *Secretary* needs document *Personal Data File* to achieve goal *Get Student Pers. Data*, and it reads document *Income Statement* to achieve goal *Obtain UTD Statement*.
- *step 1.4 Model threats*: represent the events that threaten actors' assets, namely their goals and documents. Broadly, an event threatening a goal means that the goal cannot be achieved, whereas an event threatening a document means that the document becomes not available. In our example, the document *Personal Data File* produced by *IS Manager* is threatened by an event. Since this document is read by the *Secretary* to have student's personal data, it is important to know whether this will have an effect on the secretary achieving this goal (more details on this are provided by the activities in *Phase 4*, namely *step 4.3.*).

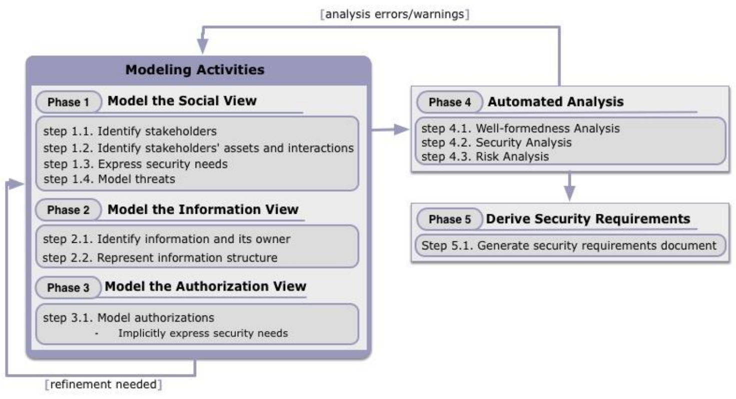

[analysis errors/warnings]
Modeling Activities
Phase 1 Model the Social View
step 1.1. Identify stakeholders
step 1.2. Identify stakeholders' assets and interactions
step 1.3. Express security needs
step 1.4. Model threats
Phase 2 Model the Information View
step 2.1. Identify information and its owner
step 2.2. Represent information structure
Phase 3 Model the Authorization View
step 3.1. Model authorizations
Implicitly express security needs
[refinement needed]
Phase 4 Automated Analysis
step 4.1. Well-formedness Analysis
step 4.2. Security Analysis
step 4.3. Risk Analysis
Phase 5 Derive Security Requirements
Step 5.1. Generate security requirements document

Figure 22: STS method

- ***Phase 2. Model the Information View***: as described earlier, the STS method distinguishes between information and its representation. Information as is cannot be transferred or manipulated, instead information entities are represented through documents, and all operations are performed over the latter. In the social view we represented actors' documents, the exchange of documents, and the different operations actors might read these documents to achieve their desired goals. We now switch to the information view and represent information entities relating them to the documents within which they are contained. The roles, agents, and their respective documents are already modeled in the social view, so they are preserved also in the information view. Hence, the designer needs just specify what information they represent (the relation *TangibleBy* is used to express that an informational entity is represented or made tangible by a given document). For instance, document Income Statement of IS Manager contains the Financial Status of the various students (see Figure 18).
  - *Model ownerships, step 2.1. Identify information and its owner*: after identifying the informational content of the modeled documents, we define who the owners of the different information entities that we have identified are. In our example, the Student is the owner of his *Personal Data* and *Financial Status*, whereas *Program Coordinator* is the owner of his own *Signature*.
  - *Step 2.2. Represent information structure*: apart from representing the different information pieces and the respective documents, the designer can capture the structure of information and documents for the given scenario. Since documents are preserved from the social view, in order to represent how the different documents are interconnected, the designer simply needs to relate the various documents (we use the *Part Of* relationship to capture this structure). The same applies to building the information structure, the different information pieces are connected through Part Of relationships. For instance, documents *Income Statement* and *Personal Data File* are part of document *Student Supporting Doc*, see Figure 18.
- ***Phase 3. Model the Authorizations View***: starting from information owners, we draw the authorizations they grant to other actors. In our example, the *Student* authorizes the *Program Coordinator* to read information *Financial Status* and *Personal Data* in the scope of goal *Write Doc for I.O* but prohibits the right to produce (see Figure 20). Security requirements over authorizations (authorization requirements) are specified implicitly from the granted authorization, so the security requirements engineer needs to understand the permissions actors want to grant together with the security concerns they have (the prohibitions they want to specify), and model them with the help of authorizations relationships. For instance, the Student requires the *Program Coordinator* not to produce *Personal Data*, since the label P for the operation produce is prohibited. Some security needs are implicitly derived, when no authorized actor grants the permission. For instance, the Program Coordinator is required not to disclose students Personal Data, since the program coordinator is not the owner of such information and there is no authorization towards him granting the right to transmit (T), see Figure 20.

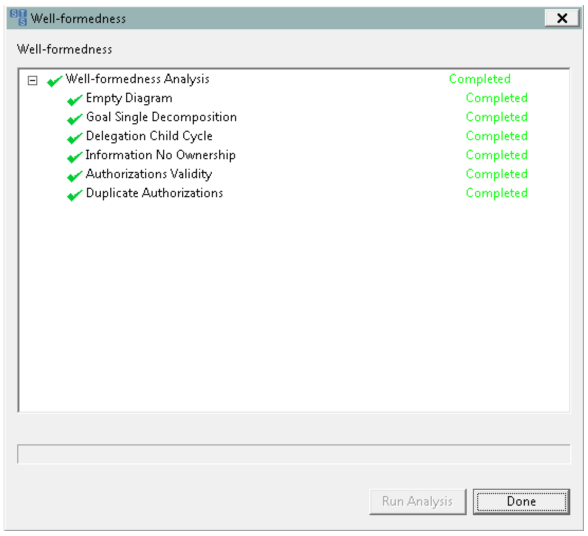
We emphasize that the modeling process dictated by the STS method (Phases 1—3) is iterative. The views can be further refined, depending on the level of detail that is needed. Therefore, in building the STS-ml model for our running example we can continue modeling by following these steps:

- *Further refine the Social View*: actors might need to *read* or *modify* particular information, but they might not have the documents that represent this information. Therefore, they might need to get this information from other actors. Thus, the next step in the modeling process consists in representing information exchange. *Information exchange* is captured by the document transmission relationship. This relationship models another type of interaction among actors, so this activity is another iteration of *step 1.2.* A document can be transmitted in STS-ml only by an actor that is in possession of that document. An actor is in possession of a document if it produces the document or the document is within the actor's rationale and there are no incoming document transmissions towards him. In Figure 9, the *IS Manager* produces document *Income Statement* and transmits it to the *Secretary*, which reads this document to achieve goal *Obtain UTD statement*. Refinement continues until the designer considers that all the important interactions have been captured and all required security requirements have been expressed.
- *Refine the Information View*: similarly to the previous identified information entities, the security requirements engineer considers for each document what its informational content is. An information could be made tangible by more documents, as well as the same document can make tangible more information entities. The designer continues modeling until all the relevant information entities have been represented.
- *Refine the Authorization View*: for all the represented actors, consider whether there are permissions being granted to them or whether they grant any permissions/specify any prohibitions to the interacting actors. Termination criteria are established depending on whether all important interactions in terms of permission flow have been captured and all the correct authorization security needs have been expressed.

Once the modeling is done and all security expectations have been expressed, the complete list of security requirements can be derived.

STS-ml's modeling and analysis abilities have been implemented in STS-Tool [6] (http://www.sts-tool.eu), to support graphical modeling of socio-technical systems, automatic derivation of security requirements, requirements document generation, as well as automated analysis. Therefore, ***Phases 4*** and ***5*** are automated by STS-Tool [6].

- ***Phase 4. Automated Analysis***: in this phase the STS-ml model built by the designer is analyzed through the three types of analysis supported by STS-Tool, namely *well-formedness analysis*, *security analysis*, and *threat analysis*. They focus on different aspects of the modeled socio-technical system, the provided steps are presented in a suggested order, and however they can be executed by the security requirements engineer in any order. Importantly, the results of the analysis can be used to improve the models, based on the feedback provided by the tool by means of visualizations of analysis results (in terms of warnings and errors identified), as indicated by the arrow going back from Phase 4 to the Modeling Activities (represented by Phases 1 – 3).
    - <u>*Step 4.1. Well-formedness Analysis*</u>: the designer launches well-formedness analysis to verify the correctness and validity of the STS-ml model under consideration. This analysis runs well-formedness rules, to verify validity of STS models. These checks are performed a posteriori, not on the fly, as they are computationally expensive and would slow down the modeling process if executed on the fly. Well-formedness analysis returns results in terms of warnings and errors. Warnings can be ignored by the security requirements engineer (though it is good practice to consider them), while errors must be corrected. So, to verify whether the model for our running example is valid, we run the well-formedness analysis, which found no errors (see Figure 23). This is comprehensible given that our running example is necessarily a small illustrative scenario. However, in larger models, it is unavoidable that there are well-formedness warnings or errors. The security requirements engineer is provided with details for all findings, both graphically over the models and textually in terms of descriptions.

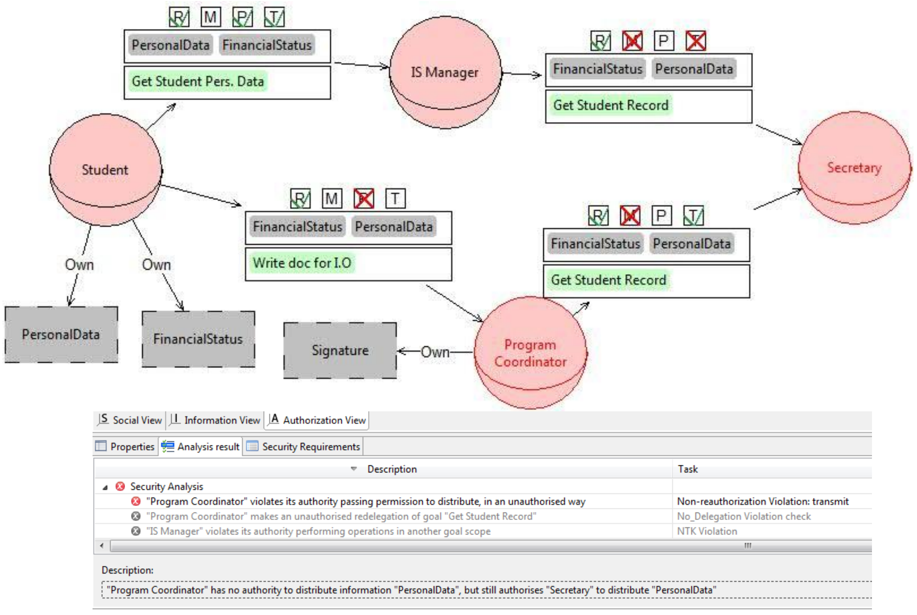


Well-formedness

Well-formedness Analysis	Completed
Empty Diagram	Completed
Goal Single Decomposition	Completed
Delegation Child Cycle	Completed
Information No Ownership	Completed
Authorizations Validity	Completed
Duplicate Authorizations	Completed

Figure 23: Well-formedness Analysis results for the running example

- *Step 4.2: Security Analysis*: verify (i) if there are any conflicting security requirements; (ii) if the diagram allows the satisfaction of the specified security requirements (i.e., there are no conflicts between stakeholders' business policies and the security requirements they should bring about). This analysis is implemented in disjunctive Datalog and consists of comparing the possible actor behaviors that the model describes against the security requirements they should satisfy. The results are enumerated in a tabular form below the diagram, and rendered visible on the diagram itself when selected (see Figure 24 and Figure 25). A textual description provides details on the identified conflicts.
  - We run the security analysis on our model to verify *whether there are any conflicting security requirements*. The analysis identified an authorization conflict *on authority to transmit* for the *Secretary*, as the *IS Manager* prohibits him the authority to transmit *Financial Status* and *Personal Data*, while the *Program Coordinator* grants this right over the same information, and in the scope of the same goal. Both these authorizations are highlighted and visualized as errors by the analysis and details are provided in the Analysis tab (see Figure 24). The designer needs to take into account the details on this finding and negotiate to resolve them. A possible way to resolve this conflict is by revoking the authorization given that the Secretary does not need it (as it can be seen from the social view).



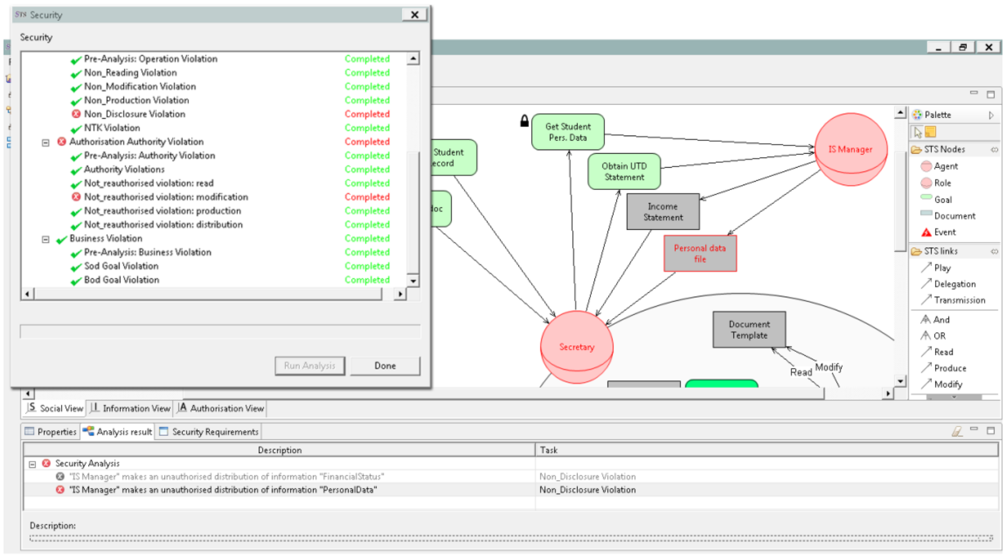
Description	Task
Security Analysis	
"Program Coordinator" violates its authority passing permission to distribute, in an unauthorised way	Non-reauthorization Violation: transmit
"Program Coordinator" makes an unauthorised redelegation of goal "Get Student Record"	No_Delegation Violation check
"IS Manager" violates its authority performing operations in another goal scope	NTK Violation

Description:

"Program Coordinator" has no authority to distribute information "PersonalData", but still authorises "Secretary" to distribute "PersonalData"

Figure 24: Security Analysis results for the running example – authorization conflicts

- After resolving authorization conflicts, we ran the security analysis again to verify the satisfaction of security needs (identify possible violations). The security analysis over our running example identified that the *non-disclosure* security need is violated (see Figure 25). These conflicts arise because of the different policies stakeholders have in achieving their goals, and the security requirements imposed to them by others. In order to resolve these conflicts, the security requirements engineer needs to find a trade-off and negotiate with stakeholders in order to relax or change one of the requirements, either the security requirement or the one from the actor's business policy.



Security

✓ Pre-Analysis: Operation Violation	Completed
✓ Non_Reading Violation	Completed
✓ Non_Modification Violation	Completed
✓ Non_Production Violation	Completed
✗ Non_Disclosure Violation	Completed
✓ NTK Violation	Completed
✗ Authorisation Authority Violation	Completed
✓ Pre-Analysis: Authority Violation	Completed
✓ Authority Violations	Completed
✓ Not_reauthorised violation: read	Completed
✗ Not_reauthorised violation: modification	Completed
✓ Not_reauthorised violation: production	Completed
✓ Not_reauthorised violation: distribution	Completed
✓ Business Violation	Completed
✓ Pre-Analysis: Business Violation	Completed
✓ Sod Goal Violation	Completed
✓ Bod Goal Violation	Completed

Description	Task
✗ Security Analysis	
"IS Manager" makes an unauthorised distribution of information "FinancialStatus"	Non_Disclosure Violation
✗ "IS Manager" makes an unauthorised distribution of information "PersonalData"	Non_Disclosure Violation

Description:

Figure 25: Security analysis results - non-disclosure

- <u>*step 4.3 Threat Analysis*</u>: Threat analysis focuses on events threatening goals or documents, to then propagate the effects over goal trees and goal/document relationships internal to the actor (read, modify, produce), as well as social relationships involving goals and documents (goal delegation and document transmission). We ran *threat analysis* on our model to identify that the event threatening *IS Manager*'s document *Personal Data File* threatens also Secretary having this document. As a consequence it threatens secretary's goal *Get Student Pers. Data*, and threatens his root goal *Get Student Record*, for which *Get Student Pers. Data* is an and-subgoal (see Figure 26). Similarly, the threat is propagated in the rest of the model. The details are provided in the Analysis tab below the model and the textual description enumerates all elements of the path along which the threatening effect of the event is propagated.

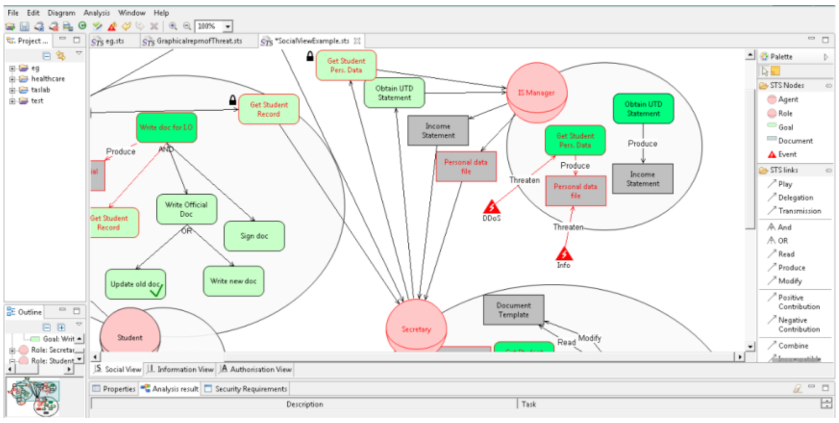
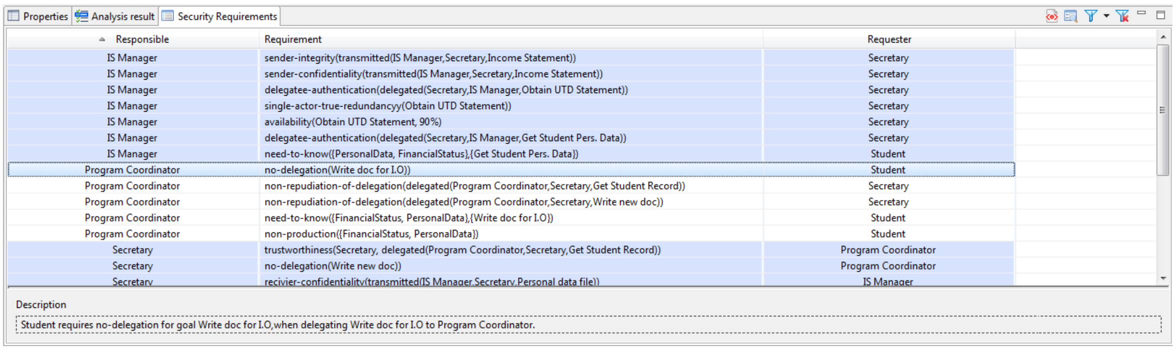


Figure 26: Threat Analysis results for the running example

More details on the checks performed by each analysis are provided in Appendix.

- ***Phase 5. Derive Security Requirements***: STS-Tool supports the automatic derivation of security requirements in terms of commitments (see Figure 27). The security requirements are listed and they can be sorted according to the various attributes. For instance, filtering the requirements with respect to the *responsible* actor, gives an idea of who are the actors responsible (debtor making the commitment) to satisfy the security requirements (commitments). On the other hand, filtering requirements according to their *requirement type* groups together security requirements (commitments) that need to be satisfied to fulfill a certain security policy.
    - *Step 5.1. Generate security requirements document:* the modeling process terminates with the generation of a security requirements document, which supports the communication between the security requirements engineer and stakeholders. This document is customizable by choosing among a number of model features to include in the report (e.g., including only a subset of the actors, concepts or relations he or she wants more information about). The diagrams are explained in detail providing textual and tabular descriptions of the models (see Figure 28 for an excerpt of the security requirements document).



Responsible	Requirement	Requester
IS Manager	sender-integrity(transmitted(IS Manager,Secretary,Income Statement))	Secretary
IS Manager	sender-confidentiality(transmitted(IS Manager,Secretary,Income Statement))	Secretary
IS Manager	delegatee-authentication(delegated(Secretary,IS Manager,Obtain UTD Statement))	Secretary
IS Manager	single-actor-true-redundancyy(Obtain UTD Statement))	Secretary
IS Manager	availability(Obtain UTD Statement, 90%)	Secretary
IS Manager	delegatee-authentication(delegated(Secretary,IS Manager,Get Student Pers. Data))	Secretary
IS Manager	need-to-know({PersonalData, FinancialStatus},{Get Student Pers. Data})	Student
Program Coordinator	no-delegation(Write doc for I.O))	Student
Program Coordinator	non-repudiation-of-delegation(delegated(Program Coordinator,Secretary,Get Student Record))	Secretary
Program Coordinator	non-repudiation-of-delegation(delegated(Program Coordinator,Secretary,Write new doc))	Secretary
Program Coordinator	need-to-know({FinancialStatus, PersonalData},{Write doc for I.O})	Student
Program Coordinator	non-production({FinancialStatus, PersonalData})	Student
Secretary	trustworthiness(Secretary, delegated(Program Coordinator,Secretary,Get Student Record))	Program Coordinator
Secretary	no-delegation(Write new doc))	Program Coordinator
Secretary	recivier-confidentiality(transmitted(IS Manager,Secretary,Personal data file))	IS Manager

Description

Student requires no-delegation for goal Write doc for I.O,when delegating Write doc for I.O to Program Coordinator.

Figure 27: Security Requirements for the running example

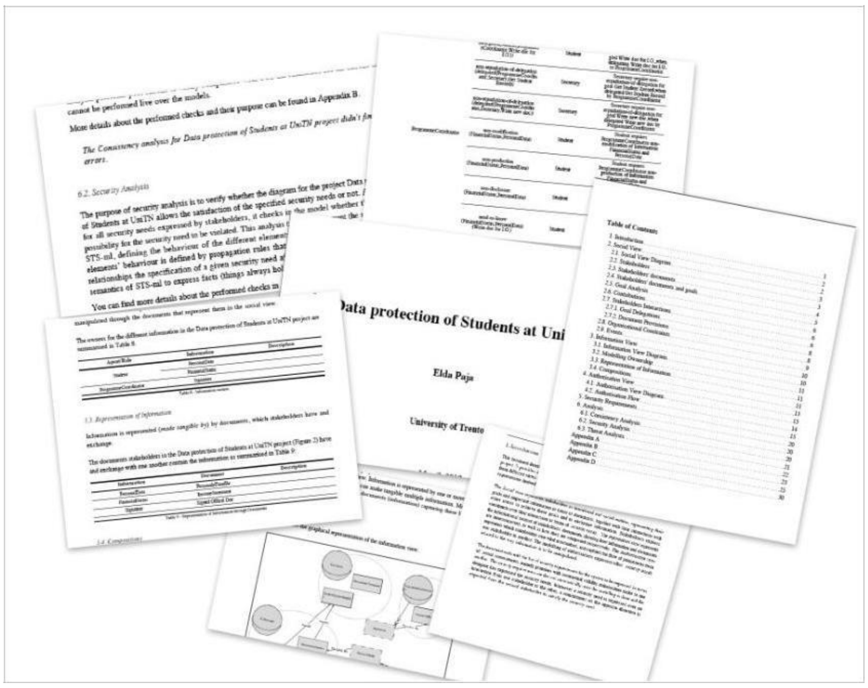
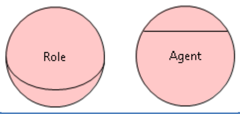
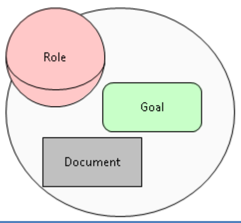
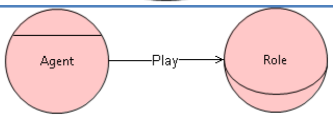
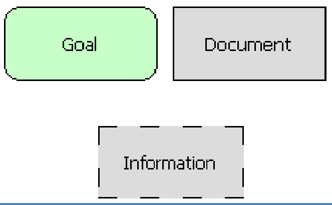
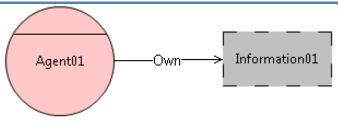
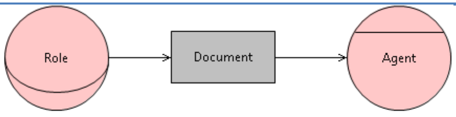
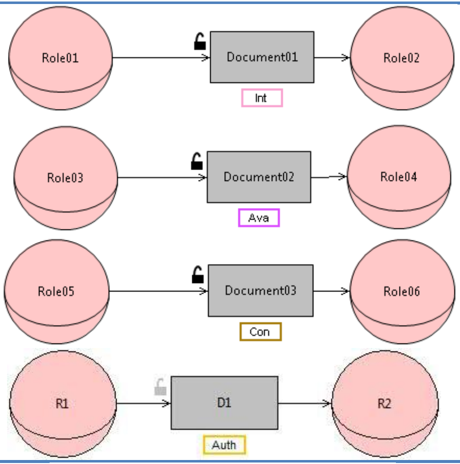
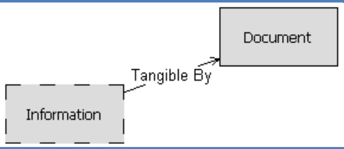
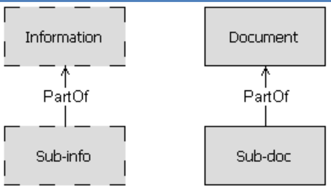
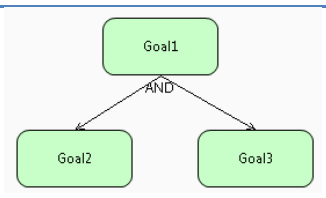
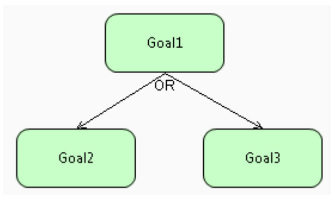
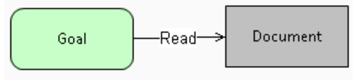
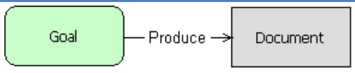
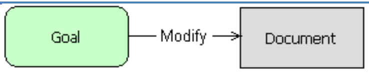
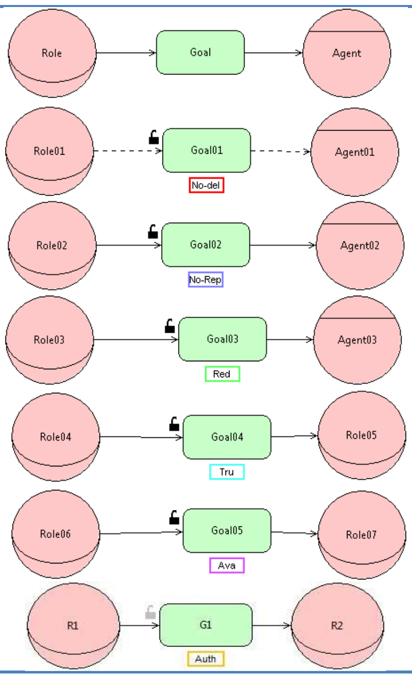


Figure 28: Security Requirements Document for the running example

5 Summary and discussion

Table 3 summarizes all the concepts that have been introduced throughout this document, together with their graphical representation.

CONCEPT	GRAPHICAL NOTATION	EXPLANATION
Role **Agent**		A role is an abstract characterization of an actor (e.g., professor, student), while an agent refers to a concrete actor (e.g., Marco, Laura).
Role/Agent scope **Possess** **Want**		The scope of a role (agent) defines the strategic construction of the role(agent): • The goals the role/agent wants to achieve; • The documents the role/agent possesses.
Play		An agent plays a role (e.g., Marco plays role Student, Laura plays role Teacher).
Goal **Document** **Information**		A goal represents a desired state of affairs (e.g.., car is bought, stay permit is given). A document represents an exchangeable entity (e.g., reference letter), which may contain some information (e.g., personal data, salary).
Own		A role/agent is the legitimate owner of some information and can freely dispose of it, as well as decide to transfer rights about it to others (e.g.., a student is the owner of her personal data).
Transmit		A role/agent transmits a document that she possesses to another role/agent. For example, a student transmits his report to the professor.
Integrity of transmission, **Availability,** **Confidentiality of transmission** **Authentication**	 Role01 Document01 Int Role02 Role03 Document02 Ava Role04 Role05 Document03 Con Role06 R1 D1 Auth R2	A role/agent requires integrity of transmission from the role/agent providing the document. The receiver requires a minimum level of availability for Document is preserved by the provider. Sender/Receiver requires authentication of the receiver/sender for getting/transmitting the document.
Tangible by		An information entity is made tangible by a document (e.g., the student's personal data is made tangible by a reference letter)

Part Of		An information entity (a document) is part of another information entity (another document). The part of relationship applies between homogeneous concepts. For example, the “date of birth” information is part of the “personal data” information. Also, the “letter header” is part of the “recommendation letter”.
AND-decomposition **OR-decomposition**	 	Decompositions enable the hierarchical refinement of goals. There are two types of decomposition: AND-decomposition, where the achievement of all the sub-goals implies the achievement of the decomposed goal. For instance, goal “reference letter written” may be AND-decomposed to “letter written” and “letter signed”. OR-decomposition, where the achievement of either sub-goal implies the achievement of the parent goal. For example, goal “letter written” can be OR-decomposed to “new letter written” or “old letter copied”.
Read		A document is read in order to achieve a goal. For example, document “letter template” is read to achieve goal “old letter copied”.
Produce		A document is produced while achieving a goal. For instance, document “reference letter” is produced while achieving goal “recommendation letter written”
Modify		The content of a document is modified while achieving a goal. For instance, document “health record” is modified while achieving goal “update patient’s health record”.
Goal delegation **No-redelegation** **Non-repudiation** **Redundancy** **Trustworthiness** **Availability** **Authentication**	 Role Goal Agent Role01 Goal01 No-del Agent01 Role02 Goal02 No-Rep Agent02 Role03 Goal03 Red Agent03 Role04 Goal04 Tru Role05 Role06 Goal05 Ava Role07 R1 G1 Auth R2	A goal delegation implies that a role/agent (delegator) delegates the fulfillment of a goal (delegatum) to another role/agent (delegatee). In STS-ml, we support some variants of delegation: No-delegation: the delegatee cannot further delegate the goal. Non-repudiation: the delegatee cannot repudiate that the delegation has taken place. Redundancy: the delegatee shall adopt measures so as to provide redundant fulfillment of a goal. Trustworthiness: the delegatee shall provide a proof of trustworthiness, e.g., issued by a certification authority. Availability: the delegatee shall ensure a minimum level of availability for the delegated goal. Authentication: the delegator/delegatee shall authenticate for delegating/getting the goal.

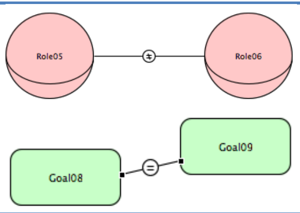
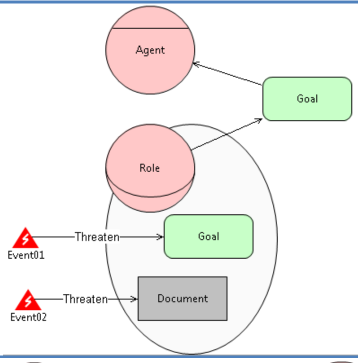
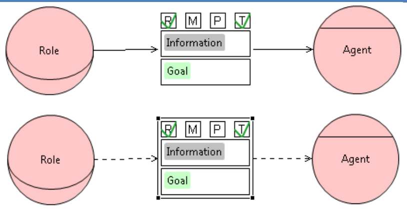
SoD Role-based Goal-based		Role-based BoD: defines bound roles. Goal-based BoD: defines goals that have to be achieved in combination.
Events	 Agent Goal Role Event01 Threaten Goal Event02 Threaten Document	An event threatens actors (agents and roles), goals, goal delegations and documents.
Authorisation	 Role R M P T Information Goal Agent Role R M P T Information Goal Agent	A role/agent authorizes another for certain operations to be performed on one or more information items, within the scope of a given goal. *Operations* are: Read (R), Modify (M), Produce (P), and Transmit (T). For example, reading could be allowed, while modification could not be granted. The *scope* is one or more goals. For example, a role may be authorized to read "personal data" in the scope of goal "recommendation letter written", but not in the scope of other goals (e.g., "bank payment made"). An authorization may deny transfer of authority (second picture). The role/agent receiving the authorization cannot re-transmit (parts of) the authorization.

Table 3: Socio-technical security modeling language: summary

6 References

[1] Elda Paja, Fabiano Dalpiaz, Paolo Giorgini (2013) *Managing Security Requirements Conflicts in Socio-Technical Systems*. In Proceedings of the 32nd International Conference on Conceptual Modeling (ER 2013), pp. 270-283, November 2013.

[2] Fabiano Dalpiaz, Elda Paja, Paolo Giorgini, *Security Requirements Engineering via Commitments*, In proceedings of the First Workshop on Socio-Technical Aspects in Security and Trust (STAST'11), pp. 1-8, September 2011.

[3] N. Zannone, "A Requirements Engineering Method for Trust, Security, and Privacy," Ph.D. dissertation, University of Trento, 2007.

[4] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modelling Security Requirements through Ownership, Permission and Delegation," in Proceedings of RE 2005. IEEE Computer Society, 2005, pp. 167–176

[5] F. Dalpiaz, A. K. Chopra, P. Giorgini, and J. Mylopoulos, "Adaptation in Open Systems: Giving Interaction its Rightful Place," in *Proceedings of ER 2010*, ser. LNCS, vol. 6412. Springer, 2010, pp. 31–45

[6] M. P. Singh. Agent Communication Languages: Rethinking the Principles. IEEE Computer, 31(12):40–47, December 1998

[7] M. P. Singh. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. Artificial Intelligence and Law, 7(1):97–113, 1999

[8] E. Paja, F. Dalpiaz, M. Poggianella, P. Roberti, P. Giorgini (2012), STS-Tool: Using Commitments to Specify Socio-Technical Security Requirements. In Proceedings of the 31st International Conference on Conceptual Modelling – Workshops (ER'12 Workshops). pp. 396-399.

[9] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, Tropos: An Agent-Oriented Software Development Method. Autonomous Agents and Multi-Agents Systems, 8(3): 203-236, 2004.

[10] STS Tool User Guide http://www.sts-tool.eu/doc/UserGuide_STS_Tool_ver1.3.3_web.pdf

[11] Aniketos EU FP7 project, http://www.aniketos.eu/ - D1.2 First Aniketos architecture and requirements specification

[12] L. Liu, E. Yu, and J. Mylopoulos. Security and privacy requirements analysis within a social setting. In Proc. of RE 2003, pages 151–161, 2003.

[13] H. Mouratidis and P. Giorgini. Secure Tropos: A security-oriented extension of the tropos method. International Journal of Software Engineering and Knowledge Engineering, 17(2):285–309, 2007.

[14] E. Bertino, S. Jajodia, and P. Samarati. A flexible authorisation mechanism for relational data management systems. ACM Transactions on Information Systems, 17(2):101–140, 1999.

[15] M. E. Whitman and H. J. Mattord. Principles of Information Security. Course Technology Press, 4th edition, 2011.

Appendix: Automated analyses supported by STS-Tool

A. Well-formedness Analysis

The purpose of well-formedness analysis is to verify whether the diagram built by the designer is consistent and valid. A diagram is considered to be well-formed if its constituent elements (concepts and relationships) are drawn and interconnected following the semantics of the modeling language (STS-ml). Thus, well-formedness analysis performs post checks to verify compliance with STS-ml semantics for all checks that cannot be performed live over the models. Currently, the following checks are performed:

- *Empty Diagram*
  This check verifies whether the given diagram is empty or not. If that is the case, then no other well-formedness checks are performed. If the diagram is not empty, the well-formedness analysis returns: "No errors found" and continues performing the rest of the checks.
- *Goal Single Decomposition*
  This check verifies the consistency of goal decompositions. Following the semantics of STS-ml a given goal is decomposed in two or more sub goals. As a result, the decomposition should specify at least two subgoals. Therefore, goal single decomposition verifies whether there are cases of decompositions to a single sub goal.
- *Delegation Child Cycle*
  This check verifies the well-formedness of goal delegations, so that no cycles or loops are identified as a result of the delegatee decomposing the delegatum (delegated goal) and re delegating back one of the subgoals. Delegation child cycle verifies exactly this and gives a warning in case of inconsistency.
- *Information No Ownership*
  This check verifies that all information have an owner. If there are cases of information without any ownership relationships from any actor in the diagram, the well-formedness analysis returns a warning.
- *Authorizations Validity*
  This check verifies that all authorization relationship between two given actors are valid. An authorization relationship specifies authorizations or permissions an actor grants to another on some information, to perform some allowed operations. The authorizations could be limited to a goal scope and they can be re-delegated or not. However, the first two attributes should be specified for an authorization relationship to be valid. If there are no information specified, the well-formedness analysis returns an error. The same applies to the cases, in which no allowed operations are specified.
- *Duplicate Authorizations*
  This check verifies that there are no duplicate authorization relationships that could be merged. There are several cases that are addressed by this check: (i) we encounter two identical authorization, i.e., between the same roles, in the same direction, for the same set of information, allowed operations and goals, and having the same value of transferability; (ii) identify authorization relationships between the same roles, in the same direction, in which one grants permissions that are subset of the other authorization's relationship.

B. Security Analysis

STS-ml allows for the specification of security needs over actors' interactions. It currently supports a non-exhaustive set of security needs and organizational constraints, namely non-repudiation, redundancy, no-redelegation, non-reading, non-modification, non-production, non-disclosure and need-to-know. The purpose of security analysis is to verify whether the drawn diagram allows the satisfaction of the specified security needs or not. As a result, for all security needs expressed by stakeholders, it checks in the model whether there is any possibility for the security need to be violated. This analysis takes into account the semantics of STS-ml, defining the behavior of the different elements represented in the models. The elements' behavior is defined by propagation rules that consider what concepts and what relationships the specification of a given security need affects. Datalog is used to define the semantics of STS-ml to express facts (things always hold) and rules. In the following are provided the details for all the checks performed during security analysis:

- *No-redelegation Violation*
  This violation is verified whenever a delegatee actor further delegates a goal, over the delegation of which a no-delegation security need is specified from the delegator actor. No delegation is specified over a goal delegation by the delegator, who requires the delegatee not to further delegate the delegated goal. Therefore, to check for any violations of no delegation, the analysis searches for redelegations of the delegatum (delegated goal) or any of its subgoals.
- *Redundancy Violation*
  This check verifies if redundancy is satisfied by controlling that single-actor-redundancy or multi-actor-redundancy are not violated. At design time we cannot make the distinction between fallback and true redundancy, so they cannot be verified at this stage. Therefore, both fallback redundancy single and true redundancy single are mapped to single actor redundancy.
  Similarly for multi-actor-redundancy; the analysis verifies a redundancy violation if one of the following occurs: (1) actor does not decompose the delegated goal in any or-subgoals, for which both types of redundancy are violated; (2) actor decomposes the goal into or-subgoals and delegates one to another actor when single actor redundancy has been specified, for which this type of redundancy is violated; (3) actor decomposes the goal into or-subgoals, but does not delegate any of the subgoals to another actor when multi actor redundancy has been specified, for which this type of redundancy is violated.
- *Pre-Analysis: Authorization Conflict*
- This task includes a set of checks that are run to verify that no conflicting authorizations are passed towards a given actor.
  - Authorization Conflict: This task identifies a conflict of authorization whenever at least two authorization relationships for the same information are drawn towards the same actor from two illegible actors (being the owner of information or another authorized actor) such that: (1) one limits the authorization to a goal scope (requiring a need-to-know security need) and the other does not (authorizing the actor without any limitations); (2) for the same goals or intersecting goal scopes, different permissions are granted in terms of operations or authority to transfer authorization. That is, one passes the actor the authority to perform operations (read, modify, produce, transmit) on a given information, and the other does not (requiring non-reading, non-modification, non-production, non-transmission); one passes the actor the authority to further transfer authorizations and the other requires no further authorizations take place.
- *Pre-Analysis: Operation Violation*
  This task includes a set of checks that verify that no unauthorized operations are performed by any actor.
  - *Non-disclosure Violation*: this violation is detected whenever an actor discloses information without having the right to transmit it. Non-disclosure expresses the need of not disclosing or further transmitting the given information to other actors, apart from the authorizer. Thus, authority to transmit the information is not passed. The way actors exchange information is through document transmission. In order to disclose some information, an actor would have to transmit to others the document(s) containing that information. Hence, to verify if there are any unauthorized disclosures of information, the analysis checks for transmissions of documents representing the given information from any unauthorized actors towards other actors.
  - *Non-reading Violation*: This violation is detected whenever there is an inappropriate modeling of the reading relationship. Transmission should encapsulate reading too. A carrier serving as a transmission point needs to have only transmission permissions, not necessarily any about reading what is being transferred.
  - *Non-modification Violation*: this violation is detected whenever an actor modifies information without having the right to modify it. Non-modification expresses the need that information should not be changed (modified), i.e. authority to modify the information is not granted. To verify if there could be any violations of non-modification, the analysis looks if the authorisee (or an actor that is not authorized by authorized party) modifies the given information. For this, it searches for modify relationships from any goal of this actor to any document representing the given information.

- *Non-production Violation*: this violation is detected whenever an actor produces information without having the right to produce it. Non-production expresses the need that information should not be produced in any form, i.e. authority to produce the information is not granted. To verify if there could be any violations of non-production, the analysis checks whether if the authorisee (or an actor that is not authorized by authorized party) produces the given information. For this, it searches for produce relationships from any goal of this actor to any document representing the given information.
- *NTK Violation*: this violation is detected whenever an actor reads, modifies or produces information for other purposes (goal achievement) than the ones for which it is authorized. Need-to-know requires that the information is used, modified, or produced in the scope of the goals specified in the authorization. This security need concerns confidential information, which should not be utilized for any other purposes other than the intended ones. To verify if there could be any violations of need-to-know, security analysis checks if the authorizee (or an actor that is not authorized by any authorized party) reads, modifies or produces the given information while achieving some goal different from the one it is authorized for. In a nutshell, it searches for need, modify, or produce relationships starting from goals different from the specified ones towards documents representing the given information.

Apart from the verification of violations of security needs, security analysis performs checks to verify that actors comply with their authorities. For this, it searches for eventual unauthorized passages of rights. For the time being, the following violations are detected:

- *Pre-Analysis: Authority Violation*
  This task includes a set of checks that verify that no actor transfers rights to others in an unauthorized way.
  - *Authority Violations*: verifies whether a given actor transfer rights to others even when it does not have the authority to further delegate rights.
  - *Unauthorized Delegation of Reading Violation*: verifies whether a given actors transfer to other actors the right to read a given information, without having itself the right to do so.
  - *Unauthorized Delegation of Modification violation*: verifies whether a given actors transfer to other actors the right to modify a given information, without having itself the right to do so.
  - *Unauthorized Delegation of Production violation*: verifies whether a given actors transfer to other actors the right to modify a given information, without having itself the right to do so.
  - *Unauthorized Delegation of Transmission violation*: verifies whether a given actors transfer to other actors the right to transmit a given information, without having itself the right to do so.
- *Pre-Analysis: Business Violation*
  This task includes a set of checks that verify that there are no violations of organizational constraints. As far as organizational constraints are concerned, security analysis verifies that the specification of SoD and BoD constraints can be satisfied in the given model.
  - Sod Violation:
    - Role-based: this violation is detected whenever a single actor plays both roles, among which an SoD constraint is expressed. Role-based SoD requires that no agent can play both roles, if it plays one of them.
    - Goal-based: this violation is detected whenever a single actor may perform both goals, between which an SoD constraint is expressed. Goal-based SoD requires that there is no actor performing both goals among which SoD is specified. To perform this verification, the analysis checks that the final performer of the given goals is not the same actor.
  - BoD Violation:
    - Role-based: this violation is detected whenever there is no agent to play both roles, among which a BoD constraint is expressed. Role-based BoD requires that the same agent plays both roles, if it plays one of them.
    - Goal-based: this violation is detected whenever a single actor may perform both goals, between which an Sod constraint is expressed. Goal-based SoD requires that there is no actor performing both goals among which SoD is specified. To perform this verification, the analysis checks that the final performer of the given goals is not the same actor.

C. Threat Analysis: threat propagation

Threat analysis focuses on events threatening goals or documents, and as such it propagates the effects over goal trees and goal/document relationships internal to the actor (read, modify, produce), as well as social relationships involving goals and documents (goal delegation, document transmission). Therefore, starting from the specified threats we identify the impact of these threats over the rest of the diagram. The analysis starts with the known events and propagates their impact over goal trees, documents and social relationships. The newly-discovered elements are treated as threatened elements. The analysis ends when no new elements are found.

The propagation rules are the following:

- If an event threatens a *goal*, then:
    - If the goal is an AND-subgoal of some other goal, then the second goal is considered to be threatened;
    - If the goal is delegated, then the threat is propagated to the goal of the delegator;
    - If the goal produces a document, then the threat is propagated to the document too.
- If an event threatens a *document*, then:
    - If the document needs to be read or modified by some goal, then the goal is considered to be threatened too;
    - If the document is transmitted to some other actor, then the receiver’s document is threatened too;
    - If the document is composed of other documents, then the threat is propagated to the parts of the document.