# Know What You Trust

## Analyzing and Designing Trust Policies with Scoll[⋆]

Fred Spiessens, Jerry den Hartog, and Sandro Etalle

Eindhoven Institute for the Protection of Systems and Information
University of Technology Eindhoven
Eindhoven, The Netherlands.
a.o.d.spiessens@tue.nl j.d.hartog@tue.nl s.etalle@tue.nl

**Abstract.** In Decentralized Trust Management (DTM) authorization decisions are made by multiple principals who can also delegate decisions to each other. Therefore, a policy change of one principal will often affect who gets authorized by another principal. In such a system of influenceable authorization a number of principals may want to coordinate their policies to achieve long time guarantees on a set of safety goals. The problem we tackle in this paper is to find minimal restrictions to the policies of a set of principals that achieve their safety goals. This will enable building useful DTM systems that are safe by design, simply by relying on the policy restrictions of the collaborating principals. To this end we will model DTM safety problems in Scoll [1], an approach that proved useful to model confinement in object capability systems [2].

## 1 Introduction

Structural (role based) decentralized trust management (DTM) systems address the problem of access and delegation control in a distributed setting where authorization emanates from multiple sources. The rights of the agents/users in/of a system are not determined by a single authority but is the effect of policies set by different parties.

Principals cannot only define roles and authorize other principals as members of these roles. They can also delegate the authorization of their roles to other principals. Several powerful role based delegation models and trust management languages have been proposed for this purpose in the literature, each with their own balance between simplicity and expressive power. In this paper we will use $\mathrm{RT}_0$, the simplest in the RT [3, 4] family of trust management languages, but our approach can easily be applied to more expressive members of that family.

**The Running Example** The following simple example will be used and elaborated throughout this paper. The chair of the Open Conference defines a reviewer role and a submitter role for the conference. The chair designates Alice as a first

⋆ This work has been supported in part by European Commission FP7 TAS3 project, nr. 216287 and the BSIK project Poseidon.

reviewer. He then delegates all authorization responsibilities for both conference roles to the members of the reviewer role. This means that the policy of every conference reviewer can now influence the conference's role assignment. The safety concern the conference chair wants to be guaranteed is a simple mutual exclusion between the submitter and the reviewer role: no submitter should ever become also a reviewer.

**The Problem** If the reviewers cooperate with each other to manage both conference roles, it is relatively easy for them to detect a breach of the mutual exclusion requirement: they would only have to check the members of both roles and raise alarm when both roles have a common member. However, it is not trivial for the reviewers to design their policies in such a way that the safety breach becomes guaranteed impossible.

For instance, it does not suffice for the reviewers to simply refrain from authorizing anyone directly to be a member of both roles. They must also watch their delegation statements, as these may have indirect effects that may not be obvious to predict. Disallowing the reviewers to assign anybody to the submitter role could be sufficient to guarantee the safety concern, but that solution would be too restrictive for the purpose of the conference.

The problem we want to solve concerning the running example is: in what way(s) can we restrict the policies of the reviewers no more than necessary to make sure that the conference roles are mutually exclusive regardless of the policies of the non-reviewers, while still allowing the submitter role to be filled.

The general form of the problem is as follows. In a DTM system in which every principal has a policy, and policies are finite sets of monotonic authorizations and delegations, let the following be given:

- let $P_k$ be the set of principals of which we know the exact policies
- let $P_u$ be the set of principals of which we do not know the policies or have no reliable way to restrict them, with $(P_u \cap P_k = \emptyset)$,
- let $P_c$ be the set of cooperating principals: the ones of which we can restrict the policies, with $(P_c \cap P_u = \emptyset)$
- a number of safety concerns: what authorizations should not be allowed
- a number of availability concerns: what authorizations should be allowed

The problem is to find all restrictions $R_i$ for the principals in $P_c$ such that:

1. As long as the principals in $P_c$ do not include any of the elements in $R_i$ into their policy, it is impossible for the principals of $P_u$ to break the safety requirements.
2. As long as the principals in $P_c$ do not restrict their policies any further than described above, it is possible for the principals of $P_u$ to reach the availability requirements .

We call $R_i$ a solution to our problem. In practice, we will only calculate the minimal sets $R_i$ for which both properties hold, and call them "optimal" solutions.

Property 1 indicates that every solution represents a set of sufficient restrictions for the cooperating principals that will guarantee safety, regardless of how the non-cooperative principals extend their policies. Property 2 merely indicates that no solution restricts the policies so strong that, even with maximally permissive policies of the principals in $P_u$, the reachability properties would not be guaranteed. Solutions are "optimal" if the restrictions are not only sufficient but also necessary for safety.

The approach we apply here will conservatively (over-)approximate all policies of the $P_u$ principals to calculate upper bounds to the policies of the $P_c$ principals. Our safety results are valid, even if the policies of the other principals are also conservatively approximated. However, this is not the case for availability requirements. That would require us to approximate the unknown policies from below and calculate lower bounds for the policies of the cooperating principals, which we regard as interesting future work.

A solvable problem will typically have multiple solutions, because a restriction in one principal's policy may render another restriction unnecessary.

**The Proposed Solution** We propose to express and analyze safety problems in DTM systems using *Scoll* (Safe Collaboration Language), a formal model designed for general safety analysis.

First, we show that, thanks to its DataLog based structure and its explicit support for behavior-based effect analysis, Scoll provides a natural way to model such problems.

Secondly, we demonstrate how *Scollar* (Scoll's analysis tool) can calculate the minimal restrictions in the behavior of a set of entities that are necessary to avoid a given set of unwanted effects, without leading to overly restrictive solutions that prevent another given set of wanted effects. Our entities will be the principals and the role names of a DTM system. Our behaviors will correspond to the $RT_0$ policies of the principals.

Scoll and Scollar are explained in dept in "Patterns of Safe Collaboration" [1].

The remainder of this paper is organized as follows. We discuss related work in section 2. In Section 3 we give a quick account on the $RT_0$ language and express the running example in it. We then give an overview of Scoll in Section 4 while translating our example into Scoll. In Section 5 we calculate and interpret the solutions to the running example. We conclude in section 6.

## 2 Related Work

In decentralized trust management [3–6] decisions are made based on statements made by multiple principals. The decision who can be trusted, e.g. to access a resource, is not made by a single principal but takes into account information from multiple principals, i.e. the decision is in part *delegated* to these other principals.

Securely sharing statements made by principals can be achieved by certificates frameworks such as X.509 which provides certified but uninterpreted statements and systems such as SPKI/SDSI [7] which link statements to authorization. The PolicyMaker [5] and KeyNote [6] systems separate trust and security concerns, allowing the specification of trust relationships in the form of assertions by the different principals.

In the RT [3, 4] family of trust management languages principals express their trust policies in the form of relationships between the principal's roles and those of other principals. The use of simple rules and a sequence of increasingly more expressive and complex optional language features allows us to express simple policies easily while also supporting more complex trust relations.

Delegation is very powerful and typically coarse-grained in trust management systems such as RT. Usually one cannot be certain that whoever you are delegating to will know, understand and adhere to your expectations about how they should use these delegated powers. Therefore you cannot be sure that the delegation works in the way you intended. RT in itself does not provide the means to express these intentions, nor to reason about what the policies should be, given your intentions of bounding the eventual authorization.

This problem does not go away if we treat delegation as a permission in itself and allow policies to restrict delegation rights as proposed by [8, 9]. Even if such policies are more refined, the original problem remains: what should these refined policies be, given your intentions to bound the eventual authorization. Moreover, approaches using delegation-as-permission typically require a more elaborate and complex enforcement mechanism.

In [10] a different approach is followed; instead of restricting the delegation, a number of constraints on its consequences are stated explicitly. Cooperative, trusted parties are then expected to help *monitor* these constraints. The approach then calculates a minimal subset of roles whose policy changes must be monitored to guarantee the early detection of constraint violations. Control over the (consequences of the) delegation should then be kept within this group of trusted parties.

In contrast to this monitoring approach, we propose to define a set of cooperating principals and calculate alternative minimal sets of ($RT_0$) policy rules that should be disallowed for these principals, to *avoid* violating the safety constraints.

Certain security analysis problems about safety and reachability were solved in [11]. That work also focusses on calculating bounds for the algorithmical complexity of such problems.

In this paper we restrict ourselves to safety problems. We only take availability constraints into account to make sure that our proposed restrictions do not make the required availability impossible. The Scoll approach is meant for safety analysis and thus calculates minimal sets of policy restrictions. Our approach can therefor not provide real insights about availability, even though that would be very useful in the context of DTM. It is interesting future work to extend Scollar to also calculate minimal DTM policies for this purpose.

## 3 Policies and Safety Concerns in $\mathrm{RT}_0$

In this section we first introduce the basics of the trust management language $\mathrm{RT}_0$, see e.g. [12] for details. Next we introduce the notion of incomplete $\mathrm{RT}_0$ policies and show how our running example can be expressed. Finally we introduce safety concerns for such policies.

In $\mathrm{RT}_0$ *principals* are uniquely identified individuals or processes, denoted by names starting with an uppercase. A principal can define *roles*, which are denoted by the principal's name, followed by the *role name*, separated by a dot. Role names start with a lowercase. For instance "A.role1" denotes the role named "role1" as defined by principal A.

A *credential* is an expression of one of the four types listed and clarified in table 1. A *policy* system is a set of credentials. The policy $\mathrm{A}_{\mathcal{P}}$ of a (group of) principal(s) A is the subset of $S$ defining roles of A.

**Table 1.** The 4 types of $\mathrm{RT}_0$ policy expressions.

| **example** | **type** | **meaning** |
|---|---|---|
| A.r ← B | membership | principal A adds principal B to role A.r |
| A.r ← B.r1 | simple inclusion | A considers every member of B.r1 to be a member of A.r |
| A.r ← A.r1.r2 | linking inclusion | A considers everybody in the r2 role of anybody in A.r1 to be a member of A.r |
| A.r ← B.r1 ∩ C.r2 | intersection | A considers every member of B.r1 who is also a member of C.r2 to be a member of A.r |

**$\mathrm{RT}_0$ Semantics:** Given a system of $\mathrm{RT}_0$ policies, the set of principals that are defined by the system to be members of the role A.r is denoted as $[\![A.r]\!]$ (see [12] for a formal definition). We will use this notation when expressing safety requirements about an $\mathrm{RT}_0$ system.

When checking safety requirements we will need to distinguish fixed or controllable parts of the policy and parts of which we cannot be sure. To this end we add a classification to the principals. We refer to the resulting system as an incomplete $\mathrm{RT}_0$ system to emphasise that to address safety concerns we will need to consider extensions of the system.

**Definition 1 (Incomplete $\mathrm{RT}_0$ system).** *An* Incomplete $\mathrm{RT}_0$ system is a $\mathrm{RT}_0$ policy system *together with a labeling which assigns to each participant one of the following three labels:*

- *label* $\mathsf{k}$ *for the principals whose policies are static and completely known.*
- *label* $\mathsf{c}$ *for the principals whose policy changes we can control and bound if necessary.*
- *label* $\mathsf{u}$ *for the principals whose policies are not completely known or can change beyond our control*

*We use $P_l$ to denote all principals with label $l$. An* extension *of the system is obtained by adding credentials for $P_c$ or $P_u$ (but not $P_k$).*

Table 2 shows our running example as an incomplete $\mathrm{RT}_0$ system with three principals: Conference, Alice, and Bob.

**Table 2.** An incomplete $\mathrm{RT}_0$ policy system.

| **Principal** | **label** | **rule** | **nr.** |
|---|---|---|---|
| **Conference** | k | Conference.reviewer ← Alice | 1 |
| | | Conference.reviewer ← Conference.reviewer.reviewer | 2 |
| | | Conference.submitter ← Conference.reviewer. submitter | 3 |
| **Alice** | c | Alice.submitter ← Bob | 4 |
| | | Alice. reviewer ← Alice. reviewer. reviewer | 5 |
| **Bob** | u | | |

Conference's label indicates that her policy is fixed and stable as described in the first three rules of table 2. In rule 1 Conference adds Alice to Conference.reviewer. In rules 2 and 3 Conference delegates the authorization decisions about both her roles to members of her reviewer role.

Alice's label indicates that her policy changes can be controlled. In rule 4 she adds Bob to Alice.submitter. Rule 5 states that Alice allows her reviewers to make authorization decision about Alice.reviewer, just as Conference did in rule 2. Bob will become a member of Conference.submitter via the combined effects of rules 1, 3 and 4.

Bob's label indicates that we have no definite knowledge about Bob's policy and/or we cannot restrict his policy changes.

**Safety and availability concerns:** When defining roles we have will have certain restrictions on who is allowed to be in what role. These restrictions can be expressed by constraints on the roles, see e.g. [10]. Here we consider two types of constraints. The first are *Safety* constraints which are expressions of the form $[\![A_1^1.r_1^1]\!] \cap \ldots \cap [\![A_n^1.r_n^1]\!] \cup \ldots \cup [\![A_1^m.r_1^m]\!] \cap \ldots \cap [\![A_k^m.r_k^m]\!] \subseteq \emptyset$.

The safety requirement in our running example is: mutual exclusion between Conference's reviewer and submitter roles. That can be expressed as:
$[\![\mathit{Conference.reviewer}]\!] \cap [\![\mathit{Conference.submitter}]\!] \subseteq \emptyset$

The second type of constraints are availability requirements which are expressions of the form $[\![A_1^1.r_1^1]\!] \cap \ldots \cap [\![A_n^1.r_n^1]\!] \cup \ldots \cup [\![A_1^m.r_1^m]\!] \cap \ldots \cap [\![A_k^m.r_k^m]\!] \supset \emptyset$.

That at least one principal should be in Conference.submitter can be expressed as: $[\![\mathit{Conference.submitter}]\!] \supset \emptyset$.

**Definition 2.** *Given an incomplete policy system $\mathcal{P}$, a set of safety and a set of availability constraints we say that a set of credentials $R$ for roles of principals in $P_c$ (called a restriction) is a* solution *if*

– *any extension of $\mathcal{P}$ not containing credentials in $R$ satisfies the safety constraints.*
– *there exists an extension of $\mathcal{P}$ not containing credentials in $R$ which satisfies all the availability constraints.*

*We say a solution is* optimal *if any strict subset of $R$ is not a solution.*

In the next section we will see how we specify incomplete $RT_0$ systems and safety and availability concerns in Scoll. After that we will show how to find optimal solutions.

## 4 Modeling DTM safety problems in Scoll

In this section we will give an intuition about Scoll's syntax and semantics while we show how the running example can be modeled.

Scoll is based on DataLog [13] and was designed to automate reasoning about the potential effects that can be caused by the (inter-)actions of entities in a system, and to calculate what limitations (to the system and/or the entities) are necessary and sufficient to avoid all unwanted effects (safety) without preventing any wanted effects (availability). For a detailed account on Scoll we refer to [1].

Scoll programs involve a static and finite set of *subjects*. Every subject conservatively models a (possibly dynamic and infinite) set of actual entities. To model our running example we have chosen to represent all the potential reviewers with a single subject Alice, and all potential submitters with a single subject Bob. Aggregating entities this way is a valid approach when analyzing safety, but it may represent an over approximation. This means that the policy restrictions we will calculate are guaranteed to be sufficient but may be refined in situations where not all reviewers are supposed to have the same policy. While Scoll provides support for iterative and selective refinement, we will not use this feature here as we don't need it to clarify our contributions.

**Core Syntax Features:** In Scoll all predicate labels and subject constants start with a lowercase letter. Variables range over all subjects, and start with a uppercase letter. Predicate labels can contain dot characters to increase readability. Behavior types are denoted in all capitals.

Figure 1 shows how we expressed the running example in Scoll. We can distinguish six parts in the Scoll program, indicated by keywords in bold. Each part will now be discussed in detail.

### 4.1 Part 1: declare

The first part declares the labels and arities of the predicates over the subjects in the program (see Figure 1). Scoll differentiates between three kinds of predicates:

**state** predicates modeling the security state,
**behavior** predicates modeling the intentions subjects can have, and

```
declare
    state:          canActAs/3 shareMember/3
    behavior:       member/3 incl/4 link/4 intersect/6
    knowledge:
system
    /* Simple Member */
    A:member(R1,B) => canActAs(B,A,R1);
    /* Simple Inclusion */
    A:incl(R1,B,R2) canActAs(C,B,R2) => canActAs(C,A,R1);
    /* Linking Inclusion */
    A:link(R,R1,R2) canActAs(B,A,R1) canActAs(C,B,R2) => canActAs(C,A,R);
    /* Intersection */
    A:intersect(R,B1,R1,B2,R2) canActAs(C,B1,R1) canActAs(C,B2,R2)
     => canActAs(C,A,R);
    /* Mutex */
    canActAs(A,B,R1) canActAs(A,B,R2) => shareMember(B,R1,R2);
behavior
    NONE {}
    UNKNOWN {       => member(_,_) incl(_,_,_) link(_,_,_) intersect(_,_,_,_,_);}
    CONFERENCE { isAlice(X) isReviewerRole(R)=> member(X,R);
                 isReviewerRole(R) => link(R1,R,R1);}
subject
    ? alice: NONE
    bob: UNKNOWN
    conference: CONFERENCE
    reviewer: NONE
    submitter: NONE
config
    conference:isAlice(alice) conference:isReviewerRole(reviewer)
goal
    ! shareMember(conference,reviewer,submitter)
    canActAs(bob,conference,submitter)
    canActAs(alice,conference,reviewer)
```

**Fig. 1.** Running example : an $RT_0$ based trust problem in Scoll

**knowledge** predicates modeling the internal state of subjects: what a subject can "know" or "learn" about the system and about the other subjects.

The *state predicates* for our running example are clarified in table 3. They will be used in the system part (Section 4.2) and in the goal part (Section 4.6).

**Table 3.** state predicates

| **predicate** | **example** | **meaning** |
|---|---|---|
| canActAs/3 | canActAs(a,b,r1) | $A \in [\![ B.r1 ]\!]$ |
| shareMember/3 | shareMember(a,r1,r2) | $[\![ A.r1 ]\!] \cap [\![ A.r2 ]\!] \neq \emptyset$ |

The canActAs predicate expresses role membership and is scenario independent. For each safety or availability constraint we add a predicate capturing violation of the constraint, such as shareMember/3 for the mutual exclusion constraint. If the constraint concerns a single role as in $[\![ Conference.submitter ]\!] \supset \emptyset$ we can omit the extra predicate as we can already express role membership.

*Behavior predicates* express the behavior of the subject in the first argument of the predicate. Similarly, knowledge predicates express knowledge available to the subject in the first argument. To emphasize this, behavior and knowledge predicates will be denoted with their first argument in front of the predicate label, separated by a colon. For example we use conference:member(reviewer,alice) rather than member(conference, reviewer, alice) to make it clear that this is a predicate on conference's behavior.

The behavior predicates of Figure 1 are clarified in table 4. They correspond exactly to the $RT_0$ policy expressions of section 3. Instead of representing credentials (e.g. as in [12]), here they represent the authorization intentions of an issuer of credentials: his $RT_0$ policy.

**Table 4.** behavior predicates

| **predicate** | **example** | **meaning in $RT_0$** |
|---|---|---|
| member/3 | a:member(r,b) | $A.r \leftarrow B$ |
| incl/4 | a:incl(r,b,r1) | $A.r \leftarrow B.r1$ |
| link/4 | a:link(r,r1,r2) | $A.r \leftarrow A.r1.r2$ |
| int/6 | a:int(r,b,r1,c,r2) | $A.r \leftarrow B.r1 \cap C.r2$ |

Notice that we did not provide a behavior predicate to express the actual use of a role by a subject. Scoll is very suitable for modeling usage behavior as well, but we will not explore that in this paper.

*Knowledge predicates* model what entities can learn from their own successful behavior. This knowledge can be used in behavior rules (Section 4.3). We will only use static, subject specific knowledge that can be declared in the config part (Section 4.5).

### 4.2 Part 2: system

This part contains the *system rules*: DataLog rules that conservatively and monotonically model *all* the mechanisms by which subject behavior can result in changes to the security state as represented by the state predicates.

All Scoll rules use a notation that is closer to logics than to logic programming: the conditions are to the left and the conclusions to the right of a logical implication sign "=>". To encourage correct conservative approximations, Scoll allows only variables in system rules. Knowledge of identity will be modeled explicitly with static, subject specific predicates in Section 4.5.

System rules typically include behavior predicates in their conditions to express that a subject's cooperation is a necessary condition to the state change. We refer to [2] for an explanation on how this approach can model discretionary access control. In our example the four types of $\mathrm{RT}_0$ credentials each appear as a behavior condition in a system rule.

The first four system rules in Figure 1 should now be self explanatory. For every behavior predicate there is a rule that states the conditions in which a subject's behavior affects the security state. These four rules have similar effects: canActAs() facts are added to the security state.

To these scenario independent rules we add a rule capturing the meaning of each of the predicates used for the constraints: the last system rule derives a state predicate that will be used later to detect a breach of mutual exclusion:

```
canActAs(A,B,R1) canActAs(A,B,R2) => shareMember(B,R1,R2);
```

**Remark:** In the actual Scoll model of this problem we added some type restrictions to the conditions in the system rules, using unary state predicates that are not shown here. Their only effect is in speeding up the calculation and avoiding variable bindings that do not make sense. We did not show them here, to avoid cluttering up the example.

### 4.3 Part 3: behavior

*Behavior rules* are DataLog rules that express in what conditions a subject is ready to show what behavior. The first argument is dropped in every predicate of a behavior rule: it *implicitly* refers to the subject who's behavior is described.

Two standard behaviors are NONE and UNKNOWN which respectively model principals which will issue no credentials at all or freely issue any of the possible credentials. The latter is how we model unknown entities conservatively: as subjects that always show every possible behavior towards all other subjects. Notice the use of anonymous variables indicated with underbar "_".

In addition we have scenario specific behaviors. For each principal with a $k$ label we define a corresponding behavior; i.e. a behavior which issues the credentials in their (fixed) policy. The CONFERENCE behavior type has two rules:

isAlice(X) isReviewerRole(R) => member(X,R); This is the way in which rule 1 of table 2 is expressed in Scoll. Basically we are saying that someone with this behavior makes Alice a member of their reviewer role. However, as no constants are allowed in Scoll behaviors, we introduce local knowledge predicates isAlice/2 and isReviewerRole/2 describing these values and initialize them in them in the **config** part (Section 4.5).

isReviewerRole(R) => link(R1,R,R1); Here we have used a shorthand. Rather than defining two rules, one for reviewer and one for submitter roles we link any role $R1$ thus capturing both rules 2 and 3 of table 2 in a single Scoll rule.

### 4.4 Part 4: subject

Every subject is listed in this part, and assigned a behavior type from the previous part. The behavior type should reflect the trust we have in the entity to not engage in any behavior other than specified in the rules of the behavior type.

? alice:NONE The question mark before alice indicates that we want to find out how we far we can safely extend alice's behavior, starting from the NONE behavior type. All principals with a $c$ label should be marked like this.

bob:UNKNOWN To safely approximate bob's behavior we assume the worst.

conference:CONFERENCE Subject conference has behavior CONFERENCE

reviewer:NONE Subject reviewer is a role name and has no behavior

submitter:NONE Subject submitter is a role name and has no behavior

### 4.5 Part 5: config

This part defines the initial configuration: a list of all state facts in the initial security state and all knowledge facts in the initial subject states. In Figure 1 this part initializes the private knowledge of subject conference.

### 4.6 Part 6: goal

The final part of a Scoll program is the "goal" part. It lists the facts that should *not* become true (safety requirements) preceded by an exclamation mark, and the facts that should become true (availability requirements) without an exclamation mark.

In the example we want one fact to *not* become true:
shareMember(conference, reviewer, submitter).

This goal corresponds to the mutual exclusion constraint: nobody should have both the reviewer role and the submitter role for this conference. Conservative modeling should guarantee that the safety properties satisfied in the Scoll model also hold in the actual system.

The availability goals are added to avoid solutions that restrict Alice's policy so much that there is no way for Bob to be in Conference.submitter, or for Alice to be in Conference.reviewer.

## 5 Scollar finds solutions for DTM safety problems

The Scoll program in Figure 1 expresses a mutual exclusion problem combined with basic availability requirements. Achieving mutual exclusions is generally a difficult problem in trust management systems. For example, in the RT family of languages a special construction (manifold roles [3]) is needed. In [10] mutual exclusion is monitored and detected early, rather than prevented, by introducing constraints and keeping control within a group of trusted, cooperating agents.

We turned our mutual exclusion constraint into a detectable state-predicate (Section 4.2), of which a particular fact should be avoided (Section 4.6).

As explained in [14], Scollar uses constraint programming to calculate the minimal sets of behavior restrictions that guarantee the safety requirements without preventing the availability requirements. By listing the ways in which Alice's policies can be restricted no more than necessary to achieve our safety goals, without preventing our availability goals, Scoll will tell us what the boundaries to Alice's allowed policies are.

When presented with the problem of Figure 1, Scollar finds two solutions (Figure 2) that minimize the restrictions on Alice's policy. To keep the table within reasonable size for a good overview, we removed the 6-ary predicate intersection() from the calculations.

| | | solution number | 1 | 2 |
|---|---|---|---|---|
| 1 | alice:member( | reviewer,alice) | | 0 |
| 2 | | reviewer,bob) | 0 | 0 |
| 3 | | submitter,alice) | 0 | 0 |
| 4 | alice:incl( | reviewer,alice,submitter) | 0 | 0 |
| 5 | | reviewer,bob,reviewer) | 0 | 0 |
| 6 | | reviewer,bob,submitter) | 0 | 0 |
| 7 | | submitter,alice,reviewer) | 0 | |
| 8 | | submitter,bob,reviewer) | 0 | 0 |
| 9 | | submitter,bob,submitter) | 0 | 0 |
| 10 | alice:link( | reviewer, reviewer,submitter) | 0 | |
| 11 | | reviewer,submitter,reviewer) | 0 | 0 |
| 12 | | reviewer, submitter, submitter) | 0 | 0 |
| 13 | | submitter, reviewer, reviewer) | 0 | |
| 14 | | submitter,submitter,reviewer) | 0 | 0 |
| 15 | | submitter, submitter,submitter) | 0 | 0 |

**Fig. 2.** Overview of the 2 possible alternatives for restricting Alice's $RT_0$ policy (excluding the intersection statements).

The table in Figure 2 contains a row for every behavior fact (policy , see table 1) of Alice that is to be avoided in at least one of the two solutions. If the expression is to be avoided in a solution, it is indicated as a zero in the column representing this solution.

Let us first check the lines that contain 0 for both solutions. In no circumstances should Alice add the corresponding $\mathrm{RT}_0$ credentials to her policy.

- line 2: Alice.reviewer $\leftarrow$ Bob
  Alice should never make Bob a member of Alice.reviewer because, since the conference's roles are delegated to Alice, that would immediately violate the mutual exclusion constraint.
- line 3: Alice.reviewer $\leftarrow$ Alice,
  Alice should never make herself member of her submitter role (line 3) because, since the conference's roles are delegated to Alice, that would immediately violate the mutual exclusion constraint.
- line 4: Alice.reviewer $\leftarrow$ Alice.submitter
  Alice should never include here submitter role in her reviewer role.
- lines 5 and 6:
  Alice.reviewer $\leftarrow$ Bob.reviewer,
  Alice.reviewer $\leftarrow$ Bob.submitter
  Alice should never include any of Bob's roles in her reviewer role.
- lines 8 and 9:
  Alice.submitter $\leftarrow$ Bob.reviewer,
  Alice.submitter $\leftarrow$ Bob.submitter
  Alice should never include any of Bob's roles in her submitter role either.
- lines 11,12,14 and 15:
  Alice.reviewer $\leftarrow$ Alice.submitter.reviewer,
  Alice.reviewer $\leftarrow$ Alice.submitter.submitter,
  Alice.submitter $\leftarrow$ Alice.submitter.reviewer,
  Alice.submitter $\leftarrow$ Alice.submitter.submitter,
  Alice should never link any of her roles via her submitter role.

Solution 1 allows Alice to include herself to her own reviewer role (line 1), at the cost of further restricting the delegation via that role (lines 7, 10, and 13). Solution 2 represents the only alternative.

For improved understanding of the results, Scoll allows the user to check out the individual solutions in detail. The user then gets a complete overview showing the state, knowledge and behavior facts that would become true for every entity.

The solutions that are found in Scoll correspond to the optimal solutions (Definition 2).

**Theorem 1 (Correctness and completeness).** *Given an incomplete policy system with a set of safety and availability constraints and the Scoll program modeling the system and constraints as described in Section 4 we have that:*

- *Any restriction set calculated by Scollar is an optimal solution.*
- *Any optimal solution is found by Scollar.*

## 6 Conclusions and Future Work

We have shown that trust management research, particularly in DTM, can benefit from general techniques for safety analysis, in particular from analysis techniques that can model entity behavior.

We have shown how authorization and delegation policies can be modeled as subject behavior in the Scoll language, and how such models can be used to calculate how the cooperating principals can limit their policies to bound their direct and indirect consequences in the presence of unknown policies.

We have applied the Scollar tool to calculate the ways to restrict a principal's policy no more than necessary to avoid unwanted authorizations effects.

We have shown that Scollar can also take availability requirements into account when calculating the necessary restrictions. Even if these availability requirements are not guaranteed in a system of which the Scoll program is a conservative model, they are useful to detect and avoid solutions that would only model systems that cannot possibly comply to the availability requirements.

The advantages of Scoll and Scollar thus become available in the domains of Trust Management as well as Security research:

– The state predicates, behavior predicates and knowledge predicates can be chosen to model the effects and influences relevant for TM systems.
– The system rules can be chosen in accordance with the protection system that controls the modeled systems.
– The behavior types can be modeled in accordance with the relevant assumptions, trust, and knowledge about the entities or principals in the system.
– The detail of modeling can be adjusted to the requirements, and adapted for different parties in the same model. Scoll supports mechanisms for refinement of state, knowledge, and behavior.

**Future Work:** Since DTM requirements include proving availability as well as safety, we intend to adapt Scollar in the near future so that it supports availability and safety equally well.

We could consider modeling use-behavior as well in Scoll, should we want to bound the role *activations* of the cooperating principals, or guarantee dynamic mutual exclusion constraints.

The problem modeled in this paper is relatively simple and calculates a-priori properties and trust requirements for the cooperating principals . Future work may also focus on applying the proposed method to analyze trust management and usage control policies in a runtime system, during (updates in) actual delegation, authorization, and use. Future applications may for instance provide for dynamic adaptation of authorization, delegation, and use policies in accordance to knowledge gained from a-posteriori auditing or reputation systems.

The TAS3 project develops trusted architectures for shared services in domains such as healthcare and employability. This architecture implements trust policies which can depend both on structural and behavioral rules.

The Poseidon project, which conducts research on secure interoperation in ad hoc coalitions of heterogeneous parties in the maritime domain, could consider applying the approach and improving its scalability to match their demands for safety and trust analysis.

Scoll is available as open source at http://www.scoll.evoluware.eu, in the hope of attracting researchers and developers to help boost the scalability of the tool to the level necessary for more demanding research.

## References

1. Spiessens, F.: Patterns of Safe Collaboration. PhD thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium (2007)
2. Spiessens, F., Van Roy, P.: A Practical Formal Model for Safety Analysis in Capability-Based Systems. In: TGC 2005. Volume 3705 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (2005) 248–278
3. Li, N., Mitchell, J., Winsborough, W.: Design of a Role-based Trust-management Framework. In: Proc. IEEE Symposium on Security and Privacy, IEEE Computer Society Press (2002) 114–130
4. Artz, D., Gil, Y.: A survey of trust in computer science and the semantic web. Journal of Web Semantics (2007)
5. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In Press, I.C.S., ed.: Proc. 1996 IEEE Symposium on Security and Privacy. (1996) 164–173
6. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The KeyNote trust-management system, version 2. IETF RFC 2704 (1999)
7. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI Certificate Theory. IETF RFC 2693 (1999)
8. Kagal, L., Cost, S., Finin, T., Peng, Y.: A framework for distributed trust management. In: Proc. of IJCAI-01 Workshop on Autonomy, Delegation and Control. (2001)
9. Kagal, L., Cost, S., Finin, T., Peng, Y.: A framework for distributed trust management. In: Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control. (2001) http://citeseer.nj.nec.com/kagal01framework.html.
10. Etalle, S., Winsborough, W.H.: Integrity constraints in trust management (extended abstract). In Ahn, G.J., ed.: 10th ACM Symp. on Access Control Models and Technologies (SACMAT), Stockholm, Sweden, New York, ACM Press (2005) 1–10
11. Li, N., Mitchell, J.C., Winsborough, W.H.: Beyond proof-of-compliance: security analysis in trust management. J. ACM **52** (2005) 474–514
12. Czenko, M.R., Etalle, S., Li, D., Winsborough, W.H.: An introduction to the role based trust management framework RT. Technical Report TR-CTIT-07-34, University of Twente, Enschede (2007)
13. Gallaire, H., Minker, J., eds.: Logic and Data Bases. Perseus Publishing (1978)
14. Spiessens, F., Jaradin, Y., Van Roy, P.: Using Constraints To Analyze And Generate Safe Capability Patterns. Research Report INFO-2005-11, Département d'Ingénierie Informatique, Université catholique de Louvain, Louvain-la-Neuve Belgium (2005) Presented at CPSec'05. Available at `http://www.info.ucl.ac.be/~fsp/rr2005-11.pdf`.