RDF Views

Brenton

May 2025

1 Introduction

Some RDF models are large, and some are enormous. They are far larger than can ever be loaded into the memory of a typical personal computer. This is likely to remain the case even when PCs have TB RAM modules as the RDF models will also be even larger.

In order to process data from these models it is necessary to create a small model that contains just the data of interest.

These small models are known as views[1].

1.1 Scope

The document covers the design of a view system. This includes the design for setting up views for a model; creating the view and populating it; and using it as a model.

1.2 Overview

1.2.1 Reason for Views

There are two reasons for using views: Providing an extract from some very large models; and providing a focussed view when displaying a model.

In the first case the view will have up to a few hundred statements, but frequently a lot less. In the second case the view might only have a dozen nodes since more than this can make for a cluttered diagram.

1.2.2 Construction

A view is constructed by starting with an RDF node, or statement, and then doing a breadth first search for connected nodes. This is repeated until the desired range is reached, or a maximum size is reached.

[1] This is my term, borrowed from the similar function in SQL.

The problem is that schema nodes can be connected to thousands, or millions, of other nodes. Just discovering that you have reached such a node can crash the process.

1.2.3 Boundary

The view system will include the ability to set up a boundary for a view. This will prevent the view from retrieving data that is out of the intended scope of the view.

1.2.4 Usage

A view should be identical to any other RDF model once created. It should be possible to create small views for use by visualisation functions. It should also be possible to combine them with other models as a submodel[2].

1.3 Audience

This document is intended for the developers of **SASSY**, and, in particular the developers of the RDF code.

[2]Currently submodels can only be constructed from the base models.

Glossary

Item	Description
Bound	A single element of a boundary. It performs a single test on a node to determine if the containing statement has reached the edge of the desired region.
Boundary	A named set defining a set of bounds, and the default size and range for the view.
Catalog(ue)	An RDF model that contains the metadata for a set of models.
Compound Model	A model combining two models, one of which is updatable and one which has a copy of a set of submodels.
Model	An RDF database that has been loaded into memory for processing.
Node	One of the three elements that make up an RDF statement.
Range	The maximum number of steps to get from the starting node to any other node in the region.
RDF	Resource Description Framework. A linked data system.
RDFGUI	A general purpose GUI program for examining and manipulating RDF models.
Region	A connected set of statements in a model.
SASSY	Software Architecture Support System.
Schema	A set of statements that define the classes and relationships used in a model.
Size	The number of statements in the model or view.
Statement	The primary record structure for RDF. It consists of three nodes called the subject, predicate, and object.

Item	Description
Submodel	A model that has been included as supporting data for a model.
View	An extract from a database selected for some purpose

2 Boundaries

2.1 Purpose

2.1.1 Large Models

Each large model will have one or more boundaries defined. These will specify where the search process should stop for a particular branch.

2.1.2 All Models

When visualising a model it should be possible to dynamically limit the region to be displayed.

2.2 Form

A boundary will consist of a set of *bounds*. These will be used to check if an RDF node is at the edge of the region.

The bounds will come in several types. A predicate bound will be used to match against the statement predicates. A resource bound will match against a resource node (subject or object). A matching bound will match the start of a URI, and a regex bound will also match a URI. (The latter is a bit more resource intensive.)

The boundary may also contain the range of the model. This is the number of steps in the breadth first search from the start node. Also a maximum number of statements may be specified.

2.3 Storage

The boundaries will be stored in the catalogue with the other metadata for the models. This is RDF data.

The catalogue schema will be extended to include boundaries for each model. Each boundary will be given an identifying name.

2.4 Configuration

2.4.1 Large Models

A GUI will be constructed in which boundaries can be set up. Initially this will be a tab in the RDFGUI program, (or a panel in the **Catalog** tab) but will eventually be a GUI library that can then be used as a standalone program, or a panel in other GUIs.

It should include the ability to copy between models.

2.4.2 Visualisation

The **Visual** tab in RDFGUI will have a popup/floating dialog for configuring the display of models.

3 Views

3.1 Purpose

3.1.1 Large Models

A view allows access to a region of models which are too large to load, or would take too long.

3.1.2 Small Models

Even small models are often too large to display diagrammatically. The rapid creation of a small view can provide a zoom effect allowing the user to focus in on some region of interest.

3.2 Construction

A view will be constructed in a memory model, probably the *Hashes* type.

3.2.1 Start Point

A view starts at some particular node (or, perhaps, statement) in the source model. For large models this node is typically found via a SPARQL query. During display of small models it would be the currently selected node.

Start points that fail the boundary check (i.e. they are on the boundary) will be ignored.

For use with submodels it may be indicated by its existence in another model. A set of potential nodes may be provided. These should be checked to confirm they are in the model.

3.2.2 Searching

From the start point the incoming and outgoing arcs are obtained and the statements added to the view. The nodes are checked for the boundary condition and if not at the edge the nodes that are not already in the view are added to the new list of starting points. The process is then repeated until the range or size constraint is reached.

3.3 Configuration

Views for large models can be preset, while those for visualisation can be configured dynamically.

The details of a view of a large model will be stored in the Catalogue. They will be entered using a GUI panel, initially in RDFGUI.

Configuration details will include the source model, the boundary to use, the defaults for range and size, and a default starting point.

4 Submodels

This section describes the integration of views of large models with the submodels system.

4.1 Purpose

Submodels allow the composition of models from a set of smaller models. This allows us to modularise the models, rather than have just a few large ones. Models can be reused, which simplifies maintenance.

For example a model can include its schema model as a submodel. The specific schema can include the core schema.

4.2 Mechanism

The submodel system uses a compound model. This consists of a normal model (main) and a temporary memory model. The temporary model is loaded (copied) from the included submodels plus the main model. The compound model performs queries (both SPARQL and programmatic) on the temporary model, and updates on both the temporary and main models. This has the effect of limiting the updates to just the main model which avoids the possibility of corrupting the schemas etc.

4.3 Configuration

The catalogue already contains the submodels for each model. The RDFGUI program has a panel for adding and removing submodels from a model.

If the view for large models appears as just another model in the catalogue then no change will be required in the configuration process.

4.4 Views

A temporary view will be created and then loaded into the temporary memory model for the submodels. It will use the boundary and default sizes recorded in the catalogue.

4.4.1 Starting Points

The starting points will come from the nodes in the already loaded statements. This implies the views should be loaded last. It also puts the initialisation of the view into the realm of the model rather than the catalogue, or as a combined effort.

It would be useful if resources that are from another model could be tagged with their source. One possibility would be for each node to have a link to its original model, but this is not supported by the C library. (Models are more

of an artifact than a core part of RDF.) Another idea is to include a statement indicating the source for just the nodes that are from a view. The experience with prefixes[3] would suggest that this is not a good idea. Following from that it seems that putting this data in the catalogue might be the best approach. But how?

During the loading of a view into the temporary model in the compound model each node will be given a link to the view, probably as a URI. When a link is made from the main model to one of these view nodes, the node will be added to a list of starting points for the main model. This list will need to be saved into the catalogue when the model is closed. Thus the catalogue will have all the information it needs to open and initialise a view.

4.4.2 Initial Starting Points

The above describes how a compound model with included views can be reloaded once it has been created. There remains the issue of how to get it started.

The basic mechanism is a SPARQL query on the large model that we need to create a view from. This can get us a starting point, but there are questions.

Lets assume we have a large model, *much-stuff*, that has been installed in a database, and defined to the catalogue. We have also defined a view in the catalogue, *much-stuff-view*, that has much-stuff as its source, and a suitable boundary installed.

There are several scenarios:

We need to do some exploration. Using any available documentation, and running queries on *much-stuff* looking for classes and properties, we determine the basic structure of the model. We then run queries to find one, or a few, nodes that are likely to be good starting points. These are added to the catalogue's description of the view, *much-stuff-view* as its default starting point(s).

If we now open *much-stuff-view* with RDFGUI we will be able to see a region around the starting points.

In the second scenario we open a model that has *much-stuff-view* as a submodel. We can now create statements that reference nodes in the view. These will be added to the catalogue when the model is saved. On re-opening they will be automatically used as starting points in addition to the defaults. The defaults can then be changed to include other parts of *much-stuff*.

[3]In an early version of librdfxx each model had a set of statements defining the prefix to URI relationship. It became obvious that metadata should be stored separately and was transferred to the catalogue.

5 Development Tasks

This is a list of the things that need to be done to the **SASSY** software to implement this idea.