Modeling Social and Individual Trust in Requirements Engineering Methodologies[*]

Paolo Giorgini[1], Fabio Massacci[1], John Mylopoulos[1,2], and Nicola Zannone[1]

[1] Department of Information and Communication Technology
University of Trento - Italy
{massacci,giorgini,zannone}@dit.unitn.it
[2] Department of Computer Science
University of Toronto - Canada
jm@cs.toronto.edu

Abstract. When we model and analyze trust in organizations or information systems we have to take into account two different levels of analysis: social and individual. Social levels define the structure of organizations, whereas individual levels focus on individual agents. This is particularly important when capturing security requirements where a "normally" trusted organizational role can be played by an untrusted individual.

Our goal is to model and analyze the two levels finding the link between them and supporting the automatic detection of conflicts that can come up when agents play roles in the organization. We also propose a formal framework that allows for the automatic verification of security requirements between the two levels by using Datalog and has been implemented in CASE tool.

Keywords: Information Technologies; Social; Security & Trust; Requirements and methodologies; Trust specification, analysis and reasoning; Realization of prototypes; Agent-Oriented Technologies.

1 Introduction

The last years have seen a major interest for methodologies for software engineering that could capture trust and security requirements from the very early stage of design [10, 12, 14, 18, 19]. Still all proposals (including our own) discuss the system or the organization looking at roles and positions rather than individual agents. From a certain viewpoint this is to be expected natural as a software engineer doesn't want to design and implement John Doe but rather the generic Cashier agent. However, in a recent study, the majority of Information Security Administrators said that their biggest worry is employee negligence and abuse [15]. Internal attacks can be more harmful than external attacks since they are

[*] This work has been partially funded by the IST programme of the EU Commission, FET under the IST-2001-37004 WASP project, by the FIRB programme of MIUR under the RBNE0195K5 ASTRO Project and by PAT MOSTRO project.

being performed by trusted users that can bypass access control mechanisms. So, we need models that compare the structure of the organization (roles and relations among them) with the concrete instance of the organization (agents playing some roles in the organization and relations among them).

Among the requirements engineering methodologies, the Tropos agent-oriented requirements methodology [1] involves two different levels of analysis: *social* and *individual*. In the organization level we analyze roles and positions of the organization, whereas in individual level the focus is on single agents. Of course there is no explicit separation between the two levels, and so Tropos is not able to maintain the consistency between the social level (roles and positions) and the individual level (agent).

In the trust management setting, as far we know, there are only few proposals that analyze both social and individual levels and compare them. Huynh et al. [8] introduce role-based trust to model the trust resulting from the role-based relationships between two agents, but no requirements methodology is proposed. Sichman et al. [17] propose an approach where agents mental attitude is characterized by their personal mental attitude and the one which they have by playing a role. However, there is not a complete separation between the two levels. Therefore, this approach is not able to identify possible conflicts that can be arise by analyzing each level separately, and so system designers cannot verify the correctness and consistency of the structure of organizations.

In this paper we focus on the problem of identifying and solving conflicts emerging between the social and the individual level in the Secure Tropos model that we have proposed. Our goal is to:

- design models at both social level and individual level, independently;
- verify correctness and consistency of social level;
- map relations at social level into models at individual level;
- solve conflicts if needed;
- verify correctness and consistency of models at individual level.

The remainder of the paper is structured as follows. Next (§2) we provide an brief description of Tropos concepts and describe the basic ones that we use for modeling security. Then, we show how Tropos concepts are mapped into the Secure Tropos framework and vice versa (§3). Next (§4) we analyze how social relationships are propagate between social and individual levels, and show how this can generate conflicts. We present a formal framework for automatically identifying and solving conflicts (§5). Finally, we discuss related works and conclude the paper (§6).

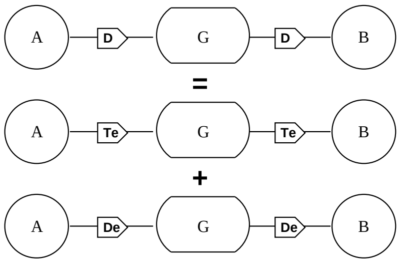
2 Tropos and Secure Tropos

Tropos [1] is a development methodology, tailored to describe both the organizational environment of a system and the system itself. Tropos uses the concepts of actor, goal, soft goal, task, resource and social dependency for defining the obligations of actors (dependees) to other actors (dependers). An actor is an active entity that performs actions to achieve goals. Actors could have dependencies on other actors as well as dependencies from other actors. Actors can be decomposed into sub-units for modeling the internal structure of an actor preserving the intentional actor abstraction provided by modeling processes in terms of external relationships. Complex social actors can be modeled using three types of sub-units: agents, roles, and positions. An agent is an actor with concrete, physical manifestations, such as a human individual. The term agent is used instead of person since it can be used to refer to human as well as artificial agents. Agents are those that have characteristics not easily transferable to other individuals. A role is an abstract characterization of the behavior of a social actor within a certain domain. Its characteristics are easily transferable to other social actors. A position represents a set of roles played by an agent. A goal represents the strategic interests of an actor. A task specifies a particular course of action that produces a desired effect, and can be executed in order to satisfy a goal. A resource represents a physical or an informational entity. Finally, a dependency between two actors indicates that one actor depends on another to accomplish a goal, execute a task, or deliver a resource. In Tropos diagrams, actors are represented as circles; services - goals, tasks and resources - are respectively represented as ovals, hexagons and rectangles.

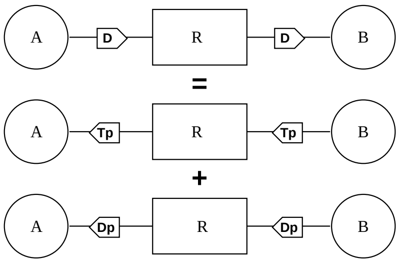
Secure Tropos [5] introduces four new concepts and relationships behind Tropos dependency: *trust*, *delegation*, *provisioning*, and *ownership*. The basic idea of ownership is that the owner of an service has full authority concerning access and disposition of his service. The distinction between owning (owns) and provisioning (provides) a service makes it clear how to model situations in which, for example, a client is the legitimate owner of his/her personal data and a Web Service provider that stores customers' personal data, provides the access to her/his data. We use the relation for delegation when in the domain of analysis there is a formal passage of authority (e.g. a signed piece of paper, a digital credential is sent, etc.). The trust relations have their intuitive meaning among agents. As for trust relations among roles or positions, the semantic is subtler as it refers to trust among organizations as we shall see in the next section.

3 Refining the concept of Dependency

The new Secure Tropos concepts allow for a refinement of the dependency concept. In particular, we can now show how the dependency (depends) between two actors can be expressed in terms of trust and delegation. In order to do that, we introduce the distinction between delegation of permission and execution. In the *delegation of permission* (del_perm) the delegatee thinks "Now, I have the permission to fulfill the service", whereas in the *delegation of execution* (del_exec), the delegatee thinks "Now, I have to get the service fulfilled". Further, we want separate the concept of trust from the concept of delegation, as we might need to model systems in which some actors must delegate permission or execution to other actors they don't trust. Also in this case it is convenient to have a suitable distinction for trust in managing permission and trust in



(a) Goal Dependency



(b) Resource Dependency

Fig. 1. Tropos dependency in terms of Secure Tropos

managing execution. The meaning of *trust of permission* (trust_perm) is that an actor (truster) trusts that another actor (trustee) uses correctly the service. The meaning of *trust of execution* (trust_exec) is that an actor (truster) trusts that another actor (trustee) is able to fulfill the service.

The distinction between *execution* and *permission* allows us to define a dependency in terms of trust and delegation. In particular, when the dependum is a goal or a task we have delegation and trust of execution, whereas when the dependum is a resource we have delegation and trust of permission. In symbols:

$$\mathsf{depends}(A, B, S) \Longleftrightarrow \mathsf{del\_exec}(A, B, S) \wedge \mathsf{trust\_exec}(A, B, S) \quad (1)$$

where S is a goal or a task, and

$$\mathsf{depends}(A, B, S) \Longleftrightarrow \mathsf{del\_perm}(ID, B, A, S) \wedge \mathsf{trust\_perm}(B, A, S) \quad (2)$$

where S is a resource.

A graphical representation of these formulas is given, respectively, in Fig. 1(a) and in Fig. 1(b). These diagrams use the label **D** for Tropos dependency and labels **De** and **Te** (**Dp** and **Tp**), respectively for delegation of execution and trust of execution (delegation of permission and trust of permission). Notice, also from Fig. 1 that the same dependency is mapped into differently oriented relations at the lower level.

Another refinement is the introduction of negative authorizations which are needed for some scenarios. Tropos already accommodates the notion of positive or negative contribution of goals to the fulfillment of other goals. We use negative authorizations to help the designer in shaping the perimeter of positive trust to avoid incautious delegation certificates that may give more powers than desired.

Suppose that an actor should not be given access to a service. In situations where authorization administration is decentralized, an actor possessing the right to use the service, can delegate the authorization on that service to the wrong actor. Since many actors may have the right to use a service, it is not always possible to enforce with certainty the constraint that a actor cannot access a particular service. We propose an explicit distrust relationship as an approach for handling this type of constraint. This is also sound from a cognitive point of view if we follow the definition of trust given by [2]: trust is a mental state based on a set of beliefs. We can say that if, on your own knowledge, you feel to trust me, then you trust me. Similarly, if you feel like distrusting me, then you distrust me. Obviously, there are various reasons of distrusting in agents such as unskillfulness, unreliability and abuse, but these situations are not treated here.

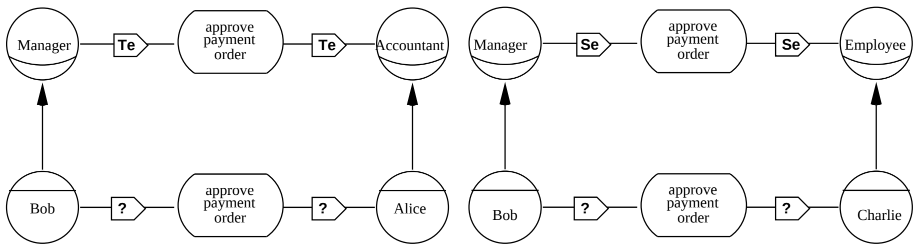
As we have done for trust, we also distinguish between distrust of execution (distrust_exec) and distrust of permission (distrust_perm). The graphical diagrams presented in this paper use the labels **Se** and **Sp**, respectively, for distrust of execution and distrust of permission. In the case there is no explicit trust relationship between agents, the label "**?**" is used.

4 Social vs Individual Trust

In Tropos, stakeholders are presented as actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. Since the concept of actor includes those of agent, role and position, the Tropos models involve two different levels of analysis: *social* and *individual*. In the social level we analyze roles and positions of the organization, whereas in individual level the focus is on single agents. In particular, at social level the structure of organizations are defined associating to every role (or position) objectives and responsibilities relating to the activities that such roles have to perform within the organizations. On the other hand, at individual level, agents not only are defined with their objectives and responsibilities, but also they are associated to role (or position) they can play.

This role-based approach takes advantage from specifying agents into two steps: assignment of objectives and responsibilities to role, and assignment of agents to roles. This allows to simplify the management of requirements. For instance, when new responsibilities are considered by the information system, the administrator needs only to decide to which roles such responsibilities have to be assigned. Then, all agents that play those roles inherit them. This means that relations spread from social level to individual level. Notice that when more agents play the same role, all instances inherit the properties associated to that role where the term property includes any relations presented above. Another advantage is that we can capture vested interested or conflict of interest explicitly during requirements phase.

Tropos supports also role hierarchy by using the relation ISA (is_a). Notice that this hierarchy is different from "standard" RBAC role hierarchy [9] where higher roles in the hierarchy are more powerful and the notion of domination is used. Instead our approach is based on the "standard" notion of hierarchy proposed in UML-base and Database-base approaches. Referring to the study case presented in [13] we have, for example, that Faculty Deans, Heads of Department and Central Directorate Managers are Data Processors according the Italian Privacy legislations.



(a) Trust of execution (b) Distrust of execution

Fig. 2. Missing (dis)trust relations at individual level

Definition 1. *Let r_1 and r_2 be roles. We say that r_1 is a* specialized sub-roles *of r_2 (or, equivalently, r_2 is* generalized super-role *of r_1) if* $\mathsf{is\_a}(r_1, r_2)$. *Then, all specialized sub-roles inherit all properties of the generalized super-role.*

In above scenario, Faculty Deans, Heads of Department and Central Directorate Managers have all properties assigned to Data Processors.

Yet, in Tropos there is no explicit separation between the two levels, and it is very difficult to analyze and maintain the consistency between the social level (dependencies between roles and positions) and the individual level (dependencies between agents). For simplicity, in the remainder of the paper we don't distinguish role and position.

4.1 Missing Requirements

When we model and analyze functional trust and security relationships, it is possible that such requirements are given only at individual level or at social level. We would like to have a CASE tool that automatically completes models given at individual level from the social one when any relations are missing. Let us see why this is needed with examples from bank policies.

Example 1. In a bank context, branch managers have the objective to guarantee the availability and correct execution of payment orders. A bank policy states that a payment order should be issued only when it has been submitted and approved. Banks have also a policy stating that a branch bank manager should trust the chief accountants who work in his branch to approve payment orders (Fig. 2(a)). Suppose that Bob is the branch manager and Alice a new chief accountant and they have never met before. Then, Bob should trust Alice for approving payment orders to guarantee the availability of the service.

Example 2. Another bank policy states that a branch bank manager should distrust normal employees to approve payment orders (Fig. 2(b)). Suppose that Bob is the branch manager and Charlie a newly employed cashier and they don't know each other. Then, Bob should distrust Charlie for approving payment orders.

We don't consider the case in which the relations are missing at social level because this level represents the structure of the organization which should be described explicitly in the requirements. The presence of a large number of trust relations at individual level that is not matched by a social level may be an indicator of a missing link at social level (or of a problem in the organization for distrust relations). On the contrary, Hannoun et al. [7] propose to detect the inadequacy of an organization regarding the relations existing among the agents involved in the system.

4.2 Conflicts on Trust Relations

In [5] we have only considered when trust is explicit, and we have not distinguished the case where there is explicit distrust and the case where no trust relation is given. Contrarily, in this paper we take in consideration all these three possibilities. The presence of positive and negative authorization at the same time could generate some conflicts on trust relationships. We define a *trust conflict* the situation where there are both a positive and a negative trust relation between two actors for the same service. Next, formal definitions are given.

Definition 2. *A conflict on trust of execution occurs when*

$$\exists x, y \in \texttt{Agent}\ \exists s \in \texttt{Service} \mid \mathsf{trust\_exec}(x, y, s) \wedge \mathsf{distrust\_exec}(x, y, s)$$

Definition 3. *A conflict on trust of permission occurs when*

$$\exists x, y \in \texttt{Agent}\ \exists s \in \texttt{Service} \mid \mathsf{trust\_perm}(x, y, s) \wedge \mathsf{distrust\_perm}(x, y, s)$$

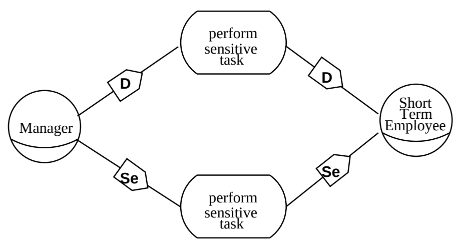
A trust conflict may exist, for example, since system designers wrongly put both a (implicit) trust relation and the corresponding distrust relation.

Example 3. A manager depends on a short-term employee for a certain sensitive task, but short-term employees are distrusted for sensitive tasks (Figure 3(a)).

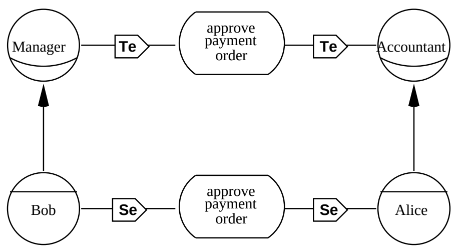
When we model and analyze security requirements, it is also possible that such requirements are specified at both individual and social levels, they could be in contrast with each other.

Example 4. Consider again Example 1 where bank managers trust accountants for approving payment orders, and Bob is the manager and Alice an accountant in a bank branch. What happen if Bob has had some problems with Alice in the past and he doesn't trust her? This scenario is presented in Fig.3(b).

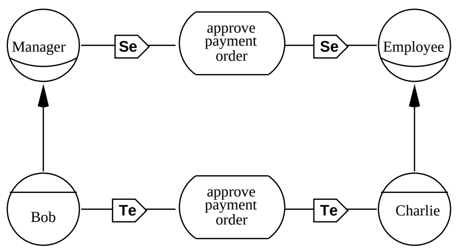
Example 5. Consider again Example 2 where bank managers distrust employees for approving payment orders, and Bob is the manager and Charlie a employee in a bank branch. What happen if Bob trusts Charlie for approving payment orders? This scenario is presented in Fig. 3(c).



(a) Conflict due to implicit trust



(b) Social Trust vs Individual Distrust



(c) Social Distrust vs Individual Trust

Fig. 3. Conflicts on (dis)trust relations

4.3 Solving Trust Conflicts

Our goal is to identify a solution in order to detect and, possibly, resolve conflicts on trust relations. To this end, we propose to use monitoring (monitoring) for solving conflicts since there is some evidence that it is good solution to prevent undesirable behaviors in information systems [6, 11]. Monitors rely on events and aim at observing, analyzing and controlling the execution of the information system in order to define its current behavior model and correct the undesirable behaviors, as well as unauthorized accesses.

Example 6. Referring to Example 4, we believe that Bob should monitor (or delegate this task to another actor) whether Alice does what she has to do since the organization imposes him to trust, but it is not his own choice.

4.4 Conflict of Interest

An agent can play (play) several roles. We assume that an agent is explicitly assigned to a given roles and this assignment gives him the rights and responsibilities assigned to that role. Conflicts of interest refer to scenarios where an individual occupies dual roles which should not be performed simultaneously. Because of the risk for abuse, performing both roles at the same time is considered to be inappropriate. In other words, the conflict of interest concerns the potential advantage an agent could take of his position.

Definition 4. *Let r_1 and r_2 be two roles that an agent cannot play at the same time. A conflict of interest occurs when*

$$\exists x \in \texttt{Agent} \mid \mathsf{play}(x, r_1) \wedge \mathsf{play}(x, r_2)$$

Example 7. In a bank context, payment orders should be issued only when they have been submitted and approved. A payment order should be submitted by an employee and approved by a different accountant. This means that an agent cannot play at the same time the role *accountant* and *employee*.

$$:\!\text{-}\ \mathsf{play}(X, \mathit{accountant}) \wedge \mathsf{play}(X, \mathit{employee}) \wedge \mathsf{agent}(X)$$

Therefore, if we assume that the set of employees is disjoint from the set of managers this kind of conflicts doesn't exist.

The above notion of conflict of interest could be refined since it doesn't show why conflicts exist. Moreover, the definition we have done above could be too strong since some time a conflict could be only for some specific instances.

Example 8. The scenario presented in Example 7 is far from real life: accountant can usually execute employee tasks, but they cannot approve their own orders.

This example also reveals that, before verifying the consistency of the individual level, we should be sure on the consistency of the organization structure: it is also possible that a conflict arises by considering just the social level.

Further, some laws issued, for example, by Antitrust Division[3] or some enterprises' policies can impose that the same person must not own (be entitled) or provide some certain services at the same time. An agent could own or provide some service himself and could own or provide other services since he plays roles that owns or is able to provide such services. Now, we refine the Definition 4 as follows:

Definition 5. *Let s_1 and s_2 be two services that an actor cannot own at the same time. An ownership conflict occurs when*

$$\exists x \in \texttt{Actor} \mid \mathsf{owns}(x, s_1) \wedge \mathsf{owns}(x, s_2)$$

Definition 6. *Let s_1 and s_2 be two services that an actor cannot provide at the same time. A provisioning conflict occurs when*

$$\exists x \in \texttt{Actor} \mid \mathsf{provides}(x, s_1) \wedge \mathsf{provides}(x, s_2)$$

In this paper we define conflicts only on primitive properties, but similar definitions can be given also for the derived ones presented in [5].

Example 9. An instance of the bank policy presented in Example 8 can be formalized with the following integrity constraints.

$$:\!\text{-}\ \mathsf{provides}(A, \mathit{submit\_order\_25}) \wedge \mathsf{provides}(A, \mathit{approve\_order\_25})$$

[3] http://www.usdoj.gov/atr/

Tropos Primitives	**Secure Tropos Primitives**
goal(Goal:g)	provides(Actor:a, Service:s)
task(Task:t)	wants(Actor:a, Service:s)
resource(Resource:r)	owns(Actor:a, Service:s)
agent(Agent:a)	del_exec(Actor:a, Actor:b, Service:s)
position(Position:a)	del_perm(id:idC, Actor:a, Actor:b, Service:s)
role(Role:a)	trust_exec(Actor:a, Actor:b, Service:s)
play(Agent:a, Role:b)	trust_perm(Actor:a, Actor:b, Service:s)
is_a(Role:a, Role:b)	distrust_exec(Actor:a, Actor:b, Service:s)
depends(Actor:a, Actor:b, Service:s)	distrust_perm(Actor:a, Actor:b, Service:s)
	monitoring(Actor:a, Actor:b, Service:s)
	trust_mon(Actor:a, Actor:b, Service:s)

Table 1. Predicates

5 Formalization

We distinguish between two main types of predicates: primitive and derived. These correspond to respectively extensional and intensional predicates in Datalog. Extensional predicates are predicates set directly with the help of ground facts and are the ones corresponding the edge and circles drawn by the requirements engineer on the CASE tool. Intensional predicates are implicitly determined with the help of rules. We start by presenting the set of extensional predicates (Table 1) and refer to [5] for all rules related to previously introduced concepts. Here we only present rules for the new concepts.

The left part of the table contains the primitives used for modeling Tropos framework. The unary predicates goal, task and resource are used respectively for identifying goals, tasks and resource. Note that type Goal, Task and Resource are sub-types of Service. We shall use letters S, G, T and R possibly with indices as metavariables ranging over the terms, respectively, of type Service, Goal, Task and Resource. The intuition is that agent(a) holds if instance a is an agent, position(a) holds if instance a is a position, and role(a) holds if instance a is a role. Note that type Agent, Position and Role are sub-types of Actor. We shall use letters X, Y and Z as metavariables ranging over the terms of type Actor, A, B and C as metavariables ranging over the terms of type Agent, and T, Q and V as metavariables ranging over the terms of type Role. Metalevel variables are used as a syntactic sugar to avoid to write the predicates that type variables. For example, when the metavariable G occurs in a rule, the predicate goal(G) should be put in the body of the rule. The predicate play(a, b) holds if agent a is an instance of role b. The intuition is that is_a(a, b) holds if role a is a specialization of role b. The predicate depends(a, b, s) holds if actor a depends from actor b for service s.

In the right part we have the additional predicates introduced by the Secure Tropos framework. When an actor has the capabilities to fulfill a service, he provides it. The intuition is that provides(a, s) holds if actor a provides service s.

From Tropos to Secure Tropos		
ST1:	trust_exec(X,Y,G) :-	depends(X,Y,G)
ST2:	del_exec(X,Y,G) :-	depends(X,Y,G)
ST3:	trust_perm(Y,X,R) :-	depends(X,Y,R)
ST4:	del_perm(ID,Y,X,R) :-	depends(X,Y,R)
From Secure Tropos to Tropos		
ST5:	depends(X,Y,G) :-	trust_exec(X,Y,G) $\wedge$ del_exec(X,Y,G) $\wedge$ **not** distrust_exec(X,Y,G)
ST6:	depends(X,Y,R) :-	trust_perm(Y,X,R) $\wedge$ del_perm(ID,Y,X,R) $\wedge$ **not** distrust_perm(Y,X,R)

Table 2. Axioms for mapping Tropos into Secure Tropos and vive versa

The predicate wants(a,s) holds if actor a has the objective of fulfilling service s. The predicate owns(a,s) holds if actor a owns service s. The owner of a service has full authority concerning access and usage of his services, and he can also delegate this authority to other actors. The predicate trust_exec(a,b,s) (resp. trust_perm(a,b,s)) holds is actor a trusts that actor b is able to fulfill (resp. uses correctly) service s where a is called truster and b trustee. The predicate distrust_exec(a,b,s) (resp. distrust_perm(a,b,s)) holds is actor a distrusts actor b for service s. The intuition is that monitoring(a,b,s) holds if actor a monitors actor b on service s. The intuition is that trust_mon(a,b,s) holds if actor a trust actor b for monitoring service s. The predicate del_perm(idC,a,b,s) holds if actor a delegates to actor b the permission on service s. The actor a is called the *delegater*; the actor b is called the *delegatee*; idC is the certificate identifier. The predicate del_exec(a,b,s) holds if actor a delegates to actor b the execution of service s.

Once the requirements engineer has drawn up the model (i.e. the extensional predicates) we are ready for the formal analysis. To derive the right conclusions from an intuitive model, we need to complete the model using *axioms* for the intensional predicates. Axioms are rules of the form L:- $L_1 \wedge ... \wedge L_n$ where L, called head, is a positive literal and $L_1,...,L_n$ are literals and they are called body. Intuitively, if $L_1,...,L_n$ are true in the model then L must be true in the model. We use the notation $\{L\}$:-$L_1,\ldots,L_n$ to indicate that if $L_1,\ldots,L_n$ are true then L *may* be true. Essentially, L will be added to the model only if some constraints demand its inclusion. This construction can be captured with a simple encoding in logic programs. Notice also that when a relation uses variables of type `Actor` the relation can apply to both social and individual levels, but separately.

In Table 2 there are the axioms to map Tropos dependency into Secure Tropos framework and vice versa. Notice that ST1-2 and ST5 have also to be repeated for the case where the dependum is a task.

Table 3 defines the intensional versions, entrust_exec and disentrust_exec (entrust_perm and disentrust_perm) of the extensional predicates trust_exec and distrust_exec (trust_perm and distrust_perm) that are used to build (dis)trust

Trust of execution		
T1:	disentrust_exec(X,Y,S) :-	distrust_exec(X,Y,S)
T2:	disentrust_exec(X,Z,S) :-	entrust_exec(X,Y,S) $\wedge$ distrust_exec(Y,Z,S) $\wedge$ not disentrust_exec(X,Y,S)
T3:	entrust_exec(X,Y,S) :-	trust_exec(X,Y,S) $\wedge$ not disentrust_exec(X,Y,S)
T4:	entrust_exec(X,Z,S) :-	entrust_exec(X,Y,S) $\wedge$ entrust_exec(Y,Z,S) $\wedge$ not disentrust_exec(X,Z,S)
Trust of permission		
T5:	disentrust_perm(X,Y,S) :-	distrust_perm(X,Y,S)
T6:	disentrust_perm(X,Z,S) :-	entrust_perm(X,Y,S) $\wedge$ distrust_perm(Y,Z,S) $\wedge$ not disentrust_perm(X,Y,S)
T7:	entrust_perm(X,Y,S) :-	trust_perm(X,Y,S) $\wedge$ not disentrust_perm(A,B,S)
T8:	entrust_perm(X,Z,S) :-	entrust_perm(X,Y,S) $\wedge$ entrust_perm(Y,Z,S) $\wedge$ not disentrust_perm(X,Z,S)
Confident of monitoring		
T9:	confident_mon(X,Y,S) :-	trust_mon(X,Z,S) $\wedge$ monitoring(Z,Y,S)

Table 3. Axioms on Trust and Distrust

chains by propagating (dis)trust of execution (permission) relations. The intuitive meaning of rules T1-2 (and T5-6 for permission) is presented in the following examples.

Example 10. A branch manager depends on accountants for performing sensitive tasks. This implies that the manager trusts accountants for it. On the other hand, accountants distrust short-term employees for this goal. Therefore, the manager distrusts short-term employees. This explains also the conflict shown in Example 3.

Example 11. A bank policy states that the bank general manager should trust branch managers for correctly managing branch cash desks. Another branch policy states that the branch manager have not to permit (distrust of permission) short-term employees for managing branch cash desks. Therefore, the general manager distrusts short-term employees.

T3-4 (respectively T7-8) rules are used to build trust chains by propagating trust of execution (permission) relations. T9 introduces the intensional predicate confident_mon(a,b,s): actor a is confident that there exists someone that monitors actor b for service s. Also this set of axions applies to both social level and individual level, independently, and so A, B and C have to be typed as roles for the social level and as agents for the individual level.

Table 4 presents the axioms for role hierarchy and for mapping relations from social level to individual level. The predicate specialize is the intensional version of is_a, whereas instance is intensional version of play. Axioms Ax1-12 have to be repeated replacing the predicate instance with specialize and predicate agent with role for completing social level with respect to role hierarchy.

Role Hierarchy		
RH1:	$\mathsf{specialize}(T,Q)$:-	$\mathsf{is\_a}(T,Q)$
RH2:	$\mathsf{specialize}(T,Q)$:-	$\mathsf{specialize}(T,V) \wedge \mathsf{is\_a}(V,Q)$
RH3:	$\mathsf{instance}(A,T)$:-	$\mathsf{play}(A,T)$
RH4:	$\mathsf{instance}(A,T)$:-	$\mathsf{instance}(A,Q) \wedge \mathsf{specialize}(Q,T)$
From social level to individual level		
Ax1:	$\mathsf{provides}(A,S)$:-	$\mathsf{provides}(T,S) \wedge \mathsf{instance}(A,T)$
Ax2:	$\mathsf{wants}(A,S)$:-	$\mathsf{wants}(T,S) \wedge \mathsf{instance}(A,T)$
Ax3:	$\mathsf{owns}(A,S)$:-	$\mathsf{owns}(T,S) \wedge \mathsf{instance}(A,T)$
Ax4:	$\mathsf{trust\_exec}(A,B,S)$:-	$\mathsf{trust\_exec}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax5:	$\mathsf{trust\_perm}(A,B,S)$:-	$\mathsf{trust\_perm}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax6:	$\mathsf{distrust\_exec}(A,B,S)$:-	$\mathsf{distrust\_exec}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax7:	$\mathsf{distrust\_perm}(A,B,S)$:-	$\mathsf{distrust\_perm}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax8:	$\mathsf{del\_exec}(A,B,S)$:-	$\mathsf{del\_exec}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax9:	$\mathsf{del\_perm}(ID,A,B,S)$:-	$\mathsf{del\_perm}(ID,T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax10:	$\mathsf{monitoring}(A,B,S)$:-	$\mathsf{monitoring}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax11:	$\mathsf{trust\_mon}(A,B,S)$:-	$\mathsf{trust\_mon}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Ax12:	$\mathsf{depends}(A,B,S)$:-	$\mathsf{depends}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$

Table 4. Axioms for role hierarchy and for mapping social level into individual level

Pro1:	:- $\mathsf{entrust\_exec}(X,Y,S) \wedge \mathsf{disentrust\_exec}(X,Y,S)$
Pro2:	:- $\mathsf{entrust\_perm}(X,Y,S) \wedge \mathsf{disentrust\_perm}(X,Y,S)$
Pro3:	:- $\mathsf{entrust\_exec}(A,B,S) \wedge \mathsf{disentrust\_exec}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Pro4:	:- $\mathsf{entrust\_perm}(A,B,S) \wedge \mathsf{disentrust\_perm}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Pro5:	:- $\mathsf{disentrust\_exec}(A,B,S) \wedge \mathsf{entrust\_exec}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$
Pro6:	:- $\mathsf{disentrust\_perm}(A,B,S) \wedge \mathsf{entrust\_perm}(T,Q,S) \wedge \mathsf{instance}(A,T) \wedge \mathsf{instance}(B,Q)$

Table 5. Properties for identifying conflicts

Properties are different from axioms: they are desirable design features, but may not be true (or too costly to implement) of the particular design at hand. Table 5 presents the properties used to identifying conflicts that occur when both a trust and a distrust relations exist among two actors for the same service. Pro1-2 are used to identify generic conflicts and correspond to Definition 2 and 3. These properties apply to both social level and individual level, independently and so A and B have to be typed as role for the social level and as agents for the individual level. Pro1-2 can be refined in order to identify conflicts of the form of Fig. 3(c) (Pro3-4) and Fig. 3(b) (Pro5-6).

Table 6 formalizes the proposal for solving conflicts when there is a trust relation at social level and a distrust relation at individual level.

To accommodate C1-2 in our framework we have to modify axioms Ax6-7 in Table 4. The new version of these axioms is given in Table 7.

C1: {monitoring(M, B, S)}:-	disentrust_exec(A, B, S) $\wedge$ entrust_exec(T, Q, S) $\wedge$ instance(A, T) $\wedge$ instance(B, Q) $\wedge$ trust_mon(A, M, S)
C2: {monitoring(M, B, S)}:-	disentrust_perm(A, B, S) $\wedge$ entrust_perm(T, Q, S) $\wedge$ instance(A, T) $\wedge$ instance(B, Q) $\wedge$ trust_mon(A, M, S)

Table 6. Axioms for solving conflicts

Ax6′: distrust_exec(A, B, S) :-	distrust_exec$(T, Q, S)$$\wedge$instance$(A, T)$$\wedge$instance$(B, Q)$$\wedge$ **not** confident_mon(A, B, S)
Ax7′: distrust_perm(A, B, S) :-	distrust_perm$(T, Q, S)$$\wedge$instance$(A, T)$$\wedge$instance$(B, Q)$$\wedge$ **not** confident_mon(A, B, S)

Table 7. Axioms in order to support monitoring

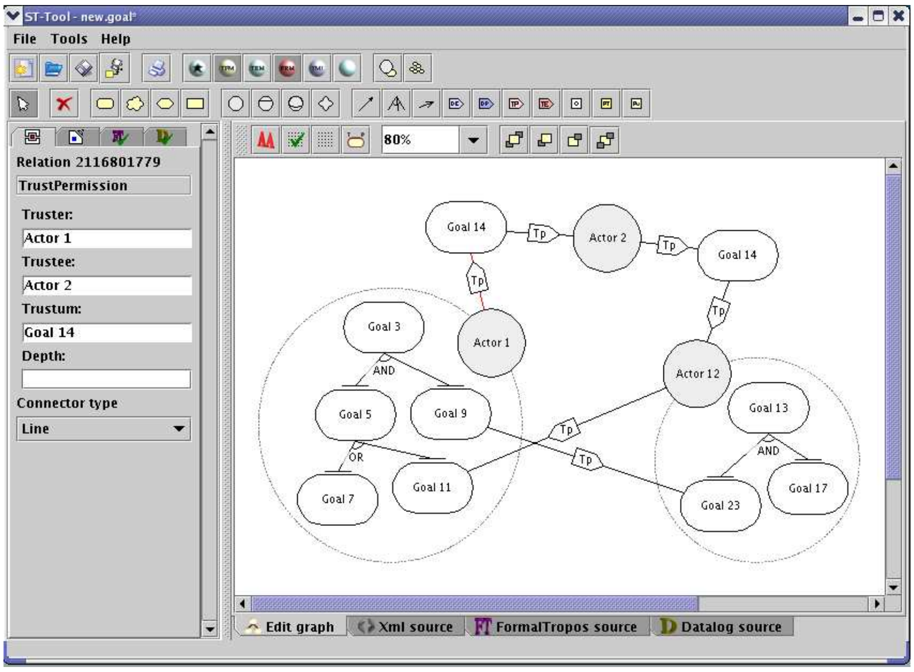
6 Related Work and Conclusions

Computer Security is one of today's hot topic and the need for conceptual models of security features have brought up a number of proposals especially in UML community [4, 10, 12, 14, 18, 16].

Lodderstedt et al. [12] present a modeling language, based on UML, called SecureUML. Their approach is focused on modeling access control policies and how these (policies) can be integrated into a model-driven software development process. To address security concerns during software design, Doan et al. [4] incorporate Mandatory Access Control (MAC) into UML. Ray et al. [16] propose to model RBAC as a pattern by using UML diagram template. Further, they represent constraints on RBAC model through the Object Constraint Language. Similarly, Jürjens [10] proposes an extension of UML to accommodate security requirements by using Abstract State Machine model and adding several stereotypes to accommodate its proposal towards security verification. McDermott and Fox adapt use cases [14] to capture and analyze security requirements, and they call the adaption an abuse case model. Sindre and Opdahl define the concept of a misuse case [18], the inverse of a use case, which describes a function that the system should not allow. CORAS [3] is a model-based risk assessment method for security-critical systems. It is essentially a risk management process based on UML and aims to adapt, refine, extend and combine existing methods for risk analysis.

We have presented here an enhanced trust and security requirements engineering methodology that is able to capture trust conflicts at social and individual level. Our framework is supported by a CASE tool called STTool.[4] A screenshot is shown in Fig. 4. This tool is implemented in JAVA and provides a user friendly interface within the DLV system to the requirements engineer for the verification of the correctness and consistency of trust and security requirements in the organization.

[4] http://sesa.dit.unitn.it/sttool/


Relation 2116801779

TrustPermission

Truster: Actor 1

Trustee: Actor 2

Trustum: Goal 14

Depth:

Connector type: Line

Fig. 4. STTool

References

1. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *JAAMAS*, 8(3):203–236, 2004.
2. C. Castelfranchi and R. Falcone. Principles of trust for MAS: Cognitive anatomy, social importance and quantification. In *Proc. of ICMAS'98*, pages 72–79. IEEE Press, 1998.
3. F. den Braber, T. Dimitrakos, B. A. Gran, M. S. Lund, K. Stølen, and J. Ø. Aagedal. The CORAS methodology: model-based risk assessment using UML and UP. In *UML and the unified process*, pages 332–357. Idea Group Publishing, 2003.
4. T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl. MAC and UML for secure software design. In *Proc. of FMSE'04*, pages 75–85. ACM Press, 2004.
5. P. Giorgini, F. Massacci, J. Mylopoulous, and N. Zannone. Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning. In *Proc. of iTrust'04*, *LNCS* 2995, pages 176–190. Springer-Verlag, 2004.
6. Z. Guessoum, M. Ziane, and N. Faci. Monitoring and Organizational-Level Adaptation of Multi-Agent Systems. In *Proc. of AAMAS'04*, pages 514–521. ACM Press, 2004.
7. M. Hannoun, J. S. Sichman, O. Boissier, and C. Sayettat. Dependence Relations between Roles in a Multi-Agent System: Towards the Detection of Inconsistencies in Organization. In *Proc. of 1st. Int. Workshop on Multi-Agent Based Simulation*, *LNCS* 1534, pages 169–182. Springer-Verlag, 1998.

8. D. Huynh, N. R. Jennings, and N. R. Shadbolt. Developing an Integrated Trust and Reputation Model for Open Multi-Agent Systems. In *Proc. of 7th Int. Workshop on Trust in Agent Societies*, pages 65–74, 2004.
9. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *TODS*, 26(2):214–260, 2001.
10. J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2004.
11. G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring Teams by Overhearing: A Multi-Agent Plan-Recognition Approach. *JAIR*, 17:83–135, 2002.
12. T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proc. of UML'02*, *LNCS* 2460, pages 426–441. Springer-Verlag, 2002.
13. F. Massacci, M. Prest, and N. Zannone. Using a Security Requirements Engineering Methodology in Practice: The compliance with the Italian Data Protection Legislation. *Comp. Standards & Interfaces*, 2005. To Appear. An extended version is available as Technical report DIT-04-103 at `eprints.biblio.unitn.it`.
14. J. McDermott and C. Fox. Using Abuse Case Models for Security Requirements Analysis. In *Proc. of ACSAC'99*, pages 55–66. IEEE Press, 1999.
15. L. Ponemon. What Keeps Security Professionals Up At Night?, April 2003. URL: http://www.darwinmag.com/read/040103/threats.html.
16. I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *Proc. of SACMAT'04*, pages 115–124. ACM Press, 2004.
17. J. S. Sichman and R. Conte. On personal and role mental attitudes: A preliminary dependence-based analysis. In *Proc. of SBIA'98*, *LNCS* 1515, pages 1–10. Springer-Verlag, 1998.
18. G. Sindre and A. L. Opdahl. Eliciting Security Requirements by Misuse Cases. In *Proc. of TOOLS Pacific 2000*, pages 120 –131. IEEE Press, 2000.
19. A. van Lamsweerde, S. Brohez, R. De Landtsheer, and D. Janssens. From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering. In *Proc. of RHAS'03*, pages 49–56, 2003.