Chapter 5

Named Graphs

The practical problems with RDF reification raise the question whether a pure triple data model is adequate for applications which need to represent meta-information about RDF data or if a variation of the RDF data model would be more suitable.

This chapter proposes the Named Graphs data model, a simple extension of the RDF data model, which allows a more efficient representation of meta-information about RDF data.

The ideas presented in this chapter are the result of joint work with Jeremy Carroll (Hewlett Packard Labs, United Kingdom), Patrick Stickler (Nokia, Finland), and Pat Hayes (Institute for Human and Machine Cognition, United States). The discussions that lead to the development of the Named Graphs data model took place between January and April 2004 and are archived on the W3C www-archive mailing list[1]. The results of our joint work have been published in [CBHS05a].

The Named Graphs data model is designed to fulfill the following core requirements:

Representation of Meta-Information. The model should allow a more efficient representation of meta-information than the RDF reification mechanism.

Unique Identification of RDF Data. The model should provide a mechanism for the globally unique identification of RDF data, so that different information providers can express meta-information about the same RDF data.

Backward Compatibility. In order to provide as much backward compatibility with existing RDF data and deployed applications as possible,

[1] http://lists.w3.org/Archives/Public/www-archive/ (retrieved 09/25/2006)

the design should keep close to the RDF recommendations.

Exchange of Meta-Information. The model should be accompanied with syntaxes for publishing and exchanging information together with meta-information.

5.1 The Named Graphs Data Model

The Named Graphs data model is a simple variation of the RDF data model. The basic idea of the model is to introduce a graph naming mechanism, which allows RDF triples to talk about RDF graphs. A named graph is an entity which consists of an RDF graph and a name in the form of a URI reference. Two named graphs which have different names but share the same RDF graph are seen as two separate entities. Two named graphs which have different RDF graphs have to be named with different URI references.

The RDF data model represents information as a single node-and-edge labeled graph. Within the Named Graphs data model, information is represented as a set of named graphs.

> A set of named graphs is a set $\{(u_1, G_1), (u_2, G_2), ...(u_n, G_n)\}$ where each G_i is an RDF graph, and each u_i is a URI reference. All u_i are distinct.

In order to enforce the blank node scoping rules [Hay04] the global assumption is made that blank nodes cannot be shared between named graphs, meaning that if ng and ng' are different named graphs then the sets of blank nodes which occur in triples in ng and in ng' are disjoint.

A named graph is an RDF resource and can be described in the usual open way using RDF statements. RDF statements about a named graph may occur in the named graph itself or in other graphs. Information which is stated about a named graph is understood to refer to each statement within the graph. For instance, the statement that somebody is the creator of a named graph implies that he is the creator of each statement within the graph. This interpretation provides a simple, but flexible alternative to RDF reification, as it enables meta-information to be stated about graphs containing only a single statement as well as about graphs containing multiple statements. As graphs are uniquely identified by being named with a URI reference, it is possible for different information providers to make statements about a graph.

Figure 5.1 shows a graphical representation of a graph set consisting of two named graphs. Graph `http://www.bizer.de/Graph235` contains information about Document1325. Graph `http://www.bizer.de/Graph365` describes the

provenance of graph `http://www.bizer.de/Graph235`. The graphset represents the same information as the reification example in Figure 4.9.

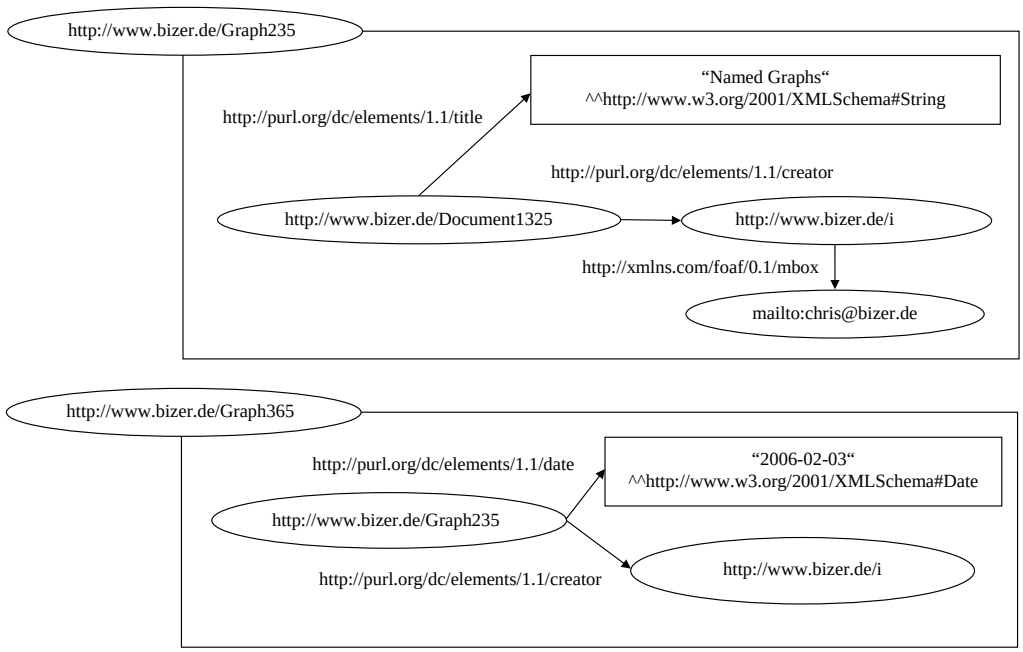

Figure 5.1: Graph set consisting of two named graphs.

5.2 Related Work

There has been an intensive discussion in the RDF community about RDF reification and several authors have proposed alternative approaches. The following sections compare the different proposals with the Named Graphs data model.

5.2.1 Quads

A quad is an RDF triple plus a further forth element. Several authors have proposed to use quads instead of RDF reification [Int06a, Bec03b, Dum03, MK03, TW05]. The different proposals vary widely in the semantics of the fourth element using it as statement ID, model ID, or generally to refer to the "context" of a statement. The proposals can be grouped into two categories depending on whether other quads are allowed to refer to the fourth element or not.

The approach to use quads without allowing them to refer to each other, is taken by Dave Beckett's Redland Application Framework [Bec03b]. Within this framework, all RDF triples belong to a single graph, but may be annotated with an additional context ID. This context ID may be a URI reference, a blank node, or a literal. As not all triples need to have a context ID, this leads to a model where some triples are contextualized while others are not. The context ID may be used in operations which change the graph, for instance delete all triples from a certain context, or as additional parameter in queries against the graph. A second tool-set from this category is Carsten Tolle's RDF-Source related Storage System[TW05], which stores the retrieval URL of the document from which a triple originates together with each triple. Both authors argue that quads should not refer to the fourth element because this would change the way information is represented in RDF and is not compatible with the RDF specifications. Therefore, the forth element should only be used for local operations within RDF tool-sets.

Robert MacGregor and In-Young Ko argue that quads should be allowed to refer to other quads [MK03]. They use the forth element to refer to the context of a statement. For them "a context consists of a set of facts (here, RDF triples) and a description of an environment within which these triples are believed to be true" [MK03]. As some quads may point to triples which are not context dependent, they propose to put a `null` value into the context position for these quads.

An example of a tool-set that implements quads which are allowed to refer to each other is Intellidimension RDF Gateway [Int06a]. Within this toolset, the forth element has to be an URI reference and is interpreted as "context" identifier. Contexts are used within the tool-set itself for representing access control restrictions.

The Named Graphs data model is very similar to the variation of quads used by Intellidimension RDF Gateway. What differentiates both approaches is the handling of blank nodes. RDF Gateway allows blank nodes to be shared between quads that belong to different contexts, while Named Graphs forbid blank nodes to be shared between graphs. Therefore, a set of Named Graphs can be decomposed losslessly into a set of separate RDF graphs, while a set of RDF Gateway quads cannot.

5.2.2 N3 Formula

A second variation of the RDF data model which may be used to represent meta-information about RDF data is N3 [BL98]. N3 has been developed by Tim Berners-Lee as a language for expressing data and rules. N3 extends RDF with features such as variables, universal and existential quantification.

N3 allows triples to be grouped to sets by using formulas. A formula may be used within triples like normal nodes. Blank nodes are not shared between formulas. Formulas can be seen as subgraphs within an outer graph formed by the N3 document. Formulas may be used within other formulas, allowing infinite subgraph chains.

Within the N3 syntax, formulas are enclosed with curly brackets. Figure 5.2 shows an example N3 document containing a formula in lines 5 and 6. This formula is used in line 6 as subject of a statement, saying that Richard is the author of the statements within the formula.

```
1.  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2.  @prefix dc: <http://purl.org/dc/elements/1.1/> .
3.  @prefix ex: <http://example.org/> .
4.
5.  { ex:Chris foaf:mbox <mailto:chris@bizer.de> .
6.    ex:document1243 dc:creator ex:Chris } dc:creator ex:Richard .
```

Figure 5.2: N3 formula example.

Allowing subgraph chains may be a useful feature for representing rules. For use cases where applications only require to represent and exchange meta-information about RDF data, it seems to be an unnecessarily far step away from the RDF specifications.

5.2.3 RDF Dataset

The Named Graphs data model has been adopted by the W3C Data Access Working Group with a slight modification as the data model underlying the SPARQL query language [PS05]. The SPARQL specification defines 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. All u_i are distinct.
>
> G is called the default graph. (u_i, G_i) are named graphs.

The main difference to the Named Graphs data model is the addition of the unnamed default graph. The default graph provides backward compatibility with RDF without named graphs, and allows the named graphs functionality of SPARQL to be optional. Thus, the SPARQL query language can be used by applications which require graph naming as well as by applications which do not require this feature.

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 repositories, while maintaining provenance information, may prove difficult. Further problems arise 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 5.3). 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, it is recommended to name all graphs before publishing them on the Web.

5.3 Syntaxes for Named Graphs

In order to exchange sets of named graphs between applications and to publish Named Graphs on the Web, serialization syntaxes are needed. A serialization syntax for named graphs has to exhibit the name, the graph, and the association between them. This chapter introduces two syntaxes which allow sets of named graphs to be serialized into single documents: TriX based on XML; and TriG as a compact plain text format based on Turtle.

5.3.1 The TriX Syntax

The RDF/XML syntax [Bec04b] provides various abbreviations which neither completely hide the underlying RDF, nor do they make it clear. The abbreviations make it impossible to describe RDF/XML with XML Schema [TBMM04] and make generic XML tools such as XPath [CD99], XSLT [Cla99], and XQuery [BCF+05] hard to use together with RDF/XML [CS04a, Bec04a].

TriX [CS04a] is an alternative XML-based syntax for RDF. TriX addresses the shortcomings of the RDF/XML syntax by having a basic syntax that corresponds closely to the RDF data model. In addition, TriX provides for naming graphs and for serializing several graphs in a single document.

The Trix syntax is described by the XML document type definition shown in Figure 5.3. The core of TriX is the `triple` element, which contains three children, the subject, predicate, and object of a 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

```
<!-- 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>
```

Figure 5.3: TriX document type definition, taken from [CS04a].

normalization is applied to `uri` and `id` element content. `typedLiteral` elements require a `datatype` attribute. `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).

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.

Figure 5.4 shows a TriX document containing two named graphs; the second graph describes the first.

5.3.2 The TriG Syntax

TriG [Biz05] is a plain-text syntax which provides for serializing several named graphs into a single document. TriG is a variation of the Turtle [Bec04c] syntax. TriG extends Turtle by introducing curly brackets to group triples into multiple graphs, and to precede each by the name of that graph. The complete EBNF grammar of TriG syntax is given in appendix B.

The TriG document shown in Figure 5.5 contains two graphs. The first graph (line 6-11) contains information about Chris and Document1325. Line 6 specifies that the graph is named `ex:Graph1`. The graph refers to itself as the graph name URI `ex:Graph1` is used within the statements in lines 11 and 12.

```
<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>
```

Figure 5.4: Example of a TriX document, taken from [CS04a].

5.4 Query Languages for Named Graphs

There are two query languages for the Named Graphs data model: TriQL [Biz04a] and RDFQ [Sti04]. Both languages extend the idea of matching triple patterns against a single graph to matching graph patterns against a set of named graphs. A graph pattern consists of an optional graph name and a set of triple patterns. The graph name as well as the components of the triple patterns may be variables or RDF nodes. If the graph name is omitted then a graph pattern matches all graph names.

TriQL [Biz04a] is based on RDQL [Sea04]. Figure 5.6 shows an TriQL query which selects people together with their email address and homepage. The query takes only information that is authored by `<http://www.richardcyganiak.de/me>` into account. Line 2-3 contain a graph

```
1.   @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2.   @prefix dc: <http://purl.org/dc/elements/1.1/> .
3.   @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4.   @prefic ex: <http://www.bizer.de/ExampleDocument/> .
5.
6.  ex:Graph1 {
7.    <http://www.bizer.de/i> foaf:mbox <mailto:chris@bizer.de> .
8.    <http://www.bizer.de/Document1325>
9.                          dc:title "Named Graphs"^^xsd:String ;
10.                         dc:creator <http://www.bizer.de/i> .
11.   ex:Graph1 dc:creator <http://www.bizer.de/i> .
12.   ex:Graph1 dc:date "2005-03-03"^^xsd:date . }
13.
14.  ex:Graph2 {
15.    <http://www.bizer.de/i> foaf:name "Chris Bizer";
16.                              foaf:homepage <http://www.bizer.de> .
17.    ex:Graph2 dc:creator <http://www.richardcyganiak.de/me> ;
18.    ex:Graph2 dc:date "2004-03-03"^^xsd:date . }
```

Figure 5.5: Example of a TriG document.

pattern, which matches named graphs that contain a `foaf:mbox` and a `foaf:homepage` statement about the same `?person`. The names of matching graphs are bound to the variable `?graph`. `?graph` is constrained by the second graph pattern in line 4 to graphs which are authored by Richard. The information that Richard authored a graph may occur in any named graph, as the second graph pattern does not constrain graph names.

```
1. SELECT ?person ?email ?homepage
2. WHERE ?graph ( ?person foaf:mbox ?email .
3.                ?person foaf:homepage ?homepage )
4.              ( ?graph dc:creator <http://www.richardcyganiak.de/me> )
5. USING foaf FOR <http://xmlns.com/foaf/0.1/>
6.         dc FOR <http://purl.org/dc/elements/1.1/>
```

Figure 5.6: TriQL query against a set of named graphs.

The TriQL query language was evaluated [Fuk04] by the W3C Data Access Working Group and TriQL's graph pattern matching mechanism has been adopted by the SPARQL query language [PS05]. Within SPARQL, the keyword `GRAPH` is used to restrict a graph pattern to match specific named graphs. Figure 5.7 shows the reformulation of the TriQL query from Figure 5.6 as a SPARQL query. In line 5, the variable `?graph` is bound to the names of all graphs that contain information about email addresses and homepages.

The second pattern in lines 8 and 9 constrains `?graph` to graphs that are authored by Richard.

TriQL and RDFQ predate SPARQL and it can be expected that the SPARQL standard will supersede both languages.

```
1. PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2. PREFIX dc: <http://purl.org/dc/elements/1.1/>
3. SELECT ?person ?email ?homepage
4. WHERE {
5.   GRAPH ?graph
6.   { ?person foaf:mbox ?email .
7.     ?person foaf:homepage ?homepage }
8.   GRAPH ?anyGraph
9.   { ?graph dc:creator <http://www.richardcyganiak.de/me> }
10. }
```

Figure 5.7: SPARQL query against an RDF dataset.