UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38123 Povo – Trento (Italy), Via Sommarive 5
http://www.disi.unitn.it

Identifying conflicts in security requirements with STS-ml

Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini

December 2012

Technical Report # DISI-12-041

Identifying Conflicts in Security Requirements with STS-ml

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

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

Abstract. Requirements are conflicting when there exist no system that satisfies them all. Conflicts often originate from clashing needs of different stakeholders. Security requirements are no exception to the rule; moreover, their violation leads to severe consequences, such as privacy infringement, which, in many countries, implies burdensome monetary sanctions. In large (security) requirements models, conflicts are hard or impossible to identify manually. In these cases, automated reasoning is necessary. In this paper, we propose a reasoning framework to detect conflicting security requirements as well as conflicts between security requirements and business policies. Our framework formalises the STS-ml requirements modelling language for socio-technical systems. These systems consist of mutually interdependent humans, organisations, and software. In addition to presenting the framework, we apply the it to a case study about e-Government, and we report on promising scalability results of our implementation.

Keywords: Security requirements; automated reasoning; requirements models

1 Introduction

Conflicting requirements are requirements that cannot be satisfied at the same time. Conflicts often occur because requirements come from multiple stakeholders that have inconsistent needs [15]. Conflicts affect security requirements too [3]: access to some information may be granted from one stakeholder, but prohibited from another. Also, security requirements can conflict with business policies: an actor's policy may specify to access some information, while no authorised is granted by the information owner.

Coping with such conflicts at requirements-time avoids designing and implementing a non-compliant and hard-to-change system. Unfortunately, security requirements models are often large, and cannot be effectively analysed manually. Ignoring conflicts is not an option: non-compliance may result in privacy laws infringements, loss of reputation, and burdensome sanctions. Automated reasoning has been proposed to detect conflicts between requirements [20,5,4,8,10], and security requirements [21,7].

Conflicting security requirements are critical in Socio-Technical Systems (STSs). An STS is a purposeful interaction among human, organisational, and technical actors. Each actor defines its individual policy, and expects others to comply with its security requirements. Being specified independently, policies and security requirements are likely to clash. When a conflict arises, an actor will inevitably violate either its

policy, or the security requirements it is requested to fulfil. Either case threatens the well-functioning of the STS, which depends on the proper interplay of the actors.

Many security requirements frameworks have been proposed (see [12] for a review). Since we are interested in STSs, our baseline is the STS-ml [1] security requirements modelling language for STSs. STS-ml represents an STS as a set of goal-oriented interacting actors, and it supports specifying a variety of security requirements between those actors. Practical experiences with STS-ml (see [19] and Sec. 2) have empirically evidenced that the resulting models are large and that they include conflicts that are difficult to identify manually.

In this paper, we propose a reasoning framework for STS-ml for detecting two families of conflicts: among security requirements, and between business policies and security requirements. 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. The contributions of the paper are:

- A formal framework for STS-ml for detecting conflicts by comparing (i) actions that actors may perform, based on their business policies; and (ii) expectations about (not) performing actions, based on security requirements;
- An implementation of the formal framework in Datalog (bundled in STS-Tool [14], the support tool for STS-ml), which shows promising scalability results;
- An experimentation on an industrial case study, which demonstrates the effectiveness of the reasoning techniques in identifying non-trivial conflicts in large models.

The rest of the paper is organised as follows. Sec. 2 presents our motivating case study about e-Government. Sec. 3 reviews our baseline: STS-ml. Sec. 4 introduces the formal framework for STS-ml. Sec. 5 describes the identification of conflicts, while Sec. 7 evaluates our framework on the case study and presents scalability results. Sec. 8 contrast our approach to related work, while Sec. 9 concludes.

2 Motivating Case Study: tax collection in Trentino

Trentino as a Lab (TasLab)[1] is an online collaborative platform to foster ICT innovation in the Trentino province [16]. Its aim is to create a community of research institutions, universities, enterprises and public administration, which collaborate in research-intensive ICT projects. TasLab provides information on local innovation trends, events, investment opportunities. It also offers an area where users can match innovation demand (from local government and municipalities) with innovation supply (by enterprises and research institutions), and they can collaboratively write project proposals.

We focus on a TasLab collaborative project about tax collection. The innovation demand comes from the Province of Trento (*PAT*) and the Trentino Tax Agency (*Trentino Riscossioni*), which require a system that verifies if correct revenues are gathered from individual (*Citizen*) and corporate (*Organisation*) taxpayers, provides a complete profile of taxpayers, generates reports, and enables online tax payments.

This is an example of an STS in which multiple actors interact via a technical system: citizens and organisations pay taxes online; municipalities (*Municipality*) furnish

[1] http://www.taslab.eu

information about citizens, addresses, and tax payments; Informatica Trentina (*InfoTN*) is the system contractor; other IT companies develop specific functionalities (e.g., data polishing, search modules); Trentino Riscossioni is the end user of the system; and PAT withholds the land register (information about buildings and lots).

These actors exchange confidential information and interact for processing such information. Each actor has its own business policy, i.e., goals achieved through processes that manipulate information, and expects others to comply with its security requirements, e.g., about integrity and confidentiality. Moreover, normative requirements apply to all actors. Different types of conflict may arise:

- Business policies can clash with security requirements. For instance, Trentino Riscossioni may authorise Informatica Trentina to use some data, but does not allow further distribution of such data. If Informatica Trentina's business policy includes relying upon an external provider to polish data, a conflict would occur;
- Security requirements can be conflicting. For instance, citizens may not want to authorise IT companies to access their personal data, while the municipality that possesses the citizen records may grant such authority;
- Normative requirements may conflict with other requirements. For instance, a local norm may prohibit private subjects from matching personal information about citizens with their tax records. This could create a conflict with the business policy of the company who polishes data, wherein such information is matched.

3 Baseline: STS-ml

STS-ml [1] is an actor- and goal-oriented security requirements engineering framework. As such, it includes high-level organisational concepts such as actor, goal, delegation, etc. Security requirements in STS-ml models are mapped to *social commitments* [17]—contracts among actors—that actors in the STS shall comply with at runtime. STS-ml modelling consists of three complementary views, so that different interactions among actors can be analysed by working on a specific perspective (view). Fig. 1 shows parts of the model for our case study (the full model is in Appendix A) .

The *social view* represents actors as intentional and social entities. Actors are intentional as they have goals they aim to attain, and they are social, for they interact with others by delegating goals and exchanging (providing) documents. Actors may possess documents, they may use, modify, or produce documents while achieving their goals. STS-ml supports two types of actors: agents, to refer to concrete participants, and roles, to refer to abstract actors (abstracted from agents, used when the actual participant is unknown). In our example, we represent Informatica Trentina (InfoTN) as agent, while TN Company Selector is modelled as a role, given that we do not know which party will take over this responsibility. InfoTN has goal online system built. Goals are refined through *AND/OR-decompositions*: online system built is AND-decomposed into system maintained, search module built and navig module built. InfoTN delegates search module built to TN Company Selector; it provides the document high quality data to Trentino Riscossioni.

The *information view* represents the owners of information, it gives a structured representation of actors' information and documents, and the way they are interconnected.

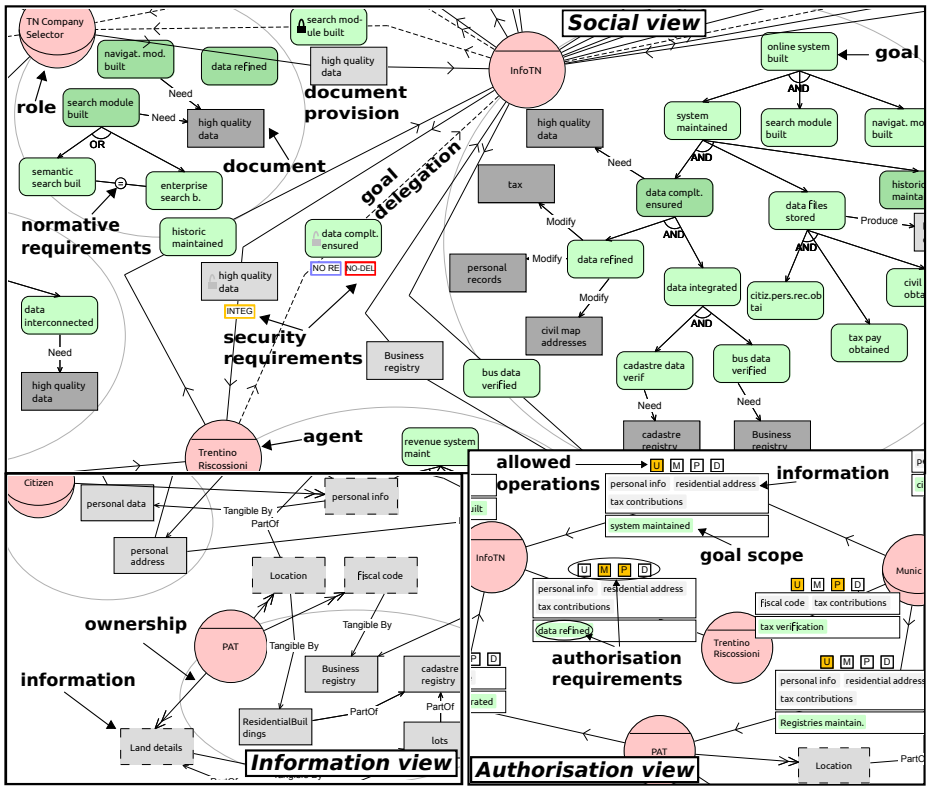

Fig. 1: Partial STS-ml model of the tax collection case study

This view helps determining how actors affect information while they manipulate documents to achieve their goals. Information can be represented by one or more documents (through the madeTangibleBy relationship), and on the other hand one or more information pieces can be part of some document. For instance, location and fiscal code are information owned by PAT; location is made tangible by residential buildings

The *authorisation view* shows the authorisations actors grant to others over information, either because they own it, or because they have been authorised to do so. In our example, Municipality authorises InfoTN to use information personal info, residential address, and tax contributions to have system maintained.

Through its three views, STS-ml supports different requirements types:

- *Business policies* are expressed by specifying actors, their goals, delegations, document provisions, and how actors manipulate documents to fulfil goals;
- *Interaction (security) requirements* are security-related constraints on delegations and provisions, e.g., non-repudiation, integrity of transmission, or redundancy;
- *Authorisation requirements* determine which information can be used, how, for which purpose, and by whom;
- *Normative requirements* constrain the adoption of roles and the uptake of responsibilities (separation / binding of duties, conflicting / combination of goals).

Together, interaction, authorisation, and normative requirements constitute the security requirements of STS-ml. The business policies of the actors shall comply with the security requirements. Security requirements are social relationships where an actor (*requester*) wants another actor (*responsible*) to comply with a requested property.

4 Formal framework

We define the formal framework for STS-ml that enables our automated reasoning techniques, and illustrate it on the model of Fig. 1. We employ the following notation: atomic variables are strings in italic with a leading capital letter (e.g., *G*, *I*); sets are strings in the calligraphic font for mathematical expressions (e.g., $\mathcal{G}$, $\mathcal{I}$); relation names are in sans-serif with a leading non-capital letter (e.g., wants, possesses); constants are in typewriter style with a leading non-capital letter (e.g., `and`, `or`). Due to space limitations, we do not define here atomic concepts and relations (e.g., goal, delegation).

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

- part-of-i(I_1, I_2)*: information* I_1 *is part of information* I_2*;*
- part-of-d(D_1, D_2)*: document* D_1 *is part of document* D_2*;*
- makes-tangible(I, D)*: document* D *materializes information* I*.* □

The information view in Fig. 1 includes, e.g., relationships makes- tangible(fiscal code, Business registry), and part-of-d(ResidentialBuildings, cadastre registry).

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

- decomposes$(A, G, \{G_1, \ldots, G_n\}, DecT)$*: A decomposes goal* G *into sub-goals* G_1 *to* G_n*, and the decomposition is of type DecT (*`and` *or* `or`*);*
- needs(A, G, D)*: A uses document* D *while achieving* G*;*
- modifies(A, G, D)*: A modifies document* D *while achieving* G*;*
- produces(A, G, D)*: A produces document* D *while achieving* G*;*
- capable-of(A, G)*: A is capable of achieving leaf-level goal* G *on its own;*
- possesses(A, D)*: A possesses document* D *(no other actor provides it to* A*).* □

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

- *IRL* $=$ decomposes$(A', G, \mathcal{S}, DecT) \rightarrow A' = A \wedge G \in \mathcal{G} \wedge \mathcal{S} \subset \mathcal{G}$
- *IRL* $=$ needs/modifies/produces$(A', G, D) \rightarrow A' = A \wedge G \in \mathcal{G}$
- *IRL* $=$ capable-of$(A', G) \rightarrow A' = A \wedge G \in \mathcal{G}$ □

An actor model defines the business policy of one actor. The social view of Fig. 1 includes an actor model where $A =$ InfoTN, $\mathcal{G}$ includes online system build, data refined, and so on, $\mathcal{IRL}$ includes decomposes(InfoTN, data complt. ensured, {data refined, data integrated}, and) and modifies(InfoTN, data refined, tax), and $T =$ `agent`.

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

- delegates(A_1, A_2, G)*: actor* A_1 *delegates goal* G *to actor* A_2*;*
- provides(A_1, A_2, D)*: actor* A_1 *provides document* D *to actor* A_2*;*
- authorises$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, \mathit{TrAuth})$*: actor* A_1 *authorises actor* A_2 *to perform operations* $\mathcal{OP}$ *on the information in* $\mathcal{I}$*, in the scope of the goals in* $\mathcal{G}$*, and allows (prohibits)* A_2 *to transfer the authorisation to others if TrAuth is true (false);*
- plays(Ag_1, R_2)*: agent* Ag_1 *plays role* R_2*;*
- owns(A_1, I_2)*: actor* A_1 *is the legitimate owner of information* I_2*.* □

Social relationships are modelled in the social and authorisation views. They define the social structure among the actors, i.e., relationships with validity in the modelled STS.

Def. 5 (Interaction requirement). *A property that an actor requires another to comply with, related to either a* delegates *or a* provides *social relationship between them.*
If Del = delegates(A_1, A_2, G)*:*

- r-not-repudiated-del(A_2, A_1, Del)*:* A_2 *requires* A_1 *not to repudiate the delegation;*
- r-not-repudiated-acc(A_1, A_2, Del)*:* A_1 *requires* A_2 *not to repudiate the acceptance of the delegation Del;*
- r-ts-red-ensured(A_1, A_2, G)*:* A_1 *requires* A_2 *to deploy concurrent redundant means for* G *(*ts-red = *true redundancy, single actor);*
- r-tm-red-ensured(A_1, A_2, G)*:* A_1 *requires* A_2 *to deploy concurrent redundant means for* G *involving at least another actor (*tm-red = *true redundancy, multiple actors);*
- r-fs-red-ensured(A_1, A_2, G)*:* A_1 *requires* A_2 *that, if the first strategy for* G *by* A_2 *fails,* A_2 *will deploy another strategy (*fs-red = *fallback redundancy, single actor);*
- r-fm-red-ensured(A_1, A_2, G)*:* A_1 *requires* A_2 *that, if the first strategy for* G *by* A_2 *(another actor* A_3*) fails,* A_3 *(*A_2*) will deploy another strategy (*fm-red = *fallback redundancy, multiple actors);*
- r-not-redelegated(A_1, A_2, G)*:* A_1 *requires* A_2 *to not redelegate* G*.*

If Prov = provides(A_1, A_2, Doc)*, then* r-integrity-ensured$(A_2, A_1, \mathit{Prov})$ *means that* A_2 *requires* A_1 *that the integrity of Doc is not compromised during its transmission.* □

Interaction requirements are security expectations that actors express on social relationships. In Fig. 1, Del_1 = delegates(Trentino Riscossioni, InfoTN, data complt. ensured) has two interaction requirements: r-not-repudiated-acc(Trentino Riscossioni, InfoTN, Del_1) and r-not-redelegated(Trentino Riscossioni, InfoTN, data complt. ensured).

Def. 6 (Normative requirement). *A property that the STS—here, intended as legal context—requires any participating actor* A *to comply with:*

- r-not-played-both$(\mathit{STS}, A, R_1, R_2)$*:* A *cannot play both roles* R_1 *and* R_2*;*
- r-not-pursued-both$(\mathit{STS}, A, G_1, G_2)$*:* A *cannot pursue both goals* G_1 *and* G_2*;*
- r-played-both$(\mathit{STS}, A, R_1, R_2)$*: if* A *plays role* R_1 *(*R_2*) shall also play* R_2 *(*R_1*);*
- r-pursued-both$(\mathit{STS}, A, G_1, G_2)$*: if* A *pursues goal* G_1 *(*G_2*),* A *should pursue* G_2 *(*G_1*) too.* □

Fig. 1 includes a normative requirement that imposes a combination of duties to any actor: r-pursued-both(STS, A, semantic search built, enterprise search b.).

Def. 7 (STS-ml model). *A tuple $M = \langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{IRQ}, \mathcal{NRQ} \rangle$ where $\mathcal{AM}$ is a set of actor models, $\mathcal{SR}$ is a set of social relationships, IKB is an informational knowledge base, $\mathcal{IRQ}$ is a set of interaction requirements, and $\mathcal{NRQ}$ is a set of normative requirements. An STS-ml model is valid iff:*

- *social relationships are only over actors with models in $\mathcal{AM}$;*
- *delegations are consistent:* $\forall \mathsf{delegates}(A_1, A_2, G) \in \mathcal{SR} \rightarrow \exists \langle A_1, \mathcal{G}_1, \mathcal{IRL}_1, T_1 \rangle, \langle A_2, \mathcal{G}_2, \mathcal{IRL}_2, T_2 \rangle \in \mathcal{AM}.\ G \in \mathcal{G}_1 \ \wedge \ G \in \mathcal{G}_2$*;*
- *provisions are consistent:* $\forall \mathsf{provides}(A_1, A_2, D) \in \mathcal{SR} \ \rightarrow \exists \langle A_1, \mathcal{G}, \mathcal{IRL}, T \rangle \in \mathcal{AM}.\ \mathsf{possesses}\ (D) \in \mathcal{IRL} \ \vee \ \exists$ *a consistent* $\mathsf{provides}(A_3, A_1, D) \in \mathcal{SR}$*;*
- *normative requirements are over roles with models in $\mathcal{AM}$ and their goals.* □

An STS-ml model is constructed from all the elements in all the views. A valid STS-ml model obeys to additional constraints. The STS-ml model sketched in Fig. 1 is valid. Note that STS-Tool does not allow creating invalid STS-ml models.

Def. 8 (Authorisation completion). *Let $M = \langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{IRQ}, \mathcal{NRQ} \rangle$ be a valid STS-ml model. The authorisation completion of $\mathcal{SR}$, denoted as $\Delta_{\mathcal{SR}}$, is a superset of $\mathcal{SR}$ that makes prohibitions explicit. Formally, $\forall A_1, A_2$ with an actor model in $\mathcal{AM}$,* $\forall \mathsf{owns}(A_1, I) \in \mathcal{SR}$. $\nexists \mathsf{authorises}(A_3, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, TrAuth) \in \mathcal{SR} \wedge I \in \mathcal{I} \rightarrow \mathsf{authorises}(A_1, A_2, I, \mathcal{G}_{A_2}, \emptyset, \mathtt{false}) \in \Delta_{\mathcal{SR}}$*, where $\mathcal{G}_{A_2}$ is the set of goals of A_2.* □

Def. 8 formalises the intuition that, if an actor A_2 has no incoming authorisation for information I, A_2 has a prohibition for I. Such prohibition is an STS-ml authorisation from the information owner that allows performing no operation and prohibits transferring authorisations. In Fig. 1, the lack of incoming authorisations to Trentino Riscossioni for information land details, implies authorises(PAT, Trentino Riscossioni, land details, $\mathcal{G}, \emptyset, \mathtt{false}) \in \Delta_{\mathcal{SR}}$, where $\mathcal{G}$ is the set of goals of Trentino Riscossioni.

Def. 9 (Authorisation requirement). *A requirement derived from an authorisation Auth* $= \mathsf{authorises}(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, TrAuth) \in \Delta_{\mathcal{SR}}$ *as follows:*

- $\mathcal{G} \neq \emptyset \ \rightarrow \ \mathsf{r\text{-}not\text{-}ntk\text{-}violated}(A_1, A_2, \mathcal{I}, \mathcal{G})$*, where $\forall I \in \mathcal{I}$, documents that make I tangible can be used / modified / produced by A_2 only for goals in $\mathcal{G}$;*
- $\mathtt{U} \notin \mathcal{OP} \ \rightarrow \ \mathsf{r\text{-}not\text{-}used}(A_1, A_2, \mathcal{I})$, $\mathsf{r\text{-}not\text{-}reauthorised}(A_1, A_2, \mathcal{I}, \mathcal{G}, \{\mathtt{U}\})$*: A_2 cannot use documents that include information in $\mathcal{I}$, or authorise others;*
- $\mathtt{M} \notin \mathcal{OP} \ \rightarrow \ \mathsf{r\text{-}not\text{-}modified}(A_1, A_2, \mathcal{I})$, $\mathsf{r\text{-}not\text{-}reauthorised}(A_1, A_2, \mathcal{I}, \mathcal{G}, \{\mathtt{M}\})$*: A_2 cannot modify documents that include information in $\mathcal{I}$, or authorise others;*
- $\mathtt{P} \notin \mathcal{OP} \ \rightarrow \ \mathsf{r\text{-}not\text{-}produced}(A_1, A_2, \mathcal{I})$, $\mathsf{r\text{-}not\text{-}reauthorised}(A_1, A_2, \mathcal{I}, \mathcal{G}, \{\mathtt{P}\})$*: A_2 cannot produce documents that include information in $\mathcal{I}$, or authorise others;*
- $\mathtt{D} \notin \mathcal{OP} \ \rightarrow \ \mathsf{r\text{-}not\text{-}disclosed}(A_1, A_2, \mathcal{I})$, $\mathsf{r\text{-}not\text{-}reauthorised}(A_1, A_2, \mathcal{I}, \mathcal{G}, \{\mathtt{D}\})$*: A_2 cannot provide to other actors any document that includes information in $\mathcal{I}$, or authorise others;*
- *TrAuth* $= \mathtt{false} \ \rightarrow \ \mathsf{r\text{-}not\text{-}reauthorised}(A_1, A_2, \mathcal{I}, \mathcal{G}, \{\mathtt{U}, \mathtt{M}, \mathtt{P}, \mathtt{D}\})$*: A_2 cannot transfer any permission on $\mathcal{I}$ and for $\mathcal{G}$ to other actors.*

We denote the set of authorisation requirements for Auth as $\mathcal{ARQ}_{Auth}$, and the set of authorisation policies for an actor A as $\mathcal{AARQ}_A$. □

In STS-ml, authorisation requirements are specified implicitly by modelling authorisations between actors. In Fig. 1, the authorisation from Trentino Riscossioni to InfoTN implies, for instance, requirements about r-not-ntk-violated (due to the non-empty goal scope), r-not-used and r-not-disclosed (no authorisation on those operations is granted).

Table 1: Security requirements and their verification against a variant $\mathcal{V}_M$. Del = delegates(A_1, A_2, G); $Prov$ = provides(A_1, A_2, D)

Requirement	**Verification at design-time**
Interaction requirements	
R_1 : r-not-repudiated-del(A_2, A_1, Del)	No
R_2 : r-not-repudiated-acc(A_1, A_2, Del)	No
R_3 : r-ts-red-ensured(A_1, A_2, G) R_4 : r-fs-red-ensured(A_1, A_2, G)	Partial. A_2 pursues goals in $\mathcal{V}_M$ that define at least two disjoint ways to support G
R_5 : r-tm-red-ensured(A_1, A_2, G) R_6 : r-fm-red-ensured(A_1, A_2, G)	Partial. Both A_2 and another actor A_3 support G, each in a different way
R_7 : r-not-redelegated(A_1, A_2, G)	$\nexists$delegates$(A_2, A_3, G') \in \mathcal{V}_M$. $G' = G$ or G' is a subgoal of G
R_8 : r-integrity-ensured$(A_2, A_1, Prov)$	No
Authorisation requirements	
R_9 : r-not-ntk-violated$(A_1, A_2, \mathcal{I}, \mathcal{G})$	$\nexists$needs/modifies/produces$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$ and $G \notin \mathcal{G}$
R_{10} : r-not-used$(A_1, A_2, \mathcal{I})$	$\nexists$needs$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{11} : r-not-modified$(A_1, A_2, \mathcal{I})$	$\nexists$modifies$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{12} : r-not-produced$(A_1, A_2, \mathcal{I})$	$\nexists$produces$(A_2, G, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{13} : r-not-disclosed$(A_1, A_2, \mathcal{I})$	$\nexists$provides$(A_2, A_3, D) \in \mathcal{V}_M$. D makes tangible (part of) $I \in \mathcal{I}$
R_{14} : r-not-reauthorised$(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP})$	$\nexists$authorises$(A_2, A_3, \mathcal{I}, \mathcal{G}, \mathcal{OP}') \in \mathcal{V}_M$. $\mathcal{OP}' \subseteq \mathcal{OP}$
Normative requirements	
R_{15} : r-not-played-both(STS, A, R_1, R_2)	$\{\text{plays}(A, R_1), \text{plays}(A, R_2)\} \nsubseteq \mathcal{V}_M$
R_{16} : r-played-both(STS, A, R_1, R_2)	$\{\text{plays}(A, R_1), \text{plays}(A, R_2)\} \subseteq \mathcal{V}_M$
R_{17} : r-not-pursued-both(STS, A, G_1, G_2)	A is not the final performer for both G_1 and G_2 or their subgoals
R_{18} : r-pursued-both(STS, A, G_1, G_2)	A is the final performer for both G_1 and G_2 or their subgoals

5 Detecting conflicts in security requirements

STS-ml models represent an analyst's *knowledge* about an STS. At design-time, the analyst can rely upon such knowledge to analyse the models. While there is no guarantee

that the agents will act as in the model, analysis still helps to identify potential conflicts. We use the framework of Sec. 4 to detect conflicts among authorisations (Sec. 5.1), and those between business policies and security requirements (Sec. 5.2). We provide examples of both types of conflicts obtained from the case study in Sec. 7.1.

5.1 Conflicts among authorisations

Before reasoning on conflicts between business policies and security requirements (interaction, authorisation, and normative requirements), we need to ensure that there are no conflicts among authorisations, i.e., that the authorisations are *consistent*. Inconsistent authorisations are ambiguous, as they include concurrent authorisations and prohibitions. Conflict resolution techniques (e.g., [18]) may be used to take a decision.

Def. 10 (Authorisation conflict). *Two authorisations* $Auth_1, Auth_2 \in \Delta_{\mathcal{SR}}$*, where* $Auth_1 = \mathsf{authorises}(A_1, A_2, \mathcal{I}_1, \mathcal{G}_1, \mathcal{OP}_1, CT_1)$ *and* $Auth_2 = \mathsf{authorises}(A_3, A_2, \mathcal{I}_2, \mathcal{G}_2, \mathcal{OP}_2, CT_2)$*, are conflicting (*$\mathsf{a\text{-}conflict}(Auth_1, Auth_2)$*) if* $\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$ *and either:*

1. $\mathcal{G}_1 \neq \emptyset \ \wedge \ \mathcal{G}_2 = \emptyset$ *, or vice versa; or,*
2. $\mathcal{G}_1 \cap \mathcal{G}_2 \neq \emptyset$*, and either (i)* $\mathcal{OP}_1 \neq \mathcal{OP}_2$*, or (ii)* $CT_1 \neq CT_2$*.* □

An authorisation conflicts occurs if both authorisations apply to the same information, and either (1) one authorisation restricts the permission to a goal scope, while the other does not (thus, one implies an r-not-ntk-violated requirement, while the other permits usage for any purpose); or, (2) the scopes are intersecting, and different permissions are granted (operations, and authority to transfer the authorisation). An authority-consistent STS-ml model (Def. 11) is a valid STS-ml model where no authorisation conflicts exist.

Def. 11 (Authority-consistent STS-ml model). *A valid STS-ml model* $M = \langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{IRQ}, \mathcal{NRQ} \rangle$ *such that* $\nexists Auth_1, Auth_2 \in \Delta_{\mathcal{SR}}$. $\mathsf{a\text{-}conflict}(Auth_1, Auth_2)$. □

5.2 Conflicts between business policies and security requirements

Given an authorisation-consistent STS-ml model, we verify if any security requirement is violated by the business policies of the actors. Such conflicts occur if (1) actors do some action they are required not to do, or (2) actors do not do something they are required to do. STS-ml models include the necessary information to check these conflicts:

- Intentional or social relationships define the actions an actor can possibly do (its business policy). For instance, given $AM = \langle A, \mathcal{G}, \mathcal{IRL}, T \rangle$, if $\mathsf{needs}(A, G, D) \in \mathcal{IRL}$, then A may possibly execute the action of using the document D to achieve G. Similarly, $\mathsf{delegates}(A_1, A_2, G)$ implies that A_1 may possibly execute the action of delegating the fulfillment of G to A_2;
- Security requirements imply commitments about (not) performing certain actions. For instance, $\mathsf{r\text{-}played\text{-}both}(STS, A, R_1, R_2)$ implies a commitment for A to execute the actions of playing both R_1 and R_2, while $\mathsf{r\text{-}not\text{-}modified}(A_1, A_2, \mathcal{I})$ implies a commitment for A_2 to not execute any $\mathsf{modifes}(A, G, D)$, where D makes tangible some $I \in \mathcal{I}$.

An STS-ml model does not explicitly specify the exact course of actions that the involved actors carry out to achieve their goals. We introduce the notion of a *variant* for an STS-ml model (see Def. 12) to denote a set of actions that the actors carry out to achieve all their root goals. These actions correspond to intentional relationships (needs, modifies, produces), social relationships (delegates, provides, authorises), and the pursues(A, G) action, telling that actor A pursues (intends to achieve) goal G.

Def. 12 (STS-ml variant). *Given an authorisation-consistent STS-ml model $M = \langle \mathcal{AM}, \mathcal{SR}, IKB, \mathcal{IRQ}, \mathcal{NRQ} \rangle$, a variant of M (denoted as $\mathcal{V}_M$) is a set of actions such that all the actors in M support their root goals. Formally:*

1. $\alpha \neq \mathsf{pursues}(\ldots) \in \mathcal{V}_M \leftrightarrow \alpha \in \mathcal{SR} \vee \exists \langle A, \mathcal{G}, \mathcal{IRL}, T \rangle \in \mathcal{AM}.\ \alpha \in \mathcal{IRL}$
2. $\forall \langle A, \mathcal{G}, \mathcal{IRL}, T \rangle \in \mathcal{AM}$:
   (a) $\forall G \in \mathcal{G}.$ *G is a root goal* $\rightarrow \mathsf{pursues}(A, G) \in \mathcal{V}_M$
   (b) $\forall \mathsf{decomposes}(G, \{G_1, \ldots, G_n\}, \mathtt{and}) \in \mathcal{IRL} \wedge \mathsf{pursues}(A, G) \in \mathcal{V}_M \rightarrow \mathsf{pursues}(A, G_1) \in \mathcal{V}_M \wedge \ldots \wedge \mathsf{pursues}(A, G_n) \in \mathcal{V}_M$
   (c) $\forall \mathsf{decomposes}(G, \mathcal{S}, \mathtt{or}) \in \mathcal{IRL} \wedge \mathsf{pursues}(A, G) \in \mathcal{V}_M \rightarrow \exists G_i \in \mathcal{S}.\ \mathsf{pursues}\ (A, G_i) \in \mathcal{V}_M$
   (d) $\forall G \in \mathcal{G}.\ \mathsf{pursues}(A, G) \in \mathcal{V}_M$:
      i. $\forall \alpha = \mathsf{delegates}(A, A', G) \in \mathcal{SR} \rightarrow \{\alpha, \mathsf{pursues}(A', G)\} \subseteq \mathcal{V}_M$
      ii. $\forall \alpha = \mathsf{needs/modifies/produces}(A, G, D) \in \mathcal{IRL} \rightarrow \alpha \in \mathcal{V}_M$
3. $\forall \alpha = \mathsf{authorises}(A_1, A_2, \mathcal{I}, \mathcal{G}, \mathcal{OP}, \mathcal{CT}) \in \mathcal{SR} \rightarrow \alpha \in \mathcal{V}_M$
4. $\forall \alpha = \mathsf{provides}(A_1, A_2, D) \in \mathcal{SR} \rightarrow \alpha \in \mathcal{V}_M$ □

Every action in the variant that does not refer to pursuing a goal shall appear in the STS-ml model (clause 1), i.e., the variant refers to that STS-ml model. For each actor model (clause 2), the actor pursues its root goals in the variant (clause 2(a)). If a pursued goal is and- (or-) decomposed, all (at least one) subgoals are pursued in the variant (clauses 2(b-c)). If a goal is pursued, and that goal is delegated to another actor (clause 2(d)i.), the delegation is in the variant and the delegatee pursues the goal in the variant. Need/produce/modify actions that relate to pursued goals are in the variant too (clause 2(d)ii.). All authorisations and provisions (clauses 3-4) are actions in the variant.

Def. 13 (Bus-Sec conflict). *Given a variant $\mathcal{V}_M$ for an STS-ml model M, there exists a conflict between business policies and security requirements iff:*

- *$\mathcal{V}_M$ contains one or more performed by A_2 that are forbidden by some requirement in $\mathcal{IRQ}$, $\mathcal{NRQ}$, or $\mathcal{AARQ}_{A_2}$ requested from some A_1 to A_2;*
- *$\mathcal{V}_M$ does not contain one or more actions performed by A_2 that are required by some requirement in $\mathcal{IRQ}$, $\mathcal{NRQ}$, or $\mathcal{AARQ}_{A_2}$ requested from some A_1 to A_2.* □

The second column of Table 1 describes semi-formally if and how security requirements can be verified at design-time. Below, we provide some more details.

Security requirements. R_1, R_2, and R_8 are verified at runtime, by checking actions that are not in STS-ml (e.g., repudiating a delegation). Redundancy requirements (R_3 to R_6) can be partially checked. While the existence of redundant alternatives can be verified,

a variant does not tell how alternatives are interleaved, i.e., if they provide true redundancy, fallback, or none. Thus, true redundancy and fallback are checked the same way. Single-agent redundancy (R_3 and R_4) is fulfilled if A_2 has at least two disjoint alternatives (via or-decompositions) for G. Multi-actor redundancy (R_5 and R_6) requires that at least one alternative involves another actor A_3. Not-redelegation (R_7) is verified if there is no delegation of G or its subgoals from A_2 to other actors in the variant.

Authorisation requirements. These prescribe actions that A_2 shall not perform in the variant. Need-to-know (R_9) is verified by the absence of needs, modifes, or produces actions on documents that make tangible some information in $\mathcal{I}$ for some goal G' that is not in $\mathcal{G}$ or in descendants of some goal in $\mathcal{G}$. Requirements R_{10} to R_{12} are verified if A_2 performs no needs, modifies, and produces action on documents that make tangible part of $I \in \mathcal{I}$, respectively. Non-disclosure (R_{13}) does a similar check but looking at document provisions. Non-reauthorisation (R_{14}) is fulfilled if A_2 does not authorise others to perform any operation in $\mathcal{OP}$ on $\mathcal{I}$ in the scope of $\mathcal{G}$.

Normative requirements. R_{15} and R_{16} require A to avoid playing or to play two roles through plays actions, respectively. R_{17} is verified if A is not the final performer[2] for both G_1 and G_2 or their subgoals. R_{18} is verified in a similar way, with the main difference that A has to be the final performer for both goals.

6 Reasoning about conflicts in STS-ml using Datalog

We have implemented our framework using Datalog, and it supports identifying conflicting authorisations as well as verifying the violation of security requirements. This implementation is integrated in STS-Tool, the modelling and analysis support tool for the socio-technical security modelling language. STS-ml models are drawn through the tool, to be then translated into Datalog textual files. Rules for the mapping each element of the model to a Datalog predicate have been specified in order to make the translation automatic. The DLV reasoner takes in input the generated STS-ml model files together the Datalog rules specifying the checks performed by the analysis to get the results. The results are parsed and visualized over the STS-ml models.

In the following we present the Datalog rules for identifying conflicts, together with the general rules necessary for defining the propagation of properties as well as for capturing actors' business requirements.

Listing 1.1 presents the rules for the model's informational knowledge base, which define when a given actor possesses a certain document (rules 1-4): an actor possess a document that is within his model (has-in-scope) (1), it is not producing the document and no other actor is providing this document to him (2), the actor has a goal that produces the document and possesses such document being the first actor to create the document(3), and finally an actor possesses a document if it is provided the document by some other actor (4). Additionally, the rules specify ownership propagation over parts of information (rule 5), that is, an actor that owns a given information, owns also its constituent pieces of information.

[2] An actor that pursues a given goal using its capabilities

Listing 1.1: Informational Knowledge Base Rules

```
1. possesses(A,D) :- has_in_scope(A,D), 0=#count{G: produce(A
    ,D,G)}, 0=#count{A1: provides(A1,A,D)}.
2. possesses(A,D) :- produces(A,D,G), has(A,G).
3. provided(A1,A2,D) :- possesses(A1,D), provides(A1,A2,D),
    A1 != A2.
4. possesses(A2,D) :- provided(_,A2,D).
5. own(A,I1) :- own(A,I), partOfI(I1,I).
```

Listing 1.2 and 1.3 present the datalog rules for the verification of r-not-redelegated and r-redundancy-ensured respectively. This check will identify a conflict if there is a conflict in at least one variant of the considered STS-ml model.

Listing 1.2: Interaction Requirements Verification: No-redelegation

```
R1 : r-not-redelegated(A1,A2,Del)
1. violate_not_redelegated(A2,A1,G,Gi ) :- delegated(A1,A2,G)
    , not_redelegated(A1,A2,G), delegated(A2,_,Gi).
2. not_redelegated(A1,A2,G,Gi) :- not_redelegated(A1,A2,G),
    has(A2,G), is_refined(A2,G,Gi).
3. has(A,Gi ) :- has(A, G), and_dec(A,G), is_refined(A,G,Gi).
4. has(A,Gi ) v - has(A, Gi ) :- has(A,G), or_dec(A,G),
    is_refined(A,G,Gi).
5. -has(A,Gi) :- or-dec(A,G), 0=#count{Gi :is_refined(A,G,Gi)
    ,has(A,Gi)}.
6. -has(A,Gi) :- or-dec(A,G),1<#count{Gi :is_refined(A,G,Gi),
    has(A,Gi)}.
7. delegated(A1,A2,Gi ) :- has(A1,G), delegates(A1,A2,Gi).
8. has(A2,Gi ) :- delegated(_,A2,Gi).
9. subgoal(Gi,G,A) :- is_refined(A,G,Gi).
10. subgoal(G1,G2,A) :- subgoal(G1,G3,A), subgoal(G3,G2,A).
```

The verification of redundancy considers goal trees, being them composed of or-decompositions of and-decompositions, to be *pursued* by the actor. This means that only one variant is generated, since we cannot verify redundancy in case only one alternative is selected to accomplish the desired goal.

Listing 1.3: Interaction Requirements Verification: Redundancy

```
R2 : r-s-red-ensured(A1,A2,G)
1. violate_s_red(A2,A1,G) :- delegated(A1,A2,G),
    s_red_ensured(A1,A2,G), 1>=#count{Gi:or_dec(A2,G),
    is_refined(A2,G,Gi).
2. violate_s_red(A2,A1,G) :- delegated(A1,A2,G),
    s_red_ensured(A1,A2,G), or_dec(A,G), is_refined(A,G,Gi),
    delegated(A2,_,Gi).
3. has(A,Gi ) :- has(A,G), and_dec(A,G), is_refined(A,G,Gi).
4. has(A, Gi ) :- has(A, G), or_dec(A, G), is_refined(A, G,
    Gi ).
```

```
5. delegated(A1,A2,Gi ) :- has(A1,G), delegates(A1,A2,Gi).
6. has(A2,Gi ) :- delegated(_,A2,Gi).
7. subgoal(Gi,G,A) :- is_refined(A,G,Gi).
8. subgoal(G1,G2,A) :- subgoal(G1,G3,A), subgoal(G3,G2,A).

R3 : r-m-red-ensured(A1,A2,G)
1. violate_m_red(A2,A1,G) :- delegated(A1,A2,G),m_red_ensured
    (A1,A2,G), 1>=#count{Gi:or_dec(A2,G),is_refined(A2,G,Gi).
2. violate_m_red(A2,A1,G) :- delegated(A1,A2,G),m_red_ensured
    (A1,A2,G), 0=#count{A3:delegated(A2,A3,Gi),subgoal(Gi,G,
    A2).
3. has(A,Gi) :- has(A,G), and_dec(A,G), is_refined(A,G,Gi).
4. has(A,Gi) :- has(A,G), or_dec(A,G), is_refined(A,G,Gi).
5. delegated(A1,A2,Gi) :- has(A1,G), delegates(A1,A2,Gi).
6. has(A2,Gi) :- delegated(_,A2,Gi).
7. subgoal(Gi,G,A) :- is_refined(A,G,Gi).
8. subgoal(G1,G2,A) :- subgoal(G1,G3,A), subgoal(G3,G2,A).
```

Listing 1.4 introduces the authorisation rules, which are necessary to capure the transfer of authorisations from actor to actor. The owner of an information has full authority over the information (rules 1 and 2); whenever an actor authorises another to perform operations over information for the scope of some goal, it authorises the actor to perform operations over information while achieving subgoals of the authorised goals (rule 3), similarly for parts of information (rule 4); whenever a given authorisation is granted the predicate hasAuthority keeps track of an actor's authority to perform operations over a given information, in the scope of some goal, having the authority to transfer authoirsations or not (rule 5). Rules 6 to 13 define when an actor could use, modify, produce or distribute a given information as well as keep track of the authority the actor has to use, modify, produce or distribute. The authorisation scope limiting an authorisation to a goal scope defines for which goals the actor has authority to perform operations on the granted information. Rule 15 instead defines the goals that are outside an authorisation's scope. These rules lay the ground for the verification of authorisation requirements.

Rules 16 to 26 define the authority an actor has as authorised by an illegible actor, for each authorised operation the authorisee is granted to perform that operation (similarly for the transfer of authorisations), and for each operation that is not granted the authorisation for that operation is not passed. Making explicit these rules facilitates capturing conflicts among authorisations.

Listing 1.4: Authorisation Rules

```
1. hasAuthority(A,1,1,1,1,I,G,1) :- own(A,I), has(A,G).
2. hasAuthority(A,1,1,1,1,I,all_goals,1) :- own(A,I), 0=#
    count{G: has(A,G)}.
3. authorise(A1,A2,I,G1,U,M,P,Di,T) :- authorise(A1,A2,I,G,U,
    M,P,Di,T), subgoal(G1,G,A2).
4. authorise(A1,A2,I1,G,U,M,P,Di,T) :- authorise(A1,A2,I,G,U,
    M,P,Di,T), partOfI(I1,I).
```

```
5. hasAuthority(A2,U,M,P,Di,I,G,T):- authorise(A1,A2,I,G,U,M,
   P,Di,T).

6. can_use(A,I,D,G) :- has(A,G), need(A,D,G), madeTangibleBy(
   I,D).
7. has_authority_to_use(A,I) :- hasAuthority(A,1,_,_,_,I,_,_)
   .
8. can_modify(A,I,D,G) :- has(A,G), modify(A,D,G),
   madeTangibleBy(I,D).
9. has_authority_to_modify(A,I) :- hasAuthority(A,_,1,_,_,I,_
   ,_).
10. can_produce(A,I,D,G):- has(A,G), produce(A,D,G),
   madeTangibleBy(I,D).
11. has_authority_to_produce(A,I) :- hasAuthority(A,_,_,1,_,I
   ,_,_).
12. can_distribute(A,I,D):- provides(A,_,D), madeTangibleBy(I
   ,D).
13. has_authority_to_distribute(A,I) :- hasAuthority(A,_,_,_
   ,1,I,_,_).

14. scope_g(A,I,G) :- hasAuthority(A,_,_,_,_,I,G,_).
15. -scope_g(A,I,G) :- hasAuthority(A,_,_,_,_,I,G1,_), has(A,
   G), has(A,G1), G != G1, 0=#count{G2: hasAuthority(A,_,_,_
   ,_,I,G2,_), G2 = G}.

16. -has_authority_to_authorise(A,I) :- hasAuthority(A,_,_,_,
   _,I,_,0).
17. authorise_usage(A1,A2,I) :- authorise(A1,A2,I,_,1,_,_,_,_
   ).
18. -authorise_usage(A1,A2,I) :- authorise(A1,A2,I,_,0,_,_,_,
   _).
19. authorise_modification(A1,A2,I) :- authorise(A1,A2,I,_,_
   ,1,_,_,_).
20. -authorise_modification(A1,A2,I) :- authorise(A1,A2,I,_,_
   ,0,_,_,_).
21. authorise_production(A1,A2,I) :- authorise(A1,A2,I,_,_,_
   ,1,_,_).
22. -authorise_production(A1,A2,I) :- authorise(A1,A2,I,_,_,_
   ,0,_,_).
23. authorise_distribution(A1,A2,I) :- authorise(A1,A2,I,_,_,
   _,_,1,_).
24. -authorise_distribution(A1,A2,I) :- authorise(A1,A2,I,_,_
   ,_,_,0,_).
25. authorise_transferibility(A1,A2,I) :- authorise(A1,A2,I,_
   ,_,_,_,_,1).
26. -authorise_transferibility(A1,A2,I) :- authorise(A1,A2,I,
   _,_,_,_,_,0).
```

Listing 1.5 defines the rules for identifying authorisation conflicts. For all actors, the incoming authorisations are considered and for every pair an authorisation conflict is detected whenever one of the authorisations grants performing an operation (authorise-usage, authorise-modification, authorise-production, and authorise-distribution, or grants the authority to further transfer authorisations through authorise-transferibility, whereas the other authorisation forbids either performing the operations or transfering authorisations.

Listing 1.5: Authorisation Conflicts Verification

```
1. authorisation_conflict(A2,I) :- authorise_usage(A1,A2,I),
    -authorise_usage(A3,A2,I).
2. authorisation_conflict(A2,I) :- authorise_modification(A1,
    A2,I), -authorise_modification(A3,A2,I).
3. authorisation_conflict(A2,I) :- authorise_production(A1,A2
    ,I), -authorise_production(A3,A2,I).
4. authorisation_conflict(A2,I) :- authorise_distribution(A1,
    A2,I), -authorise_distribution(A3,A2,I).
5. authorisation_conflict(A2,I) :- authorise_transferibility(
    A1,A2,I), -authorise_transferibility(A3,A2,I).
```

After detecting authorisation conflicts, the analysis verifies if there are any conflicts among business requirements and authorisation requirements. Listing 1.6 presents the rules for identifying these conflicts, grouping them by requirement. All the violations are propagated through the information structure (following the part of relationships).

Listing 1.6: Authorisation Requirements Verification

```
Need to know: r-not-ntk-violated(A1,A2,I,G)
1. violate_ntk(A2,I,G) :-  -scope_g(A2,I,G), used(A2,I,G),
    not violate_non_usage(A2,I,G).
2. violate_ntk(A2,I,G) :- -scope_g(A2,I,G), modified(A2,I,G),
     not violate_non_modification(A2,I,G).
3. violate_ntk(A2,I,G) :- -scope_g(A2,I,G), produced(A2,I,G),
     not violate_non_production(A2,I,G).
4. violate_ntk(A2,I1,G) :- violate_ntk(A2,I,G), partOfI(I1,I)
    .
5. violate_ntk(A2,I,G) :- violate_ntk(A2,I1,G), partOfI(I1,I)
    .

Non usage: r-not-used(A1,A2,I)
1. violate_non_usage(A2,I,G) :- not has_authority_to_use(A2,I
    ), used(A2,I,G).
2. used(A2,I,G) :- possess(A2,D), can_use(A2,I,D,G).
3. violate_non_usage(A2,I1,G) :- violate_non_usage(A2,I,G),
    partOfI(I1,I).
4. violate_non_usage(A2,I,G) :- violate_non_usage(A2,I1,G),
    partOfI(I1,I).
```

```
Non modification: r-not-modified(A1,A2,I)
1. violate_non_modification(A2,I,G) :- not
    has_authority_to_modify(A2,I), modified(A2,I,G).
2. modified(A2,I,G) :- possess(A2,D), can_modify(A2,I,D,G).
3. violate_non_modification(A2,I1,G) :-
    violate_non_modification(A2,I,G), partOfI(I1,I).
4. violate_non_modification(A2,I,G) :-
    violate_non_modification(A2,I1,G), partOfI(I1,I).

Non production: r-not-produced(A1,A2,I)
1. violate_non_production(A2,I,G) :- not
    has_authority_to_produce(A2,I), produced(A2,I,G).
2. produced(A2,I,G) :- can_produce(A2,I,D,G).
3. violate_non_production(A2,I1,G) :- violate_non_production(
    A2,I,G), partOfI(I1,I).
4. violate_non_production(A2,I,G) :- violate_non_production(
    A2,I1,G), partOfI(I1,I).

Non disclosure: r-not-disclosed(A1,A2,I)
1. violate_non_disclosure(A2,I,D) :- not
    has_authority_to_distribute(A2,I), distributed(A2,I,D).
2. distributed(A2,I,D) :- possess(A2,D), can_distribute(A2,I,
    D).
3. violate_non_disclosure(A2,I1,G) :- violate_non_disclosure(
    A2,I,G), partOfI(I1,I).
4. violate_non_disclosure(A2,I,G) :- violate_non_disclosure(
    A2,I1,G), partOfI(I1,I).
```

Listing 1.7 on the other hand, enumerates the rules for identifying all actors which violate their authorities, while reauthorising other actors: (1) without having the right to tranfer authorisations; (2) authorising others on operations they do not have themselves.

Listing 1.7: Unauthorised Reauthorisations

```
Authority violation: r-not-reauthorised(A1,A2,I,G,OP)
1. violate_del_of_authority(A1,A2,I) :- -
    has_authority_to_authorise(A1,I), authorise_usage(A1,A2,I
    ).
2. violate_del_of_authority(A1,A2,I) :- -
    has_authority_to_authorise(A1,I), authorise_modification(
    A1,A2,I).
3. violate_del_of_authority(A1,A2,I) :- -
    has_authority_to_authorise(A1,I), authorise_production(A1
    ,A2,I).
4. violate_del_of_authority(A1,A2,I) :- -
    has_authority_to_authorise(A1,I), authorise_distribution(
    A1,A2,I).
5. unauth_del_of_usage(A1,A2,I) :- not has_authority_to_use(
    A1,I), authorise_usage(A1,A2,I), not
    violate_del_of_authority(A1,A2,I).
```

```
6. unauth_del_of_mod(A1,A2,I) :- not has_authority_to_modify(
    A1,I), authorise_modification(A1,A2,I), not
    violate_del_of_authority(A1,A2,I).
7. unauth_del_of_prod(A1,A2,I) :- not
    has_authority_to_produce(A1,I), authorise_production(A1,
    A2,I), not violate_del_of_authority(A1,A2,I).
8. unauth_del_of_distr(A1,A2,I) :- not
    has_authority_to_distribute(A1,I), authorise_distribution
    (A1,A2,I), not violate_del_of_authority(A1,A2,I).
```

As far as organisational constraints are concerned, security analysis verifies whether the specification of r-not-played-both, rmbox−played-both, r-not-pursued-both, and r-pursued-both brings up conflicts with the actors business requirements. The analysis defines a final performer actor, and propagates the normative requirements over an actor's model and over social relationships it has with others, to identity conflicts.

Listing 1.8: Normative Requirements Verification

```
Role based separation of duty
1. - played(A,R2) :- sod_role(R1,R2), played(A,R1), role(R1),
     role(R2), R1!= R2.
2. - played(A,R1) :- sod_role(R1,R2), played(A,R2), role(R1),
     role(R2), R1!= R2.
3. violate_sod_role(A,R1,R2) :- sod_role(R1,R2), played(A,R1)
    , played(A,R2).

Goal rules
1. has(A,Gi) :- has(A,G), and_dec(A,G), is_refined(A,G,Gi).
2. has(A,Gi) :- has(A,G), or_dec(A,G), is_refined(A,G,Gi).
3. delegated(A1,A2,Gi) :- has(A1,G), delegates(A1,A2,Gi).
4. has(A2,Gi) :- delegated(_,A2,Gi).
5. subgoal(Gi,G,A) :- is_refined(A,G,Gi).
6. subgoal(G1,G2,A) :- subgoal(G1,G3,A), subgoal(G3,G2,A).

7. finalPerformer(R,G) :- has(R,G), 0=#count{R1: can_delegate
    (R,R1,G)}.
8. finalPerformer(R,G) :- has(R,G), can_delegate(R,R1,G), not
     delegated(R,R1,G).

Separation of duty: r-not-played-both(STS,A,R1,R2)
1. violate_sod_goal(A,R1,G1,R2,G2) :- sod_goal(G1,G2),
    finalPerformer(R1,G1), finalPerformer(R2,G2), play(A,R1),
     play(A,R2).
2. violate_sod_goal(R,R,G1,R,G2) :- sod_goal(G1,G2),
    finalPerformer(R,G1), finalPerformer(R,G2), 0=#count{A:
    play(A,R)}.
3. violate_sod_goal(A,A,G1,R,G2) :- sod_goal(G1,G2),
    finalPerformer(A,G1), finalPerformer(R,G2), agent(A),
    role(R), play(A,R).
```

```
4. sod_goal(Ga,G2) :- sod_goal(G1,G2), or_dec(R,G1),
    isRefined(R,G1,Ga), finalPerformer(R,Ga).
5. sod_goal(G1,Ga) :- sod_goal(G1,G2), or_dec(R,G2),
    isRefined(R,G2,Ga), finalPerformer(R,Ga).

Binding of duty: r-played-both(STS,A,R1,R2)
1. violate_cod_goal(A,R1,G1,R2,G2) :- cod_goal(G1,G2),
    finalPerformer(R1,G1), finalPerformer(R2,G2), agent(A),
    role(R1), role(R2), play(A,R2), not play(A,R1).
2. violate_cod_goal(A,R1,G1,R2,G2) :- cod_goal(G1,G2),
    finalPerformer(R1,G1), finalPerformer(R2,G2), agent(A),
    role(R1), role(R2) ,play(A,R1), not play(A,R2).
3. violate_cod_goal(R1,R1,G1,R2,G2) :- cod_goal(G1,G2),
    finalPerformer(R1,G1), finalPerformer(R2,G2), 0=#count{A:
     agent(A)}.
4. violate_cod_goal(R1,R1,G1,R2,G2) :- cod_goal(G1,G2),
    finalPerformer(R1,G1), finalPerformer(R2,G2), agent(A),
    not play(A,R1), not play(A,R2).
5. violate_cod_goal(A,A,G1,R,G2) :- cod_goal(G1,G2),
    finalPerformer(A,G1), finalPerformer(R,G2), agent(A),
    role(R), not play(A,R).
6. cod_goal(Ga,G2) :- cod_goal(G1,G2), or_dec(R,G1),
    isRefined(R,G1,Ga), finalPerformer(R,Ga).
7. cod_goal(G1,Ga) :- cod_goal(G1,G2), or_dec(R,G2),
    isRefined(R,G2,Ga), finalPerformer(R,Ga).
```

7 Evaluation

We evaluate our framework in two ways. One, we show its effectiveness in identifying conflicts by applying it to the case study about tax collection (Sec 7.1). Two, we assess its efficiency by reporting on scalability experiments with large models (Sec 7.2).

7.1 Findings from the case study

We first modelled the case study using STS-Tool (Fig. 1). Then, we used the tool's automated reasoning capabilities—based on a disjunctive datalog solver—to identify *authorisation conflicts*. The analysis returned a number of conflicts that we had not identified during the modelling, among which:

- Authority to produce: Trentino Riscossioni authorises InfoTN to produce information personal info, residential address and tax contributions to obtain refined data, whereas Municipality requires this information is only used, and not produced.
- Authority to modify: InfoTN grants Okkam Srl the authority to modify information personal info to obtain interconnected data, whereas TN Company Selector requires no document representing this information is modified.

These conflicts exist due to the different authorisation policies we elicited from the stakeholders. These conflicts, which went unnoticed at modelling time, became evident

after performing the reasoning. One possible strategy to resolve them is to consider the need for authorisation for the authorised party, and negotiate the necessary rights with the authorising parties. This way, the first conflict would be solved by negotiating with the Municipality. The second conflict, instead, can be fixed by informing InfoTN to revoke the authorisation, given that Okkam Srl does not need it (from the social view).

After fixing authorisation conflicts, we used the tool's capabilities to identify *Bus-Sec conflicts*. This activity provided us with further useful insights:

- r-not-redelegated: TN Company Selector relies on Okkam Srl to build a semantic search module (delegation of semantic search built). However, while relying on TN Company Selector, InfoTN wants this company to build the search modules, requiring it not to redelegate goal semantic search built. This interaction requirement is in conflict with the business policy about delegating semantic search built.
- r-not-modified: Engineering Tribute Srl makes an unauthorised modification of Citizen's personal info, violating the authorisation requirement r-not-modified specified by Citizen and passed on by TN Company Selector.
- r-not-produced: Citizen makes an unauthorised production of addresses, for this information is owned by the Municipality and no authorisation is granted to Citizen.
- r-not-reauthorised: Citizen wants only the Municipality to use and produce his personal info and does not allow transfer of authority, however the Municipality further authorises InfoTN to use this information.
- r-pursued-both: goals semantic search built and enterprise search b. should be pursued by the same actor, since a r-pursued-both normative requirement is specified between these goals. A conflict occurs because TN Company Selector is not the final performer for both goals (semantic search built is delegated to Okkam Srl).

The Bus-Sec conflicts that we identified mainly originate from the different policies of the companies in the province. Resolving these conflicts necessarily requires trade-off analysis [3], by comparing the importance of business policies for the stakeholders and the impact of relaxing the security requirements. Notice that relaxation is often not an option, especially if a requirement derives from norms in the legal context.

7.2 Scalability study

We performed a scalability study to assess the effectiveness of our automated reasoning, and to determine how well it would scale up to large models. To such extent, we investigate how the execution time is affected by the model size.

Design of experiments. We take the model in Fig. 1 as a basic building block, and clone it to obtain larger models. We increase the size of a model in two ways: first, we augment the *number of elements* (nodes and relationships) in the model; second, we increase the *number of variants* in the model. The latter is motivated by our reasoning techniques, which rely upon the generation of STS-ml model variants (Def. 12).

To obtain bigger models, we (1) create an identical copy (clone) of the given model; (2) add a fictitious leaf goal to a randomly chosen actor; (3) delegate this goal to the clone of the chosen actor; and (4) decompose the delegated goal in the cloned actor model into the root goal of his existing goal model and another fictitious goal. This process increases the number of variants, for the initial model contains variability.

We run tests on models with *zero*, *medium* and *high* variability, by customising the decomposition types in the original model. For each model, we run the analysis 7 times, discard the fastest and slowest executions, and compute the average execution time.

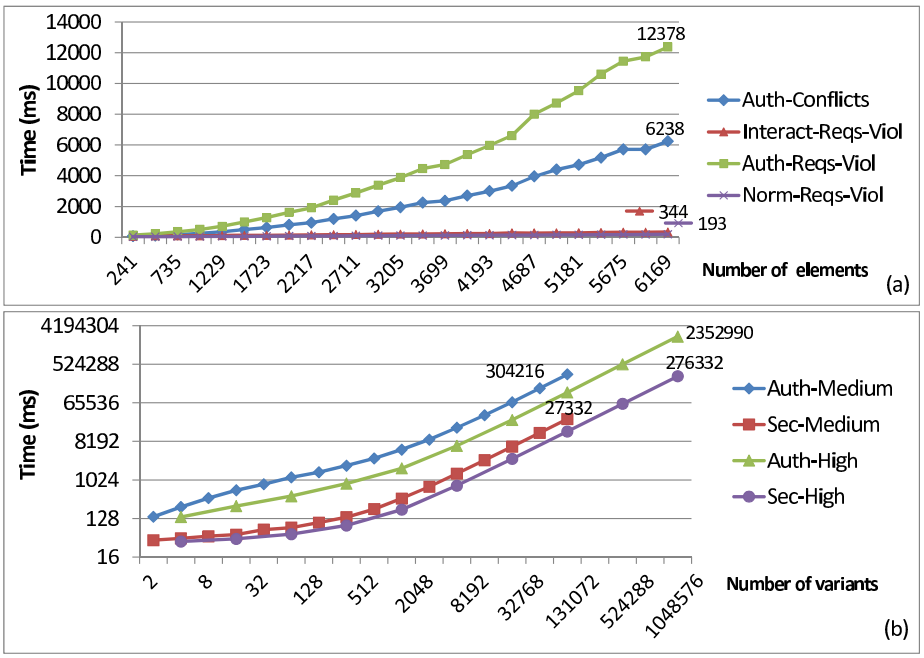

Fig. 2: Scalability analysis: increasing the number of elements (a) and variants (b)

Results. We have conducted experiments on a DELL Optiplex 780 machine, Pentium(R) Dual-Core CPU E5500 2.80GHz, 4Gb DDR3 399, powered by Windows 7. Fig. 2 summarises the results of our scalability experiments. Below, we detail the results and draw conclusions for the two scalability dimensions we have considered:

- *Number of elements* [Fig. 2(a)]: we present results for all the conflict types we can detect, i.e., authorisation conflicts, and violation of interaction, authorisation, and normative requirements. As noticeable by the plot, all techniques scale very well (linear growth). Furthermore, the tool is able to reason about extra-large models (>6000 elements) in about twelve seconds.
- *Number of variants* [Fig. 2(b)]: this dimension affects execution time the most. We show only violations of authorisation and interaction requirements; the other checks do not increase the number of variants. While the growth is still linear in the number of variants, it is exponential in the number of elements (the model with 1,048,576 variants consists of 2,500 elements). The reason why *medium* variability tests seem to have longer execution times than *high* is that, for a given number of variants, a

medium variability model contains twice the elements in a *high* variability model. Notice that the tool deals with dozens of thousands of variants in less than a minute.

The results are very promising, especially considering the fact that the size of real world scenarios is smaller than the extra-large models we produced with our cloning strategy.

8 Related work

We review related work about identifying conflicting requirements, reasoning about security requirements, and methodologies for security requirements engineering.

Conflicts between requirements. The importance of identifying conflicting requirements is well-known by practitioners and has been widely acknowledged by the research community [20,5]. Several formal frameworks have been proposed, especially in goal-oriented requirements engineering.

Giorgini et al. [8] use SAT solvers to analyse the satisfaction or denial of goals in goal models. They propose both qualitative and quantitative analysis techniques that determine evidence of goal satisfaction/denial by using label propagation algorithms. Conflicts are identified when propagation implies both positive and negative evidence. Their approach inspired further research. Horkoff and Yu [10] deal with conflicts in an interactive fashion, i.e., the analyst has to resolve conflicting sources of partial or conflicting evidence. Fuxman et al. [5] translate *i** models to Formal Tropos, and use first-order linear-time temporal logic to identify scenarios with conflicts. KAOS [20] includes analysis techniques to identify and resolve inconsistencies that arise from the elicitation of requirements from multiple stakeholders with different viewpoints.

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

Reasoning about security requirements. SI* [6] is a security requirements engineering framework that relies upon organisational concepts. It builds on *i** [22] and adds security-related concepts, among which delegation and trust of execution or permission. SI* uses automated reasoning to check security properties of a model, reasoning on the interplay between execution and permission of trust and delegation relationships. Our framework supports a wider set of security requirements (featuring sophisticated authorisations), and clearly separates security requirements from business policies.

De Landtsheer and van Lamsweerde [2] model confidentiality claims in terms of specification patterns, representing properties that unauthorised agents should not know. Their reasoning identifies violations of confidentiality claims in terms of counterexample scenarios present in requirements models. Diagnosis algorithms are used to generate the unauthorised agents reasoning to infer knowledge that is claimed to be confidential. While their approach represents confidentiality claims in terms of high-level goals, ours represents authorisation requirements as social relationships, and we identify violations by looking at the business policies of the actors.

Security requirements methodologies. These approaches provide methodological guidance to identify possible conflicts, as opposed to exploiting automated reasoning techniques. Secure Tropos [13] models security concerns throughout the whole develop-

ment process. The framework expresses security requirements as *security constraints*, considers potential threats and attacks, and provides methodological steps to validate these requirements and overcome vulnerabilities.

Liu et al. [11] extend *i** to deal with security and privacy requirements. Their methodology defines security and privacy-specific analysis mechanisms to identify potential attackers, derive threats and vulnerabilities, thereby suggesting countermeasures.

Haley et al. [9] propose a framework to determine adequate security requirements by constructing the context of the system, defining security requirements as constraints over functional requirements, and developing a structure of satisfaction arguments to verify the correctness of security requirements. This approach focuses mainly on system requirements, while ours is centred on the interaction among actors.

9 Conclusions

We have proposed a formal framework to detect conflicts in security requirements. Our framework formalises STS-ml [1], a security requirements modelling language for STS. The formal framework defines the semantics of the modelling language as well as that of the security requirements it can express (interaction security requirements, authorisation requirements, and normative requirements).

Based on such framework, we have shown how to detect two types of conflicts: (i) among authorisation requirements; and (ii) between business policies and security requirements. We have illustrated the effectiveness of our conflict identification techniques on an industrial case study, and we have reported on a scalability study that shows the efficiency of our framework even with very large models.

Additionally, the formal framework constitutes a theoretical foundation for extending the language, as well as to develop further analysis techniques. Our future work includes: (1) devising further reasoning techniques to identify inconsistencies among security requirements (so far, we identify inconsistencies only among authorisation requirements); and (2) exploring possible ways to resolve conflicts and inconsistencies.

Acknowledgments

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

References

1. F. Dalpiaz, E. Paja, and P. Giorgini. Security requirements engineering via commitments. In *Proc. of STAST'11*, pages 1–8, 2011.
2. R. De Landtsheer and A. Van Lamsweerde. Reasoning about confidentiality at requirements engineering time. In *Proc. of FSE'05*, pages 41–49, 2005.
3. G. Elahi and E. Yu. A goal oriented approach for modeling and analyzing security trade-offs. *Proc. of ER 2007*, pages 375–390, 2007.
4. N. A. Ernst, A. Borgida, J. Mylopoulos, and I. J. Jureta. Agile requirements evolution via paraconsistent reasoning. In *Proc. of CAiSE'12*, pages 382–397. 2012.
5. A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in tropos. In *Proc. of RE'01*, pages 174–181, 2001.
6. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling security requirements through ownership, permission and delegation. In *Proc. of RE'05*, pages 167–176, 2005.
7. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *Int. J. Inf. Sec.*, 5(4):257–274, 2006.
8. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Proc. of ER 2002*, pages 167–181, 2003.
9. C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.
10. J. Horkoff and E. Yu. Finding solutions in goal models: An interactive backward reasoning approach. *ER 2010*, pages 59–75, 2010.
11. 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.
12. D. Mellado, C. Blanco, L.E. Sánchez, and E. Fernández-Medina. A systematic review of security requirements engineering. *Computer Standards & Interfaces*, 32(4):153–165, 2010.
13. H. Mouratidis and P. Giorgini. Secure Tropos: A security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2):285–309, 2007.
14. E. Paja, F. Dalpiaz, M. Poggianella, P. Roberti, and P. Giorgini. STS-Tool: socio-technical security requirements through social commitments. In *Proc. of RE'12*, pages 331–332, 2012.
15. W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements interaction management. *ACM Computing Surveys (CSUR)*, 35(2):132–190, 2003.
16. P. Shvaiko, L. Mion, F. Dalpiaz, and G. Angelini. The taslab portal for collaborative innovation. In *Proc. of ICE 2010*, 2010.
17. M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
18. M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4):333–360, 1994.
19. S. Trösterer, E. Beck, F. Dalpiaz, E. Paja, P. Giorgini, and M. Tscheligi. Formative user-centered evaluation of security modeling: Results from a case study. *International Journal of Secure Software Engineering*, 3(1):1–19, 2012.
20. A. Van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.
21. A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26:978–1005, 2000.
22. E. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, Canada, 1996.

A Multi-view modelling of TasLab Case Study

We provide the complete model for the scenario extracted from the tax collection case study. We represent here the different views as modelled in STS-Tool for this case study. Fig. 3 represents the complete social view, which represents all the involved actors together with their interactions and captures the complete list of elicited interaction (security) needs; Fig. 4 represents the complete information view, capturing the informational content of the documents actors have and possess, as modelled in the social view. Finally, Fig. 5 shows all the authorisations passed from actor to actor in this case study.

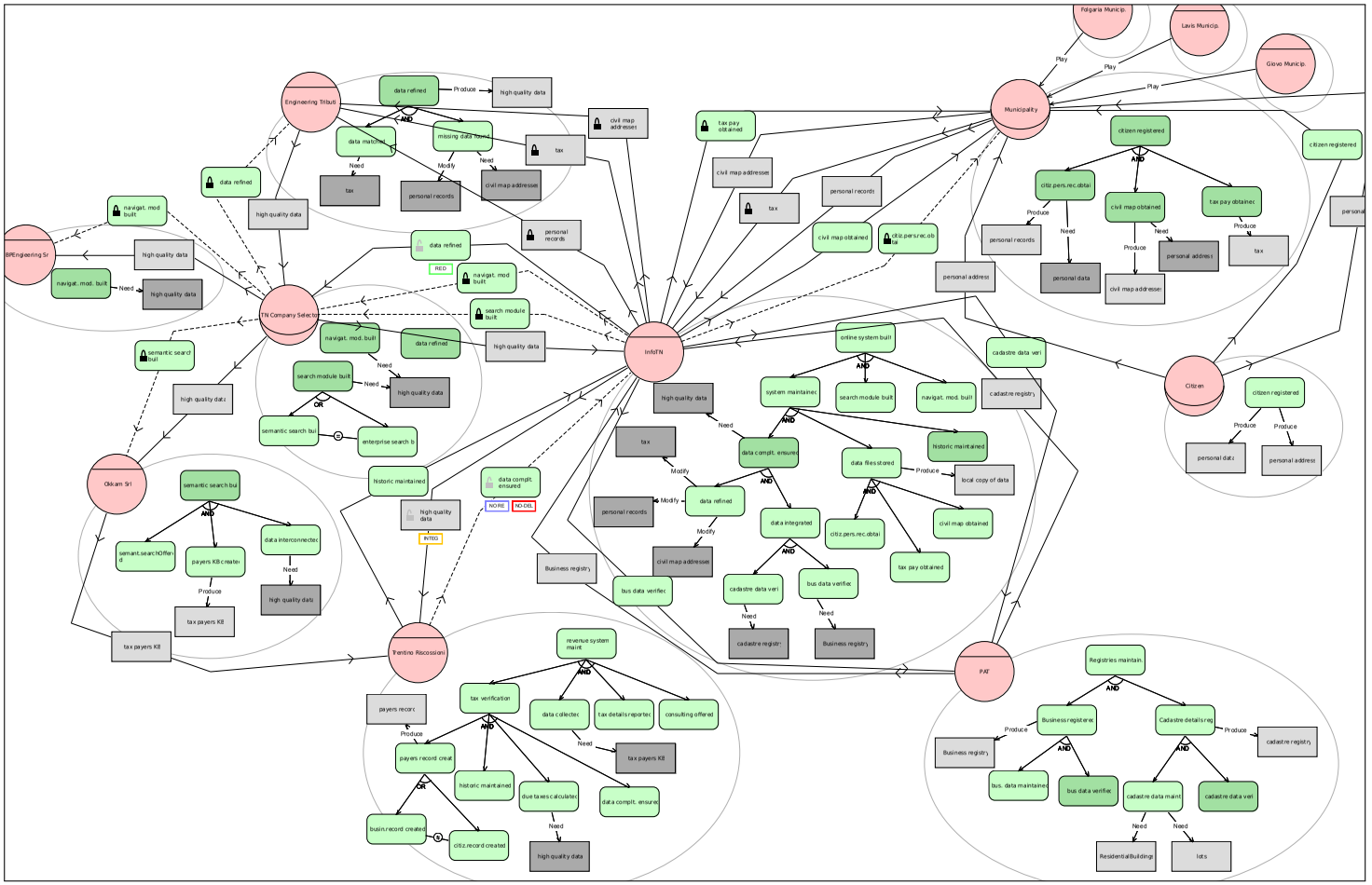

Fig. 3: TasLab Social View

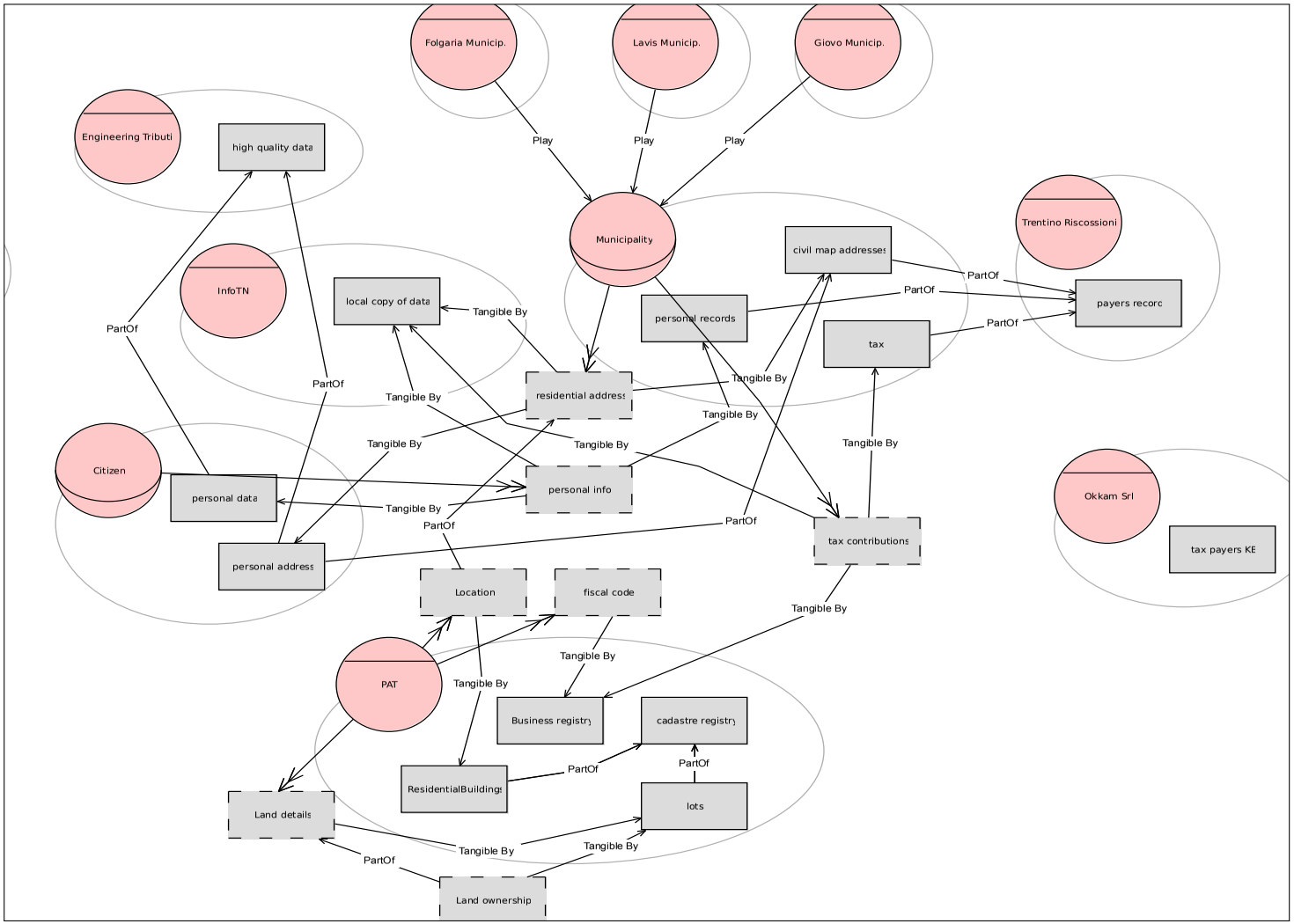

Fig. 4: TasLab Information View

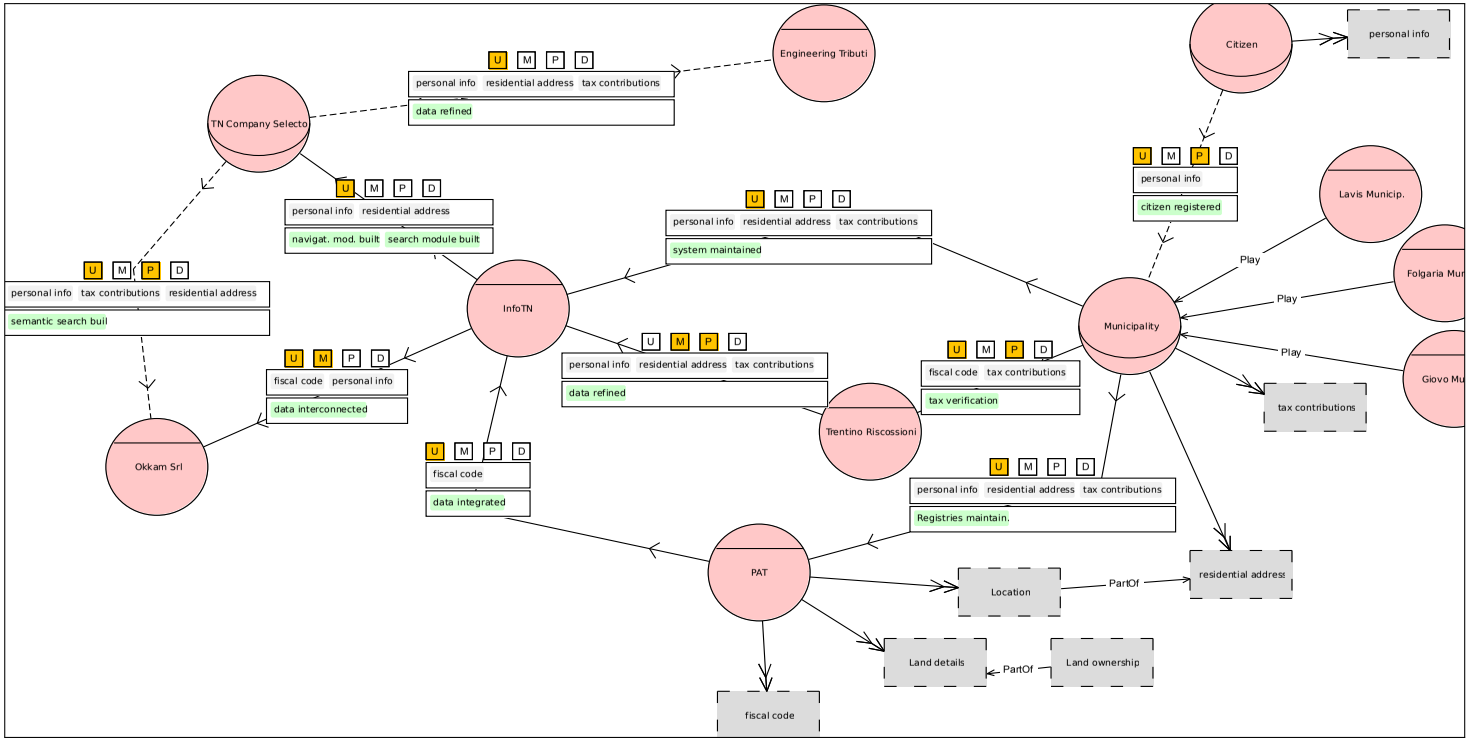

Fig. 5: TasLab Authorisation View