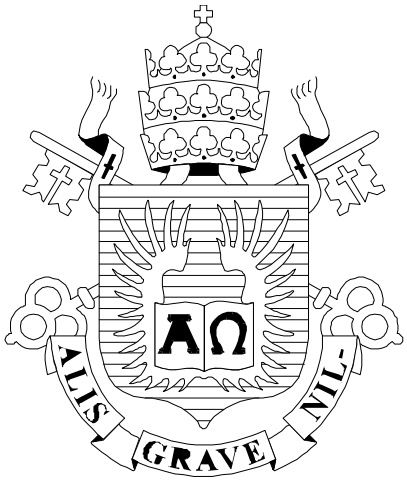

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 42/05

Toward Real-world Trust Policies

Vinicius da Silva Almendra
Daniel Schwabe
Marco Antonio Casanova

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 42/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December, 2005

Toward Real-world Trust Policies *

Vinicius da Silva Almendra, Daniel Schwabe,
Marco Antonio Casanova

almendra@inf.puc-rio.br, dschwabe@inf.puc-rio.br, casanova@inf.puc-rio.br

Abstract. The increasing reliance on information gathered from the Web and other Internet technologies (P2P networks, e-mails, blogs, wikis, etc.) raises the issue of trust. Trust policies are needed to filter out untrustworthy information. This filtering task can be leveraged by the increasing availability of Semantic Web metadata describing retrieved information. It is necessary, however, to adequately model the concept of trustworthiness; otherwise one may end up with operational trust measures that lack a clear meaning. It is also important to have a path from one's trust requirements to concrete trust policies through Semantic Web technologies. This paper proposes a classical logic model for trust policies, grounded on real-world models of trust, and some guidelines to map trust requirements to trust policies. It also presents the implementation of a case study using Prolog.

Keywords: Semantic Web, Trust Policies, Trust Model.

Resumo. A crescente dependência de informações obtidas através da Web e de outras tecnologias da Internet (redes P2P, correio eletrônico, blogs, wikis, etc.) levanta a questão da confiança. Políticas são necessárias para descartar informação não-confiável. Esta tarefa pode ser alavancada pela crescente disponibilidade de metadados descrevendo as informações obtidas. É necessário, no entanto, modelar adequadamente o conceito de confiabilidade; caso contrário, pode-se terminar com medidas de confiança operacionais sem uma semântica clara. Também é importante prover um caminho dos requisitos de confiança a políticas de confiança concretas, passando pelas tecnologias da Web semântica. Este artigo propõe um modelo baseado em lógica clássica para políticas de confiança, apoiado em modelos de confiança do mundo real, e algumas pautas para mapear requisitos de confiança em políticas de confiança. Também apresenta a implementação de um estudo de caso usando Prolog.

Palavras-chave: Web Semântica, Políticas de Confiança, Modelo de Confiança.

---

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

In charge for publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introduction

One of the great challenges of the Web is the problem of trust. Operational measures of trustworthiness are needed to separate relevant and truthful data from those that are not [Guha, 2004]. However, to be correctly interpreted, these measures must be linked with real-world concepts of trust. They must also meet the trust requirements of their users. Building on the trust concept found in [Gerck, 1998] and [Castelfranchi, 2001], our work aims to pave the path leading from a user's trust requirements to operational trust policies that can be applied to Semantic Web data, while preserving the relation between the resulting policies and the trust requirements we started with. This relation is important as it enables the user to find out why a piece of data was found trustful.

We focus on the Semantic Web scenario, where an agent receives some data described using Semantic Web metadata. The agent must decide which metadata can be trusted and then decide if the data is trustful. Our contributions include a concept of real-world trust policies, a model to represent these policies using classic logic and a case study applying the proposed model in Prolog. It represents an improvement over [Almendra, 2005].

In [Bizer, 2005] we find a similar work that describes a Semantic Web browser which filters information based on trust policies that the user selects. It also offers explanations of why each piece of information was trusted. This proposal offers facilities to express trust policies as pieces of TriQL queries in such a way that it offers an explanation of why a triple was found trustful. However, it does not address the problem of building the trust policy. Our work deals with representing trust requirements as trust policies that preserve real-world trust relationships.

Another similar work is [Nejdl, 2004], which presents an approach for building trust among unknown agents by the exchange of credentials, or trust negotiation. A credential asserts that one or more entities hold some property (e.g. being a university student). An agent is trusted when he can prove (through the exchange of credentials) that he has certain properties, as defined by the other party. [Nejdl, 2004] uses logic clauses (Guarded Distributed Logic Programs) to define trust policies. It also offers a conceptual framework for the trust negotiation process, which eases the task of building trust policies.

In section 2 we describe the concept of real-world trust on which we based our work. In section 3 we present a formal model to build trust policies following the concept of trust shown in section 2. In section 4 we present a case study: an implementation of some trust policies using Prolog. In section 5 we conclude our work and point future directions.

2 A Model of Trust

2.1 A Motivating Scenario

The scenario we focus on is based on the Semantic Web Publishing scenario [Carroll, 2004], the DBin project [Tummarello, 2005] and the Piggy Bank software [Huynh, 2005]. The Semantic Web Publishing scenario has agents embodying two roles: information providers and information consumers. An information provider publishes RDF

graphs, which contain information and its metadata, such as provenance, publishing date, etc. An information consumer gathers these graphs and decides what to do with them, provided that these graphs can be seen as claims of the information provider, rather than definitive facts. The formal meaning of these claims, that is, what statements about the world are being made, is given by a set of accepted graphs, which is a subset of the graphs the information consumer receives. It is assumed that the agent will act based solely on information contained in accepted graphs.

The Semantic Web Publishing proposal also enables the user to specify a trust policy, that is, a set of conditions that the received information should meet to be accepted. An example of a policy would be "trust all information that comes from direct friends and is about computers".

This scenario can be integrated with the one outlined in [Tummarello, 2005], which is a P2P network where people exchange RDF graphs of interest and store all the received graphs in a local database. Filtering can be applied to hide triples that do not match the user's criteria. One use for this is the implementation of trust policies. The set of visible triples, which we call accepted triples, is similar to the set of accepted graphs described above.

The Piggy Bank software shows a possible path to integrate the World Wide Web and the Semantic Web, based on the idea of a browser extension that stores semantic data collected from Web pages. This offers a scenario for trust policies, where an application will analyze the trustfulness of metadata much in the same way a human analyzes the trustfulness of data (i.e. the Web pages).

These scenarios are only examples of possible uses of trust and trust policies within the Semantic Web context. Other scenarios are possible, such as Semantic Social Desktops [Decker, 2004].

2.2 A Concept of Trust

To build a suitable trust model, we start eliciting attributes of real-world trust, trying to capture its essence. We do this based on [Gerck, 1998] and [Castelfranchi, 2001].

[Castelfranchi, 2001] defines trust in the context of multi-agent systems, where agents are endowed with goals. In this context, he asserts that trust is "a mental state, a complex attitude of an agent x towards another agent y about the behavior/action α relevant for the result (goal) g. This attitude leads the agent x to the decision of relying on y having the behavior/action α, in order to achieve the goal g.

[Gerck, 1998] presents a definition of trust as "what an observer knows about an entity and can rely upon to a qualified extent". This definition is closely related to the previous one: the observer is the agent who trusts; the entity is trusted agent; the qualified extent is the behavior/action. Both also link trust with reliance. However, the former definition mentions explicitly the goal-oriented nature of trust, which is an important aspect, as agents lacking goals do not really need trust [Castelfranchi, 2001].

From both definitions, we observe that trust implies reliance: when an agent trusts something, he relies on its truth to achieve some goal without further analysis – even if he is running the risk of taking an inappropriate or even damaging action if the object of trust is false.

Trust implies reliance, but not necessarily action. For example, John may trust Mary's bookstore without buying anything there, nevertheless, if John needs a book and Mary offers it under good conditions of price, placement, payment etc., John will

buy the book without further questions. At the same time, he may refuse to buy the same book under better conditions at a bookstore that he does not trust. So, the trust attitude comprises a "potential" relying action on the object of trust.

We may also ask what the object of trust is. In this case, it could be described as "Mary's bookstore is a good one". John believes this and will act upon it when necessary to achieve some goal; using the definition of [Gerck, 1998], John knows that "Mary's bookstore is a good one" and is relies on this. So, trust is particular form of knowledge: a reliable knowledge.

However, there is another important question: how did John decide to trust Mary's bookstore? This is the problem of justification [Gerck, 1998]. [Castelfranchi, 2001] grounds the trust decision on the beliefs of the trusting agent. In our example, John may have decided to trust Mary's bookstore because he believes Mary is an honest and competent person and that the business runs under her strict control. If one of these beliefs were absent, then John would not trust.

The problem is not solved yet, as we may ask where these beliefs come from. John relies on those beliefs to take a decision (in this case, the decision to trust Mary's bookstore), what characterizes John's trust on those beliefs. So, the trust decisions may be recursive: to trust Mary's bookstore, John also has to trust that she is competent and honest. Nevertheless, these trust decisions do not need to be simultaneous: John may have decided to trust Mary's competence many years before she had a bookstore.

There are some kinds of beliefs widely used to justify trust. One of these is the self-trust belief: a person normally trusts facts that are evident to him. Provenance belief is also an important one: when deciding the truthfulness of a statement, one of the first questions is who stated it. In fact, the word statement implies a provenance: a statement has been stated by someone.

The justification of trust based on beliefs links trust with belief revision: if some of the beliefs that justified trust are discredited, trust may eventually be lost. If John discovers that several friends bought defective books with Mary, the belief of competence could be revised. Then, trust on Mary's bookstore would become unjustified and might be lost. This is a situation where new evidence hampers previously acquired trust, as this trust was based on the assumption that Mary was competent. It might have been a good assumption, but was shown to be false due to contradictory evidence.

Another characteristic is that trust is subjective: different agents may have different beliefs, different goals and require different degrees of justification to trust something. Continuing with the example, Mike might not trust Mary's bookstore, as he beliefs she is not competent, she does not know Japanese literature well and she does not worry about the decoration of the bookstore. Here, we face contradictory beliefs and also different demands to consider a bookstore to be trustful. The difference between beliefs may be due to the goals: John might be an occasional reader, while Mike is an artist interested in Japanese culture. Note that this is not just a matter of opinions: both take decisions based on these beliefs.

Trust also evolves with time: John may lose his trust on Mary's bookstore even without any change in his beliefs about her. At the same time, may start trusting airplanes as a safe transportation means, again without any change is his beliefs about airplanes. What changes herein these cases is the justification required for trustfulness.

3 Formalizing Trust

3.1 Outline of a Trust Model

Based on the trust concept formulated above, we build an informal model of trust, which comprises the following elements: facts, contexts, knowledge bases, trust policies, trust decisions, justification, beliefs and, trusting agents,

A fact is a statement about the reality, following the semantic of RDF triples. We recognize that a triple's interpretation demands a fourth element: the context [Guha, 2004b; Decker, 2005; Bizer, 2004]. Contexts help define the provenance of the facts (who stated), the circumstances (date, time, reason, etc. [Marchiori, 2004]), and, more generally, help situating a fact in order to allow its correct interpretation.

The set of facts that an agent knows is its knowledge base. An asserted fact is a known fact and a trusted fact is an asserted fact that can be trusted. For example, when someone reads a newspaper, he may augment his knowledge base with several asserted facts, but he may only trust some of them (or none!).

A trust policy is a set of rules hat the trusting agent uses to test the trustfulness of a fact. Different trusting agents may use different trust policies and, hence, they can make different trust decisions, even when exposed to the same facts, characterizing the subjective nature of trust. The same trusting agent may change his trust policy in order to match his current goals. This characterizes trust dynamism.

A trust decision is the act of testing if an asserted fact meets a trust policy, that is, a decision to rely on that fact's truthfulness. A trust decision is in fact the process of finding a deduction, which we call a justification, that the asserted fact can indeed be trusted. Only trusted facts can be used in a justification. Here we go through a simplification: we will use only classic logic, whereas in real world, people also reason with uncertain facts.

3.2 A Formal Model

We will proceed to formalize the model outlined in the previous section, using definite (horn) clauses to model trust policies. We will later discuss the introduction of "negation as failure" and its consequences.

We assume that the knowledge base of the trusting agent has a *domain theory* that defines all predicates unrelated to the trust model.

Facts will be modeled as RDF triples (*subject, predicate and object)* plus a *context*, similarly to other approaches [Carroll, 2004; Decker, 2005], yielding a quadruple. A context may also be the subject of a fact. The knowledge base of the trusting agent contains a set of unit clauses of the form *assertedFact(S,P,O,C)*, which indicate that the fact having subject *s*, predicate *p*, object *o* and context *c* is asserted.

A *trust policy* is a predicate that fulfills the following conditions:

- It evaluates facts, that is, it has the form *policy(S,P,O,C)*, where *policy* is the name of the trust policy and *S, P, O, C* are the components of a fact.
- It does not depend on facts that are asserted, but are not trusted.
- It asserts the truthfulness of a fact, given the current set of trusted facts of the trusting agent.

In other words, a trust policy is a predicate that asserts the truthfulness of a fact based only on trusted facts or on the domain theory of the agent's knowledge base.

An *assertedFact(S,P,O,C)* is *trusted* iff *trustedFact(S,P,O,C)* can be deduced from the other trusted facts in the knowledge base and the trust policy adopted by the trusting agent.

The following clause illustrates the use of a trust policy:

$$trustedFact(S,P,O,C) \leftarrow assertedFact(S,P,O,C) \wedge policy(S,P,O,C)$$

The definition of a trust policy may use the predicate *trustedFact* to test if a fact is trusted. The following example theory defines the trust policy *policy* using the predicate *trustedFact*. Notice that this is a recursive policy: the definition of *policy* uses *trustedFact*, which in turn use *policy*, as in the following theory:

$$\begin{aligned}
&policy(Person_1, worksWith, Person_2, C) \leftarrow \\
&\quad trustedFact(Person_2, knows, Person_1, C) \wedge \\
&\quad trustedFact(Person_1, worksAt, Organization, AnyContext) \wedge \\
&\quad trustedFact(Person_{21}, worksAt, Organization, AnyContext2) \\
&policy(Document, type, email, Context) \leftarrow \\
&\quad trustedFact(Context, hasProvenance, Person, Context) \wedge \\
&\quad trustedFact(myself, knows, Person, myContext)
\end{aligned}$$

where variables are capitalized and constants are not.

The first clause states that the statement "$Person_1$ works with $Person_2$" can be trusted if $Person_2$ knows $Person_1$ and they work at the same organization; the second clause states that the statement "This *document* is an e-mail" can be trusted if the trusting agent knows the provenance of the e-mail.

Trust policies can be used together, either in the same clause or in different clauses, as in the following clauses:

$$\begin{aligned}
&p_1(S,P,O,C) \leftarrow p_2(S,P,O,C) \wedge p_3(S,P,O,C) \\
&p_1(S,P,O,C) \leftarrow p_4(S,P,O,C)
\end{aligned}$$

A trust policy P is *elementary* iff P is about the domain theory in the knowledge base of the trusting agent. Hence, P must not use either the predicate *trustedFact* or other trust policies

$$\begin{aligned}
&p_1(Document, creationDate, Date, C) \leftarrow \\
&\quad isOlderThan(Date, Today) \wedge isCurrentDate(Today) \\
&p_1(Context, P, O, Context) \leftarrow belongsToDublinCore(P)
\end{aligned}$$

The first clause states that the statement "this document's creation date is *Date*" is trustful if *Date* is a past date. The second clause states that a fact about its context is trustful if the property belongs to Dublin Core ontology.

We can infer some important properties of this model for trust policies. The first is the *interdependence of facts*: any non-elementary trust policy uses trusted facts; in this case, trust in a fact is always dependent on trust in other facts.

When trust policies appear together in a clause and share the same variables, the resulting trust policy will be the *intersection* of the sets of facts that would be trusted by each of them. When policies appear in *different* clauses for the same trust policy, then the result is the *union* of the sets of trusted facts.

3.3 Using Negation

The use of definite (Horn) clauses to express trust policies restricts the use of negation. Intuitively, only positive facts can be tested. This limitation can be overcome by the use of *negation as failure*: the negation of a formula is true iff one cannot prove that formula's truth.

The use of negation as failure improves the expressiveness of the formalism, allowing trust policies to reason with the *absence* of a trusted fact. A direct application of this is capturing the notion of *exception*: trust something under some conditions, *except* when a certain fact is trusted. For example, a person may trust financial reports coming from any bank, except when there is a complaint coming from a friend about a particular bank. This is a common reasoning in daily life: trust based on the absence of contrary evidence.

However, negation as failure must be used with caution. Formally speaking, negation as failure is based on the *closed world assumption*: something that cannot be proven is assumed to be false. In other words, there are no "unknown" facts. This is an optimistic premise that may not be desirable when dealing with reliance.
Therefore, it is important for the user to understand whether this assumption is consistent with his desired meaning for the policy, given its goals.

3.4 Building Trust Policies

The formal model described in Section 3.3 gives only a structure to build rules that justify trust. However, it would be interesting to describe how these rules could be built in order to reflect real-world semantics. Towards this end, we propose some guidelines elicited from the trust concept, as discussed in Section 2.

3.4.1 Representing Trust Policies

From now on we will use Prolog to represent trust policies, allowing a fast prototyping and testing of trust policies. To do so, we will make the following mappings:

- Variables will be capitalized;
- Atoms will be used to denote URI's. Note that atoms can be quoted, e.g. 'foaf:knows'. To avoid cluttering, we will use namespaces or even just single words to denote URI;
- The predicates assertedFact and trustedFact will be mapped to predicate terms with the same name.

The following Prolog code shows an example with some facts and a simple trust policy:

```
/* Asserted facts */
assertedFact(john,'foaf:knows',mary,my_context).
assertedFact(  news1, 'rdf:type', 'news:News',news1_context).
assertedFact(  news1_context,dc:creator,erick,my_context).

/* trustedFact predicate */
trustedFact(S,P,O,C) :- assertedFact(S,P,O,C), myCurrentPolicy(S,P,O,C).
myCurrentPolicy(_,_,_,my_context).
myCurrentPolicy(_,'rdf:type', 'news:News',C) :-
    trustedFact(C,dc:creator,_,my_context).
```

3.4.2 Trust Policy Structure

The formal definition of trust policies is too wide to provide useful hints about building policies. In order to simplify this task, we propose a stricter structure.

Trust policies may be divided into *composite* policies and *atomic* policies. A composite policy is a trust policy built using solely other trust policies through union or intersection. Any policy that does not fulfill this is an atomic policy: it uses the predicate *trustedFact* and user-defined predicates. The code below exemplifies this.

```
/* Atomic policies */

/* Friendship must be reciprocal */
friends1(S, 'foaf:knows',myself,_) :-
  trustedFact(myself, 'foaf:knows',S,_).

/* The provenance of a context must be
   a person URI's. Notice the use of
   user-defined predicates */
validContext(C,Provenance,O,C) :-
  isProvenancePredicate(Provenance),
  trustedFact(O,'rdf:type',Type,_),
  isPersonType(Type).

/* A person can state his friends */
friends2(S,'foaf:knows',_,C) :-
  trustedFact(C,'dc:creator',S,C).

/* Composite policies */
myPolicy(S,P,O,C) :-
  validFriendship(S,P,O,C),
  validContext(S,P,O,C).

validFriendship(S,P,O,C) :- friends1(S,P,O,C).
validFriendship(S,P,O,C) :- friends2(S,P,O,C).
```

In this example, all facts that fulfill the policies *friends1* or *friends2* also fulfill *validFriendship*; and all facts that fulfill *validFriendship* and *validContext* also fulfill *myPolicy*. This structure can be depicted as an *and/or tree*.

This division of policies into atomic and composite is arbitrary. However, it provides a segmentation of trust policies that can be used in a *top-down* approach (subdivision of trust policies until it is easy to define them atomically), or in a bottom-up approach (build atomic policies and then combine then to form more and more complex policies).

This decomposition does not imply independence: the justification of a fact by one policy may demand facts that are found trustful due to any other policy in the hierarchy of the trust policy used by the trusting agent, even facts unrelated with the policy we started with. For example, the justification of a statement about a bank account may require the name of the bank, which can be treated by a completely unrelated trust policy. This is a corollary of the property of interdependence of facts.

3.4.3 Building Atomic Policies

To further simplify atomic policies, we may break them into two pieces: its *scope* and its *justification*. The scope is the set of facts whose trustfulness is evaluated by the policy. In the code above, we see that policy *friends1* only deals with facts whose predicate is "foaf:knows" and whose object is "myself". All other facts are not trustful (from this policies viewpoint). So, we could describe the scope of this policy as being "statements saying that somebody is my friend". The most direct way to provide scope is to express

the values of the fact's components, as was done in the example. Nevertheless, predicates could also be used, as shown in the policy *validContext*: the predicate *isProvenancePredicate* tests whether the predicate part of the fact is a "provenance predicate". Note that the scope provides an interesting hint for policy subdivision: trust policies dealing with unrelated sets of facts (e.g., a trust policy about airplanes and food) should be divided into policies about each one of these sets of facts.

The *justification* is the set of conditions imposed by the policy to accept a fact – that is in the policy's scope – as trustful. In the above example, the policy *friends1* justifies trust on facts (belonging to its scope) by testing whether or not the trusting agent is a friend of the subject of the statement.

Justification is done based on beliefs (see Section 2.2). There are some beliefs ubiquitous in trust policies: *provenance* and *circumstances* [Carroll, 2004; Bizer, 2004b]. These beliefs can be translated into trusted facts about contexts. A discussion about representing provenance of RDF graphs using contexts can be found at [Castelfranchi, 2001]. In the code above, we find an example of representing provenance: an asserted fact stating the creator of the context named *news1_context*. Circumstances can be represented similarly: there can be statements asserting date, time, placement, reason etc., of a context. It is even possible to have a hierarchy of contexts, where one contexts "inherits" statements made about another. Nonetheless, this discussion is out of the scope of this paper.

A particular case of provenance is when the source of a statement is the trusting agent. Based on the self-trust belief, trust policies will accept these facts as trustful by default. However, there is nothing in the model that obliges this: one may not trust himself on some subjects, e.g. on fixing cars. A special context, like *my_context*, may be used to represent self-provenance.

4 An Example & Case Study

4.1 Definition of the Trust Policies

We have tested the proposed model crafting an example trust policy and implementing it as a logic program using XSB Prolog [Rao, 1997], as it gives a better support to recursive predicates and to the use of negation in clauses.

The scenario is as follows. Andrew runs a small business: a web-based news clipping service, daily updated. He relies heavily on Web and e-mail to obtain relevant news. From the web he extracts news from news-related websites, blogs, discussion lists, etc. From e-mail he receives information related to his job from friends, acquaintances, sources and even from strangers. He is also interested in finding relationships between people, as these relationships can be explored to gather more information for his clippings.

Andrew cannot check thoroughly all the information he receives, as he does not have time to do it in a timely fashion. On the other hand, his clients are concerned with the quality of the news clippings. So, he has to rely on information provided by other people, which he needs to trust.

Here trust policies can help: instead of making himself repetitive analysis to check the veracity and truthfulness of the information, Andrew may delegate a substantial part of this job to the computer.

We will assume that Websites provide some degree of semantic information about their content, that e-mails also carry semantic information related to the e-mail content (FOAF profiles, for example), and that all this semantic content is stored in Andrew's knowledge base, which is available as RDF triples, plus a context. This context is treated as a blank node that is the subject of metadata, like provenance (which can be a person's e-mail, the URL of a Web page or the Permalink of a blog's post). Every time some Semantic Web content is retrieved, a new context is built containing the above information and it is assigned to all the retrieved triples.

Now we'll follow the proposed guidelines to build trust policies that help Andrew is his job. The first task is to identify the trusting agent**.** Although it seems obvious that Andrew is, we could also think that Andrew's clients are the trusting agents. Nonetheless, we will use with Andrew.

We can elicit two goals: prepare free clippings, which are placed is the news service Website for free access, and prepare paid clipping, which will be sold to Andrew's clients. These two policies may be decomposed using an and/or tree as depicted in Figure 1.

The next step is to build the atomic policies. In Table 1 we show a possible specification of the scope and the required justification for each atomic trust policy.

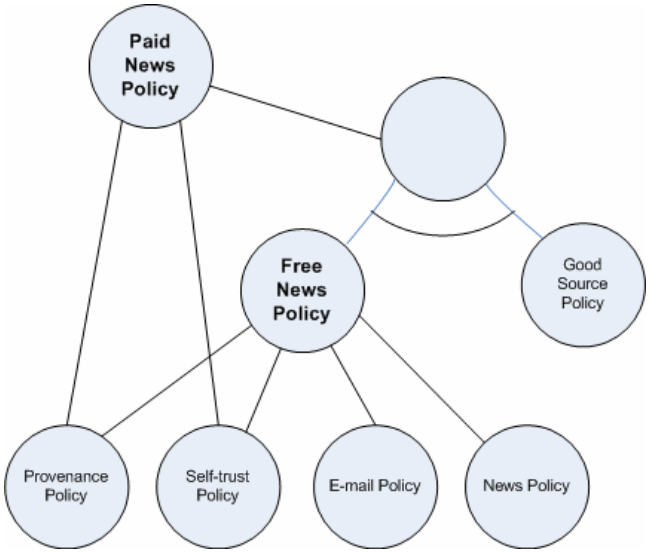

Figure 1. Policy decomposition

Atomic Policy name	Scope	Justification required
provenancePolicy	Provenance information attached to contexts	None, as in this scenario provenance information is attached by Andrew
selfTrustPolicy	Self-asserted information	Facts must be in Andrew's context, which is a special context, different from all others.
newsPolicy	Statements about news	Facts must come from a trusted news source or from a friend.
emailPolicy	Statements about e-mails	Facts must come from friends or from people Andrew usually exchanges e-mails with and are reachable through friendship relation.
goodSourcePolicy	All statements	Facts must come from a person who has at least two friends in common with Andrew and is not in Andrew's black list.

Table 1. Andrew's atomic trust policies

We can now proceed to map the policies to Prolog code. The description of the composite trust policies (*freeNewsPolicy* and *paidNewsPolicy*) follows.

```
/* "Free News" policy */
freeNewsPolicy(S,P,O,C) :- provenancePolicy(S,P,O,C).

freeNewsPolicy(S,P,O,C) :- selfTrustPolicy(S,P,O,C).

freeNewsPolicy(S,P,O,C) :- newsPolicy(S,P,O,C).

freeNewsPolicy(S,P,O,C) :- emailPolicy(S,P,O,C).
freeNewsPolicy(S,P,O,C) :- contactInfoPolicy(S,P,O,C).
freeNewsPolicy(S,P,O,C) :- newsSourcePolicy(S,P,O,C).
/* "Paid news" policy */
paidNewsPolicy(S,P,O,C) :- freeNewsPolicy(S,P,O,C),
goodSourcePolicy(S,P,O,C).

paidNewsPolicy(S,P,O,C) :- selfTrustPolicy(S,P,O,C).

paidNewsPolicy(S,P,O,C) :- provenancePolicy(S,P,O,C).
```

Finally, we can go to the atomic trust policies. For sake of brevity, we will show only some of them. Note that trust policy *goodSourcePolicy* uses negation as failure to test for the absence of a fact.

```
/* Policy used for trustful  provenance data */
provenancePolicy(C,dc:creator,_,C).

/* Trust that something is news when
   its provenance is a news source */

newsPolicy(_,P,O,C) :-
    P = rdf:type, O ='ex:News',
    trustedFact(C,dc:creator,Provenance,C),
    trustedFact(Provenance,rdf:type,'ex:NewsSource', _).

/* Trust any property about trusted news
   when both belong to the same context */
newsPolicy(S,_,_,C) :- trustedFact(S,rdf:type,'ex:News',C).

/* A good source is one that shares at least
   two friends with myself and is not in my
   black list */
goodSourcePolicy(_,_,_,C) :-
    trustedFact(C, dc:creator, Source, C),
    trustedFact(myself, foaf:knows, P1, _),
    trustedFact(myself, foaf:knows, P2, _),
    trustedFact(P1, foaf:knows, Source, _),
    trustedFact(P2, foaf:knows, Source, _),
    P1 \= P2,
    not trustedFact(Source, rdf:type,'ex:BadSource', my_context).
```

4.2 Running Trust Policies

We have executed these policies for an ordered list of facts, simulating the situation where Andrew visits Web pages, read e-mail, etc. and Semantic Web information is collected in each activity. We tested with two policies, *freeNewsPolicy* and *paidNewsPolicy*. The results are summarized in Table 2.

The first column shows the events and their associated Semantic Web data, that is, RDF triples plus a context. These data is inserted in the trusting agent's knowledge base as asserted facts. To save space, the context of the triples is stated only once in each event, as all triples share the same context. The second and the third columns show the facts that were found trustful with each policy. Note that each line shows only new asserted facts and new trusted facts, as the previously asserted or trusted ones continue to be so. An exception is shown in the last line, where the use of negation is *paidNewsPolicy* introduces nonmonotonicity: the addition of an asserted fact causes previously trusted facts to lose this condition.

5 Conclusions and Future Work

Our goal was to build a model to capture, represent and apply trust policies of an agent in the scenario of Semantic Web, while preserving real-world semantics of trust.

We first outlined a model capturing relevant aspects of the trust concept, such as reliance, subjectivity, dynamism, justification, and then proceeded to building a formal model for trust policies. We also presented some guidelines to create real-world trust policies using that model. Finally, we presented an example of an implementation in Prolog of trust policies using the formalism and the guidelines previously described.

The proposal outlined offers the possibility of building large trust policies using a divide-and-conquer approach, where more complex policies are composed from simpler ones. Another feature is the possibility to build trust policies incrementally.

Event/facts	Added Trusted fact with *freeNewsPolicy*	Added Trusted fact with *paidNewsPolicy*
Andrew starts his knowledge base stating friendship relations. (context=*my_context*) myself foaf:knows John myself foaf:knows Bob John foaf:knows Adam Bob foaf:knows Adam	myself foaf:knows John myself foaf:knows Bob John foaf:knows Adam Bob foaf:knows Adam	myself foaf:knows John myself foaf:knows Bob John foaf:knows Adam Bob foaf:knows Adam
Andrew receives an e-mail from Mary (context=*email_mary*) email_mary rdf:type ex:Email email_mary dc:creator mary	email_mary dc:creator mary *(notice that it was not trusted as an e-mail)*	email_mary dc:creator mary
Andrew receives an e-mail from Adam with FOAF information and with information about a Website that is a news source. (context=*email_adam*) email_adam dc:creator Adam email_adam rdf:type 'ex:Email' Adam foaf:knows mike Reuters rdf:type 'ex:Website' Reuters rdf:type 'ex:NewsSource'	email_adam dc:creator Adam	email_adam dc:creator Adam
Andrew states that Adam is his friend. (context=*my_context*) myself foaf:knows Adam	myself foaf:knows Adam email_adam dc:creator Adam email_adam rdf:type 'ex:Email' Adam foaf:knows mike Reuters rdf:type 'ex:Website' Reuters rdf:type 'ex:NewsSource' (*note the interdependence of facts: these trusted facts depended on friendship with Adam).*	myself foaf:knows Adam email_adam dc:creator Adam email_adam rdf:type 'ex:Email' Adam foaf:knows mike Reuters rdf:type 'ex:Website' Reuters rdf:type 'ex:NewsSource'
Andrew visits Soccer blog. (context=*blog_data*) blog_data dc:creator soccer_blog news_blog rdf:type 'ex:News' news_blog dc:description "World cup news"	blog_data dc:creator soccer_blog	blog_data dc:creator soccer_blog
Andrew visits Reuters web news service. (context=*reuters_data*) reuters_data dc:creator reuters soccer_blog rdf:type 'ex:NewsSource'	reuters_data dc:creator reuters	reuters_data dc:creator reuters
Andrew reads an e-mail from Mike, giving a hint about a blog with soccer news. (context=*email_mike*) email_mike dc:creator Mike email_mike rdf:type 'ex:Email' soccer_blog rdf:type 'ex:Blog' soccer_blog rdf:type 'ex:NewsSource'	email_mike dc:creator mike email_mike rdf:type 'ex:Email' soccer_blog rdf:type 'ex:Blog' soccer_blog rdf:type 'ex:NewsSource' news_blog rdf:type 'ex:News' news_blog dc:description "World cup news"	email_mike dc:creator mike *(Here the policies diverge: with paidNewsPolicy, Mike is not considered a good source)*
Andrew has trouble working with Adam and decides he is not a good source anymore. (context= *my_context*) Adam rdf:type 'ex:BadSource'	*Nothing happens*	*The following statements will not be trustful anymore:* email_adam rdf:type 'ex:Email' email_adam rdf:type 'ex:Email' Adam foaf:knows mike Reuters rdf:type 'ex:Website' Reuters rdf:type 'ex:NewsSource'

Table 2. Results of Trust Policy Application

The next steps in this work include giving explanations of why a fact was deemed trustful, similarly to [Bizer, 2005]; a refinement of the guidelines for making trust poli-

cies, possibly yielding a method; a deeper evaluation of the proposed formalism's properties (expressiveness and computational complexity, among others), specially when using negation as failure. We also plan to develop a case study in a realistic scenario, such as Social Semantic Desktops and Semantic Web Browsing, with large trust policies using RDF data.

References

ALMENDRA, V. S.; SCHWABE, D. Real-world Trust Policies. In: SEMANTIC WEB AND POLICY WORKSHOP, held in conjunction with INTERNATIONAL SEMANTIC WEB CONFERENCE – ISWC'05, 4., 2005, Galway, Ireland. Proceedings … Available at < http://www.csee.umbc.edu/swpw/papers/almendra.pdf>. Visited December 13, 2005.

BIZER, C.; CARROLL, J. Modeling Context using Named Graphs. In: SEMANTIC WEB INTEREST GROUP MEETING, 2004, Cannes. Available at <http://lists.w3.org/Archives/Public/www-archive/2004Feb/att-0072/swig-bizer-carroll.pdf>. Visited May 18, 2005.

BIZER, C.; OLDAKOWSKI, R. Using Context- and Content-Based Trust Policies on the Semantic Web. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, Alternate track papers & posters – WWW Alt.'04, 13., 2004, Hiroshima, Japan. Proceedings … New York: ACM Press, 2004. p. 228-229.

BIZER, C.; CYGANIAK, R.; MARESCH, O.; GAUSS, T. TriQL.P - Trust Policies Enabled Semantic Web Browser (2005). Available at <http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQLP/browser/>. Visited December 10, 2005.

CARROLL, J. J.; BIZER, C.; HAYES, P.; STICKLER, P. Named Graphs, Provenance and Trust. Bristol: HP Laboratories, Digital Media Systems Laboratory, 2004. 16 p. (Technical report HPL-2004-57).

CASTELFRANCHI, C.; FALCONE, R. Social Trust: A Cognitive Approach. In: Castelfranchi, C.; Yao-Hua Tan, eds. **Trust and Deception in Virtual Societies**. New York: Springer, 2001.

DECKER, S.; FRANK, M. R. The Networked Semantic Desktop. In: WWW2004 WORKSHOP ON APPLICATION DESIGN, DEVELOPMENT AND IMPLEMENTATION ISSUES IN THE SEMANTIC WEB, 2004, New York. Proceedings … CEUR-WS.org, 2004. Available at < http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-105/DeckerFrank.pdf>. Visited May 18, 2005.

DECKER, S.; SINTEK, M.; BILLIG, A. et al. TRIPLE - an RDF Rule Language with Context and Use Cases. In: W3C WORKSHOP ON RULE LANGUAGES FOR INTEROPERABILITY, 2005, Washington, DC, USA. Proceedings … W3C, 2005. Available at < http://www.w3.org/2004/12/rules-ws/paper/98>. Visited December 13, 2005.

GERCK, E. Toward Real-World Models of Trust: Reliance on Received Information. Available at <http://www.safevote.com/papers/trustdef.htm>. Visited May 16, 2005

GUHA, R. Open Rating Systems. In: Workshop on Friend of a Friend, Social Networking and the Semantic Web, 1., 2004, Galway, Ireland. Proceedings … Available at <http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/open_rating_systems/wot.pdf >. Visited May 31, 2005.

GUHA, R.; MCCOOL, R.; FIKES, R. Contexts for the Semantic Web. In INTERNATIONAL SEMANTIC WEB CONFERENCE – ISWC'04, 3., 2004, Hiroshima, Japan. Proceedings … New York: Springer, 2004. p. 32-46 .

HUYNH, D.; MAZZOCCHI, S.; KARGER, D. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In: INTERNATIONAL SEMANTIC WEB CONFERENCE – ISWC'05, 4., 2005, Galway, Ireland. Proceedings … New York: Springer, 2005.

MARCHIORI, M. W5: The Five W's of the World Wide Web. In: INTERNATIONAL CONFERENCE ON TRUST MANAGEMENT – iTrust'04, 2., 2004, Oxford, UK. Proceedings … New York: Springer. p. 27-32.

NEJDL, W.; OLMEDILLA, D.; WINSLETT, M. PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. In: WORKSHOP ON SECURE DATA MANAGEMENT IN A CONNECTED WORLD – SDM'04, 2004, Toronto, Canada. Proceedings … New York: Springer. p. 118-132.

RAO, P.; SAGONAS, K. F.; SWIFT, T.; WARREN, D. S.; FREIRE, J. XSB: A System for Efficiently Computing Well-Founded Semantics. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING AND NON-MONOTONIC REASONING – LPNMR'97, 4., 1997, Dagstuhl, Germany. Proceedings … New York: Springer. p. 431-441.

TUMMARELLO, G.; MORBIDONI, C.; PULITI, P.; PIAZZA, F. The DBin Semantic Web platform: an overview.
Available at <http://www.instsec.org/2005ws/papers/tummarello.pdf>. Visited July 1st, 2005.