Trust Policies for Semantic Web Repositories

Vinicius da S. Almendra[1], Daniel Schwabe[1]

(1) Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
Rio de Janeiro - Brazil
{almendra,dschwabe}@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 that describes the information retrieved. 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 horn logic model for trust policies, grounded on a real-world model of trust that offers justification for trust decisions and controlled trust measurement. We also propose the use of this model to enhance existing Semantic Web repositories with a trust layer.

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. 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 concepts found in the work of Gerck [10] and Castelfranchi et al. [7], 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 correspondence between these policies and the trust requirements we started with. This correspondence is important because 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 aggregates metadata from various sources and the agent must decide which metadata can be trusted and to what extent. Our contributions include a concept of real-world trust, a model to represent this trust concept in the scenario of interest, a simple model to express trust policies using horn logic, including justification for trust decisions, and an implementation, written in Java and Prolog, for the proposed model. It represents an improvement over our previous works [1, 2], where trust degrees and justification were not present.

In section 2 we describe the scenario in which this work fits in. In section 2 we describe the concept of real-world trust on which our work is based, and a model for trust on Semantic Web, based on this trust concept; we also describe a motivating ex-

ample. In section 4 we describe a model to express trust policies using horn logic. In section 4.3 we describe the implementation of the trust model using Java and Prolog. In section 6 we present works more closely related to our, discussing some differences. In section 7 we conclude our work and point to future directions.

2 A Motivating Scenario

The scenario we focus on is based on the Semantic Web Publishing scenario [6], the DBin project [17] and the Piggy Bank plugin [14]. They all work with the idea of a local repository of information that aggregates new statements from various sources. The first two also include the idea of trustfulness.

The Semantic Web Publishing scenario has 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, treating these graphs as claims by 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.

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 about computers that comes from direct friends".

This scenario can be integrated with the one of DBin's project, 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. The set of visible triples, which we call *accepted triples*, is analogous to the set of accepted graphs described above.

These tools can be integrated, giving rise to the scenario we are working with: an agent that continuously aggregates Semantic Web descriptions from various sources and uses these descriptions, together with trust policies, to decide which other descriptions are to be trusted. Recommender systems, reputation management systems, autonomous agents and Social Semantic Desktops can all be seen as particular instances of this scenario, when they are built on top of Semantic Web technologies.

The integration of Piggy Bank and the Semantic Web Publishing scenario, discussed in [4], exemplifies how trust filtering can be integrated to web browsing.

3 A Model for Trust

3.1 A Concept of Trust

To build a suitable trust model, we start by eliciting attributes of real-world trust, trying to capture its essence. Castelfranchi et al. [7] define 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 [10] presents a definition of trust as "what an observer knows about an entity and can rely upon to a qualified extent". This definition has a close parallel with Castelfranchi's: the observer is the agent who trusts; the entity is the trusted agent; the qualified extent is the behavior/action. Both associate 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 [7].

From both definitions, we observe that trust implies *reliance*: when an agent trusts something, s/he relies on its truth to achieve some goal without further analysis – even if s/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 sale conditions – 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 sale conditions at a bookstore that he does not trust[1]. So, the trust attitude entails a "potential" reliance on the *object* of trust.

We may also ask what the object of trust is. In this case, it could be described as the statement "Mary's bookstore is good". John *believes* this and will act upon it when needed. Using the definition of Gerck, John *knows* that "Mary's bookstore is good" is true and relies on this.

There is another question to be considered: how did John *decide* to trust Mary's bookstore? This is the problem of *justification* [10]. Castelfranchi et al. [7] ground the trust decision on the beliefs of the trusting agent. In our example, one reason John may have decided to trust Mary's bookstore is 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 might not trust Mary's bookstore, according to these premises. Notice that this does not preclude John from trusting Mary's bookstore for other reasons besides this one.

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), which characterizes *John's trust on those beliefs*. So, trust decisions may be *chained*: 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 certain 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 (directly observed by) 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*.

---

1 Absence of trust is different from *distrust*, which is a *positive evaluation* of negative qualities: one may not trust a stranger, but will almost certainly distrust a liar.

The justification of trust based on beliefs links trust with *belief revision*: if some of the beliefs that justified a trust decision are discredited, this trust may eventually be lost. If John discovers that several friends bought defective books from Mary, the belief about competence could be revised. Then, trust on Mary's bookstore could 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*. Here we use the underlying assumption that sometimes the absence of a belief (in this case, the belief that some people bought defective books from Mary) is treated as a positive belief (in this case, the belief that no one has ever bought defective books from Mary or, if someone did, it is not relevant to my decision). This is grounded on the assumption that, if something "wrong" happens (that is, something that may impact an agent's decisions), the agent will eventually be informed about it before he can make damaging decisions.

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 believes she is not competent, she does not know Japanese literature well and she does not worry about the tidiness of the bookstore. Here, we face contradictory beliefs and also different demands to consider a bookstore to be trustful. The difference between beliefs held by John and Mike may be due to the goals: John might be an occasional reader, whereas Mike might be an artist interested in Japanese culture. Note that this goes beyond being a matter of opinion: both make decisions and act based on these beliefs.

Trust also changes over time. John may lose his trust on Mary's bookstore even without any change in his beliefs about her. What changes in such cases is the justification required for trustfulness.

It is of common sense that trust is scalable [7], but this apparently conflicts with the concept of trust as a binary decision (to rely or not). Following Gerck's reasoning [10], the scalability of trust lies is the *degree of justification* required to trust. Stronger trust means stronger evidences. This implies an ordering on the possible justifications for trusting a fact: a better justification assigns a greater trust level to a trusted fact.

3.2 Trust and Semantic Web

At the core of Semantic Web technologies lays RDF (Resource Description Framework) and languages and formalisms based on it, most notably OWL (Web Ontology Language). RDF allows one to describe things using a controlled vocabulary, based on URIs (Uniform Resource Identifiers), through *statements* which are triples in the form (subject, property, object), meaning that the resource identified by *subject* has *property* with value *object*. An RDF document is a set of statements about some reality.

The fact that an agent (human or not) states something does not mean that it is true: one might state that Brazil's capital is Buenos Aires, which is not true (it is Brasília). When an agent uses (that is, relies on) an RDF document, it is implicitly trusting the source of the document on the statements contained in it, which means that s/he trusts

every RDF triple in it. Trusting an RDF triple (*S, P, O*) simply means that s/he believes *S* has the property *P* with value *O*. This trust decision is based on the information contained in the document and on other information available to the agent. From the work of [6], this trust decision may be called *accepting* a triple. After this decision, the triple becomes a belief of the trusting agent.

The decision of whether or not to accept a triple can be modeled as a trust policy which specifies which triples are to be trusted, depending on its components and on other triples the agent has already trusted.

Another important information when dealing with trust is provenance. There are some proposals [6, 9] to add provenance to RDF documents using a fourth element in RDF statements: the context. Although this is not an essential element for trust (one might gather provenance information from other sources), it enhances the expressivenesses of trust policies.

RDF triples may be stored anywhere (in a web page, in an email, in a document in the local file system etc.). In our work we focus on the idea of a repository which holds all triples that are going to be subject to trust evaluation. This solution avoids problems of distributed systems (e.g. lack of network connectivity) and circumscribes the universe of facts used in trust evaluation.

3.3 Outline of a Trust Model

Based on the trust concept formulated in section 3.1 and on Semantic Web concepts discussed in section 3.2, we build an informal model of trust, which comprises the following elements: facts, contexts, knowledge bases, trust policies, trust decisions, justification and trust layers.

A *fact* is a statement about reality, following the semantic of RDF triples. Some authors recognize the need for a fourth element: the *context* [3, 9, 13]. Contexts help define the provenance of the facts (who stated it), the circumstances (date, time, reason, etc.), and, more generally, help situating a fact in order to allow its correct understanding. As these elements are relevant to trust, we will assume the presence of this fourth element. This raises the need for an extension of RDF, such as the named graphs formalism [6]. We will not propose a new extension, but simply assume the availability of contexts and the possibility to obtain provenance and circumstantial information through it.

The set of facts that an agent knows constitutes 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 that 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 based on 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, which characterizes trust dynamism.

A *trust decision* is the act of testing if an asserted (or inferred according to the domain theory) 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. The trust policy specifies trusting agent decides what justifications are acceptable to trust a certain fact. From now on we will call justification any necessary and sufficient set of trusted facts needed to accept an asserted fact as trustworthy. The deduction is done by the trust policy, so a justification is always relative to a specific trust policy and to a specific asserted fact.

Not all justifications grant the same degree of confidence to the trusting agent. Two or three mentions in different web sites might be enough to justify buying a CD from an unknown internet dealer, but would not give enough confidence to buy a car. In both cases, what is in stake is the quality of service offered by the vendor and his/her honesty, but the degree of reliance exhibited in each decision is clearly different. Greater reliance levels demand better justifications.

Given a trust policy and a fact, a *justification level* is an equivalence class of all justifications that are equally good to the trusting agent for that fact with respect to that policy. The set of all justification levels is a partial order: in some situations we say that some justification is better than other, in others situations this statement does not make sense, as one would be comparing apples with oranges. An example of the former is a reputation management system: a person is more reliable than other if its reputation score is greater. An example of the latter is to compare his/her reputation score with the number of citations of papers authored by the other: although both share the idea of refereeing, a direct comparison of these scores does not make sense. A *justification class* is a set of justification levels that form a total order, that is, they are all comparable. Each fact that can have a varying degree of justification may yield a justification class.

A trust policy must assign some justification level to trusted facts. This justification level should reflect some property displayed by the justification that has varying degrees, e.g. the number of positive reviews. When this is not the case, the trust policy may assign the same justification level regardless of the justification facts.

The use of justification levels allows one to rank competing trusted facts in order to choose the one with more evidence. This feature allows the implementation of reputation systems based on trust policies, where the entities of interest are relevant facts from the trusting agent point of view.

The *trust layer* stands between a repository of RDF statements and the application and yields a list of trusted facts together with their justification levels. It does not alter the information in the repository: it just gives trust information under request.

Figure 1 shows the relationship among these concepts. The trust layer augments the knowledge base with justification information, represented by the circles (sets of facts), arcs (justification relationship) and labels on arcs (point out the policy used and the justification level achieved). Colored circles denote trusted facts. Notice that fact 9 was not trusted; policy 1 asserts the trustfulness of fact 4 with justification level A1 due to the presence of the trusted facts 1, 2 and 3. Fact 8 has two different justification levels.

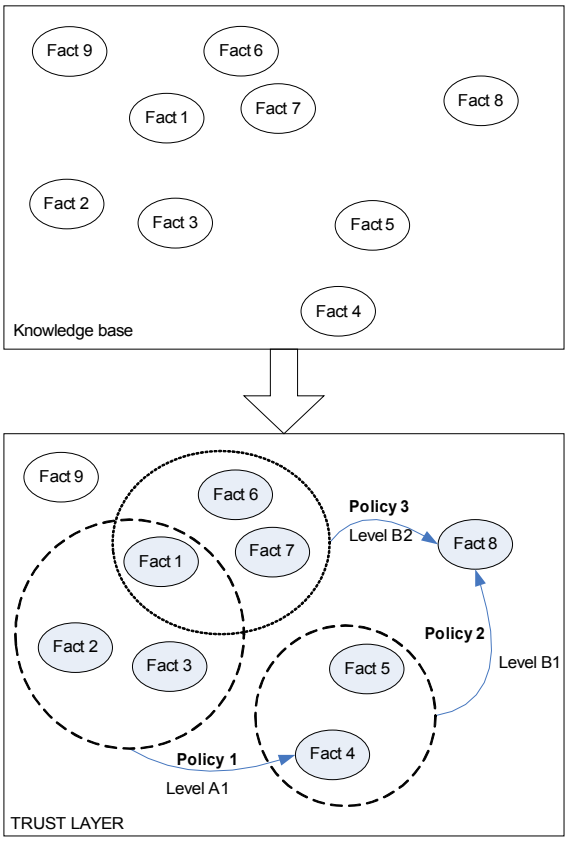

Fig. 1: A Knowledge base and its Trust Layer

3.4 A Motivating Example

Large companies normally have a purchasing department that takes care of buying supplies and purchasing services for the organization. This department deals with a wide variety of products and with many internal clients, that is, people that actually demand the purchasing of new goods or services. It also has a set of suppliers that may change over time, although at a slow rate. As suppliers' performance can vary widely in terms of quality of service, an important function of the purchasing department is to select the right suppliers, given budget restrictions and minimum quality of service requirements.

This task involves dealing of lots of information coming from suppliers, from previous purchases, coming from internal feedback about purchased goods or services, marketing information coming from specialized media etc. Feedback information can be regarded as kind of recommendation. The use of Semantic Web technologies together with trust policies fits well in this scenario, as it uses lots of structured information from various sources and needs to repeatedly assess the trustfulness of state-

ments related to the quality of service and recommendations. The proposed model does not improve over other possible solutions for this problem, but it enables a simpler and cheaper implementation, based on the reuse of existing standards, tools and ontologies like ROR (Resource of a Resource)[2] and eClassOWL[3].

This scenario naturally leads to Semantic Web Services solutions to e-business. Nevertheless, we are assuming a situation were the process is not fully automated: the system makes recommendations based on trust levels, among other factors.

In this scenario, we can devise some trust policies:

- One important factor when buying supplies is the delivery time, so the buyer needs to trust this information. This depends on the past performance of the vendor: if great delays already happened, the information supplied by the vendor is untrustworthy. Here we link data generated when supplies are received to advertised delivery time. The buyer can also choose among vendors with similar delivery times based on the justification level: the chosen vendor is the one whose delivery time has the greatest level of justification.
- Another important factor is the internal feedback from the actual users of supplies. This could be done using a simple ontology, linking a review to a specific purchase, which, in turn, is linked to a vendor. This review should point out some aspect advertised by the vendor that was contradicted (or corroborated) by reality. The system would then assign different justification levels depending on the reviews available.
- The reviews could also be subject of trust evaluation, as sometimes the internal clients may have biases toward some vendors. The reviews of an internal client that are always contradictory to other's reviews might lose their justification level.

The justification for trust is also useful for the buyer. Sometimes s/he might get puzzled with the results of trust evaluation, e.g. when the system assigns a high justification level to a vendor s/he dislikes or when a vendor known to be very good gets a low justification level. The buyer may ask for the facts used to derive trustfulness and may be convinced that the levels assigned were right or may change the trust policies in order to take into account factors that were not contemplated in the trust decision.

4 Building Trust Policies

4.1 A Horn Logic Approach

We will express the model outlined in the previous section, using definite horn clauses to build 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, object)* plus a *context*, similarly to other approaches [4, 7], yielding a quadruple, as discussed above. A context may also be the subject of a fact. We assume that there exists a repository from

2 http://www.rorweb.com/

3 http://www.heppnetz.de/eclassowl/

which all RDF triples come from. There may be some "context" layer that adds contextual information to triples, in the case the repository used does not support this.

A justification class *JC* is represented by a tuple *(J, R)*, where *J* is a nonempty set and *R* is a total order on *J*. The elements of *J* are justification levels. There must be at least one justification class with at least one justification level. The union of all justification levels of all justification classes is called *justification space* (*JS*). The *support set* of a fact is the set of facts that is used to justify trust on it. A fact may have many support sets. A *trust policy* is a predicate with the following signature:

p (fact, j, supportSet)

This predicate means that *fact* can be trusted with justification level *j* given the support set *supportSet*, which is a list of facts.

Trust policies have the following characteristics:

- A fact can be trusted with more than one pair of justification level-support set, as the justification level depends on the support set.
- The policy will find *fact* trustful only if it can deduce the truthfulness of every fact contained in the support set.

The process of finding the trustfulness of an arbitrary fact is recursive and raises the question: which trust policy should be used?

A first solution would be to assume that a single trust policy should decide trustfulness of all facts. So it would use itself recursively to decide trustfulness of the facts contained in the support set. However, as the policy is not tied to specific facts, it does not show clearly the link between the support set and the fact being analyzed. Another solution would be to have specialized policies, each one for a small set of facts. This raises another problem: each policy should be aware of all other policies in order to choose the right one.

We adopt an intermediate solution, assuming the existence of a "general" trust policy that decides trustfulness of all facts, but building this policy through aggregation of smaller (i.e., more restricted) ones. Each component policy must be aware only of the general policy. This general policy identifies itself with the trusting agent's adopted trust policy, as it decides the trustfulness of all facts. From now on we will call it the *root trust policy*. The aggregation can be done using multiple clauses, each one aggregating a new policy, so the root trust policy becomes a disjunction of other policies. For example:

$$rootTrustPolicy(fact, j_1, support1) \leftarrow foafPolicy(fact, j_1, support1)$$
$$rootTrustPolicy(fact, j_2, support2) \leftarrow dcPolicy(fact, j_2, support2)$$
$$rootTrustPolicy(fact, j, support) \leftarrow systemPolicy(fact, j, support)$$

Policies other than the root trust policy will be called elementary trust policies or simply trust policies. In this work we will not go further in the issue of policy composition, as we are investigating the possibility of adopting Bonatti's [5] algebra of policies.

Now we must describe elementary trust policies. An elementary trust policy has the following components: scope definition, a justification, a trustfulness check and a justification level assignment.

The *scope definition* is a set of logical conditions that circumscribes the set of facts that are examined by this policy; facts outside this set are ignored. The simplest restriction is to specify the value of some of the components of the fact, e.g. the predicate. More sophisticated scopes could be ones like "all facts belonging to the FOAF ontology", "all facts about books written by J. R. R. Tolkien". The *justification* is composed by a set of conditions that link the fact being examined to other asserted facts (and their components) and put constraints on their values, just like an ordinary database query. The *trustfulness check* consists of checking all facts used in justification for trustfulness using the root trust policy. The *justification level assignment* is the determination of the justification level of the fact being examined. It can be calculated using facts gathered in the justification step. For example, the justification degree of a fact related to a person might depend to the number of people that stated friendship with this person.

In this model, the trusting agent has no "privilege": it is a source of information like any other. The default policy for self-asserted information would be of accepting it with a high justification level, but nothing prevents the use of policies that give more trust to some other person's statements than to self-asserted ones.

Trust policies are built around sets of facts. For each relevant (from the trusting agent point of view) set of facts, one or more trust policies can be built. As policies are specified on a per-fact basis, the relationship among them is always done through the root trust policy. This makes the trust policy set easily maintainable, as there are no direct references from one policy to another.

Trust policies may use whatever data is available in the repository. Some policies might demand strong evidences, like PGP signatures or similar mechanisms; other may accept lightweight evidence, like FOAF statements gathered from web pages without further warrants of validity.

Next we show an example of the trust policy "trust the e-mail of a person if s/he is known by a friend of mine", along with some explanation. Variables are in italics.

foafMboxPolicy(*testFact, justificationLevel, supportSet*) ←	
testFact = (*person1*, foaf:mbox, *email*, *context1*) AND	Fact to be tested
fact1 = (*person2*, foaf:knows, *person3*, *context2*) AND *fact2* = (*person3*, foaf:mbox_sha1sum, *sha1*, *context2*) AND hasSHA1(*email*, *sha1*) AND *fact3* = (myself, foaf:knows, *person2*, *myContext*) AND *fact4* = (*context2*, dc:creator, *person2*, *context2*) AND *fact5* = (*myContext*, dc:creator, myself, *myContext*) AND	Finds a person that has the same email and is known by a friend of mine
supportSet = *{fact1*, *fact2*, *fact3*, *fact5*, *fact5}* AND isInRepository(*supportSet*) AND	Tests if the support set is in the knowledge base
checkTrustPolicy(*supportSet*) AND	Checks support set trustfulness
justificationLevel = foaf_mbox	Assigns justification level

This example policy supposes a network of friends that exchange lists of hashed emails, in order to avoid spam without compromising other people's privacy. Notice the use the predicate hasSHA1, which must be defined elsewhere.

To illustrate the recursive behavior, we can show how a policy that accepts a person as the author (i.e. provenance) of a context would look like:

dcCreatorPolicy(*testFact, justificationLevel, supportSet*) ←
testFact = (*context*, dc:creator, *person*, *context*) AND
fact1 = (*context*, wot:assurance, *signature*, *context*) AND
fact2 = (*person*, foaf:mbox, *email*, *context*) AND
getPublicKey(*email*, *pubkey*, public_key_server) AND
checkSignature(*context*, *signature*, *pubkey*) AND
supportSet = {*fact1*, *fact2*} AND
isInRepository(*supportSet*) AND
checkTrustPolicy(*supportSet*) AND
justificationLevel = default

This policy uses the Web of Trust vocabulary and custom predicates in order to check provenance using public key infrastructure.

However, there is a circular dependence: the first policy checks trustfulness of foaf:mbox facts but needs the second one to check dc:creator statements; the second one needs to check foaf:mbox statements, which is done by the first.

The solution lies in the use of justification levels: the second policy does not need a strong verification, as this will be provided by the signature check. So, it can rely on another policy that assigns a default (and lower) trust level to foaf:mbox statements. Notice the use of negation as failure:

foafMboxPolicySimple(*testFact, justificationLevel, supportSet*) ←
testFact = (*person1*, foaf:mbox, *email*, *context1*) AND
supportSet = {} AND
justificationLevel = default

This sample presumes that foaf_mbox > default

4.2 Distrust and the Use of Negation

The use of definite (Horn) clauses to express trust policies restricts the use of negation. Intuitively, only positive facts can be tested. However, there may be situations where one understands that the absence of some facts conveys some meaning, as noted in Section 3.1. This can be represented in trust policies by the use of *negation as failure*: the negation of a formula is true iff one cannot prove that formula's truth.

In our model, distrust can be modeled as a trust policy that puts a fact into a unique justification class, different from all others. Then we can use negation as failure to says that a fact can only be trusted if it is *not* trusted within that justification class, that is, distrust always prevents trust, no matter the justification level achieved.

However, the use of negation raises some issues. Semantic Web technologies are based on the Open World Assumption, where the absence of a fact does not imply its falseness. So, when using negation as failure, one assumes a completeness of knowledge that is not warranted by the underlying model of RDF. The only guarantee that

s/he has is the control over the repository used to make trust evaluations. Therefore, it is important for the user to understand whether this assumption is consistent with his desired meaning for a policy that uses negation, given its goals.

4.3 Limitations of Horn Clauses

The use of horn clauses allows one to express easily reasoning based on material implication, linking a fact to sets of necessary and sufficient evidences. Nonetheless, they may not be adequate to express some conditions, e.g. those based on quantity constraints and on statistical calculations.

One possible solution is to use features available in the Prolog implementation, like `findall/3` and `is/2` predicates, to overcome some limitations, especially those related to quantity restrictions. More complex conditions, like ones related to social network analysis, may be evaluated outside Prolog, given the restriction that only trusted facts are used in their evaluation. A similar approach is taken in TriQL.P [4].

5 Implementation

We implemented the idea of trusted repositories using a Java library that wraps an existing RDF repository, runs trust policies on demand and offers programmatic access to its conclusions: which facts were trusted, the facts used to justify each trusted fact and their justification levels.

To apply the trust policies, we used XSB Prolog [16]. We choose this Prolog implementation due to its ability to cope nicely with recursive predicates, avoiding infinite loops. The XSB Prolog runs the inference to find out which RDF triples are trustworthy; the Java library interfaces with the XSB Prolog code, converting the RDF triples to be analyzed to Prolog and converting the results back. These results are not added directly to the repository, although this could be easily done using reification. However, we opted for a less intrusive approach, leaving to the client application the decision to store or not the computed trust information in the repository.

Trust information is generated on demand. When new facts are added to the repository, it is necessary to run the trust engine again to update trust information. We did not develop yet a strategy for incremental update.

Justification levels are mapped to a pair (URI, number), where *URI* denotes the justification class and *number* denotes the *justification degree*, which is an integer that fulfils the following relation:

$$j_1 = (URI_1, n_1), j_2 = (URI_2, n_2), URI_1 = URI_2, n_1 \geq n_2 \rightarrow j_1 \geq j_2$$

This conveys the formal model's restriction that two justification levels are comparable when they belong to the same justification class. The use of an integer eases the task of computing and using the results of trust. However, this number just expresses ordering: there is nothing in the formal model that guarantees meaningfulness of arithmetic operations on these numbers.

RDF statements are converted to Prolog facts with the following form:

```
fact(Subject, Predicate, Object, Context, ID).
```

The values of these elements can be URIs, which are mapped to Prolog atoms, strings, which are also mapped to atoms, and numbers. Blank nodes are mapped back and forth to fake URIs. Trust policies are represented as Prolog clauses. Logical conditions translate to terms; Prolog's dynamic database is used as the knowledge base (in fact, just those terms whose head are of the form fact/5).

Right now, we have a prototype of this library that wraps Jena repositories, Sesame local repositories, Named Graphs sets [6] and DBin [17] trusted repositories.

6 Related Work

The work of Gerck [10] provides a lengthy discussion on the concept of trust as reliance on information. He contrasts this concept with other commonly used definitions for trust. He also offers a conceptual framework to reason about trust levels grounded on common sense reasoning, where trust on some information is justified if the trust agent is "convinced" of its truth. Castelfranchi et al. [7] present a cognitive model of trust, rooted in multi-agent systems domain. They also characterize trust as reliance, corroborating Gerck's model with some minor modifications.

Carroll et al. [6] proposes the named graphs extension to RDF, which adds URIs to graphs under a well-defined semantics. Among other things, this extension opens interesting possibilities to express trust policies based on provenance information, as shown in their work. These policies are used to build a set of accepted graphs, which is the set of graphs that contribute to the meaning of the entire knowledge base (in their terms, the set of named graphs); when specified by trust policies, it is really the set of trusted graphs. So, an entire named graph could be trusted or not. Our previous work [2] adopted a similar view: we worked with the set of trusted RDF triples.

The application of named graphs to the trust policies field continued with the work of Bizer et al. [4] the TriQL.P browser, which enhances the Piggy Bank browser [14] with trust policies based on the named graphs extension and on TriQL.P query language [4]. Our first work on trust modeling [1] used TriQL.P to describe and apply elementary trust policies. A fundamental difference among this work and ours is the recursive nature of our model: trust is always decided based on previously trusted statements. As we needed recursive queries to implement our trust model, we moved away from TriQL.P to XSB Prolog, in order to express recursive queries more naturally. Even so, our elementary trust policies greatly resemble this policy language and may benefit from the explanation engine built into TriQL.P browser.

As our work also uses the idea of trust measurement, it can be compared to other ones that embrace this idea when specifying trust policies. Golbeck's work [11] is one of them. It offers an approach to calculate trust on a web-based social network, grounded on the concept of trust as reliance among agents in the network. Each agent states trust on some other agents; this leads to a social network whose directed edges express trust of an agent on other. This trust attitude has degrees, which are weights of the graph edges. From this model, Golbeck proposes algorithms to infer trust

among people connected only indirectly through the network. We adopt a different approach to trust measurement, as we believe that assigning numbers to trust relationships and than making computation with these numbers can be misleading, as the semantics of the result is not clear, even if it obeys some kind of intuition about how trust works in real life. We use the idea of justification levels in order to incorporate the notion of trust levels while keeping the semantics clear, e.g. forbidding direct comparison of different justification classes without some underlying theory that justifies this.

The implementation of trust engines grounded on logic is shared by works like PeerTrust [15] and SULTAN [12]; they also share the idea of collecting evidence to decide trustfulness. However, none of them is concerned with Semantic Web data.

7 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 knowledge bases, while preserving real-world semantics of trust. We first specified a model capturing relevant aspects of the trust concept, such as reliance, subjectivity, dynamism, justification, measurement. Then proceeded to build a horn logic model to express trust policies that may be built incrementally through the concepts of elementary trust policies and root trust policies. We presented a motivating example where the described model offers a solution and described a test implementation we have made, which can be used as a trust layer for an RDF repository.

The next steps in this work include a deeper study of justification levels, in order to provide a more solid theoretical background to this concept; building guidelines for defining trust policies, possibly yielding a method; the explicit inclusion of non-monotonocity in the model, especially negation as failure; exploring the idea of trust delegation, i.e. using trust information from other agents. We also plan to develop a case study in a realistic scenario, such as Social Semantic Desktops, Semantic Web Browsing and Social Semantic Collaborative Filtering, with large trust policies using RDF data.

8 Acknowledgements

This research was sponsored by UOL (www.uol.com.br), through its UOL Bolsa Pesquisa program, process number 20060601215400a. We would like to thank the reviewers for their valuable comments and suggestions.

9 References

1. Almendra, V. S., Schwabe, D. Real-world Trust Policies. In Proceedings of Proceedings of the Semantic Web and Policy Workshop, 4th International Semantic Web Conference (Galway, Ireland, November 2005).
2. Almendra, V. S., Schwabe, D., Casanova, M. A. Towards Real-world Trust Policies. Technical report MCC42/05, Informatics Department, PUC-Rio (2005).
3. Bizer, C., Carroll, J. Modeling Context using Named Graphs. Semantic Web Interest Group Meeting, March 2004, Cannes, France.
4. Bizer, C., Cyganiak, R., Maresch, O., Gauss, T. TriQL.P - Trust Policies Enabled Semantic Web Browser. http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQLP/browser/
5. Bonatti, P., Vimercati, S. C., Samarati, P. An Algebra for Composing Access Control Policies. ACM Transactions on Information and System Security, Vol. 5, No. 1, February 2002, Pages 1–35.
6. Carroll, J. J., Bizer, C., Hayes, P., Stickler, P. Named Graphs, Provenance and Trust. Technical report HPL-2004-57 (2004).
7. Castelfranchi, C., Falcone, R. Social Trust: A Cognitive Approach. In: Castelfranchi, C.; Yao-Hua Tan (Eds.): Trust and Deception in Virtual Societies. Springer-Verlag (2001).
8. Decker, S., and Frank, M. R. The Networked Semantic Desktop. In Proceedings of the WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web (New York, NY, USA, May, 2004).
9. Decker, S., Sintek, M., Billig, A. et al. TRIPLE - an RDF Rule Language with Context and Use Cases. In Proceedings of W3C Workshop on Rule Languages for Interoperability, (Washington, DC, USA, April 2005), W3C, 27-28.
10. Gerck, E. Toward Real-World Models of Trust: Reliance on Received Information. http://www.safevote.com/papers/trustdef.htm.
11. Golbeck, J., Parsia, B., Hendler, J. Trust Networks on the Semantic Web. ISWC'03.
12. Grandison, T., Sloman, M. Trust Management Tools for Internet Applications. Proc 1st Intl. Conference on Trust Management, May 2003, Crete.
13. Guha, R., McCool, R., and Fikes, R. Contexts for the Semantic Web. In Proceedings of the ISWC'04 (Hiroshima, Japan, November 2004), Springer.
14. Huynh, D., Mazzocchi, S., and Karger, D. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In Proceedings of ISWC'05 (Galway, Ireland, November 2005), Springer.
15. Nejdl, W., Olmedilla, D., Winslett, M. PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. Workshop on Secure Data Management in a Connected World (SDM'04) in conjunction with 30th International Conference on Very Large Data Bases, Aug.-Sep. 2004, Toronto, Canada
16. Rao, P., Sagonas, K. F., Swift, T., Warren, D. S. and Freire, J. 14: A System for Efficiently Computing Well-Founded Semantics. http://citeseer.csail.mit.edu/rao97xsb.html
17. Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F. The DBin Semantic Web platform: an overview. http://www.instsec.org/2005ws/papers/tummarello.pdf.