Nonmonotonic Trust Management for P2P Applications

Marcin Czenko [1,3], Ha Tran [4], Jeroen Doumen [1,5], Sandro Etalle [1,2,6],
Pieter Hartel [7], Jerry den Hartog [2,8],

Department of Computer Science
University of Twente
Enschede, The Netherlands

Abstract

Community decisions about access control in virtual communities are non-monotonic in nature. This means that they cannot be expressed in current, monotonic trust management languages such as the family of Role Based Trust Management languages (RT). To solve this problem we propose $RT_\ominus$, which adds a restricted form of negation to the standard RT language, thus admitting a controlled form of non-monotonicity. The semantics of $RT_\ominus$ is discussed and presented in terms of the well-founded semantics for Logic Programs. Finally we discuss how chain discovery can be accomplished for $RT_\ominus$.

Key words: Distributed Trust Management (DTM), Virtual Communities (VC), Peer to Peer (P2P), Role Based Trust Management (RT), Non-monotonic Policies, Chain Discovery.

1 Introduction

Languages from the family of Role Based Trust Management Framework (RT), like most Trust Management (TM) languages are monotonic: adding a credential to the system can only result in the granting of additional privileges. Usually, this property is desirable in policy languages [23]. However, banishing negation from an access control language is not a realistic option. In fact, as stated by Li et al. [16] "many security policies are non-monotonic, or more easily specified as non-monotonic ones"; similar views are expressed by Barker and Stuckey [2] and by Wang et al. [26] in the context of logic-based access control. This is also true for complex distributed systems such as virtual communities. In particular, as we will show, modeling access control decisions by a community, as opposed to access control decisions by an individual member, cannot be made without at least a form of negation, which we call negation-in-context. As pointed out by Dung and Thang [6] a TM system should be monotonic with respect to the credential submitted by the client but could be non-monotonic with respect to the site's local information about the client. Our extension allows a TM system to be non-monotonic not only in a local setting, but also when the context for negation can be provided.

Contributions

We present a significant enhancement to the power of the RT family of trust management languages by proposing $RT_\ominus$, an extension of RT_0. More specifically we:

- add a single new statement type adding negation-in-context to standard RT;
- present and discuss the declarative semantics of $RT_\ominus$;
- show that the extension is essential to specify access control policies for virtual communities.
- describe a chain discovery algorithm for $RT_\ominus$.

Currently, we are using $RT_\ominus$ to specify and implement virtual community packages in the context of the Freeband project I-SHARE. In the next section we discuss how access control policies in virtual communities motivate us to add negation-in-context to RT. In Section 3 the syntax and informal semantics of $RT_\ominus$ is introduced. The formal semantics of $RT_\ominus$ is presented in Section 4. We present related work in Section 7 and conclusions and future work in Section 8.

2 Virtual Communities

Virtual communities are groups of individuals with a shared interest, relationship or fantasy [15]. The majority of current virtual communities is interested in sharing audio/video content using P2P systems [21]. Taking into account the distributed nature of virtual communities, special mechanisms for access control must be provided to ensure secure operations at both intra- and inter-community levels. As it is often impossible to identify strangers [20], trust must be established between community members and entities from outside the community prior to allowing a specific access.

As an example imagine that Alice (A), Bob (B), and Carol (C) decide to form a virtual community (or just a community for short). At the beginning they are the only members of the community, but they welcome others to join. We represent a community by a list with an entry for each member. Each entry names the community member and the members it knows about. For example the following list represents the community of Alice, Bob, and Carol:

$$A[B, C] \ B[A, C] \ C[A, B]$$

In this community all members know each other. As the community grows it becomes harder and harder for each member to have complete information about all other members. Yet the community would like to protect its integrity. Rather than to require involvement of all members in decision making, a more practical and scalable approach is to allow decisions about membership to be taken by a group of coordinators selected from the community members. This group of coordinators itself forms a (sub)community. To find all the coordinators we require that each coordinator has a relationship with *at least* one other coordinator in such a way that all coordinators can be reached. For example in the list below A knows B, B knows C and C knows B and A:

$$A[B] \ B[C] \ C[B, A]$$

To become a member of a community or to become a new coordinator *all* the existing coordinators of a given community must approve. Trust management languages based on logic programming semantics do not support queries of this kind directly. If one wants to know "if all coordinators approve entity A" without explicitly enumerating these coordinators, one must check if the *negation* of this statement - "is there any coordinator that does not approve entity A" - holds. If not, one can conclude that all coordinators approve entity A. Existing trust management languages [17] are strictly monotonic, thus do not allow for negation. For this reason they are not sufficiently expressive to efficiently model complex collaborations that commonly appear in virtual communities.

Before we can elaborate on this using the example just presented, we need to review the definition of RT_0, and then present our extension $RT_\ominus$.

3 $RT_\ominus$

We first summarise and then extend the RT_0 language.

3.1 The RT_0 language

RT_0 contains two basic elements: *entities* and *role names*. Entities represent uniquely identified principals, individuals, processes, public keys, etc. Entities are denoted by names starting with an uppercase letter, for example: A, B, D, and $Alice$. A role name begins with a lower case letter. In RT_0, roles are denoted by the entity name followed by the role name, separated by a dot. For instance $A.r$ and Company.testers are roles. To define role membership, RT_0 provides four kinds of policy statements:

- $A.r \longleftarrow D$ *(Simple Membership)*. Entity D is a member of the role $A.r$.
- $A.r \longleftarrow B.r_1$ *(Simple Inclusion)*. Every member of $B.r_1$ is also a member of $A.r$. This represents delegation from entity A to entity B.
- $A.r \longleftarrow A.r_1.r_2$ *(Linking Inclusion)*. For every entity X who is a member of $A.r_1$, every member of $X.r_2$ is also a member of $A.r$. This statement represents a delegation from entity A to all the members of the role $A.r_1$. The right-hand side $A.r_1.r_2$ is called a *linked role*.
- $A.r \longleftarrow B_1.r_1 \cap B_2.r_2$ *(Intersection Inclusion)*. Every entity which is a member of both $B.r_1$ and $B.r_2$ is a member of $A.r$. This statement represents partial delegation from the entity A to B_1 and to B_2. The right-hand side $B_1.r_1 \cap B_2.r_2$ is called an *intersection* role. In a policy statement $A.r \longleftarrow e$ we call $A.r$ the head and e the body. The set of policy statements having the same head $A.r$ is called the *definition* of $A.r$.

3.2 Extending RT_0 with negation

RT_0 and other languages from the RT framework do not support negation. As argued in Section 2, this limits expressiveness. Let us first see an example of negation to enforce the following separation of concerns policy: "developers cannot be testers of their own code". We would like to express in *RT* something similar to the LP clause:

$$\text{verifycode}(?A) \text{ :- tester}(?A), not \text{ developer}(?A).$$

where $?A$ denotes a logical variable. This clause states that A can verify the code if A is a tester and A is not the developer responsible for the code. RT^{DT} - another member of the RT framework [17] - supports thresholds and delegation of role activations; to some extent, RT^{DT} allows to model separation of concerns without using negation. However, this comes at the cost of having to define manifold roles (cumbersome to work with, in practice). In any case, the examples we present in the sequel cannot be modeled in RT^{DT}. We define a new type of statement based on RT_0 and a new role-exclusion operator $\ominus$:

- $A.r \longleftarrow B_1.r_1 \ominus B_2.r_2$ *(Exclusion)* All members of $B_1.r_1$ which are not members of $B_2.r_2$ are added to $A.r$.

Example Using the $\ominus$ operator we can solve the separation of concerns problem as follows:

$$\text{Company.verifycode} \longleftarrow \text{Company.tester} \ominus \text{Company.developer.} \qquad (1)$$

Suppose that both *Alice* and *Bob* are testers but Alice is also a developer of the code:

$$\text{Company.tester} \longleftarrow \text{Alice} \quad \text{Company.tester} \longleftarrow \text{Bob} \quad \text{Company.developer} \longleftarrow \text{Alice}$$

We see that credential 1 does not make Alice be a member of the Company.verifycode role. Thus, only Bob can verify the code.

3.3 Modeling virtual communities using $RT_\ominus$

Having given a simple example and its representation in $RT_\ominus$, we now return to the more complex scenario of community decision making from Section 2.

Recall that we have a community of coordinators - *Alice* (A), *Bob* (B), and *Carol* (C). Assume that another entity - say D - wants to join this community and asks *Alice* for approval. *Alice* can accept D as a new coordinator locally, but before making the final decision she must check if there is no objection from other coordinators. A coordinator expresses the

objection using a so called *black list*. An entity that is on the black list of one of the coordinators will not be accepted as a new coordinator.

Table 1

Roles used by coordinators

Definition (for coordinator A)	Description	Op-tio-nal
A.agreeToAdd ⟵ [set of entities]	A coordinator uses this role to express that she approves an entity. The role has a local meaning. It is not sufficient to be a member of the *agreeToAdd* role to become a coordinator. It is necessary that no other coordinators says that an entity is a member of her *disagreeToAdd* role. The *agreeToAdd* role, through the *allCandidates* role, provides context for the ⊖ operator in the definition of the the *addCoord* role.	
A.disagreeToAdd ⟵ [see description in the text]	This role is used by a coordinator as a black list.	
A.coord ⟵ [set of entities]	This role contains all the coordinators known by a coordinator.	
A.allCoord ⟵ A A.allCoord ⟵ A.allCoord.coord	This role allows a coordinator to iterate over all entities connected by the *coord* role. This role, if defined, contains all the coordinators.	✓
A.objectionToAdd ⟵ A.allCoord.disagreeToAdd	A coordinator can use this role to obtain all entities for which there is any objection.	✓
A.allCandidates ⟵ A.allCoord.agreeToAdd	This role, if defined, contains all the candidate coordinators locally accepted by any of the coordinators. Used as the context for the ⊖ operator in the body of the *addCoord* role.	✓
A.addCoord ⟵ A.allCandidates ⊖ A.objectionToAdd	After becoming a member of this role, a candidate coordinator becomes a new coordinator and becomes a member of the *coord* role.	✓

Table 1 shows the minimal definition, and the descriptions of the roles used by coordinators. We see from Table 1 that some roles are mandatory while the others are not. For instance the role *disagreeToAdd* must be defined by each coordinator. On the other hand, the roles *allCoord*, *allCandidates*, and *addCoord* can be defined as needed by a coordinator. Special attention must be given to the definition of the *disagreeToAdd* role. For example, a coordinator can use the following credential to say that she distrusts any entity she does not accept locally:

$$A.\text{disagreeToAdd} \longleftarrow A.\text{allCandidates} \ominus A.\text{agreeToAdd}.$$

If a coordinator trusts other coordinators to select candidates she can leave the *agreeToAdd* role empty and use her *disagreeToAdd* role to block some candidates. For example, *Alice* can put E on her black list to disallow E to become a coordinator, and simultaneously accept all other candidates proposed by other coordinators:

$$A.\text{disagreeToAdd} \longleftarrow E.$$

Table 2 shows the roles and their members as seen by Alice, Bob, and Carol. In this table, we assume that Alice agrees locally to add D as a new coordinator. Also, Bob and Carol have no objection to add D as a new coordinator, but E is on Alice's black list and F is on the black list of Bob and Carol. As a consequence, only D is the member of the *addCoord*

4

role of Alice. Bob and Carol do not have to define the *allCoord*, *allCandidates* [9] , *objectionToAdd*, and *addCoord* unless they themselves add a new coordinator.

Table 2
Adding new coordinator - D is successful, E, F fail

	coord	agreeToAdd	allCoord	allCandidates	disagreeToAdd	objectionToAdd	addCoord
Alice (A)	{B}	{D}	{A,B,C}	{D}	{E}	{E,F}	{D}
Bob (B)	{C}	{}	Not Defined	Not Defined	{F}	Not Defined	Not Defined
Carol (C)	{B,A}	{}	Not Defined	Not Defined	{F}	Not Defined	Not Defined

4   Semantics

The semantics of trust management languages is typically given by a translation into Logic Programming (LP) [17]. We will follow the same route. Trust management credentials are by definition distributed among different principals. The use of negation creates an additional difficulty, also because in logic programming there various different semantics exist to cope with negation. We now sketch some of the reasons why we refer to the Well-Founded semantics. The first reason is syntactic: in a TM system it is impossible to avoid circular references, and we cannot expect policies to be (locally) *stratified*. Stratification basically means that one can restructure a logic program into separate parts in such a way that negative references from one part refer only to previously defined parts. Without the possibility of local stratification we cannot refer to the *perfect model semantics* [22]. For the same reason, we certainly have to refer to a *three valued semantics*: next to the truth values *true* and *false*, we have to admit the valued *undefined*. In short, this is because we cannot expect the completion of a policy to be a consistent logic program in the sense described in [24]. A second point is handling of positive circular references, as in $\{A.r \longleftarrow B \quad A.r \longleftarrow B.r \quad B.r \longleftarrow A.r\}$. In this case, in accordance with the semantics of RT_0, we want to be able to say that C does *not* belong to $A.r$. This forces us to exclude Kunen's semantics [14] (i.e. the semantics of logical consequences of the completion of the program together with the weak domain closure assumptions), and Fitting's semantics [8]: in both semantics the query "does C belong to $A.r$?" would return *undefined*. These reasons brought us to adopt the well-founded (WF) semantics for LP [9]. The WF semantics has the advantage that it imposes no restrictions on the syntax of programs, that any prolog program has a *unique* well-founded model associated to it (as opposed to e.g. the stable model semantics [10]), and that the WF semantics enjoys an elegant fixed-point construction.

4.1   Well-founded Semantics

We now introduce the WF semantics [9] for general logic programs. A general logic program is defined as follows:

Definition 4.1 A general logic program is a finite set of clauses of the form: A :- $L_1, \ldots, L_n$.

Here A is an atom (the *head* of the clause) and $L_1, \ldots, L_n$ with $n \geq 0$ are literals forming the *body*. A literal is an atom A (positive literal) or a negated atom $\neg A$ (negative literal). We refer to general logic programs (GLP) simply as programs. A *fact* is a clause with an empty body. The *Herbrand universe* of a program P, denoted by $\mathcal{U}_P$, is the set of all ground terms (i.e. variable-free) constructed from constants and function symbols in P. The *Herbrand base* of P, denoted by $\mathcal{B}_P$, is the set of ground atoms obtained from predicates in P and terms in $\mathcal{U}_P$. An *instantiated clause* of P is a ground clause obtained by substituting terms in $\mathcal{U}_P$ for variables in the clause in P. The *Herbrand instantiation* $Ground(P)$ of P is the set of all instantiated clauses. A set I of ground literals is *consistent* if there is no atom A such that both A and $\neg A$ are in I. An *interpretation* of P is a consistent set of ground literals in $\mathcal{B}_P$. Intuitively, atom A is true in I if $A \in I$, false in I if $\neg A \in I$, and undefined in I if neither $A \in I$ nor $\neg A \in I$.

---

[9] A coordinator must define the *allCandidates* role if she defines the *disagreeToAdd* role in terms of the *agreeToAdd* role.

The well-founded semantics uses *unfounded sets* to derive atoms that are false:

Definition 4.2 Let P be a program, I be an interpretation of P, and U be a subset of $\mathcal{B}_P$. U is an *unfounded set of P with respect to I* if every atom $A \in U$ satisfies the following condition: for every instantiated clause A :- $L_1, \ldots, L_n \in Ground(P)$ whose head is A, either (1) some literal L is false in I or (2) some positive literal L is also in U.

Intuitively, an unfounded set is a set of atoms which can be simultaneously declared false without having to assume anything to be true. The union of unfounded sets is an unfounded set, and the greatest unfounded set of P with respect to I, denoted by $\mathbf{U}_P(I)$, is the union of all unfounded sets of P with respect to I.

Definition 4.3 Let P be a program, I be an interpretation of P. Transformations $\mathbf{T}_P$ and $\mathbf{W}_P$, and the *well-founded semantics* of P are defined as follows:
- $A \in \mathbf{T}_P(I)$ if and only if there is A :- $L_1, \ldots, L_n \in Ground(P)$ such that all literals $L_1, \ldots, L_n$ are true in I;
- $\mathbf{W}_P(I)$ is a union of $\mathbf{T}_P(I)$ and $\neg\mathbf{U}_P(I)$ which contains the negation of each element in $\mathbf{U}_P(I)$: $\mathbf{W}_P(I) = \mathbf{T}_P(I) \cup \neg\mathbf{U}_P(I)$.
- The *well-founded model* of P is the least fixed point of $\mathbf{W}_P(I)$.

Transformations $\mathbf{T}_P(I)$, $\mathbf{U}_P(I)$, and $\mathbf{W}_P(I)$ are monotonic [9]. Let α range over all countable ordinals, the set I_α, whose elements are literals in the Herbrand base of P, is defined recursively by: $I_{\alpha+1} = \mathbf{W}_P(I_\alpha)$. I_α is also a monotonic sequence. $I_\infty = \cup_\alpha I_\alpha$ is the least fixed point of $\mathbf{W}_P$.

4.2 Translating $RT_\ominus$ to GLP

We first give the translation to LP for RT_0 and, using this translation, the semantics of a set of RT_0 policy statements. Next we extend this to a translation from $RT_\ominus$ to GLP and the semantics for a set of $RT_\ominus$ policy statements.

The semantics of a set of RT_0 policy statements is commonly defined by translating it into a logic program [17]. Here, we depart from the approach of Li et al. [17] by referring to the role names as predicate symbols. The statement $A.r \longleftarrow D$ is, for example, translated to $r(A, D)$ in the Prolog program. Intuitively, $r(A, D)$ means that D is a member of the role $A.r$.

Definition 4.4 Given a set $\mathcal{P}$ of RT_0 policy statements, the *semantic program*, $SP(\mathcal{P})$, for $\mathcal{P}$ is the logic program defined as follows (recall that symbols starting with "?" represent logical variables):
- For each $A.r \longleftarrow D \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $r(A, D)$
- For each $A.r \longleftarrow B.r_1 \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $r(A, ?Z)$:- $r_1(B, ?Z)$
- For each $A.r \longleftarrow A.r_1.r_2 \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $r(A, ?Z)$:- $r_1(A, ?Y), r_2(?Y, ?Z)$
- For each $A.r \longleftarrow B.r_1 \cap B.r_2 \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $r(A, ?Z)$:- $r_1(B_1, ?Z), r_2(B_2, ?Z)$

The semantics of a role is defined as follows:

Definition 4.5 Given a set $\mathcal{P}$ of RT_0 policy statements, the semantics of a role $A.r$ is a set of members Z that make the predicate $r(A, Z)$ true in the semantic program: $[\![A.r]\!]_\mathcal{P} = \{Z \mid SP(\mathcal{P}) \models r(A, Z)\}$

We write $SP(\mathcal{P}) \models r(A, Z)$ for a ground atom $r(A, Z)$ if $r(A, Z)$ is true in $SP(\mathcal{P})$. We now extend the translation of RT_0 to that of $RT_\ominus$ by adding the translation of the exclusion rule.

Definition 4.6 Given a set $\mathcal{P}$ of $RT_\ominus$ policy statements, the *semantic program*, $SP(\mathcal{P})$, for $\mathcal{P}$ is the *general* logic program defined as follows:
- For each $A.r \longleftarrow B.r_1 \ominus B.r_2 \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $r(A, ?Z)$:- $r_1(B_1, ?Z), \neg r_2(B_2, ?Z)$
- All other rules are as in definition 4.4.

We also extend the definition of the semantics of a role.

Definition 4.7 Given a set $\mathcal{P}$ of $RT_\ominus$ policy statements, the semantics of a role $A.r$ is a set of members Z that make the predicate $r(A, Z)$ true in the semantic program: $[\![A.r]\!]_\mathcal{P} = \{Z \mid SP(\mathcal{P}) \models r(A, Z)\}$

6

Note that, unlike before, the value of the semantical program may give value 'unknown' for $r(A, Z)$. In this case the agent Z is not considered to be a member of the role, nor of the negated role. The following example shows that care is required when reasoning about polices which have negative circular dependencies.

Example 4.8 Consider a system with entities A, B, C, D, roles $A.r, B.r$ and $C.r$ and the following policy rules:

$$A.r \longleftarrow B.r \ominus C.r \quad C.r \longleftarrow B.r \ominus A.r \quad B.r \longleftarrow D$$

Here D is a member of $B.r$, however, D is not a member of either $A.r$ or $C.r$ (as we do not have a way to decide in which of the two it should go). Note that as a result we have that despite the presence of the rule $A.r \longleftarrow B.r \ominus C.r$ the role $B.r$ can have members that are neither in $A.r$ nor in $C.r$.

The rules for $A.r$ and $C.r$ in the example above are referred to as negative circular dependencies; $A.r$ depends negatively on $C.r$ and $C.r$, in turn, depends negatively on $A.r$. The example shows that care is required when reasoning about policies which have negative circular dependencies.

4.3 Virtual Communities - translation to GLP

Having introduced an example of virtual community decision making in Section 2, its formalism in Subsection 3.3, we now give the GLP semantics of the example. Translating $RT_\ominus$ credentials to GLP is straightforward using the rules presented in Subsection 4.2. For the convenience of the reader we present a complete policy and the corresponding GLP rules in Appendix A. If one asks Alice to add D to the group of coordinators she needs to check if D is a member of the *A.addCoord*. This is equivalent to checking whether *addCoord(A,D)* holds after the translation to GLP. She does this by checking whether D is a logical consequence of the semantic program $SP(\mathcal{P})$ by first finding the semantics of the role *A.addCoord* and checking if it contains entity D. The semantics of the role *A.addCoord* with respect to the program $\mathcal{P}$ is as follows:

$$[\![A.\text{addCoord}]\!]_\mathcal{P} = \{D\}.$$

The semantics of the roles *A.allCandidates* and *A.objectionToAdd* (these roles define the role *A.addCoord*) are shown below:

$$[\![A.\text{allCandidates}]\!]_\mathcal{P} = \{D\} \quad [\![A.\text{objectionToAdd}]\!]_\mathcal{P} = \{E, F\}.$$

The semantics of a role may also be an empty set: $[\![B.\text{agreeToAdd}]\!]_\mathcal{P} = \{\}$.

5 Credential Chain Discovery

In this section we extend the standard chain discovery algorithm to $RT_\ominus$ following the construction of the well-founded semantics. Recall that the definition of a role $A.r$ is the set of all credentials with head $A.r$. We assume that A stores (or at least, is able to find) the complete definition of each of her roles $A.r$, i.e. that the credentials involved are issuer-traceable. The main difficulty in the chain discovery is to obtain that B is not a member of a linked role $A.r.r'$. For this we need to check that every potential member C of $A.r$ does not have B in its role $C.r'$. So who are the potential members of $A.r$? Thanks to negation in context we can provide a reasonable overestimation of this set using chain discovery for RT_0:

Definition 5.1 For a policy $\mathcal{P}$ the context policy $\mathcal{P}+$ is the policy obtained by replacing each credential of the form $A.r \longleftarrow B_1.r_1 \ominus B_2.r_2 \in \mathcal{P}$ by $A.r \longleftarrow B_1.r_1$ and leaving the other credentials unchanged. We call $[\![A.r]\!]_{\mathcal{P}+}$ the *context* of the role $A.r$.

The following lemma relates roles with their contexts.

Lemma 5.2 *For any policy $\mathcal{P}$ and role $A.r$ we have: If $SP(\mathcal{P}) \models r(A, B)$ then $SP(\mathcal{P}+) \models r(A, B)$ and if $SP(\mathcal{P}+) \not\models r(A, B)$ then $SP(\mathcal{P}) \models \neg r(A, B)$.*

The first part of this lemma states that any role is contained in its context, $[\![A.r]\!]_\mathcal{P} \subseteq [\![A.r]\!]_{\mathcal{P}+}$. If $B \notin [\![A.r]\!]_\mathcal{P}$ this means that $r(A, B)$ is undefined or false in $SP(\mathcal{P})$. The second part of the lemma states that if $B \notin [\![A.r]\!]_{\mathcal{P}+}$ it must be the latter, $r(A, B)$ is false in $SP(\mathcal{P})$. In the algorithm below we build a set of credentials $\mathcal{C}$ together with a set of context membership facts $\mathcal{I}+$ and a set of positive and negative membership facts $\mathcal{I}$.

7

Step 1. Initialise $\mathcal{I} = \emptyset, \mathcal{I}+ = \emptyset$ and $\mathcal{C}$ = the definition of role $A.r$.

Step 2. Discover context and credentials (classical chain discovery for $\mathcal{I}+$ and $\mathcal{C}$).

We look for new credentials top down; any credential that could possibly be relevant for role $A.r$ is added to $\mathcal{C}$. We look for the context of $A.r$ bottom up; any fact that can be derived from the credentials that we have found is added to $\mathcal{I}+$. Repeat the following until no changes occur: For each credential of the following form in $\mathcal{C}$:

$\quad$ [$B.r_0 \longleftarrow C$] add $r_0(B,C)$ to $\mathcal{I}+$

$\quad$ [$B.r_0 \longleftarrow C.r_1$] add the definition of $C.r_1$ to $\mathcal{C}$ and add $r_0(B,D)$ to $\mathcal{I}+$ for all $r_1(C,D)$ in $\mathcal{I}+$

$\quad$ [$B.r_0 \longleftarrow C_1.r_1 \cap C_2.r_2$] add the definitions of $C_1.r_1$ and $C_2.r_2$ to $\mathcal{C}$ add $r_0(B,D)$ to $\mathcal{I}+$ whenever $r_1(C_1,D)$ and $r_2(C_2,D)$ in $\mathcal{I}+$.

$\quad$ [$B.r_0 \longleftarrow C.r_1.r_2$] add the definition of $C.r_1$ and, for each $r_1(C,D) \in \mathcal{I}+$, the definition of $D.r_2$ to $\mathcal{C}$. Add $r_0(B,D)$ to $\mathcal{I}+$ whenever for some Y we have $r_1(C,Y)$ and $r_2(Y,D)$ in $\mathcal{I}+$.

$\quad$ [$B.r_0 \longleftarrow C_1.r_1 \ominus C_2.r_2$] add the definitions of $C_1.r_1$ and $C_2.r_2$ to $\mathcal{C}$, add $r_0(B,D)$ to $\mathcal{I}$ for every $r_1(C_1,D)$

Step 3. Discover positive facts in $\mathcal{I}$ (extended chain discovery 1).

We update $\mathcal{I}$ similar to $\mathcal{I}+$ in the previous step, only the last case ($\ominus$) changes. Repeat until $\mathcal{I}$ does not change, for credentials in $\mathcal{C}$ of the following form:

$\quad$ [$B.r_0 \longleftarrow C$] add $r_0(B,C)$ to $\mathcal{I}$

$\quad$ [$B.r_0 \longleftarrow C.r_1$] add $r_0(B,D)$ to $\mathcal{I}$ for all $r_1(C,D)$ in $\mathcal{I}$

$\quad$ [$B.r_0 \longleftarrow C_1.r_1 \cap C_2.r_2$] add $r_0(B,D)$ to $\mathcal{I}$ whenever $r_1(C_1,D)$ and $r_2(C_2,D)$ in $\mathcal{I}$.

$\quad$ [$B.r_0 \longleftarrow C.r_1.r_2$] Add $r_0(B,D)$ to $\mathcal{I}$ whenever for some Y we have $r_1(C,Y)$ and $r_2(Y,D)$ in $\mathcal{I}$.

$\quad$ [$B.r_0 \longleftarrow C_1.r_1 \ominus C_2.r_2$] add $r_0(B,D)$ to $\mathcal{I}$ whenever $r_1(C_1,D) \in \mathcal{I}$ and either $(\neg r_2(C_2,D)) \in \mathcal{I}$ or $r_2(C_2,D) \notin \mathcal{I}+$.

Step 4. Extended chain discovery for negative facts in $\mathcal{I}$: Building an unfounded set $\mathcal{U}$ (extended chain discovery 2)

We search for facts which are useful when negated in $\mathcal{I}$ and try to show that this fact is false by building an unfounded set $\mathcal{U}$ containing the fact. We start with $\mathcal{U} = \emptyset$.

We call an atom $r(X,Y)$ an *unfounded-candidate (UC)* if $r(X,Y) \in \mathcal{I}+$, $r(X,Y) \notin \mathcal{U}$ and $\neg r(X,Y) \notin \mathcal{I}$. (Basically members of the context which are not known to be false.)

Choose a rule $B.r_0 \longleftarrow C_1.r_1 \ominus C_2.r_2 \in \mathcal{C}$ and an entity D such that $r_1(C_1,D) \in \mathcal{I}$ and $r_2(C_2,D)$ founded and add $r_2(C_2,D)$ to $\mathcal{U}$.

$\quad$ For every $r(A,B)$ in $\mathcal{U}$, and every matching rule $A.r \longleftarrow e$ [10] in $\mathcal{C}$ perform:

$\quad$ [$A.r \longleftarrow C$] fail if $C = B$.

$\quad$ [$A.r \longleftarrow C.r_1$] if $r_1(C,B)$ is UC then add $r_1(C,B)$ to $\mathcal{U}$.

$\quad$ [$A.r \longleftarrow C_1.r_1 \cap C_2.r_2$] if $r_1(C_1,B)$ and $r_2(C_2,B)$ both are UCs then choose one to add to $\mathcal{U}$.

$\quad$ [$A.r \longleftarrow C_1.r_1 \ominus C_2.r_2$] if $r_1(C_1,B)$ UC and $r_2(C_2,B) \notin \mathcal{I}$ then add $r_1(C_1,B)$ to $\mathcal{U}$.

$\quad$ [$A.r \longleftarrow C.r_1.r_2$] for all Y with $r_1(C,Y)$ is UC: If $r_2(Y,B)$ UC choose one of $r_1(C,Y)$ and $r_2(Y,B)$ and add to $\mathcal{U}$.

$\quad$ Fail if $\mathcal{U}$ and $\mathcal{I}$ contain any common elements. [11]

$\quad$ $\star$ Do each possible choice in the substep above and add $\neg\mathcal{U}$ to $\mathcal{I}$ if it succeeds.

Repeat steps 3 and 4 until $\mathcal{I}$ remains unchanged.

(End of algorithm.) The algorithm correctly finds the members of the role $A.r$.

Theorem 5.3 (Soundness and completeness) *The output $\mathcal{I}$ of the algorithm satisfies:*

$\quad \forall B : r(A,B) \in \mathcal{I} \iff B \in [\![A.r]\!]_{\mathcal{P}}.$

where $\mathcal{P}$ is the policy containing all credentials and $A.r$ is the role being discovered.

The algorithm follows exactly the construction of the well-founded semantics [12] except that only part of the interpretation is found. Thus clearly $\mathcal{I}$ will be a subset of the well-founded model for $SP(\mathcal{P})$ (giving soundness). However,

---

[10] The number of atoms and matching rules may grow as we add atoms to $\mathcal{U}$.

[11] Of course, in an actual implementation this is checked as soon as an element is added to $\mathcal{U}$.

[12] Steps 3 and 4 construct parts of the transformations $\mathbf{T}_{\mathcal{P}}$ and $\mathbf{U}_{\mathcal{P}}$ respectively.

as the part of the interpretion used basically covers the contexts of $A.r$ and all roles used to define $A.r$ it also covers any membership facts for $A.r$ (giving completeness).

6 Implementation

In the current prototype storage is centralised and we assume that all credentials can be traced by the issuer. In such a case, SLG resolution of XSB prolog can be used to compute answers to queries according to the WF model for $RT_\ominus$ [5]. DLV datalog [7] and the Smodels system [19] can also be used to provide an initial implementation of $RT_\ominus$. The Smodels system [19] provides an implementation of the well-founded and stable model semantics for range-restricted function-free normal programs. The Smodels system allows for efficient handling of non-stratified ground programs. The DLV system [7] is a system for disjunctive logic programs. DLV is capable of dealing with disjunctive logic programs without function symbols allowing for strong negations, constraints and queries. We implemented the program introduced in sub-section 4.3 on three systems: XSB, SMODELS and DLV. To test the performance of the program on these systems, we used two parameters: number of coordinators (Coords) and number of iterations (Iters). The higher the number of coordinators is, the more complex the program is. The program is also executed repeatedly to compare performance more correctly. Table A.1 in the appendix reports the execution time of the program measured by the CPU time obtained. We cannot compare the execution time between XSB and the other two DLV and SMODELS because XSB is the goal-oriented system while DLV and SMODELS build and return the whole model for the program. Because of this XSB is much faster than the other two systems. DLV provides better execution time than SMODELS, especially when the complexity of the program increases.

7 Related Work

So far little attention has been given to trust management in virtual communities. Most of the existing approaches focus on reputation-based trust models in P2P networks [25]. Abdul-Rahman and Hailes [1] propose a trust model that is based on real world social trust characteristics. They also find formal logic based trust management to be ill suited as a general model of trust. To prove this claim they refer to the early work of Burrows and Abadi [4], and Gong, Needham, and Yahalom [11], which are more relevant to formal protocol verification than to formal reasoning on trust management. To support their work they claim that logic based trust management systems are not suitable to be automated - the existing literature on automated trust negotiation (ATN) yields a contradictory statement (see Seamons et al. [23]). Pearlman et al. [20] present a Community Authorization Service - a central management unit for a community that helps to enforce the policy of a virtual community. Such a central point of responsibility does not fit well in the spirit of P2P networks because of their highly distributed nature. Pearlman et al. also require that there a centralised policy exists for a virtual community. However, the policy of a virtual community may have a distributed character and can be seen as a product of the policies of the community members. Boella and van der Torre [3] take the same direction and emphasise the distinction between authorisations given by the Community Authorization Service and permissions granted by resource providers in virtual communities of agents. They regard authorisation as a means used by community authorities to regulate the access of customers to resources that are not under control of these authorities. According to Boella and van der Torre, permission can be granted only by the actual resource owner.

As we conclude in Section 2, virtual communities are also not supported by the existing trust management languages, even though the general requirements for such languages have been investigated [23].

Herzberg et al. propose in [12] a prolog-based trust management language (DTPL) together with a non-monotonic version of it (TPL). Their approach is very different from ours in the sense that TLP allows for *negative certificates* namely "certificates which are interpreted as suggestions not to trust a user". This far-reaching approach leads to a more complex logical interpretation, which includes conflict resolution. As opposed to this, our approach is technically simpler and enjoys a well-established semantics. Jajodia et al. [13], Wang et al. [26], Barker and Stuckey [2], have in common that they impose a *stratified* use of negation. Because of this, they can refer to the perfect model semantics. As we explained in Section 4, in the context of DTM, we cannot expect policies to be stratified. Our approach is thus more powerful than the approaches based on the stratifiable negation. Dung and Thang in [6] propose a DTM system based on logic programming and the *stable model semantics* [10].

8 Conclusions and future work

We present the language $RT_\ominus$, which adds a construct for 'negation-in-context' to the RT_0 trust management system. We argue the necessity of such a construct and illustrate its use with scenarios from virtual communities which cannot be expressed within the RT framework.

We provide a semantics for $RT_\ominus$ by translation to general logic programs. We show that, given the complete policy, the membership relation can be decided by running the translation in systems such as XSB, DLV datalog and Smodels. We also show how, for the case that credentials are issuer traceable [18], the chain discovery algorithm for RT_0 can be extended to $RT_\ominus$. We are currently employing $RT_\ominus$ to specify virtual community policies in the Freeband project I-SHARE. In the future we plan to examine the complexity of the presented chain discovery algorithm, ad hoc methods to minimise communication overhead, and safe methods for chain discovery in non-'issuer traces all' scenarios. A comparison with reputation systems will also be made.

References

[1] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *HICSS '00: Proc. of the 33rd Hawaii International Conference on System Sciences-Volume 6*, page 6007. IEEE Computer Society Press, 2000.

[2] S. Barker and P. J. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. on Information and System Security*, 6(4):501–546, 2003.

[3] G. Boella and L. van der Torre. Permission and authorization in policies for virtual communities of agents. In *Proc. of Agents and P2P Computing Workshop at AAMAS'04*. Springer Verlag, 2004.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[5] W. Chen, T. Swift, and D.S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–199, 1995.

[6] P.M. Dung and P.M. Thang. Trust Negotiation with Nonmonotonic Access Policies. In *Proc. IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 04)*, volume 3283 of *LNCS*, pages 70–84. Springer-Verlag, 2004.

[7] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving using the DLV system. pages 79–103, 2000.

[8] M. Fitting. A Kripke-Kleene semantics for Logic Programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[9] A.V. Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.

[10] M. Gelfond and V. Lifschitz. The Stable Model Semantics For Logic Programming. In *Proc. 5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[11] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[12] A. Herzberg, Y. Mass, J. Michaeli, Y. Ravid, and D. Naor. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *Proc. 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, pages 2–14, Oakland, CA, 2000. IEEE Computer Society Press.

[13] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for Multiple Access Control Policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260, 2001.

[14] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[15] F. Lee, D. Vogel, and M. Limayem. Adoption of informatics to support virtual communities. In *HICSS '02: Proc. of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 8*, page 214.2. IEEE Computer Society Press, 2002.

[16] N. Li, J. Feigenbaum, and B. N. Grosof. A Logic-based Knowledge Representation for Authorization with Delegation (Extended Abstract). In *Proc. 1999 IEEE Computer Security Foundations Workshop*, pages 162–174. IEEE Computer Society Press, June 1999.

[17] N. Li, J. Mitchell, and W. Winsborough. Design of A Role-based Trust-management Framework. In *Proc. 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[18] N. Li, W. Winsborough, and J. Mitchell. Distributed Credential Chain Discovery in Trust Management. *Journal of Computer Security*, 11(1):35–86, 2003.

[19] I. Niemel and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal lp. In *LPNMR '97: Proc. of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 421–430. Springer-Verlag, 1997.

[20] L. Pearlman, V. Welch, I. T. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *POLICY*, pages 50–59. IEEE Computer Society Press, 2002.

[21] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS05)*, February 2005.

[22] T. C. Przymusinski. Perfect model semantics. In *Fifth international Conference and Symposium on Logic programming, Seattle, U.S.A.*, pages 1081–1096. MIT Press, 1988.

[23] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for Policy Languages for Trust Negotiation. In *Proc. 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, pages 68–80. IEEE Computer Society Press, 2002.

[24] J. C. Shepherdson. Negation in Logic Programming. In *Foundation of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, 1988.

[25] V. Shmatikov and C. L. Talcott. Reputation-based trust management. *Journal of Computer Security*, 13(1):167–190, 2005.

[26] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proc. 2004 ACM workshop on Formal methods in security engineering FMSE'04*, pages 45–55. ACM Press, 2004.

Appendix A   Virtual Community - translation from $RT_\ominus$ to GLP

The table below shows the $RT_\ominus$ credentials and their translation to GLP.

$RT_\ominus$ rules	GLP semantics
A.addCoord $\longleftarrow$ A.allCandidates $\ominus$ A.objectionToAdd	addCoord$(A, ?Y)$:- allCandidates$(A, ?Y)$, ¬objectionToAdd$(A, ?Y)$.
A.allCandidates $\longleftarrow$ A.allCoord.agreeToAdd	allCandidates$(A, ?Y)$:- allCoord$(A, ?Z)$, agreeToAdd$(?Z, ?Y)$.
A.objectionToAdd $\longleftarrow$ A.allCoord.disagreeToAdd	objectionToAdd$(A, ?Y)$:- allCoord$(A, ?Z)$, disagreeToAdd$(?Z, ?Y)$.
A.disagreeToAdd $\longleftarrow$ A.allCandidates $\ominus$ A.agreeToAdd	disagreeToAdd$(A, ?Y)$:- allCandidates$(A, ?Y)$, ¬agreeToAdd$(A, ?Y)$.
A.allCoord $\longleftarrow$ A.allCoord.coord	allCoord$(A, ?Y)$:- allCoord$(A, ?Z)$, coord$(?Z, ?Y)$.
A.allCoord $\longleftarrow$ A	allCoord(A, A).
A.coord $\longleftarrow$ B	coord(A, B).
B.coord $\longleftarrow$ C	coord(B, C).
C.coord $\longleftarrow$ B	coord(C, B).
C.coord $\longleftarrow$ A	coord(C, A).
A.agreeToAdd $\longleftarrow$ D	agreeToAdd(A, D).
A.disagreeToAdd $\longleftarrow$ E	disagreeToAdd(A, E).
B.disagreeToAdd $\longleftarrow$ F	disagreeToAdd(B, F).
C.disagreeToAdd $\longleftarrow$ F	disagreeToAdd(C, F).

	10 Coords			30 Coords			50 Coords		
	1 Iter.	10 Iters.	20 Iters.	1 Iter.	10 Iters.	20 Iters.	1 Iter.	10 Iters.	20 Iters.
DLV	0.05s	0.81s	1.54s	0.06s	0.83s	1.55s	0.07s	0.86s	1.60s
SMODELS	0.12s	1.22s	2.32s	0.16s	1.35s	2.66s	0.19s	1.53s	2.94s
XSB	≈ 0								

Table A.1
The time execution of the program on the XSB, SMODELS, and DLV systems