Named Graphs

Jeremy J. Carroll [a] Christian Bizer [b] Pat Hayes [c] Patrick Stickler [d]

[a] *Hewlett-Packard Labs, Bristol, UK*
[b] *Freie Universität Berlin, Germany*
[c] *IHMC, Florida,USA*
[d] *Nokia, Finland*

Abstract

The Semantic Web consists of many RDF graphs nameable by URIs. This paper extends the syntax and semantics of RDF to cover such named graphs. This enables RDF statements that describe graphs, which is beneficial in many Semantic Web application areas. Named graphs are given an abstract syntax, a formal semantics, an XML syntax, and a syntax based on N3. SPARQL is a query language applicable to named graphs. A specific application area discussed in detail is that of describing provenance information. This paper provides a formally defined framework suited to being a foundation for the Semantic Web trust layer.

Key words: Semantic Web, RDF, SPARQL, Provenance, Digital Signatures

1 Introduction

A simplified view of the Semantic Web is a collection of web retrievable RDF documents, each containing an RDF graph. The RDF Recommendation [1–4] explains the meaning of any one graph, and how to merge a set of graphs into one, but does not provide suitable mechanisms for talking about graphs or relations between graphs. The ability to express metainformation about graphs is required for:

Data syndication systems need to keep track of provenance information and provenance chains.

Restricting information usage Information providers might want to attach information about intellectual property rights or their privacy preferences to graphs in order to restrict the usage of published information [5,6].

Access control A triple store may wish to allow fine-grain access control, which appears as metadata concerning the graphs in the store [7].

Signing RDF graphs As discussed in [8], it is often necessary to keep the graph that has been signed distinct from the signature, and other metadata concerning the signing, which may be kept in a second graph.

Expressing propositional attitudes such as modalities and beliefs [9].

Scoping assertions and logic where logical relationships between graphs have to be captured [10–12].

Ontology versioning and evolution OWL [13] provides various properties to express metadata about ontologies. In OWL Full, these ontologies are RDF graphs. Ontology versioning and evolution is discussed in [14,15].

RDF reification has well-known problems in addressing these use cases as previously discussed in [16]. To avoid these problems the use of quads has been proposed by several authors [7,17–19]. These consist of an RDF triple and a further URIref or blank node or ID. The proposals vary widely in the semantics of the fourth element, using it to refer to information sources, to model IDs or statement IDs or more generally to 'contexts'.

We propose a general and simple variation on RDF, called *named graphs*. A named graph is an RDF graph which is assigned a name in the form of a URIref. The name of a graph may occur either in the graph itself, in other graphs, or not at all. Graphs may share URIrefs but not blank nodes.

Named graphs can be seen as a reformulation of quads in which the fourth element's distinct syntactic and semantic properties are clearly distinguished, and the relationship to RDF's triples, abstract syntax and semantics is clearer.

Named graphs are a deliberately small step on top of the RDF and OWL Recommendations. This allows their use with tools built as implementing those recommendations, in a backward compatible way, with little or no code modifications.

The first half of the paper covers: the abstract syntax and semantics of named graphs; their relationship with RDF, OWL, TRIPLE, Guha's contexts and SPARQL RDF dataset. We then discuss the TriX, TriG and RDF/XML syntaxes for named graphs and the query language SPARQL.

The second half describes how named graphs can be used for Semantic Web publishing, looking in particular on provenance tracking and how it interacts with the choices consumers of Semantic Web information make about which information to trust. We provide a vocabulary for Semantic Web publishing with its formal semantics. The vocabulary includes support for digital signatures and addresses performative acts, such as asserting RDF.

This paper is an extended version of the paper presented at the World Wide Web Conference (WWW 2005) [20].

2 Abstract Syntax

RDF syntax is based on a mathematical abstraction: an RDF graph is defined as a set of triples. These graphs are stored in documents which can be retrieved from URIs on the Web. Often these URIs are also used as a name for the graph, for example with an `owl:imports`. To avoid confusion between these two usages we distinguish between named graphs and the RDF graph that the named graph encodes or represents. A named graph is an entity with two functions *name* and *rdfgraph* defined on it which determine respectively its name, which is a URI, and the RDF graph that it encodes or represents. These functions assign a unique name and RDF graph to each named graph. In this way, a named graph is a resource, identified by its name, and so it can be described in the usual open way using RDF.

More formally, let U be the set of all URI references, B an infinite set of RDF blank nodes, and L the set of all legal RDF literals (all three sets as defined in [4]); U, B and L are pairwise disjoint; let $V = U \cup B \cup L$ be the set of *nodes*; then the set $T = V \times U \times V$ is the set of all RDF triples [1] . The set of RDF graphs G is the power set of T. A named graph is a pair $ng = (n, g)$ with n in U and g in G. We write $name(ng) = n$ and $rdfgraph(ng) = g$. To enforce the blank node scoping rules ([3]) we make the global assumption that blank nodes cannot be shared between different named graphs, i.e. if ng and ng' are different named graphs then the sets of blank nodes which occur in triples in $rdfgraph(ng)$ and in $rdfgraph(ng')$ are disjoint.

All of the above definitions may be relativized to a particular set of URIrefs, or to a particular set of named graphs. Any set of named graphs can be thought of as a partial function from U into the power set of T.

3 Formal Semantics

The semantics of graph naming are a simple semantic extension of the RDF(S) semantics: we will say that an RDF(S) interpretation I (as in [3]) *conforms* with a set of named graphs N when: For every named graph $ng \in N$, $name(ng)$ is in the vocabulary of I and $I(name(ng)) \equiv ng$, where $\equiv$ is equivalence ignoring blank node names. Note that the named graph itself, rather than the RDF graph it intuitively 'names', is the denotation of the name. We consider the RDF graph to be related to the named graph in a way analogous to that in which a class extension is related to a class in RDFS. This intensional (c.f. [3]) style of modelling allows for distinctions between several copies of a single RDF graph (with distinct names) and avoids pitfalls arising from accidental identification of similar named graphs.

[1] We have removed the legacy constraint that a literal cannot be the subject of a triple.

The RDF recommendations [4] defines a notion of graph equivalence, which treats two RDF graphs which differ only in the identity of their blank nodes as being the 'same' graph. We will make a similar assumption. The mathematical details of 'renaming' functions are addressed below. In practice, this amounts to permitting RDF processors to freely re-name any blank node identifiers when required in order to maintain the no-sharing condition.

The intuitive meaning of a named graph G is the standard RDF meaning [3] of its associated RDF graph *rdfgraph*(G), which we will refer to as the *graph extension*. Any assertions in RDF about the graph structure of named graphs are understood to be referred to these graph extensions, just as the meanings of the RDFS class vocabulary are referred to relationships between the class extensions. As an example of this meaning, we can define two properties `rdfg:subGraphOf` and `rdfg:equivalentGraph`, with semantics defined as follows:

$\langle f, g \rangle$ is in *IEXT*$(I($`rdfg:subGraphOf`$))$
iff *rdfgraph*(f) is a subset of *rdfgraph*(g)

where the subset holds in a manner performing any necessary blank node renaming, as discussed below.

$\langle f, g \rangle$ is in *IEXT*$(I($`rdfg:equivalentGraph`$))$
iff *rdfgraph*$(f) \equiv$ *rdfgraph*(g)

where, again, equivalence is understood as renaming blank nodes as appropriate.

3.1 Details of Renaming

Equivalent graphs are discussed in [21]. Graph f and g are equivalent when there is a bijection between the blank nodes of f and the blank nodes of g, which extends to a bijection between the triples in f and the triples in g.

Thus, $I($*name*$($*ng*$)) \equiv$ *ng* means that $I($*name*$($*ng*$))$ is a pair $($*name*$($*ng*$)), f)$, where there is such a bijection between the blank nodes of $I($*rdfgraph*$($*ng*$))$ and the blank nodes of f.

rdfgraph$(f) \equiv$ *rdfgraph*(g) in the definition of `rdfg:equivalentGraph` is similarly understood.

For `rdfg:subGraphOf`, the phrase "*rdfgraph*(f) is a subset of *rdfgraph*(g)" intends that there is an injection ϕ from the blank nodes of *rdfgraph*(f) to the blank nodes of *rdfgraph*(g), such that when ϕ is extended with the identity over the URI reference and literal nodes of *rdfgraph*(f), we have $\phi($*rdfgraph*$(f)) \subset$ *rdfgraph*(g).

As shown in [21], RDF graph equivalence is GI-complete (i.e. in the same isomorphism class as graph isomorphism). This is generally believed to lie strictly between the P and NP complexity classes. Similar argument would show that the `rdfg:subGraphOf` relationship is equivalent to standard subgraph isomorphism, which is known to be NP-complete.

3.2 Accepting Graphs

A set of named graphs **N** has not been given a single formal meaning. Instead, the formal meaning depends on an additional set $A \subset \text{domain}(N)$. A identifies some of the graphs in the set as *accepted*. Thus there are $2^{|\text{domain}(N)|}$ different formal meanings associated with a set of named graphs, depending on the choice of A. The meaning of a set of accepted named graphs $\langle A, \mathbf{N} \rangle$ is given by taking the graph merge $\bigcup_{a \in A} N(a)$, and then interpreting that graph with the RDF semantics [3], subject to the additional constraint that all interpretations I conform with **N** ($N(a)$ is the graph associated with the URI reference a in N). Named graphs can be used with any of the various levels of semantic theories for RDF: simple, RDF, RDFS or datatyped interpretations from [3], or OWL Full interpretations from [22]. It is a deliberate choice to work in this way with the deployed Semantic Web recommendations, rather than inventing a new semantics with special features, perhaps from modal logic, to reflect potential conflict between different graphs on the Semantic Web.

The choice of A reflects that the individual graphs in the set may have been provided by different people, and that the information consumers who use the named graphs make different choices as to which graphs to believe. Thus we do not provide one correct way to determine the 'correct' choice of A, but provide a vocabulary for information providers to express their intensions, and suggest techniques with which information consumers might come to their own choice of which graphs to accept. Issues as to how to resolve conflicts between different graphs, and how to determine A, are seen as pragmatic issues, to be dealt with by application developers, rather than logical issues to be dealt with by formal semantics. A motivation is that different applications will have different tolerances to errors, inconsistencies and variability between the data, and a unified formal approach is likely to be overkill for some, yet may miss key features required by another (e.g. some more formal approaches to context [23–25] fail to address digital signatures, vital for financially sensitive applications).

We consider three further issues of detail in the relation between named graphs and RDF and OWL: the open world assumption; RDF reification, and OWL imports.

3.3 The Open World Assumption

Both RDF and OWL operate with the open world assumption. RDF Concepts [4] says:

> RDF is an open-world framework that allows anyone to make statements about any resource. In general, it is not assumed that complete information about any resource is available.

The OWL Guide [26]:

> OWL makes an open world assumption. That is, descriptions of resources are not confined to a single file or scope. While class C_1 may be defined originally in ontology O_1, it can be extended in other ontologies.

As is clear from these quotations, openness here means that a description of a resource is considered to be open-ended. Actual web objects such as files and RDF graphs can however be identified and rigidly named, so that the name uniquely identifies the resource. Named graphs utilize this ability to attach a name rigidly to a graph. Thus the mapping between names and graphs fixes the graph corresponding to a name in a rigid, non-extensible way. Two different Web documents asserting different graphs named by the same URI contradict one another. However, two different graphs with different names may make statements about the same resources. Thus the named graph framework facilitates the open world of the Semantic Web; not only can different people make different (hopefully complementary) statements about the same resource, but it is possible to keep these statements separate, and it is possible to combine them. The choice of which of these two is more appropriate is explicitly application specific.

Summarizing, if document A contains a graph g named u making statements about a resource r, a further document B that is consistent with A cannot use the name u for a different graph g'. However, B can contain a graph g' named u' making further statements about r. Thus the named graphs framework maintains the open-world framework of RDF, but treats graph naming as a form of rigid identification.

3.4 RDF Reification

A 'reified statement' [3] is a single RDF statement described and identified by a URI reference. Within the framework of this paper, it is natural to think of this as a named graph containing a single triple, i.e. a *named triple*. With this convention, the subject of `rdfg:subGraphOf` can be a reified triple, and the property can be used to assert that a named graph contains a particular triple. This provides a useful connection with the traditional use of reification and a potential migration path.

However, the semantics of a single triple graph differ from the (lack of) semantics offered to a reified statement by the RDF recommendation [3], better addressing traditional uses of reificiation such as providing metadata about triples and quoting.

RDF reification operates at the semantic level, not the syntactic, so that:

```
_:r rdf:type rdf:Statement .
_:r rdf:subject eg:s1 .
_:r rdf:predicate eg:p .
_:r rdf:subject eg:o .
_:r dc:create "Jeremy Carroll" .
eg:s1 owl:sameAs eg:s2 .
```

which informally says that someone named “Jeremy Carroll” is the creator of the statement `eg:s1 eg:p eg:o`, both entails and is entailed by (with the OWL semantics for `owl:sameAs`):

```
_:r rdf:type rdf:Statement .
_:r rdf:subject eg:s2 .
_:r rdf:predicate eg:p .
_:r rdf:subject eg:o .
_:r dc:create "Jeremy Carroll" .
eg:s1 owl:sameAs eg:s2 .
```

which informally says that someone named “Jeremy Carroll” is the creator of the statement `eg:s2 eg:p eg:o`. This difference of intent behind the two graphs, is not reflected in the formal semantics: the two graphs are semantically equivalent. Thus, RDF reification fails to make this simple distinction.

Named triples solve this, for example as follows. The syntax for this example is TriG explained in section 5.3.

```
:t1 {
 eg:s1 eg:p eg:o .
}
:t2 {
 eg:s2 eg:p eg:o .
}
:g1 {
 :t1 dc:creator "Jeremy Carroll" .
 eg:s1 owl:sameAs eg:s2 .
}
:g2 {
 :t2 dc:creator "Jeremy Carroll" .
 eg:s1 owl:sameAs eg:s2 .
}
```

Despite the `owl:sameAs` triple, `g1` and `g2` are saying different things, which is reflected in them not entailing one another.

The RDF reification of a statement neither implies, nor is implied by, the statement itself. The relationship between a named triple and the related statement (the interpretation of the triple), is slightly different. In the example above, there are four named graphs. The semantics given to this collection of named graphs depends on which of these are accepted. If either of the named triples is accepted then the related statement is true in all associated interpretations, if a named triple is not accepted then there may be interpretations in which it the related statement is not true.

Named triples may be combined with RDF reification, noting the possibility expressed in RDF Sematics [3]:

> Semantic extensions MAY limit the interpretation of these so that a triple of the form
> `aaa rdf:type rdf:Statement .`
> is true in I just when $I(aaa)$ is a token of an RDF triple in some RDF document, and the three properties, when applied to such a denoted triple, have the same values as the respective components of that triple.

In this case, any interpretation conforming with a set of named graphs including a named triple, will satisfy the reification of that triple. For example, any interpretation conforming with the four named graphs above, with the above extension, would entail:

```
:t1 rdf:type rdf:Statement .
:t1 rdf:subject eg:s1 .
:t1 rdf:predicate eg:p .
:t1 rdf:object eg:o .
:t2 rdf:type rdf:Statement .
:t2 rdf:subject eg:s2 .
:t2 rdf:predicate eg:p .
:t2 rdf:object eg:o .
```

This includes interpretations that do not accept any of the graphs; and as before there are interpretations that accept `:g1` and do not accept `:g2`, so that the named triples preserve the syntactic intent of most use cases for reification.

Named triples also allow named graphs to include claims about the contents of other named graphs. For example:

```
:t1 {
 eg:s1 eg:p eg:o .
}
```

```
:g3 {
 :t1 rdfg:subGraphOf :g4 .
}
```

Here, `:g3` is satisfied only by interpretations in which `:g4` is a named graph and contains the triple `:t1`. This does not make any claims about whether or not that triple is true.

3.5 OWL Imports

OWL imports processing uses the URI object of an `owl:imports` triple to locate an additional RDF/XML file to be included in an ontology, as in [22], with K a collection of RDF graphs:

> K is imports closed iff for every triple in any element of K of the form x `owl:imports` u . then K contains a graph that is the result of the RDF processing of the RDF/XML document, if any, accessible at u into an RDF graph.

Using the named graphs it is more natural to use the name of a graph as the object of `owl:imports`; so that the notion of imports closure is applied to a collection K of named graphs, and the definition is reworded as:

> K is imports closed iff for every triple in any element of K of the form x `owl:imports` u . then K contains a graph that is named u.

The URI u still may act as a locator, used to retrieve an RDF/XML document that is parsed to give a graph named u. The retrieval is unnecessary if the graph is available through other means, e.g. a cache (like with Jena's OntDocumentManager), or a local copy, or as part of a TriX document (see section 5.2 for the TriX syntax). There is a consistency question: do two different copies of a named graph agree? This can perhaps be resolved by phrasing it as: does a copy of a named graph agree with the graph found by the retrieval action?

4 Related Work

Previous authors of research work addressing the semantics of a collection of documents on the Semantic Web have tended to have rich theories for addressing the relationship between multiple contexts.

4.1 TRIPLE

TRIPLE [24] provides graphs named with resources, and a Horn clause language for defining inferences etc. e.g.

```
@dfki:documents {
   dfki:d_01_01 [
     dc:title → "TRIPLE";
     dc:creator → "Michael Sintek";
     dc:creator → "Stefan Decker" ;
     ...   ].
   ∀ S,D search(S,D) ←
      D[dc:subject → S].
}
```

By mixing up the data representation (i.e. the named graph), the implementation of core RDF and DAML-OIL semantics (through Horn rules) and application semantics (through further Horn rules), TRIPLE becomes, as they say a 'novel query and transformation language'. Horn rules are allowed to reference data from multiple models. In as much as TRIPLE could be seen as mandating a single approach to implementing RDF(S) and OWL semantics (Horn rules), this must be seen as a weakness. The ongoing work on query languages for the Semantic Web indicates that other developers are more confortable with a specification that does not presuppose a Horn implementation, but permits different developers to implement in different ways. Named graphs can be seen as taking one aspect of this language, noting that it is particularly useful, and not addressed by the Semantic Web recommendations, and pursuing that.

4.2 Contexts in RDF

Guha et al. [23] provide contexts, aggregate contexts, lifting rules, selective importing, preference rules etc. They modify the RDF model theory to have additional context parameters both in the abstract syntax being interpreted and in the universe of interpretation. They interpret sets of graphs, rather than an individual graph. Unfortunately this step is sufficiently large to require significant new effort for implementors of RDF and OWL inference systems.

Their approach shares with ours the style of expressing some of the richer semantic constraints as extensions which constrain interpretations of certain new vocabulary (for them e.g. `importsFrom`, for us e.g. `signature`).

A significant difference is the approach to aggregation. For Guha et al. certain contexts are aggregate contexts, which use lifting rules, possibly simple imports, pos-

sibly complex non-monotonic rules, to combine RDF data from multiple sources. They have some universal restrictions built into the model theory, for example, lifting rules must be defined within their target aggregate context. For us, aggregation is only ever a monotonic merging operation, but the choice of what to aggregate is seen as a pragmatic, application level decision.

We find their approach to be overly complex. Unlike Guha et al. we do not propose complex logic for contexts, merely the minimum step needed to record knowledge about provenance and other aspects of graphs needed for applications which need to address problems of trust. Using knowledge recorded with named graphs, applications will be able to use heuristics appropriate to them, to select the graphs they wish to trust for specific purposes.

The simple approach that we take, permits substantially quicker deployment of applications that need to take provenance information into account, uses the flexibility and expressiveness of RDF, and is, we believe, fully adequate for Semantic Web applications in the near future. Web technology, designed to be deployed on a world wide scale, needs to put a high value on simplicity, and on incremental steps. This ensures enough development effort can be made, in a number of systems, with different implementation strategies, to support the widespread deployment needed for the Web. The first steps of the Semantic Web are completed: systems implementing the RDF and OWL recommendations are deployed. Knowledge is published on the Semantic Web in these formats. To be effective, proposals for new Semantic Web features must build on these foundations, and must be parsimonious in the additional implementation effort required. A key feature of named graphs, lacking in TRIPLE or Guha's contexts work, is parsimony.

4.3 RDF Dataset

The concept of named graphs has been adopted by the W3C Data Access Working Group with a slight modification as the basis for the SPARQL query language [27]. Recent drafts of the SPARQL query language specification define RDF datasets as:

> An RDF dataset is a set $= \{G, (u_1, G_1), (u_2, G_2), ...(u_n, G_n)\}$ where G and each G_i are graphs, and each u_i is a URI. Each u_i is distinct.
>
> G is a called the default graph. (u_i, G_i) are named graphs.

The main innovation over our work is the addition of one unnamed default graph. This provides backward compatibility with RDF without named graphs, and allows the named graphs functionality of SPARQL to be optional.

The addition of the default graph to a collection of named graphs may have the side effect of reintroducing some of the difficulties that named graphs address. For example, merging both default graphs and named graphs from different reposito-

ries, while maintaining provenance information, may prove difficult. The problem of distinguishing between documents and document content is reintroduced when serializations of RDF datasets are published on the Web using syntaxes like TriX or TriG which serialize multiple graphs into a single document (see section 5). What does the document URL refer to if such a document contains an unnamed default graph? The document, the RDF dataset or the default graph? As this question is unanswered, we recommend naming all graphs before publishing them on the Web.

4.4 Concise Bounded Descriptions and Minimum Self-contained Graphs

Named graphs allow a large monolithic knowledge base, consisting of a single RDF graph, to be divided up into a collection of smaller graphs, each individually named. The named graphs framework provides very little guidance as to how to make this division: the only constraint is that blank nodes cannot be shared between different named graphs, so that all triples involving the same blank node must occur in the same named graph.

Two other approaches to doing such divisions are Stickler's *Concise Bounded Descriptions* (CBD) [28], and Tummarello et al.'s *Minimum Self-contained Graph* (MSG) [29]. Both of these use an algorithmic approach to determining a subgraph. A concise bounded description of a URIref u node from a graph G, is the smallest subgraph containing all the triples from G whose subject is either the given URIref or a bnode used as an object by some other triple in the CBD (there is a further rule concerning reification, which we omit here). We will write this as $c_G(u) \subset G$. An MSG is a minimal subgraph of G containing every triple in G whose subject or object is a bnode occurring in the MSG. We will write $m_G(t) \subset G$ for the MSG of G containing a triple $t \in G$.

Given a collection of named graphs N, and considering its merge $M(N)$, we see that any MSG $m \subset M(N)$ is necessarily a subgraph of some specific graph $g \in N$, i.e. an MSG from a collection of named graphs is an MSG of one of the graphs in the collection. More precisely, $m_{M(N)}(t) = m_g(t)$ for $g \in N$ such that $t \in g$. Each graph is the disjoint union of the MSGs within it, making MSGs a useful syntactic building block. This is reflected in the above decomposition of a collection of named graphs into MSGs. In contrast, the CBD $c_{M(N)}(u)$ of a URIref u in $M(N)$ may have components from different graphs in the collection of named graphs. i.e. the URIref u may have descriptions in many of the graphs in the collection of named graphs, and the CBD will draw from all of the them. More formally, $c_{M(N)}(u) = \cup_{g \in N} c_g(u)$. Unlike with MSGs, a graph cannot be decomposed into CBDs. The CBDs of two different resources may overlap, and there may be parts of the graph that are not in any CBD. This reflects CBDs being motivated by application needs rather than from purely syntactic considerations.

Compared with MSGs named graphs are a more useful way of breaking up a large graph, because the MSGs are often too small to be useful. For example, every ground triple (with no blank nodes) is an MSG. Information such as provenance information or access control is likely to be applied to subgraphs that are bigger than a single MSG.

In comparing named graphs with concise bounded descriptions it is significant that they are intended for different purposes. Named graphs allow collections of triples to be published as independent units, and to retain the integrity of the publication unit. This permits metadata to be added about the publication unit, such as metadata about the publication process etc. Concise bounded descriptions are intended primarily as a mechanism to provide a fragment of knowledge about a particular resource, typically in a response from a server to a client that asked about that resource. Hence concise bounded descriptions divide a knowledge base up into resource-centric subgraphs; whereas named graphs divide a knowledge base up in a publication oriented way. The two approaches are complementary and could be used together.

5 Concrete Syntaxes

A concrete syntax for named graphs has to exhibit the name, the graph and the association between them. We offer three concrete syntaxes: TriX and RDF/XML both based on XML; and TriG as a compact plain text format.

5.1 RDF/XML

Named graphs are backward compatible with RDF. A collection of RDF/XML [1] documents on the Web map naturally into the abstract syntax, by using the first `xml:base` declaration in the document or the URL from which an RDF/XML file is retrieved as a name for the graph given by the RDF/XML file. Using RDF/XML has disadvantages:

- The set of named graphs is in many documents rather than one.
- The known constraints and limitations of RDF/XML apply [16,30]. For instance, it is not possible to serialize graphs with certain predicate URIs, nor is it possible to use literals as subjects.
- The URI at which an RDF/XML document is published is used for three different purposes: as a retrieval address, with an operational semantics; as a means of identifying the document; and as a means of identifying the graph described by the document. There is potential for confusion between these three uses.
- Because it has reached recommendation status and is in use, it is hard to change.

None of these disadvantages is present in the TriX and TriG syntaxes described below. In balance, the major advantage of using RDF/XML is the deployed base, and current technology.

5.2 TriX

The RDF/XML syntax has been designed as a compromise of requirements from different communities. While encoding the RDF abstract syntax [4], RDF/XML tries to hide the underlying triple structure in order to make it easier for humans to read the code and to get into RDF applications such as OWL with only partial understanding of their representation in RDF. Thus RDF/XML provides a lot of different serialization options that neither complete hide the underlying RDF, nor does it make them clear. Allowing this variety of serialization options implies the drawback that it is impossible to describe RDF/XML with a DTD or XML Schema, forbids generic XML tools such as XPath [31], XSLT [32] and XQuery [33] to be used and complicates parsing the syntax [16,30].

TriX [16] addresses these shortcomings by having a basic syntax corresponding closely to the RDF abstract syntax, with hiding achieved by using alternative syntactic forms that are transformed into the basic syntax. In addition, TriX provides for naming graphs and for serializing several graphs in a single document.

The core of TriX is the `triple` element, which contains three children, the subject, predicate and object of the triple. Each of these children is either a `uri` element, an `id` element, a `plainLiteral` or a `typedLiteral` element depending on whether the corresponding node in the graph is an RDF URI reference, a blank node or a literal (plain or typed). The element content contains the label of the node (or the blank node identifier). Whitespace normalization is applied to `uri` and `id` element content. We strongly prefer the use of absolute URI references in `uri`. This ensures that XML based tools can easily compare two `uri` nodes for equality. Relative URIs, if used, are resolved against the base URL used to retrieve the document (as in RDF/XML without `xml:base`).

`plainLiteral` elements can be modified by an `xml:lang` attribute. `xml:lang` is prohibited elsewhere in the document (for example, it is not permitted on the root element). This avoids any confusion as to whether it applies to typed literals. It does not. `typedLiteral` elements require a `datatype` attribute. As in RDF/XML, no whitespace processing is performed. We note it is difficult to write the legal lexical forms for `rdf:XMLLiteral` which have to be exclusive canonical XML [34], which is escaped either with a `CDATA` block, or using XML character escaping conventions.

A `graph` element starts with an optional `uri` child element which names the graph, and then has any number of `triple` elements as children. The root element of

the document is a `trix` element, which has zero or more `graph` elements as its children.

The example below shows a TriX document containing two named graphs; the second describes the first.

```
<trix xmlns="http://www.w3.org/2004/03/trix/trix-1/">
 <graph>
  <uri>http://example.org/graph4</uri>
  <triple>
   <uri>http://example.org/aBook</uri>
   <uri>http://purl.org/dc/elements/1.1/title</uri>
   <typedLiteral datatype=
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral">
     &lt;ex:title xmlns:ex="http://example.org/">
      A Good Book
     &lt;/ex:title>
   </typedLiteral>
  </triple>
  <triple>
   <uri>http://example.org/aBook</uri>
   <uri>http://www.w3.org/2000/01/rdf-schema#comment</uri>
   <plainLiteral
    xml:lang="en">This is a really good book!</plainLiteral>
  </triple>
 </graph>
 <graph>
  <uri>http://example.org/graph5</uri>
  <triple>
   <uri>http://example.org/graph4</uri>
   <uri>http://example.org/source</uri>
   <uri>http://example.org/book-description.rdf</uri>
  </triple>
 </graph>
</trix>
```

TriX is described by the following DTD:

```
<!-- TriX: RDF Triples in XML -->
<!ELEMENT trix         (graph*)>
<!ATTLIST trix         xmlns CDATA #FIXED
                   "http://www.w3.org/2004/03/trix/trix-1/">
<!ELEMENT graph        (uri?, triple*)>
<!ELEMENT triple       ((id|uri|plainLiteral|typedLiteral),
                       uri,
                       (id|uri|plainLiteral|typedLiteral))>
<!ELEMENT id           (#PCDATA)>
```

```
<!ELEMENT uri          (#PCDATA)>
<!ELEMENT plainLiteral (#PCDATA)>
<!ATTLIST plainLiteral xml:lang CDATA #IMPLIED>
<!ELEMENT typedLiteral (#PCDATA)>
<!ATTLIST typedLiteral datatype CDATA #REQUIRED>
```

The requirements related to ease of writing and reading XML syntax for RDF tend, in general, to conflict with the core requirements of giving a transparent representation of the graph in a way that can easily be processed with XML tools. Therefore TriX defines a general purpose interoperable extensibility mechanism. Each community can then define and use whatever syntactic extensions they wish, declaring the extensions they are using at the top of the data files. As long as the extensions are described in a standard way and are identified with URLs, any processor can apply them. We use XSLT [32] as the syntactic extensibility mechanism, and the stylesheet processing instruction [35] as the declaration. Examples of how the extensibility mechanism can be used to extend the TriX basic syntax with `xml:base`, `rdf:XMLLiteral`, collections, and typed literals are found in [16].

5.3 TriG

In this paper we use TriG as a compact and readable alternative to TriX. TriG is a variation of Turtle [36], which in turn is based on N3 [10]. TriG extends Turtle by using '{' and '}' to group triples into multiple graphs, and to precede each by the name of that graph. The following TriG document contains two graphs. The first graph contains information about itself. The second graph refers to the first one (namespace prefix definitions omitted).

```
:G1 { _:Monica ex:name "Monica Murphy" .
      _:Monica ex:email <mailto:monica@murphy.org> .
      :G1 pr:disallowedUsage pr:Marketing }

:G2 { :G1 ex:author :Chris .
      :G1 ex:date "2003-09-03"^^xsd:date }
```

TriG provides syntactic compatibility with N3 [10] by allowing additionally '=' and '.' to write the same pair of graphs thus:

```
:G1 = { _:Monica ex:name "Monica Murphy" .
        _:Monica ex:email <mailto:monica@murphy.org> .
        :G1 pr:disallowedUsage pr:Marketing } .

:G2 = { :G1 ex:author :Chris .
        :G1 ex:date "2003-09-03"^^xsd:date } .
```

This is N3, with the '=' being read as `owl:sameAs`. However, unlike in N3, the following features are not permitted:

- Blank nodes cannot be shared between graphs.
- Each graph should be named with a URIref.
- A formula for a graph cannot be embedded within another graph as a node (although its name can be used in this way).

These differences mean that basic syntactic operations such as comparing two N3 documents for differences in their abstract syntax, or signing the abstract form of an N3 document, differ significantly, and are substantially more difficult, than the corresponding operations on RDF graphs. These restrictions permit named graphs to use RDF graph comparison [21] and RDF graph signatures [8].

A further difference is that when read as N3, each named graph is the object of an `owl:sameAs` triple in an outer graph, the graph of the document. Thus, N3 commits to having graphs within graphs, and the additional complexity that that entails. This complexity is not motivated by applications, but seems to be an unnecessary elegance, making the underlying mathematical framework significantly more difficult, and less well understood. Reading the document as a TriG document, reads it as two separate named graphs, and does not include the extra layer of an outer graph.

6 Query Languages

There are two query languages for named graphs: TriQL [37] and RDFQ [38]. The W3C is developing a new RDF query language SPARQL [27], which will also allow querying across multiple named graphs. RDFQ and TriQL predate SPARQL and we expect that SPARQL will supersede both languages once it has become a final W3C recommendation.

The GRAPH keyword in SPARQL allows access to the URIs naming the graphs in an RDF dataset, or restricts a graph pattern to be applied to a specific named graph.

The following SPARQL query identifies people having email addresses, selecting and extracting the person identifier and email address value pairs; furthermore, the query is restricted to statements occurring in graphs having Chris as author:

```
PREFIX ex: <http://example.org/>

SELECT ?person ?email
WHERE
  {
  GRAPH ?graph
```

```
   { ?person ex:email ?email }
GRAPH ?anyGraph
   { ?graph ex:author ex:Chris }
}
```

The variable `?graph` is bound to the names of all graphs that contain information about email addresses. The second pattern restricts `?graph` to graphs authored by Chris. The information that Chris authored a graph may occur in any named graph.

7 Implementations

Because named graphs are only a small addition on top of the Semantic Web recommendations it is easy to implement them extending existing Semantic Web tools.

7.1 NG4J

One of these extensions is NG4J [39] which builds on the Jena Semantic Web toolkit [40]. NG4J provides developers with an API for manipulating sets of named graphs and an API for signing named graphs using the Semantic Web Publishing Vocabulary described in section 8. NG4J implements the TriX and the TriG syntax and the TriQL query language. A set of named graphs can also be viewed and manipulated as a provenance-enabled Jena model, allowing applications to track the origin of statements. By retrofitting Jena with an extended abstract syntax while staying compartible with the existing Jena API, NG4J aims at providing an migration path for existing applications based on Jena.

7.2 Jena MultiModel

One application which uses named graphs is a faceted browser [41], `http://www.swed.org.uk/`. This harvests RDF graphs from potentially many sites, and stores them in a `MultiModel` object which embodies the named graph abstraction on top of Jena's `Model` class [40], which implements the RDF abstract syntax [4]. The source of any triple can be used during the faceted browse for visual styling of that part of the data. The end-user can apply varying levels of trust to different information presented on a single page. The style indicates the different authors, who can be treated with varying levels of caution.

8 Semantic Web Publishing

One application area for named graphs is publishing information on the Semantic Web. This scenario implies two basic roles embodied by humans or their agents: Information providers and information consumers. Information providers publish information together with meta-information about its intended assertional status. Additionally, they might publish background information about themselves, e.g. their role in the application area. They may also decide to digitally sign the published information. Information providers have different levels of knowledge, and different intentions and different views of the world. Thus seen from the perspective of an information consumer, published graphs are claims by the information providers, rather than facts. An information consumer may accept some of these claims and reject others. We represent these choices using the concept of the information consumer *accepting* named graphs (as discussed formally in section 3.2).

Different tasks require different levels of trust. Thus information consumers will use different trust policies to decide which graphs should be accepted and used within the specific application. These trust policies depend on the application area, the subjective preferences and past experiences of the information consumer and the trust relevant information available. A naive information consumer might, for example, decide to trust all graphs which have been explicitly asserted. This trust policy will achieve a high recall rate but is easily undermineable by information providers publishing false information. A more cautious consumer might require graphs to be signed and the signers to be known through a Web-of-Trust mechanism. This policy is harder to undermine, but also likely to exclude relevant information, published by unknown information providers.

8.1 Authorities, Authorization and Warrants

Information providers using RDF do not have any explicit way to express any intention concerning the truth-value of the information described in a graph; RDF does not provide for the expression of *propositional attitudes*, such as asserting, denying, commenting on, or otherwise expressing an opinion about the content of a graph. Information consumers may require this, however. Note that this is in addition to trust policies, and may be required in order to put such policies into operation. For example, a simple policy could be: believe anything asserted by a trusted source. In order to apply this, it is necessary to have a clear record of what is *asserted* by the source. Not all information provided by a source need be asserted by that source. We propose here a vocabulary and a set of concepts designed to enable the uniform expression of such propositional attitudes using named graphs.

We take three basic ideas as primitive: that of an *authority*, a relationship of *autho-*

rizing, and a *warrant*. An authority is a 'legal person'; that is, any legal or social entity which can perform acts and undertake obligations. Examples include adult humans, corporations and governments. The 'authorizing' relationship holds between an authority or authorities and a named graph, and means that the authority in some sense commits itself to the content expressed in the graph. Whether or not this relationship in fact holds may depend on many factors and may be detected in several ways (such as the named graph being published or digitally signed by the authority). Finally, a warrant is a resource which records a particular propositional stance or intention of an authority towards a graph. A warrant asserts (or denies or quotes) a named graph and is authorized by an authority. One can think of warrants as a way of reducing the multitude of possible relationships between authorities and graphs to a single one of authorization, and also as a way of separating questions of propositional attitude from issues of checking and recording authorizations. The separation of authority from intention also allows a single warrant to refer to several graphs, and for a warrant to record other properties such as publication or expiry date.

To describe the two aspects of a warrant we require vocabulary items: a property `swp:authority` (where `swp:` is a namespace for Semantic Web publishing) relating warrants to authorities, and another to describe the attitude of the authority to the graph being represented by the warrant. We will consider two such intentions expressed by the properties `swp:assertedBy` and `swp:quotedBy`. These take a named graph as a subject and a `swp:Warrant` as object; `swp:authority` takes a warrant as a subject and a `swp:Authority` as an object. Each warrant must have a unique authority, so `swp:authority` is an OWL functional property. Intuitively, `swp:assertedBy` means that the warrant records an endorsement or assertion that the graph is true, while `swp:quotedBy` means that the graph is being presented without any comment being made on its truth. This latter is particularly useful when republishing graphs as part of a syndication process, the original publisher may assert a news article, but the syndicator, acting as a common carrier, merely provides the graph as they found it, without making any commitment as to its truth. Warrants may also be signed, and the property `swp:signatureMethod` can be used to identify the signature technique.

8.2 Warrant Descriptions as Performatives

A warrant, as described above, is a social act. However, it is often useful to embody social acts with some record; for example, a contract (which is a social act) may be embodied in a document, which is identified with that act, and is often signed. In this section, we introduce the notion of a *warrant graph*, which is a named graph describing a warrant, that is identified with the social act. Thus, this is a resource which is both a `swp:Warrant` and an `rdfg:Graph`. Consider a graph containing a description of a warrant of another named graph, such as:

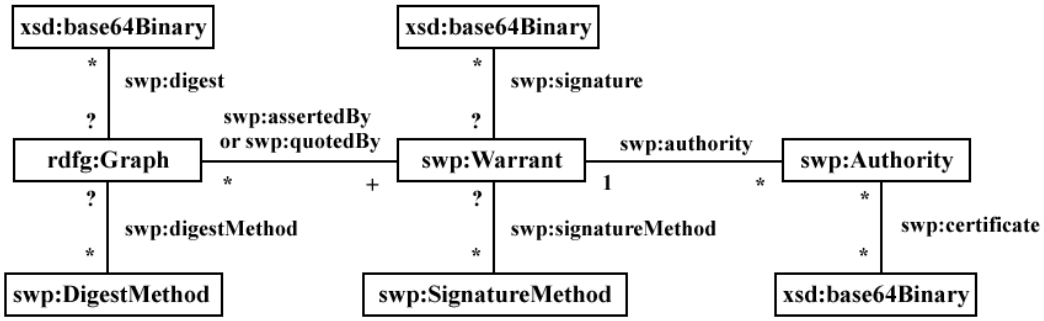

Fig. 1. The Semantic Web Publishing Vocabulary

```
{ :G2 swp:assertedBy _:w .
  _:w rdf:type swp:Warrant [2] .
  _:w swp:authority _:a .
  _:a rdf:type swp:Authority .
  _:a foaf:mbox <mailto:chris@bizer.de> }
```

The graph is true when there is a genuine warrant; but so far we have no way to know whether this is in fact the case. A slight modification identifies the graph with the warrant itself:

```
:G1 { :G2 swp:assertedBy :G1 .
      :G1 swp:authority _:a .
      _:a foaf:mbox <mailto:chris@bizer.de> }
```

Suppose further that such a *warrant graph* is in fact authorized by the authority it describes - in this case, by Chris, the owner of the mailbox: this might be established by, for example, being published on Chris' website, or by being digitally signed by him, or in some other way, but all that we require here is that it is in fact true. Under these circumstances, the warrant graph has the intuitive force of a first-person statement to the effect "I assert `:G2`" made by Chris.

In natural language, the utterance of such a self-describing act is called a *performative*; that is, an act which is performed by saying that one is doing it. Other examples of performatives include promising, naming and, in some cultures, marrying [42]. The key point about performatives are that while they are descriptions of themselves, they are not only descriptions: rather, the act of uttering the performative is understood to be the act that it describes. Our central proposal for how to express propositional attitudes on the Web is to treat a warrant graph as a record of a performative act, in just this way.[3] With this convention, Chris can assert

[2] The type triples are implied by domain and range constraints and can be omitted.

[3] The Bank of England uses this technique, by having each twenty pound note bear the text: "I promise to pay the bearer on demand the sum of twenty pounds."

the graph :G2 by authorizing the warrant graph shown above, for by doing so he creates a warrant: the warrant graph becomes the (self-describing) warrant of the assertion of :G2 by Chris. In order for others to detect and confirm the truth of this warrant requires some way to check or confirm the relationship of authorization, of course: but the qualification of the warrant graph as a warrant depends only on the relationship holding.

A graph describing a warrant is not required to be self-describing in order to be true (it may be true by virtue of some other warrant) and a warrant graph may not in fact be a performative warrant (if it is not authorized by the authority it claims). In the latter case the graph must be false, so self-describing warrant graphs whose authorization cannot be checked should be treated with caution. The warrant graph may itself be the graph asserted. Any named graph which has a warrant graph as a subgraph and is appropriately authorized satisfies the conditions for being a performative warrant of itself. For example:

```
:G2 { :Monica ex:name "Monica Murphy" .
      :G2 swp:assertedBy :G2 .
      :G2 swp:authority _:a .
      _:a foaf:mbox
      <mailto:patrick.stickler@nokia.com> . }
```

when authorized by Patrick Stickler, becomes a performative warrant for its own assertion, as well as being warranted by the earlier example. As this example indicates, a named graph may have a number of independent warrants.

8.3 Publishing with Signatures

Information providers may decide to digitally sign graphs, when they wish to allow information consumers to have greater confidence in the information published. For instance, if Patrick has an X.509 certificate [43], he can sign two graphs in this way:

```
:G1 { :Monica ex:name "Monica Murphy" .
      :G1  swp:assertedBy _:w1 .
      _:w1 swp:authority _:a .
      _:a  foaf:mbox
         <mailto:chris@bizer.de> }
:G2 { :G1 swp:quotedBy :G2 .
      :G1 swp:digestMethod
          swp:JjcRdfC14N-sha1 .
      :G1 swp:digest
          "..."^^xsd:base64Binary .
      :G2 swp:assertedBy :G2 .
      :G2 swp:signatureMethod
```

```
    swp:JjcRdfC14N-rsa-sha1 .
:G2 swp:signature
    "..."^^xsd:base64Binary .
:G2 swp:authority _:s .
_:s foaf:mbox
<mailto:patrick.stickler@nokia.com> .
_:s swp:certificate
    "..."^^xsd:base64Binary  }
```

Note that `:G2` is a warrant graph. The `swp:signature` gives a binary signature of the graph related to the warrant[4]. The canonicalization and signature algorithms which have been used to calculate the signature are indicated by the value of the `swp:signatureMethod` property. SWP uses a similar mechanism as XML-Signature [44] for signing several resources using a single signature: Including the graph digest of `:G1` into `:G2` and signing `:G2` afterwards also assures the integrity of `:G1`.

The information publisher indicates the methods used for forming digests and signatures. We require the methods to be identified by URIs, which can be dereferenced on the Web to retrieve a document, describing the method in detail. The signature method `swp:JjcRdfC14N-rsa-sha1`, for example, specifies a variation of the graph canonicalization algorithms provided in [8], and chooses one of the digest/signature algorithm combinations defined by XML-Signature [44]. Rather than make a set of decisions about digest and signature methods, SWP provides terms for describing the chosen combination.

Using signatures does not modify the theoretical semantics of assertion, which is boolean; but it will modify the operational semantics, in that without signatures, any assertions made, will only be acted on by the more trusting Semantic Web information consumers, who do not need verifiable information concerning who is making them.

The formal semantics of the Semantic Web publishing vocabulary are described in section 9.

8.4 The Information Consumer

The information consumer needs to decide which graphs to accept. This decision may depend on information concerning who said what, and whether it is possible to verify such information. It may also depend on the content of what has been said. We consider the use case in which an information consumer has read a set of named

[4] It is necessary to exclude the last `swp:signature` triple, from the graph before signing it: this step needs to be included in the method.

graphs off the Web. In terms of the semantics of named graphs (section 3.2), the information consumer needs to determine the set A. Information about the graphs may be embedded within the set of named graphs, hence most plausible trust policies require that we are able to provisionally understand the named graphs in order to determine, from their content, whether or not we wish to accept them. This is similar to reading a book, and believing it either because it says things you already believe, or because the author is someone you believe to be an authority: either of these steps require reading at least some of the book.

The trust policy an information consumer chooses for determining his set of accepted graphs depends on the application area, his subjective preferences and past experiences and the trust relevant information available. Trust policies can be based on the following types of information [45]:

First-hand information published by the actual information provider together with a graph, e.g. information about the intended assertional status of the graph or about the role of the information provider in the application domain. Example policies using the information provider's role are: "Prefer product descriptions published by the manufacturer over descriptions published by a vendor" or "Distrust everything a vendor says about its competitor."

Information published by third parties about the graph (e.g. further assertions) or about the information provider (e.g. ratings about his trustworthiness within a specific application domain). Most trust architectures proposed for the Semantic Web fall into this category [46–48]. These approaches assume explicit and domain-specific trust ratings. Providing such ratings and keeping them up-to-date puts an unrealistically heavy burden on information consumers in many application domains.

The content of a graph together with rules, axioms and related content from graphs published by other information providers. Example policies following this approach are "Believe information which has been stated by at least 5 independent sources." or "Distrust product prices that are more than 50% below the average price."

Information created in the information gathering process like the retrieval date and retrieval URL of a graph or whether a signed warrant attached to a graph is verifiable or not.

As illustrative examples we show how two different trust policies can be implemented. The first policy relies on a SPARQL query, the second on a custom algorithm. Further example trust policies are found in [45,49].

Let's assume that Patrick has a collection of several graphs from the Web plus one graph authored by himself containing a list of the people he knows. He could now decide to filter the information from the Web using the policy to trust only people he knows. This trust policy is implemented by the following SPARQL query:

```
CONSTRUCT { ?sub ?pred ?obj }
```

```
WHERE
   {
    GRAPH ?w1
      { ?sub ?pred ?obj .
        ?w1 swp:assertedBy ?w1 .
        ?w1 swp:authority ?patricksFriends . }
    GRAPH <http://example.org/patricksTrustedGraph>
      {  _:s foaf:knows ?patricksFriends .
         _:s foaf:mbox <mailto:stickler@nokia.com> . }
   }
```

As a second example we sketch an algorithm that allows the agent to implement a trust policy of trusting any RDF that is explicitly asserted. More cautious variation may require performative assertions or digital signatures.

The agent has an RDF knowledge base, K, which may or may not be initially populated. The agent is presented with a set of named graphs **N**, and augments the knowledge base with some of those graphs (determining the set A of accepted graphs).

(1) Set $A := \phi$
(2) Non-deterministically choose $n \in \text{domain}(N) - A$, if no choices remain terminate.
(3) Set $K' := K \cup N(n)$, provisionally assuming $N(n)$.
(4) If K' entails:
`n swp:assertedBy w .`
then set $K := K'$ and $A := A \cup \{n\}$, otherwise backtrack to 2.
(5) Repeat from 2.

If initially K is empty, then the first graph added to K will be one that includes its own assertion, by an arbitrary warrant and authority. All such graphs will be added to K, as will any that are asserted as a consequence of the resulting K. The algorithm is equivalent to one that seeks to accept a graph by finding a statement of its assertion either within itself, or within some other accepted graph, or the initial knowledge base. The algorithm is sound with respect to the goal of only adding graphs that are explicitly asserted, as verified by step 4. However, it is incomplete against the same criterion, since two graphs each of which explicitly assert the other, would satisfy the criterion if both were accepted, but the algorithm misses that. We see the self-asserting performative warrant as the basic communicative act, and more sophisticated phrasings (such as the mutually asserting graphs), are less likely to be understood.

At step 4, w is unconstrained, reflecting the simple policy of trusting everybody. A slightly more sophisticated query could implement a policy that, for example, only trusted a set of named individuals, or require that any self-asserting graph actually be a warrant graph.

This algorithm does not take consistency into account. As we merge internally consistent graphs in step 3 we may introduce inconsistencies that occur between the graphs. In some cases, it may not be possible to even detect this, for example, in OWL Full which has an undecidable theory. For a semantics with a complete and terminating consistency checker [50] (such as for OWL Lite), inconsistency could be detected immediately. We do not propose any particular response to inconsistency. Some applications, such as the faceted browser of [41], may not care, whereas others, may wish to use inconsistency to reject some of the graphs, in favour of a maximal consistent subset. Mechanisms such as those used in truth maintenance systems would be useful for these applications.

8.4.1 Using a Public Key Infrastructure

The policies described above would believe fraudulent claims of assertion. That is, any of the named graphs may suggest that anyone asserted any of the graphs, whether or not that is true, and the above policies has no means of detecting that.

We have described how a publisher can sign their graphs and include such signatures in the published graphs. We will continue to explore the X.509 certified case; in general the PGP [51] case is similar, and the approach taken does not assume a particular PKI.

Graph signatures can be checked by modifying the query in step 4 of the second policy to be:

```
SELECT  ?certificate ?method ?sign
WHERE
   {
    GRAPH ?w1
      { ?w1 swp:assertedBy ?w1 .
        ?w1 swp:authority ?s .
        ?w1 swp:signatureMethod ?method .
        ?w1 swp:signature ?sign }
    GRAPH ?anyGraph
      { ?s swp:certificate ?certificate }
   }
```

where this is understood as being over the interpretation of the graph, rather than as a syntactic query over the graph. The signatures must be verified following the given method. If this verification fails then the graph is false and can be rejected. If the verification succeeds then the certification chain should be considered by the information consumer. If the agent trusts anyone in the certificate chain [5], then the graph is accepted, otherwise not. More sophisticated algorithms would consider

[5] For PGP, the specific method of determining whether the certificate is trusted is different.

whether the person asserting the graph, who has now been verified, is in the group of persons which the information consumer trusts on the topic the graph discusses.

A graph may have more than one warrant. If any warrant contains an incorrect signature, then the warrant is simply wrong, and indicates data or algorithmic corruption. A graph containing such a warrant (but not necessarily the named graph misasserted) should be rejected. The choice of which warrant to check is nondeterminismic and hence should consider any valid warrant whose certification chain is trusted.

9 Formal Semantics of Publishing and Signing

This section provides an extension of RDF semantics [3] which: allows persons to be members of the domain of discourse; allows interpretations to be constrained by the identifying information in a digital certificate; allows the `swp:assertedBy` triple to have a *performative* semantics; and makes `swp:signature` triples true or false depending on whether the signature is valid or not. Together these extensions underpin the publishing framework of the previous section.

9.1 Persons in the Domain of Discourse

The two frameworks of digital signatures we have considered both tie a certificate to a legal person (e.g. a human or a company), or similar. In X.509, a certificate includes a distinguished name [52,53], which is chosen to adequately identify a legal person, and is verified as accurate by the certification authority. In PGP, a certificate contains unspecified identifying information, "such as his or her name, user ID, photograph, and so on" [51]; this is usually an e-mail address.

The class extension of `swp:Authority` is constrained to be a set P of legal persons and software agents acting on behalf of legal persons. Thus, our formal semantics requires the universe of discourse to contain such persons as resources. Such a requirement goes beyond the usual 'logical' bounds of model-theoretic semantics. We expect that Web languages will further extend their semantics into the real world of agents, acts and things as they become applied in real-world applications. This first step is, in itself, not very interesting since we have not constrained which person in the real world corresponds to which URIref or blank node in the graph.

The second step, is to constrain the property extension of `swp:certificate` to $\{(p, c)|p \in P, c$ a sequence of binary octets, with c being an X.509 or PGP certificate for $p\}$. The binary octets are represented in a graph using `xsd:base64Binary`. The interpretation of these sequences as X.509 is specified in [43], which gives a

distinguished name from RFC 2253 [53], which identifies a person. If c gives a PGP certificate then given the potential vagueness of the identifying information we allow all pairs of in which the person matches the identifying information. For example, if the identifying information is only a GIF image, then all people who look like that image are paired with the certificate. [6]

This definition does *not* depend on whether or not the certificate is trusted. If the graph containing the `swp:certificate` triple is accepted, using mechanisms such as those discussed in section 8.4, then the triple's meaning is as above. The certificate chain in the certificate is only checked when deciding which graphs to accept.

9.2 Performative warrants

A formal model-theoretic semantics specifies truth conditions. To fully capture the meaning of a performative, however, we need to go beyond truth-conditions, since the very same form of words may be true whoever uses them, but will only count as a performative if used by the authority it mentions. For example "Patrick promises..." uttered by Patrick is a promise - a performative act - but uttered by Chris is merely a description of the act; yet it may well still be true, and for the same reasons. We will deal with this by considering a warrant graph to be a 'genuine' warrant just when it describes its authority accurately, and to be true in any interpretation under which a genuine warrant actually exists.

The relationship of authorizing, and sets of authorities and warrants, are taken as primitive, and we will identify them respectively with the property extension of `swp:authority` and the class extensions of `swp:Authority` and `swp:Warrant`. All the remaining semantic conditions are defined in terms of these, so their correctness in any application depends on that of the interpretation of `swp:authority` together with its range and domain. Thus a triple

```
ex:a swp:authority ex:b .
```

is true in I just when $I($`ex:a`$)$ is a warrant which is authorized by $I($`ex:b`$)$.

The performative role of a properly authorized warrant graph can then be described by simply declaring that any named graph *ng* containing a triple

name$($*ng*$)$ `swp:authority` *bbb* .

is a warrant. Then any interpretation I of *rdfgraph*$($*ng*$)$ (conforming to the naming of *ng*) under which *ng* is authorized by $I($*bbb*$)$ makes this triple true, and hence

[6] This shows why it is unwise to only provide an image in your PGP certificate.

requires *ng* to be in *ICEXT*(I(`swp:Warrant`)): call this an *authorizing interpretation* of the named graph. Fixing the referent of the object of such a triple to be an authorizing authority thus means that the graph can be satisfied only by authorizing interpretations under which the named graph is a warrant.

The self-realizing quality of performatives is extended to the triples which express propositional attitudes by making these trivially self-fulfilling when they occur under the right conditions, in an authorized warrant graph. For example if *ng* is a warrant graph which contains a triple

```
aaa swp:assertedBy bbb .
```

where $I(bbb) = ng$, then if I is an authorizing interpretation of *ng*, then I must satisfy that triple; similarly for `swp:quotedBy` and indeed for any other property expressing a propositional attitude of an authority towards a graph.

Note that this does not imply that a named graph is *true* in an authorizing interpretation of a warrant which asserts it. The fact of an authority asserting a graph can be true independently of the actual truth of the graph. However, the attitude expressed can be utilized by trust policies. It may be appropriate to treat graphs asserted by trusted authorities as being true, but not to extend this to graphs quoted by trusted authorities. One could express this trust policy by a semantic rule to the effect that if I satisfies

```
aaa swp:assertedBy bbb .
bbb swp:authority ccc .
```

and $I(ccc)$ is trusted, then I satisfies *rdfgraph*(I(*aaa*)).

The algorithm for choosing which graphs to accept, presented in section 8.4, interacts with this performative semantics, by essentially assuming that a graph has been asserted, and then verifying that in that case the performative is true.

Using `rdfs:subPropertyOf` or `owl:equivalentProperty` to introduce aliases of `swp:assertedBy` may be misleading and should be avoided. Information consumers should be suspicious of any graphs that attempt this, except when they are also asserted by the persons using the aliases so introduced.

9.3 Graph Digests and Signatures

The final specialized vocabulary we consider is that for graph digests and signatures. Strictly speaking this is not necessary for Semantic Web publishing, but just as a signed document has greater social force than an unsigned one, a signed `swp:assertedBy` triple is more credible than an unsigned one. Thus, this section

is specifically intended to be used to sign graphs that are either the subject of, or include `swp:assertedBy` triples.

The two properties `swp:digest` and `swp:signature` are treated in a similar fashion: we start with the simpler `swp:digest`.

A pair (g, d) is in the property extension of `swp:digest`, if and only if,

(1) d is a finite sequence of octets.
(2) There is a pair (g, m) in the property extension of `swp:digestMethod`, and m is a URI which can be dereferenced to get a document.
(3) The method described in the document retrieved from m calculates the digest d for the graph $I(g)$.

This means that an `swp:digest` triple is true if and only if the value is the appropriate digest. Hence, if the graph which is the subject of the triple has been tampered with, such tampering is detected by the `swp:digest` triple being false.

Similarly, a pair (w, s) is in the property extension of `swp:signature`, if and only if,

(1) s is a finite sequence of octets.
(2) There is a pair (w, m) in the property extension of `swp:signatureMethod`, and m is a URI which can be dereferenced to get a document.
(3) There is a pair (w, a) in the property extension of `swp:authority` and a pair (a, c) in the property extension of `swp: certificate`, and c is a finite sequence of octets.
(4) There is a pair (g, w) in the property extension of `swp: quotedBy` or `swp:assertedBy`, and $I(g)$ is a named graph.
(5) The method described in the document retrieved from m calculates the signature s for the graph $I(g)$ using c understood as an X.509 or PGP certificate.

This definition does not depend upon verifying the certificate chain for c.

10 Conclusions

Having a clearly defined abstract syntax and formal semantics, named graphs provide greater precision and potential interoperablity than the variety of *ad hoc* RDF extensions currently used. Combined with specific further vocabulary, this will be beneficial in a wide range of application areas and will allow the usage of a common software infrastructure spanning these areas.

The adoption of named graphs as part of SPARQL's RDF dataset is making it likely that an increasing number of RDF stores implement named graphs, allowing Semantic Web applications to use this additional functionally through a standardized

interface.

The ability of self-reference combined with the Semantic Web Publishing vocabulary addresses the problem of differentiating asserted and non-asserted forms of RDF and allows information providers to express different degrees of commitment towards published information.

Linking information to authorities and optionally assuring these links with digital signatures gives information consumers the basis for using different task-specific trust-policies. We have shown how operational trust can depend on what is being said, rather than simply on who said it, and the trust-rating of the author.

Named graphs provide a high-value but small and incremental change to the Semantic Web Recommendations. Thus they should be preferred over more complex, all-embracing approaches to context that are more likely to face substantial barriers to adoption.

Further related work can be found at the named graphs web-site `http://www.w3.org/2004/03/trix/`.

11 Acknowledgements

Thanks to the W3C Semantic Web Interest Group for their interest and comments on this work as it has developed. During the initial development of this work, Christian Bizer has been a visiting researcher at HP Labs in Bristol and Jeremy Carroll has been a visiting researcher at ISTI, CNR in Pisa. Christian Bizer thanks his host Andy Seaborne. His visit was supported by the Leonardo Da Vinci grant no. D/2002/EX-22020. Jeremy Carroll thanks his host Oreste Signore. Pat Hayes' work was supported in part by the US Department of Defence under contract 2507-282-22. Previous publications about this work include the paper at the WWW 2005 conference [20] which this paper extends, a paper at the Trust workshop at the International Semantic Web Conference 2004 [54], a paper on TriX at Extreme Markup 2004 [16], and various HP Labs Technical Reports [55,56]. Thanks to the anonymous reviewers of various earlier papers for their helpful feedback.

References

[1] D. Beckett, RDF/XML Syntax Specification (Revised) - W3C Recommendation, `http://www.w3.org/TR/rdf-syntax-grammar/` (2004).

[2] D. Brickley, R. V. Guha, RDF Vocabulary Description Language 1.0 - W3C Recommendation, `http://www.w3.org/TR/rdf-schema/` (2004).

[3] P. Hayes, RDF Semantics - W3C Recommendation, http://www.w3.org/TR/rdf-mt/ (2004).

[4] G. Klyne, J. J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax - W3C Recommendation, http://www.w3.org/TR/rdf-concepts/ (2004).

[5] Creative Commons Website, http://creativecommons.org/ (2003).

[6] M. Marchiori, The Platform for Privacy Preferences - W3C Recommendation, http://www.w3.org/TR/P3P/ (2002).

[7] Intellidimension, RDF Gateway - Database Fundementals, http://www.intellidimension.com/pages/rdfgateway/dev-guide/db/db.rsp (2003).

[8] J. J. Carroll, Signing RDF Graphs, in: 2nd International Semantic Web Conference, 2003.

[9] A. Ibrahim, Agent Communication Languages (ACL), http://www.engr.uconn.edu/~ibrahim/publications/acl.htm (2000).

[10] T. Berners-Lee, Notation 3, http://www.w3.org/DesignIssues/Notation3 (1998).

[11] R. V. Guha, Contexts: A Formalization and Some Applications, Ph.D. thesis, Stanford (1995).

[12] G. Klyne, Circumstance, provenance and partial knowledge, http://www.ninebynine.org/RDFNotes/UsingContextsWithRDF.html (2002).

[13] M. Dean, G. Schreiber, OWL Web Ontology Language Reference - W3C Recommendation, http://www.w3.org/TR/owl-ref/ (2004).

[14] J. Heflin, Z. Pan, A Model Theoretic Semantics for Ontology Versioning, in: 3rd International Semantic Web Conference, 2004.

[15] C. Catton, D. Shotton, The use of Named Graphs to enable ontology evolution, http://www.bioimage.org/pub/paradigm.htm (2004).

[16] J. J. Carroll, P. Stickler, TriX: RDF Triples in XML, in: Proceedings of Extreme Markup Languages 2004, 2004.

[17] D. Beckett, Redland Notes - Contexts, http://www.redland.opensource.ac.uk/notes/contexts.html (2003).

[18] E. Dumbill, Tracking Provenance of RDF Data, Tech. rep., ISO/IEC (2003).

[19] R. MacGregor, I.-Y. Ko, Representing Contextualized Data using Semantic Web Tools, in: Practical and Scalable Semantic Systems (workshop at 2nd ISWC), 2003.

[20] J. J. Carroll, C. Bizer, P. Hayes, P. Stickler, Named Graphs, Provenance and Trust, in: 14th World Wide Web Conference, 2005.

[21] J. J. Carroll, Matching RDF graphs, in: International Semantic Web Conference, 2002.

[22] P. F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax - W3C Recommendation, http://www.w3.org/TR/owl-semantics/ (2004).

[23] R. Guha, R. McCool, R. Fikes, Contexts for the Semantic Web, in: 3rd International Semantic Web Conference, 2004.

[24] M. Sintek, S. Decker, TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web, in: 1st International Semantic Web Conference, 2002, pp. 364–378.

[25] A. Gangemi, P. Mika, Understanding the semantic web through descriptions and situations, in: Proceedings of DOA/CoopIS/ODBASE 2003 Confederated International Conferences DOA, CoopIS and ODBASE, LNCS, Springer, 2003.

[26] M. K. Smith, C. Welty, D. L. McGuinness, OWL Web Ontology Language Guide - W3C Recommendation, http://www.w3.org/TR/owl-guide/ (2004).

[27] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/ (Feb 2005).

[28] P. Stickler, CBD - Concise Bounded Description, http://www.w3.org/Submission/CBD/ (2005).

[29] G. Tummarello, C. Morbidoni, P. Puliti, F. Piazza, Signing Individual Fragments of an RDF Graph, in: 14th World Wide Web Conference (Poster), 2005.

[30] D. Beckett, Modernising Semantic Web Markup, in: XML Europe, 2004.

[31] J. Clark, S. DeRose, XML Path Language (XPath) Version 1.0 - W3C Recommendation, http://www.w3.org/TR/1999/REC-xpath-19991116 (1999).

[32] J. Clark, XSL Transformations (XSLT) - W3C Recommendation, http://www.w3.org/TR/xslt (1999).

[33] S. Boag, D. Chamberlin, M.F.Fernández, D. Florescu, J. Robie, J. Siméon, XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/ (2005).

[34] J. Boyer, D. E. Eastlake 3rd, J. Reagle, Exclusive XML Canonicalization Version 1.0 - W3C Recommendation, http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/ (2002).

[35] J. Clark, Associating Style Sheets with XML documents - W3C Recommendation, http://www.w3.org/1999/06/REC-xml-stylesheet-19990629/ (1999).

[36] D. Beckett, Turtle - Terse RDF Triple Language, http://www.ilrt.bris.ac.uk/discovery/2004/01/turtle/ (2004).

[37] C. Bizer, TriQL - A Query Language for Named Graphs, http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQL/ (2004).

[38] P. Stickler, RDFQ, http://sw.nokia.com/rdfq/RDFQ.html (2004).

[39] C. Bizer, R. Cyganiak, R. Watkins, NG4J - Named Graphs API for Jena, in: 2nd European Semantic Web Conference (Poster), 2005.

[40] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: 13th World Wide Web Conference, 2004.

[41] D. Reynolds, P. Shabajee, S. Cayzer, D. Steer, Semantic Portals Demonstrator - Lessons Learnt, http://www.w3.org/2001/sw/Europe/reports/demo_2_report/ (2004).

[42] J. L. Austin, How to do things with words, Harvard University Press, 1962.

[43] ITU-T, Information Technology - Open Systems Interconnection - The Directory Authentication Framework. X.509 (1997).

[44] D. Eastlake, J. Reagle, D. Solo, XML-Signature Syntax and Processing, RFC 3275 - W3C Recommendation , http://www.w3.org/TR/xmldsig-core/ (2002).

[45] C. Bizer, R. Oldakowski, Using Context- and Content-Based Trust Policies on the Semantic Web, in: 13th World Wide Web Conference (Poster), 2004.

[46] R. Agrawal, P. Domingos, M. Richardson, Trust Management for the Semantic Web, in: 2nd International Semantic Web Conference, 2003.

[47] C. Bizer, Semantic Web Trust and Security Resource Guide, http://www.wiwiss.fu-berlin.de/suhl/bizer/SWTSGuide (2004).

[48] J. Golbeck, B. Parsia, J. Hendler, Trust Networks on the Semantic Web, in: 7th International Workshop on Cooperative Intelligent Agents, CIA2003, 2003.

[49] C. Bizer, TriQL.P - A Query Language for Querying Named Graphs Published by Untrustworthy Sources, http://www.wiwiss.fu-berlin.de/suhl/bizer/triqlp/ (2004).

[50] J. J. Carroll, J. De Roo, Web Ontology Language (OWL) Test Cases - W3C Recommendation, http://www.w3.org/TR/owl-test/ (2004).

[51] P. Zimmerman, Network Associates Inc., An Introduction to Cryptography, ftp://ftp.pgpi.org/pub/pgp/6.5/docs/english/IntroToCrypto.pdf (1999).

[52] ITU-T, The Directory – Models X.501 (1993).

[53] M. Wahl, S. Kille, T. Howes, Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names, RFC 2253 (1997).

[54] J. J. Carroll, C. Bizer, P. Hayes, P. Stickler, Semantic Web Publishing using Named Graphs, in: Trust, Security, and Reputation on the Semantic Web, (workshop at 3rd ISWC), 2004.

[55] J. J. Carroll, C. Bizer, P. Hayes, P. Stickler, Named Graphs, Provenance and Trust, Tech. Rep. HPL-2004-57, HP Labs (2004).

[56] J. J. Carroll, P. Stickler, TriX: RDF Triples in XML, Tech. Rep. HPL-2004-56, HP Labs (2004).