N3 Rules Processing and HTML Translation for the Semantic web

Chun Yip Leung
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2003-199
October 2nd , 2003*

E-mail: chun.leung@doc.ic.ac.uk

N3, forward chained rules, RDF translation, semantic web

The semantic web is an emerging technology that could totally change how web works. It is based on the Resource Description Framework (RDF) which defines graphs to represent links between data. RDF/XML is the common format to store RDF graphs while N3 is another instant to represent RDF and provides powerful function like rules representation.

* Internal Accession Date Only
Approved for External Publication
© Copyright Hewlett-Packard Company 2003

N3 Rules Processing and HTML Translation for the Semantic web

Author: Chun Yip Leung (chun.leung@doc.ic.ac.uk)

Keyword: N3; Forward Chained Rules; RDF Translation; Semantic Web

Abstract

The semantic web is an emerging technology that could totally change how web works. It is based on the Resource Description Framework (RDF) [0] which defines graphs to represent links between data. RDF/XML is the common format to store RDF graphs while N3 is another instant to represent RDF and provides powerful function like rules representation.

In this report we will discuss how to connect N3 rules to the Jena2's rule processing system and also use the new system to solve the visual representation problem on browsing RDF.

Acknowledgements

This report and the project I have been working on within my six-month placement in Hewlett-Packard Labs Bristol gave my invaluable experience and I would like to say thank you to the Jena development team, especially Andy Seaborne, Chris Dollin and Dave Reynolds who gave me many advice throughout.

1 Introduction

In the current web, machines cannot understand the meaning of the links. [1]

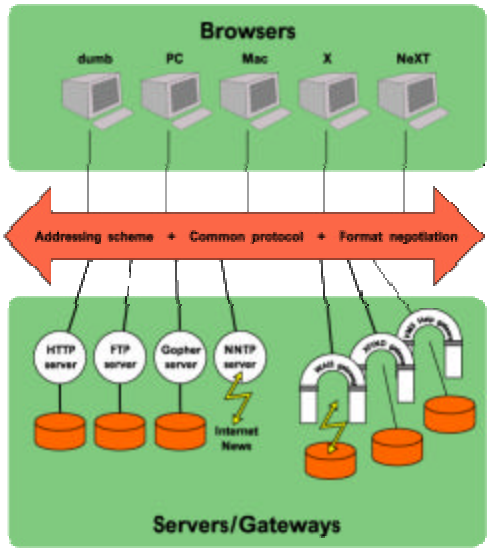


Figure 1 - The current web

Web browsers use HTTP protocol to access data served by different servers. Documents and resources are linked by hyperlinks and Hypertext is transmitted to present data over

web browsers. When the World Wide Web was designed, the assumption was a person will read the page, not machines. This makes it very hard for information management and processing, especially across different documents.

The Semantic Web makes a simple extension to the current web by using binary relationships to capture the meaning between links and data. [2] That makes it possible to represent machine processable information and lead to automation, integration and reuse of data across various applications. [3]

The computer industry has agreed and uses XML standards to give a syntactic structure for describing data. However, XML can be used in many different ways to describe the same data, this makes it too open and arbitrary to support the type of widespread and ad hoc data integration envisaged for the Semantic Web. [4] Therefore a standard syntax - Resource Description Framework (RDF) was introduced.

RDF defines a graph model to provide a consistent, standardised way of describing and querying internet resources, from text pages and graphics to audio files and video clips. It gives semantic interoperability, and provides the base layer for building a Semantic Web. [4]

2 Problem

"The Semantic Web is a vision: the idea of having data on the Web defined and linked in such a way that it can be used by machines not just for display purposes."[6]

2.1 Display information represented by RDF in web browsers

In the semantic web, RDF is the base for defining meaning, or relationships, between data. The most common way to store RDF data is to use RDF/XML; however XML syntax is not very good at the visual presentation of information to human beings; when we see XML files written according to the RDF specifications, we know how to *interpret* them. But that does not mean we could *see* the meaning straight away.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/">

<foaf:Person rdf:about="http://www.chunleung.org/">
 <foaf:name>Johnny Leung</foaf:name>
 <foaf:workplaceHomepage rdf:resource="http://www.hp.com/"/>
 <foaf:schoolHomepage rdf:resource="http://www.doc.ic.ac.uk/"/>
</rdf:RDF>
```

RDF/XML may be much harder to read than this:

```
Johnny Leung
        workplaceHomepage:    http://www.hp.com/
        schoolHomepage:       http://www.doc.ic.ac.uk/
```

Therefore, a better means of visual presentation would be necessary towards the success of semantic web. The most common way to make visual presentation over the Internet is

the hypertext. It would be a good idea if it is able to use HTML on displaying the information modelled by RDF, in particular RDF/XML, on web browsers.

One of the examples is using web server to serve FOAF (Friend of a Friend) data stored in RDF over the internet. And throughout this report, I will illustrate how this actually could be done.

Moreover, we normally do not want to display all the information stored within the RDF but only part of the information. For example, we have a collection of FOAF RDF about placement students working in Hewlett-Packard Labs but we only want to display FOAF data on a particular person, say about "Johnny". And within Johnny's data, we might only want a certain bit being displayed.

The principle is same as report generation in general I.T. industry but our data source is any data modelled in RDF. This could be done in XSLT up to a point if data is stored in XML format, but the XML's tree structure and centric design make it not so effective.

This happened because for the same graph, we could have different RDF/XML representation, as RDF focused on representing semantic, that means there are infinity ways to store the same semantic within the RDF/XML documents, that means we have to obtain the data within the semantic level.

The good thing about RDF is that there is a graph represented by it. The logical meaning, or semantic will not be affected on the representation method and therefore the translation should not be effected if the RDF data is provided in RDF/XML, N3, N-triples or in different type of Jena model.

2.2 Customizable system

When the data was presented over web browsers, it would also be great if the layout is not limited but any customizable layout.

The primary aim is different types of template could be easily created and the system could generate the HTML output according to the template so we could have whatever needed.

The further aim would be minimize the learning curve on how the configuration could be done. The system developer might need to understand the semantic web technology deeply and programming on declarative rules are required on serving different RDF data based on different schema, but the learning curve of the system administrator should be much shorter in order to configure the system.

The HTML template is based in normal HTML but with embedded tags, known as taglibs, and normal web developer or website administrator should be able pick up the technique on template writing in a very short time and create web pages to present RDF data.

3 Background

3.1 Semantic Web Technologies

Semantic Web technologies are based on a global naming scheme (URIs), a standard syntax for describing data (RDF), standard means of describing the properties of that data (rdf-schema) and standard means of describing relationships between data items (ontologies). [5]

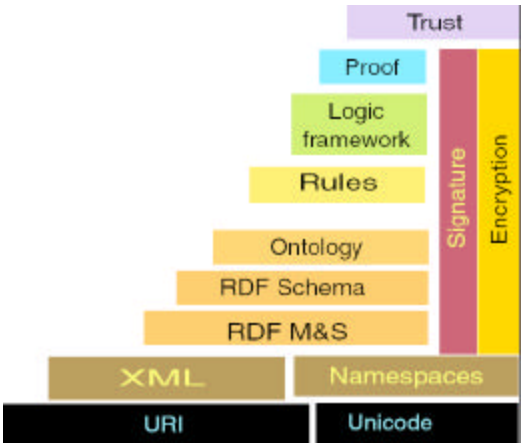


Figure 2 - The Layer Cake

The principles are:

- Everything Identifiable is on SW by URI (Universal Resource Identifiers).
- Some properties allow unambiguous identification
- Allow effective combination of the independent work of diverse communities.
- Support the ability to add new information without insisting that the old be modified.
- Provide communities the ability to resolve ambiguities and clarify inconsistencies.
- Use descriptive conventions that can expand as human understanding expands.

3.1.1 RDF / XML

RDF provides a consistent, standardized way of describing and querying internet resources and it provides the base layer for building a Semantic Web.

For instance, telnet(janet_bruten, 3128700) represents the fact that the person object Janet Bruten has the telnet number 312-8700. For example, if we have the following graphs representing relationships: [4]

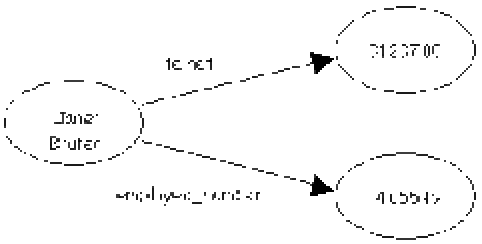


Figure 3 - A simple directed graph

This means:

http://www.hp.com/JanetBruten **has** telnet **whose value is** 31287002
and
http://www.hp.com/JanetBruten **has** employee_number **whose value is** 405549

The standard way to record these relationships is via RDF/XML:

```
<?xml version='1.0'?>
<rdf:RDF xmlns:rdf = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
   xmlns:person = " http://www.hpl.hp.com/semweb/person" >
        <rdf:Description rdf:about='http://www.hp.com/JanetBruten'>
                <person:telnet>31287002</person:telnet>
                <person:employee_number>405549</person:employee_number>
        </rdf:Description>
</rdf:RDF>
```

3.1.2 Notation 3 - N3 [7]

N3 is based on RDF standards and equivalent to RDF/XML syntax, but have extra features like rules and formula. N3 is not designed as an alternative to RDF's XML syntax but a language designed for a human-readable and scribblable language. [10]

It is easier and simpler than the RDF/XML approach. In N3, information is also a collection of statements; each with a subject, verb and object - and nothing else.

```
<subject> <verb> <object> .
```

In a statement, everything is identified by URI. And every statement is ended with a period. For example:

```
<#johnny> <#knows> <#mei>.
```

is a valid N3 statement. However, Object can be literal to represent different kind of knowledge, we could also have:

```
<#johnny> <#age> "22".
```

Another N3 syntax used for saving space is the comma and semicolon. The comma means the following object has the same subject and predicate, and semicolon means another predicate for same subject. For example,

```
<#johnny> <#knows> <#mei>, <#cissy>, <#jo> ;
    <#age>   "22";
    <#eyecolor> "black".

Is equivalent to:
<#johnny> <#knows> <#mei>.
<#johnny> <#knows> <#cissy>.
<#johnny> <#knows> <#jo>.
<#johnny> <#age>   "22".
<#johnny> <#eyecolor> "black".
```

This shows the use of comma and semicolon. By using the basic N3 syntax, for the above Janet Bruten's example; we could represent the data in N3 format:

```
@prefix :  <http://hpl.hp.com/> .
@prefix person: <http://www.hpl.hp.com/semweb>.
:JanetBruten
        person:employee_number "405549";
        person:telnet "31287002".
```

Sometimes there are things involved in a statement don't actually have any identifier you want to give them - you know one exists but you only want to give the properties. They are blank nodes, and we might represent this by square brackets with the properties inside.

```
<#johnny> <#child> [ <#age> "4" ] , [ <#age> "3" ].
```

That means <#johnny> has <#child> which one is <#age> 4, and one <#age> "3".

Beside N3, there is also another format, called N-triples, for representing RDF data in plain text format. It was designed to be a fixed subset of N3 and N-Triple is the same as N3 until now. So what make N3 so different from normal RDF and powerful?

3.1.3 N3 rules and Formula

In N3, we have rules. A simple rule might say something like (in some central heating vocabulary) "If the thermostat temperature is high, then the heating system power is zero", or

```
{:thermo :temp :high } => {:heating :power "0"}.
```

The curly brackets here enclose a set of statements. We call it a formula. All formulae are enclosed by the curly brackets. Apart from the fact that the subject and object of the statement are formulae, the thermo example shown above is just a single statement.

The "=>" used here is a special predicate, means "implies". This is used to links formulae. It is actually the short hand of the URI log:implies, or: **http://www.w3.org/2000/10/swap/log#implies**. When two formulas linked with log:implies, it is a rule. Therefore all rules are just a different kind of statements.

Formulas take us out of the things we can represent using the current RDF/XML; these Rules are not part of standard RDF syntax.

As stated above, we could see formula as a collection of statements, for examples:

```
{:thermo :temp :high. :heating :power "1". } => {:heating :power "0"}.
```

is another instant of valid N3 rules. In fact, a formula could be more than just a collection of statements. We could have variable within a formula. Variable is start with a "?". Examples or variables are:

```
{ ?x rdfs:subClass ?a, ?a rdfs:subclass ?b} => {?x rdfs:subClass ?b}
```

The rule says, if x is subclass of a, and if a is subclass of b then x is subclass of b.

3.2 FOAF

FOAF allows the expression of personal information and relationships, by using FOAF vocabularies it is possible to share information about different persons.

Here is a very basic document describing a person:

```
<foaf:Person>
        <foaf:name>Chun Leung</foaf:name>
        <foaf:mbox_sha1sum>1234567890</foaf:mbox_sha1sum>
         <foaf:homepage rdf:resource="http://www.doc.ic.ac.uk/~cyl00" />
         <foaf:img rdf:resource="http://www.doc.ic.ac.uk/~cyl00/photo/chun.jpg" />
</foaf:Person>
```

This brief example introduces the basics of FOAF. It basically says, "there is a foaf:Person with a foaf:name property of “Chun Leung” and a foaf:mbox_sha1sum property of 1234567890; this person stands in a foaf:homepage relationship to a thing called http://www.doc.ic.ac.uk/~cyl00 and a foaf:img relationship to a thing called http://www.doc.ic.ac.uk/~cyl00/photo/chun.jpg.

The full descriptions of all FOAF vocabularies could be found in http://xmlns.com/foaf/0.1/ .

3.3 Current Jena2 rules system implementation

There are two different type of rules system; forward chained and backward chained. Backward chaining systems are goal-directed, query will run against the data and the rule engine will either prove the statement as true or false, or finds the facts that satisfy the constraints. For example, Prolog is backward chaining.

Forward chaining rules systems are data-directed. Rules will run against the current data and new facts will be found and inserted into the dataset if the preconditions have been satisfied. The new rules will also be fired until no new fact have been found. [8]

For customizing the output to the selected data, some form of querying is needed. And the intention is to use N3 rules with a fixed set of vocabularies to create the necessary model. Therefore a forward chaining rule processing engine is needed for HTML translation..

Jena2 has a build-in rules system with forward chain reasoner, backward chaining reasoner, RETE reasoner and hybrid reasoner. However it only runs on its own language, not N3 rules.

3.4 Related Work

Currently, there are a few RDF rule processor and visualization tools available on the web.

3.4.1 CWM

CWM is a general-purpose data processor for the semantic web, somewhat like sed, awk, etc. for text files or XSLT for XML. It is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3 (see Notation3 Primer) serializations as required.

CWM is written in python; it is part of SWAP, a Semantic Web Application Platform. And there is another backward chained engine call Euler which is interoperable with CWM at the language level.

Official web site: **http://www.w3.org/2000/10/swap/doc/cwm.html**

3.4.2 IsaViz

The most common graphical visualization for RDF is IsaViz. It is a visual environment written by W3C using Jena, for browsing and authoring RDF models represented as graphs. It features:

- a 2.5D user interface allowing smooth zooming and navigation in the graph
- creation and editing of graphs by drawing ellipses, boxes and arcs
- RDF/XML, Notation 3 and N-Triple import
- RDF/XML, Notation 3 and N-Triple export, but also SVG and PNG export

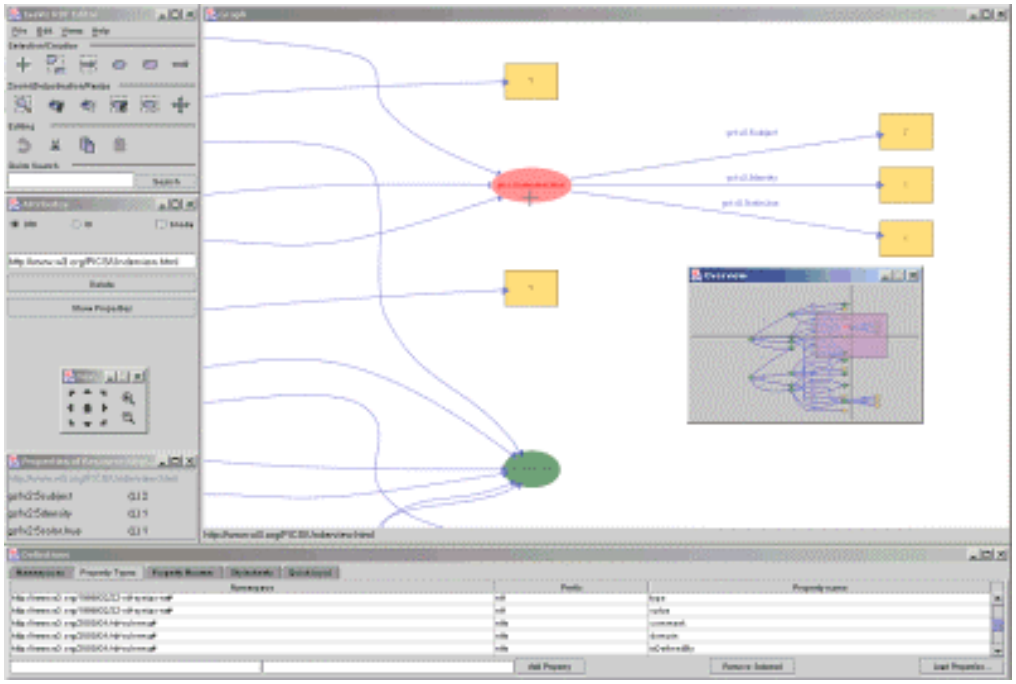


Figure 4 - IsaViz

The current version is 2.0, and since then IsaViz can render RDF graphs using GSS (Graph Stylesheets), a stylesheet language derived from CSS and SVG for styling RDF models represented as node-link diagrams.

Official web site: **http://www.w3.org/2001/11/IsaViz/**

3.4.3 BrownSauce

BrownSauce is a generic RDF browser. It was written by Damian Steer whilst employed at HP Labs Bristol. BrownSauce is free software, released under a BSD style licence.

Figure 5 - BrownSauce

Official web site: **http://brownsauce.sourceforge.net/**

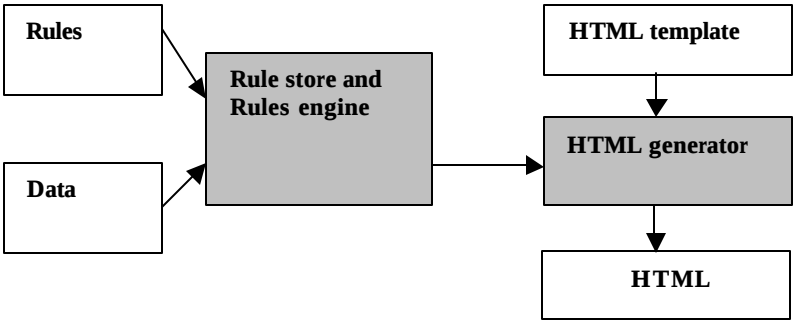
3.4.4 RDF Template language [9]

RDF Templates work in a similar fashion to XSLT. It will take RDF graphs and translate it into XML according to the stylesheets. The problem the RDF Template (RDFT) dedicated to solve is exactly the same as our problem. We will discuss RDFT in details later.

4 Solution

4.1 Overview of the design – the JRPJ system

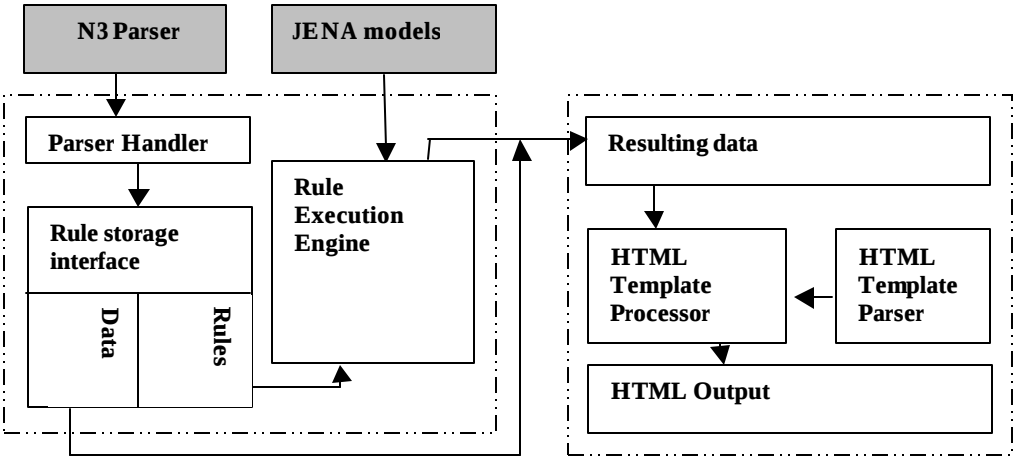
The Jena Rule Processing in Java (JRPJ) system consists of two main components; the rules store/ execution engine and the HTML generator.



The system takes in set of rules and collection of RDF data model. The system will use the forward chaining rules engine to process all the data with the given rules.

Then user could get the result or if the user uses the specific defined set of N3 rules and by feeding the result from the rules engine into the HTML generator, in combination of the HTML template; the system could translate the given RDF data into HTML.

Here is a more detailed diagram on the system design:



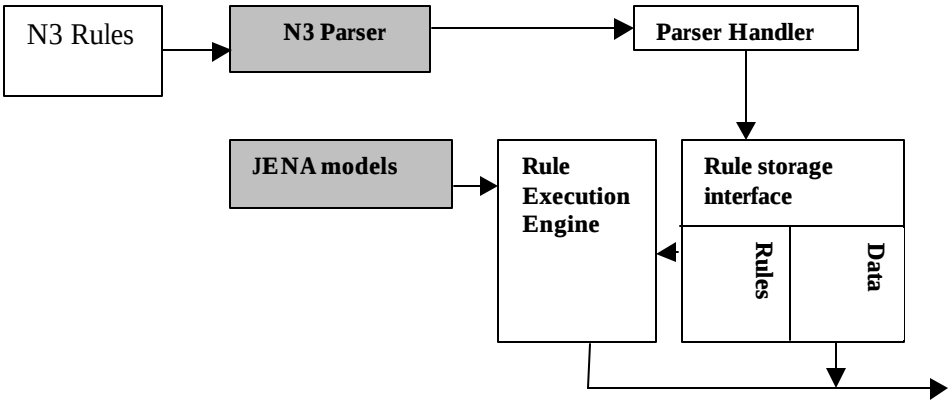
The N3 parser is used to read the N3 rules. By connecting the parser to the rule store, rules and data could be retrieved for execution engine.

After the data and rules being read, the execution engine, which extends the Jena's build-in forward chain reasoner, will process the rules using the data from the Jena model sit on the other end. There could be more than one Jena models but the Jena API already provided an easy way to deal with this with a Union Model. So we could always treat many graphs as one graph.

The execution engine will return a resulting graph which contains all the statements of the result of execution. This graph could be passed to other system, or most importantly, the HTML generator.

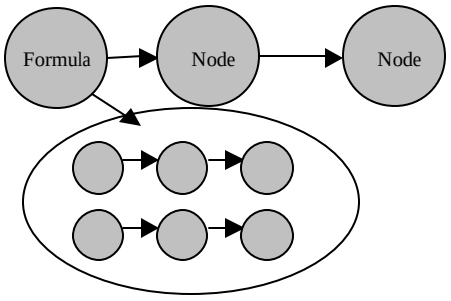
The HTML generator will read in the HTML template file, and using the information from the resulting RDF graph to generate the HTML. The HTML could be printed out to the system's standard output, or a java servlet could be used to server the HTML over a servlet enabled web server.

4.2 Graph and rules handling



We know that N3 is just a different, extended version of RDF, therefore the best and simplest way to hold N3 statements is to extend the current Jena's mechanism of holding graph – which is the *Graph* interface and the *GraphMem*, the memory implementation.

The main difference of N3 to normal RDF is N3 have formulas. So a new type of node – *Node_Formula* is added into the Jena's Node hierarchy. The *Node_formula* had an enclosed graph to store the collection of statements within this formula and the details will be described in the following section.



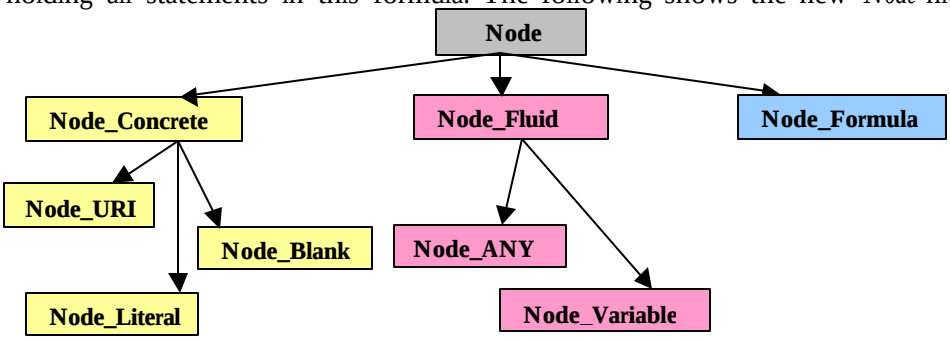
4.2.1 JRPJ Rule Store

Although N3 formulas are just another kind of RDF statements, unfortunately, it is not possible to use Jena model to store the rules directly. However, the similarity makes it possible to extend the Jena's *Graph* interface to store N3 statement and this will be the JRPJ rule store.

As mentioned before, we will use the *GraphMem* implementation provided in Jena. It is because normally we will run a small set of rules against a large set of data. So using *GraphMem* should provide a balance on speed on memory usage.

The main task is to implement a new Node type; the *Node_Formula*, which had an enclosed graph to store the collection of statements within this formula.

The *Node_formula* will extends the Jena's *Node* class and provide an enclosed graph for holding all statements in this formula. The following shows the new *Node* hierarchy:

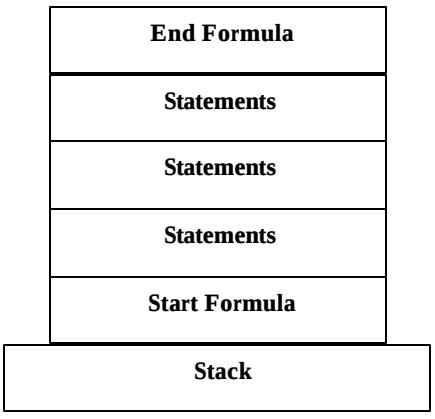


4.2.2 N3 Parser and formula handling

Jena2 has a build-in N3 parser to read normal N3. A new parser handler has been written to handle N3 and its formulas because N3 is not complete to RDF.

The new *N3RulesReader* will be a subclass of the *N3Parser*. The major change is the used of a new *ParserHandler*. The new parser handler could deal with formula and feed the parsed N3 statements into the JRPJ rule store.

The parser will read the N3 rules file and the handler will construct back all the statement and a stack will be used the obtain all the statements enclosed in formula. *Node_formula* contain an enclosed graph for the enclosed statement, a new graph will be constructed and put into the *Node_formula.*

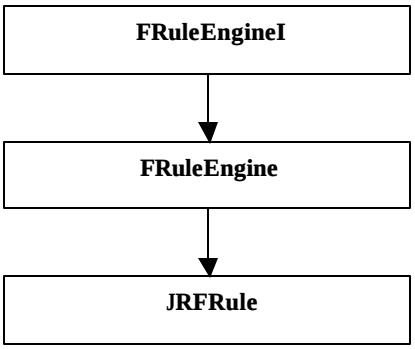


Stack was used to handle formula. When the parser saw the start of formula, i.e. a '{', it will put the token into the stack. Then the following statements will be put into the stack as well until the end of formula, '}' reach. Then all statements in the stack will be dumped into the Node_forumla

4.2.3 Connecting and interfacing to Jena's Forward Rules System

The current version of Jena2 has a simple reasoner that I could reuse the codes and probably extends its functionality. This reasoner does run on use the standard N3 rules but the idea behind is the same so I could extend it and do a little translation on N3 to use this.

The *JRFRule* extends the *FRuleEngine* which implements the *FRuleEngineI* interface *RETEEngine* could also be used if needed but we use the *FRuleEngine* as start.



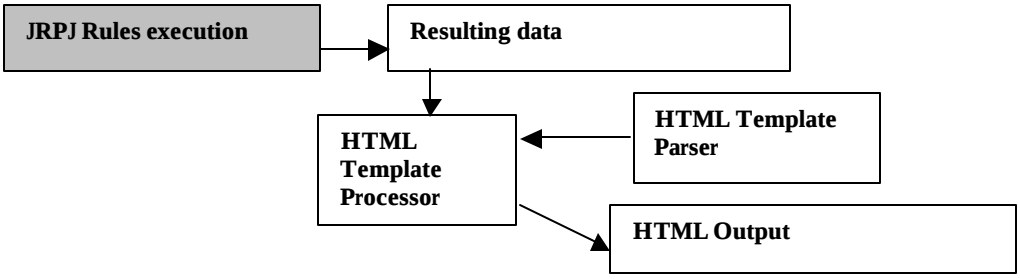
To connect the reasoner with the N3 rule store, what have been done is make the rule store as an implementation of the List interface as well. The main idea behind is the Jena's *ForwardChainReasoner* expect a *List* of *Rule* as argument, and therefore the easiest way is to make the rule store compatible with what the reasoner expected.

The following code illustrates how we could translate a N3 statement with *Node_formula* into *Rule* easily: And by translating all statements in N3 into the expected *Rule*, the in-built reasoner could be used for this system.

```
Rule rule = new Rule(translateForToList((Node_Rules) t.getObject()),
                     translateForToList((Node_Rules) t.getSubject()));

public List translateForToList(Node_Rules node) {
        List list = new ArrayList();
        if  (node.isFormula()) {
                Node_Formula nf = (Node_Formula) node;
                Iterator i = GraphUtil.findAll(nf.getGraph());
                while (i.hasNext()) {
                        Triple t = (Triple) i.next();
                        ClauseEntry c = new TriplePattern(t.getSubject(),
                                                          t.getPredicate(),
                                                          t.getObject());
                        list.add(c);
                }
        } else {list.add(n);}
        return l;
}
```

4.3 HTML Translation



It is the time to move onto the HTML translation. The Rule Processing Unit mentioned above will take a collection of RDF data and user defined N3 Rules for execution, resulting a new set of RDF in Jena model.

The idea of the HTML translation is to use a template language. The template language consists of a well defined set of N3 rules and a HTML templating syntax. When the rules run against the data, we have a well structured model and using the HTML Template to transform the data into HTML.

Moreover, beside HTML generate, it is possible to generate XML but the system is primary designed for HTML generation, so extensions might be needed on the current system to handle a wider set of template language.

4.3.1 Template Language – The HTML template

Here is an example of HTML template we used for displaying FOAF data.

```
<HTML>
        <Body>
                <?JRPJ :line1 /> br <Br>
                Name: <?JRPJ :name /><br>
                Full Name: <?JRPJ :firstName /> <?JRPJ :surname><br>
                <?JRPJ :nick>
                        Multilline – Nick name: <?JRPJ :nick /> <br>
                </JRPJ><br>
                Weblog: <?JRPJ :weblog /><br>
                mbox: <?JRPJ :mbox_sha1sum/><br>
                Homepage: <?JRPJ :homepage/><br>
                Image(desc): <?JRPJ :depiction/><br>
                wphp: <?JRPJ :workplaceHomepage/><br>
                schhp: <?JRPJ :schoolHomepage /><br>
                Known Person: (table)<br>
                <?JRPJ :table1 /><br>
        </Body>
</HTML>
```

All variables are of the form <?JRPJ **URI**>. And the URI will be used by the HTML generator to match the result from the rules. For example, for the variable <?JRPJ :name/>, the HTML generator will replace this variable with the result of executing the rules about tmp:name. However the "/" is very important, it separate out single variables from multi-line variables.

For example, when the HTML system see the following:

```
<?JRPJ :nick>
        Multilline – Nick name: <?JRPJ :nick /> <br>
</JRPJ>
```

.

That means there would be more than one nick name and the system will repeat the content enclosed within the tag.

4.3.2 Template Language – Syntax of N3 rules

In this section we will discuss the details of the special operators. The prefix used is tmp: <**http://www.hpl.hp.com/2003/jrpj/htmlTemplate#**>. For example, **tmp:head** is the short hand of **http://www.hpl.hp.com/2003/jrpj/htmlTemplate#head**.

tmp:head Indicates all information will go into head section of a HTML document. The object in the triple indicates which tag will be used to put that information.

tmp:body Indicates all information will go into the body of the HTML target. The Object will be the location where this will go.

In order for the system to work, you have to specify the head and body, like this:

```
_:doc rdf:type tmp:Template;
                tmp:head _:head;
                tmp:body _:body.
```

Which means we have a document type **tmp:template** and it has both a head and body.

tmp:title indicates information goes into <title> tag, for example:

```
_:head tmp:title "FOAF data".
```

Means this document has a title of “FOAF data”

tmp:link indicates information goes into <link> tag, for example:

```
_:head tmp:link 'rel="stylesheet" href="/main.css" type="text/css"'.
```

tmp:meta indicates information goes into <meta> tag, for example:

```
_:head tmp:meta “some meta data”.
```

To illustrate the effect of the first three rules, if say we have the above three rules ran, i.e.

```
_:doc rdf:type tmp:Template;
                tmp:head _:head;
                tmp:body _:body.

_:head tmp:title "FOAF data".
_:head tmp:link 'rel="stylesheet" href="/main.css" type="text/css"'.
_:head tmp:meta “some meta data”.
```

the resulting HTML will be the following:

```
<head>
        <title>FOAF data</title>
        <link rel="stylesheet" href="/main.css" type="text/css">
        <meta some meta data>
</head>
```

The most common operation on the body are: **tmp:variable,** which indicate the system should replace the variable in the html template, **tmp:location** tell the system where the content goes to and **tmp:content** tell the system what is the value

So, using the above three URI together, we could have:

```
_:body tmp:variable [ tmp:content "<br>"
                      tmp:location tmp:line1].
```

Which means the body have a variable named **tmp:line1**, and if the system found this variable, it will be replaced by the string “
”.

So, for example, if we the system sees “<?JRPJ :line1/>” in the HTML template, the output will be “
”.

tmp:tag and **tmp:value** are used to support any unsupported tags and XML generation, you could also specify the tag name used using **tmp:tag** and **tmp:value** to specify attributes of the tag. For example, if we have the following N3 rules:

```
{?x rdf:type foaf:Person. ?x foaf:surname ?y}
        =>
                { _:body tmp:variable [ tmp:content ?y;
                                        tmp:location tmp:surname;
                                        tmp:tag "font";
                                        tmp:value "size=12";
                                        tmp:value "color=red". ] }.
```

That means if we have the surname on the FOAF data, then create a font tag with arguments size=12 and color=red, with content value the value of variable y, and replace the tmp:surname variable on the HTML template.

The output of the above rule would therefore be:

```
<font size=12 color=red>content extracted from the RDF</font>
```

tmp:a, Create HTML <a> tag, for example:

```
{?x rdf:type foaf:Person. ?x foaf:schoolHomepage ?y} =>
        {_:body tmp:variable [ tmp:location tmp:schoolHomepage;
                               tmp:content "homepage";
                               tmp:a ?y]  }.
```

would create

homepage

tmp:img will create <img> tag for images.

```
{?x rdf:type foaf:Person. ?x foaf:depiction ?y} =>
        { _:body tmp:variable [ tmp:location tmp:depiction;
                                tmp:image ?y]
        }.
```

tmp:list for multi-valued properties, we might use **tmp:list** to print it out.. It will just print each value out line by line. However, it is not same as the RDF list.

```
{<http://www.chunleung.org/> foaf:nick ?y}=>
        {_:body tmp:list [ tmp:location tmp:nick;
                           tmp:content ?y]
        }.
```

For examples if we have multi-valued properties, then we could use list to handle it.

```
 <foaf:nick>look_yau</foaf:nick>
 <foaf:nick>Johnnnnnny</foaf:nick>
 <foaf:nick>@_@</foaf:nick>
 <foaf:nick>The fat John</foaf:nick>

{<http://www.chunleung.org/> foaf:nick ?y}=>
        {_:body tmp:list [ tmp:location tmp:nick; tmp:content ?y]}.

<?JRPJ :nick>
        Nick name: <?JRPJ :nick /> <br>
</JRPJ>
```

4.3.3 Template Language – Tables RDF to HTML translation

Tables are more complicated as it has multiple values on multi-rows with orders. So I think one of the best way is rules programmer must define/declare any tables they will be using before the rules that put data into the table.

Declaration of tables:

```
_:doc tmp:hasTable [  tmp:name tmp:table1;
                      tmp:location tmp:somewhere;
                      tmp:hasColumns ( [ tmp:name :someURI;
                                         tmp:content “some text”]
                                       [ tmp:name :someURI;
                                         tmp:content “some text”]
                                     )
                      tmp:orderBy tmp:someURI
                    ].
```

The idea is basically the same as before. However, we introduced two new operations, **tmp:hasTable** and **tmp:hasColumns** into the our HTML generation system.

tmp:hasTable tell the system that the programmer is declaring a table here, and **tmp:hasColumns** define the columns of this table. To define the columns, RDF list is used, which everything is surround ed by ( ) and in items are separated by space.

```
( [ tmp:name :someURI;  tmp:content “text”] [ tmp:name :someURI; tmp:content “text”] )
```

By using the RDF list, we could make define the order of the column of the table and the system could take care of the ordering when we actually fill in the data using the N3 rules.

Using the same style as before, everything inside columns list are blank node. (i.e. [ ]) Within these node, we use **tmp:name** to define the name of this cell and **tmp:content** to define the text of the header. As usual, **tmp:tag and tmp:value** could still be used to create tags.

The advantage of using list is everything could be in a defined order of the columns. And **tmp:orderBy** could be used to define the sorting of rows using which column in this table.

We use **tmp:hasRow** to indicate some data will be a row in this table, then use **tmp:hasCell** to tell which data will go into which cell.

To put data into the table, we could use something like:

```
:table1 tmp:hasRow [ tmp:hasCell [tmp:name :someURI; tmp:content ?x];
                     tmp:hasCell [ N3 nodes];
                     tmp:hasCell[N3 nodes]
].
```

Or, in our FOAF example:

```
{<http://www.chunleung.org/> foaf:knows ?y. ?y rdfs:seeAlso ?z. ?y foaf:name ?a} =>
        { tmp:table1 tmp:hasRow [ tmp:hasCell
                                        [ tmp:location "col1";
                                          tmp:content ?z];
                                  tmp:hasCell
                                        [ tmp:location "col2";
                                          tmp:content ?a]
                                ]
        }.
```

The usage of **tmp:location** and **tmp:content** are the same, but the location here means the location within the table, in stead of normal variable in the document.

4.3.4 The translation

After discussing the language itself, the rest it how the translation actually be done. The system simply iterate through the given model, search for know properties and use it to create the HTML trmplate.

```
        // head
        StmtIterator itHead = head.listProperties();
        while (itHead.hasNext()) {
                // set head
        }
        // variables
        StmtIterator itVariable =
        body.listProperties(
                m.createProperty(NameSpaces.Template, "variable"));
        //for each tmp:varibale
        while (itVariable.hasNext()) {
                //create tag
        }
        // lists -- basically doing the same thing as variable
        StmtIterator itList =
        body.listProperties(
                    m.createProperty(NameSpaces.Template, "list"));
        // for each tmp:list
        while (itList.hasNext()) {
                //create tag
        }
```

and it is the same for tables, but before that, tables must be created using table declaration. Then the data will be filled into the table.

```
// create table
StmtIterator itTables =
doc.listProperties(
      m.createProperty(NameSpaces.Template, "hasTable"));
while (itTables.hasNext()) {
      // create table
}
// fills in tables data
Iterator itTablesArray = tables.iterator();
while (itTablesArray.hasNext()){
      String tableName = (String) itTablesArray.next();
      HtmlTable hTable = (HtmlTable) map.get(tableName);
      Resource table = m.getResource(
            NameSpaces.Template + tableName);
      StmtIterator itRows =
            table.listProperties(
                  m.createProperty(NameSpaces.Template, "hasRow"));
      while(itRows.hasNext()){
            //fill in the row
      }
}
```

5 Analysis / Review

5.1 Prototype

Two prototypes had been built to demonstrate this system; they are the command line and the servlet version for HTML generation.

5.1.1 Command line

The command line version is to test out the N3 parsing and rules processing ability. The basic usage is:

```
[-N3] [-HTML] [N3Rules_filename] [RDF_data_filename] [HTML_template]
```

-N3: The RDF data file is in N3 format.

-HTML: Enable HTML processing

The system will depend on the arguments passed in and set up the environment. Then will parse the rules, execute the rules against the data using the engine and print the result.

```
//parse must be done after everything been setup
jrpj.parse();

//execute
Graph resultingGraph = jrpj.execute();

//output the result
try{
      jrpj.flush(resultingGraph);
}
catch(HTMLNotReadyException e){}
```

After testing, this prototype successfully parses any common N3 Rules and could execute against the data if HTML is not enabled. And the result will be printed to the standard output.

5.1.2 Servlet

A general servlet had also been developed for serving RDF over web server. All subclass implementation must implement the following functions:

```
abstract protected HtmlParser getHtml(HttpServletRequest request) throws Exception;

abstract protected Jrpj getJrpj(HttpServletRequest request) throws Exception;
```

These two functions tell the system how to find the HTML Template and the RDF data from the given arguments. Normally the HTML template is located at fixed location, for example:

```
static String templatePath = "FOAF/FOAFtemplate.html";

protected HtmlParser getHtml(HttpServletRequest request)
                                                throws FileNotFoundException {
        return new HtmlParser(templatePath);
}
```

Programmer could just return a new *HtmlParser* without looking at the arguments fed into the system. And for the *getJrpj* methods, it could be as simple as looking up the files in a specificed location, for example:

```
static String FOAFpath    = "FOAF/";

protected Jrpj getJrpj(HttpServletRequest request) throws FileNotFoundException{
        String[] values = request.getParameterValues("id");
        try {
                return new Jrpj(rules, FOAFpath + values[0] + ".rdf");
        } catch (RDFException e) {
                printErrorMessage(e);
        }
}
```

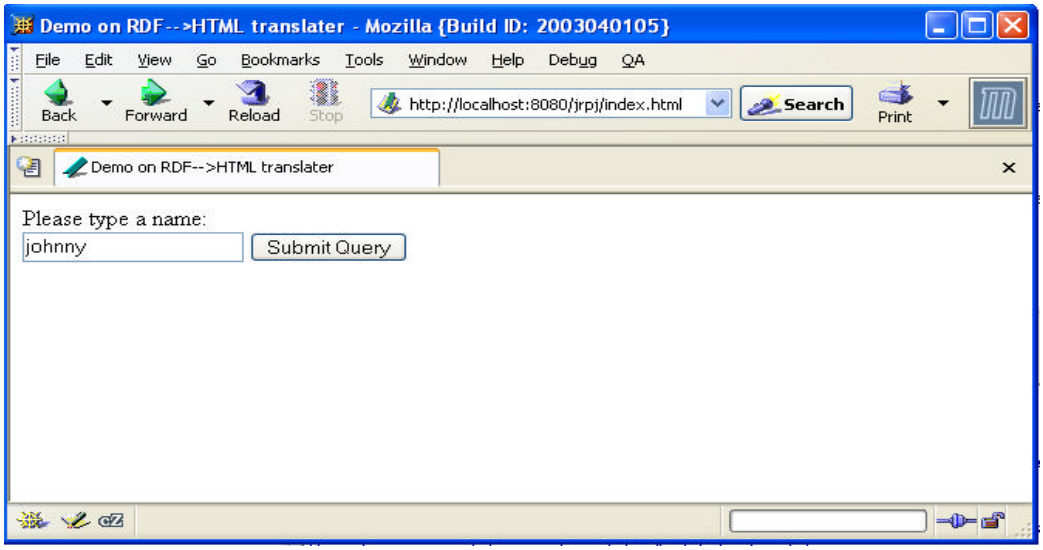
After the programmers have defined the necessary methods, the following code was used to serve the data.

```
//setup
HtmlIO io = new BasicHtmlIO(inFile, out);
HtmlTemplate template = new BasicHtmlTemplate(io);
jrpj.setHtmltemplate(template);

// parse could only be done after everything been setup
jrpj.parse();
// execute
Graph resultingGraph = jrpj.execute();

// write result
jrpj.flush(resultingGraph);
```

Here are some screen shots for the demo program: (Please see Appendix A for full data used in the demo)



This HTML page simply provides a text box to input the name of person you want to view:

```
<html>
        <head><title>Demo on RDF-->HTML translator</title></head>
        <Body>
                <form method="get" action="/jrpj/servlet/FOAF">
                        Please type a name:<br>
                        <input type="text" name="id">
                        <input type="submit">
                </form>
        </Body>
</html>
```

For this example, we want to read the FOAF's data about "johnny". And the following was part of the RDF data we want to display:

```
  <foaf:name>Johnny Leung</foaf:name>
  <foaf:firstName>Chun</foaf:firstName>
  <foaf:homepage rdf:resource="http://SWEB-PRJ-7/index.html" />
  <foaf:depiction rdf:resource="http://SWEB-PRJ-7/cat.jpg" />
……..
  <foaf:knows>
    <foaf:Person>
      <foaf:name>Jon Hayman</foaf:name>
      <rdfs:seeAlso rdf:resource="http://www.doc.ic.ac.uk/~jmh00/" />
    </foaf:Person>
  </foaf:knows>

  <foaf:knows>
    <foaf:Person>
      <foaf:name>Mei Leung</foaf:name>
      <rdfs:seeAlso rdf:resource="http://www.doc.ic.ac.uk/~cyl00/mei/" />
    </foaf:Person>
  </foaf:knows>
……..
```

The system will return the FOAF data in HTML format by running the rules against the given data and transform it into HTML. For example

```
{<http://www.chunleung.org/> foaf:name ?y} =>
        { _:body tmp:variable [ tmp:content ?y; tmp:location tmp:name]}.
```

will exact the **foaf:name**, which means the full name, of the person. And this will replace the **tmp:name** from the template into the result, which is "Johnny Leung". And

```
{<http://www.chunleung.org/> foaf:firstName ?y} =>
        { _:body tmp:variable [ tmp:content ?y;
                                tmp:location tmp:firstName;
                                tmp:tag "font";
                                tmp:value "size=12";
                                tmp:value "color='yellow'"]
        }.
```

will exact the **foaf:firstName** from the data; i.e. "Chun", and will create a font tag with attributes "size=12" and "color=yellow". Therefore we will have

```
  <font size=12 color=red>Chun</font>
```

be replaced in the final output. The same principle will also apply to the **foaf:homepage**:

```
{<http://www.chunleung.org/> foaf:homepage ?y} =>
        { _:body tmp:variable [ tmp:location tmp:homepage;
                                tmp:content "weblog";
                                tmp:a ?y]
        }.
```

will create the <a> tag with the href attribute be the exacted value from the RDF data, which is http://SWEB-PRJ-7/index.html.

```
{<http://www.chunleung.org/> foaf:depiction ?y} =>
        { _:body tmp:variable [ tmp:location tmp:depiction;
                                tmp:image ?y]
        }.
```

will create a <img> tag for the picture of this FOAF data. For table, we have to use the following code:

```
_:doc tmp:hasTable [ tmp:name "table1";
                     tmp:location tmp:table1;
                     tmp:headers ( [tmp:name "col1";
                                    tmp:tag "a";
                                    tmp:content "see_also"]
                                    [tmp:name "col2";
                                    tmp:tag "a";
                                    tmp:content "name"] )
                   ].

{<http://www.chunleung.org/> foaf:knows ?y. ?y rdfs:seeAlso ?z. ?y foaf:name ?a}=>
        { tmp:table1 tmp:hasRow [ tmp:hasCell [ tmp:location "col1";
                                                tmp:content ?z];
                                  tmp:hasCell [ tmp:location "col2";
                                                tmp:content ?a]
                                ]
        }.
```

The **tmp:hasTable** statement defines the table. It says there is a table at **tmp:table1**, which have two column, the first one's called col1 with header see_also, the second one's called col2 with header name.

Then the second statement fills in the data by saying that fills col1 with data found in **rdfs:seeAlso** and col with data found in **foaf:name**.

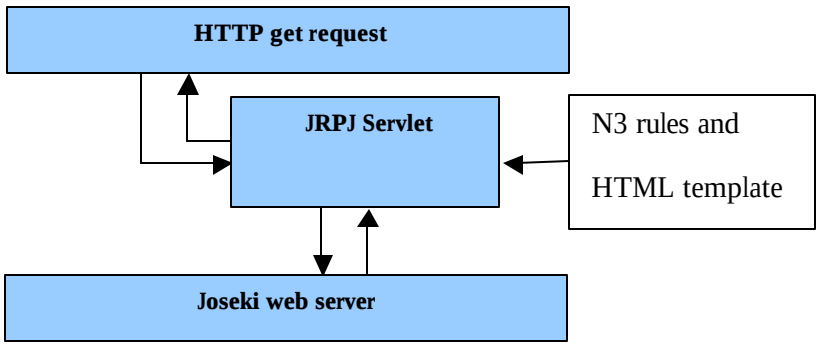
5.2 Future

At this point, the core system has been done and a few extension could be possible to enhance the system.

5.2.1 Joseki connection

The current implementation of the system allows the source of the data fetched from RDF represented in RDF/XML, N3, or any Jena model. One of the possible extension is connection to Joseki web server.

Joseki is a server for publishing RDF models on the web. Models have URLs and they can be accessed by query using HTTP GET. The system therefore could be:



5.2.2 Extends the Template Language

I have mentioned before that the system could handle XML generation but it would be better to extend the template language to make it more user friendly on XML.

5.2.3 Database connection

Another possible extension is the connection with Jena's database model. It would be very nice if the system could acquire user specified data from the database and return HTML.

This should not be too difficult to implement by using Jena's Database Model. The current system does not include database support is because time was not enough for doing all the testing.

5.3 Comparison to RDFT

As RDFT is very similar to the JRPJ HTML translate system (JHTML), now should be a good point to make comparison between the two systems.

There are few things that the same:

- Translate RDF into XML
  Both systems are targeted to transform RDF to XML
- Act as the RDF graph level
  Both systems will apply the rules on the RDF graph level, not the presenting level. This is due to RDF's is for storing the semantic so it is more sensible to act on the RDF graph level.

- Use pattern matching to select data
  Both systems use pattern matching to select data from the source graph. In our JHTML system, the subject in the forward chaining N3 rules will pattern match with the source to generate the object, which will be used to generate the resulting HTML. In RDFT, NodePattern has been used to do the pattern matching bit.

Before comparing the different between the two systems, it would be a good idea to review how the two different translation language works. For example, if we have the following very simple FOAF data:

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/">

        <foaf:Person rdf:about="http://www.chunleung.org/">
                <foaf:name>Johnny Leung</foaf:name>
        </foaf:Person>
</rdf:RDF>
```

In the RDFT system, we would use the following stylesheet to generate the HTML:

```
<?xml version="1.0"?>
<rt:stylesheet xmlns:rt="http://purl.org/vocab/2003/rdft/">
        <rt:macro-set rt:prefix="foaf">
                <rt:macro
                        rt:name="Person"
                        rt:value="resource('http://xmlns.com/foaf/0.1/Person')" />
        </rt:macro-set>
        <rt:macro-set rt:prefix="rdf">
                <rt:macro
                        rt:name="type"
                        rt:value="resource(
                                'http://www.w3.org/1999/02/22-rdf-syntax-ns#type')/*" />
        </rt:macro-set>

        <rt:macro-set rt:prefix="match">
                <rt:macro
                        rt:name="AnyPerson"
                        rt:value="resource()[rdf:type = foaf:Person]" />
        </rt:macro-set>

<rt:root-template>
        <rt:apply-templates rt:select="/resource()[rdf:type = foaf:Person]" />
</rt:root-template>

<rt:template rt:pattern="match:AnyPerson">
        <rt:value-of
                rt:select="node()/resource('http://xmlns.com/foaf/0.1/name')/literal()" />
</rt:template>
</rt:stylesheet>
```

In the JRPJ HTML translation system, we would use the following N3 rules:

```
{<http://www.chunleung.org/> foaf:name ?y} =>
        { _:body tmp:variable [ tmp:content ?y; tmp:location tmp:name]}.
```

and the following HTML template to generate the HTML.

```
<html>
        <Body>
                <?JRPJ :name/>
        </Body>
</html>
```

The different between the two systems would be:

- Language style
  The RDFT is based on XSLT and the JHTML translation system uses N3 for rules. As JHTML use N3 rules to do the pattern matching, it is more expressive as we use variable and could avoid infinite loop graph. On the other have, the RDFT's style make table and multi-valued proprieties much easier by using the XSLT's <for-each> operator.
- Template file
  The RDTS stylesheet combine the rules and the template into one file while JHTML has separate files for rules and template. Therefore, as mentioned before, the JHTML's template file make it simpler for web developer to deploy the web pages for handling RDF data as the do not necessary know much about semantic web. But the learning curve for RDFT is very short for people already familiar with XSLT.

6 Summary

The prototype demonstrated that it is possible to use N3 rules for translating RDF into HTML and using Tomcat server to serve it over the internet. The system is very simple at the moment and the mentioned extensions would make it a complete application and suitable to use in the real world.

7 Reference

[0] Resource Description Framework (RDF) / W3C Semantic Web Activity
http://www.w3.org/RDF/

[1] The Semantic Web - slide "The Current Web"
http://www.w3.org/2002/Talks/04-sweb/slide4-0.html

[2] The Semantic Web - slide "The Semantic Web - A Simple Extension to the Current Web"
http://www.w3.org/2002/Talks/04-sweb/slide6-1.html

[3] Semantic Web Points
http://www.w3.org/2001/sw/EO/points

[4] Introduction to Semantic Web Technologies: Standard Syntax – RDF
http://www.hpl.hp.com/semweb/sw-technology.htm#Standard Syntax – RDF

[5] Introduction to Semantic Web Technologies
http://www.hpl.hp.com/semweb/sw-technology.htm

[6] Semantic Web Activity Statement
http://www.w3.org/2001/sw/Activity

[7] Primer - Getting into the semantic web and RDF using N3
http://www.w3.org/2000/10/swap/Primer

[8] Knowledge-Based Systems
http://www.cs.uct.ac.za/courses/CS300W/DP/slides-2003/chapter%2017-part1.ppt

[9] RDF Template Language 1.0
http://www.semanticplanet.com/2003/08/rdft/spec-20030904

[10] Notation 3 -- Ideas about Web architecture
http://www.w3.org/DesignIssues/Notation3.html

Appendix A – The FOAF data and N3 rules used in demo

FOAFtemplate.html – the HTML template file

```
==========================================================
<HTML>
        <Head>
                <title></title>
        </Head>

        <Body>
                Name: <?JRPJ :name /><br>
                Full Name: <?JRPJ :firstName /> <?JRPJ :surname /><br>
                Weblog: <?JRPJ :weblog /><br>
                mbox: <?JRPJ :mbox_sha1sum /><br>
                homepage: <?JRPJ :homepage /><br>
                <b>Image:</b><br>
                <?JRPJ :depiction /><br>
                wphp: <?JRPJ :workplaceHomepage /><br>
                schhp: <?JRPJ :schoolHomepage /> ><br>
                <?JRPJ :nick>
                        Multilline -- Nick name: <?JRPJ :nick /><br>
                </JRPJ><br>
                <?JRPJ :table1 /><br>
        </Body>
</HTML>
```

Johnny.rdf – the RDF/XML file holding FOAF data about Johnny

```
==========================================================
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/">

<foaf:Person rdf:about="http://www.chunleung.org/">
 <foaf:name>Johnny Leung</foaf:name>
 <foaf:firstName>Chun</foaf:firstName>
 <foaf:surname>Leung</foaf:surname>
 <foaf:nick>look_yau</foaf:nick>
 <foaf:nick>Johnnnnnny</foaf:nick>
 <foaf:nick>@_@</foaf:nick>
 <foaf:nick>The fat John</foaf:nick>
 <foaf:weblog rdf:resource="http://SWEB-PRJ-7" />
 <foaf:mbox_sha1sum>0294fa59419cd2a52c0c88b8dae19d765521998b</foaf:mbox_sha1sum>
 <foaf:homepage rdf:resource="http://SWEB-PRJ-7/index.html" />
 <foaf:depiction rdf:resource="http://SWEB-PRJ-7/cat.jpg" />
 <foaf:workplaceHomepage rdf:resource="http://www.hp.com/" />
 <foaf:schoolHomepage rdf:resource="http://www.doc.ic.ac.uk/" />
```

```
 <foaf:knows>
  <foaf:Person>
    <foaf:name>William Kong</foaf:name>
    <rdfs:seeAlso rdf:resource="http://www.doc.ic.ac.uk/~whk00/about.xrdf"/>
  </foaf:Person>
 </foaf:knows>

 <foaf:knows>
  <foaf:Person>
    <foaf:name>Jing Zhang</foaf:name>
    <rdfs:seeAlso rdf:resource="http://www.doc.ic.ac.uk/~jz00/" />
  </foaf:Person>
 </foaf:knows>

 <foaf:knows>
  <foaf:Person>
    <foaf:name>Jon Hayman</foaf:name>
    <rdfs:seeAlso rdf:resource="http://www.doc.ic.ac.uk/~jmh00/" />
  </foaf:Person>
 </foaf:knows>

 <foaf:knows>
  <foaf:Person>
    <foaf:name>Mei Leung</foaf:name>
    <rdfs:seeAlso rdf:resource="http://www.doc.ic.ac.uk/~cyl00/mei/" />
  </foaf:Person>
 </foaf:knows>

</foaf:Person>
</rdf:RDF>
```

FOAF.n3 – the N3 rules

```
==================================================
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#>.
@prefix airport: <http://www.daml.org/2001/10/html/airport-ont#>.
@prefix pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

@prefix : <http://www.hpl.hp.com/2003/jrpj/htmlGeneratorTesting#>.
@prefix tmp: <http://www.hpl.hp.com/2003/jrpj/htmlTemplate#>.

# tmp:head indicate information goes into <head> section
# tmp:body indicate information goes into <body> section
 _:doc rdf:type tmp:Template;
          tmp:head _:head;
          tmp:body _:body.

###head###
# tmp:title indicate information goes into <title> tag
_:head tmp:title "Testing on HTML generator".

# tmp:link indicate information goes into <link> tag
_:head tmp:link
'rel="stylesheet" href="http://rdfweb.org/ssi/main.css" type="text/css"'.

# tmp:meta indicate information goes into <meta> tag
_:head tmp:meta "some meta data".
```

```
###body###
# tmp:variable  will replace the string of tmp:content (i.e. ?<br>?)
# at tmp:location tmp:line1 with given string
_:body tmp:variable [ tmp:content "<br>"; tmp:location tmp:line1].

#full name
{<http://www.chunleung.org/> foaf:name ?y} =>
        { _:body tmp:variable [ tmp:content ?y; tmp:location tmp:name]}.

#first name
{<http://www.chunleung.org/> foaf:firstName ?y} =>
        { _:body tmp:variable [ tmp:content ?y;
                                tmp:location tmp:firstName;
                                tmp:tag "font";
                                tmp:value "size=12";
                                tmp:value "color='yellow'"]
        }.

#surename
{<http://www.chunleung.org/> foaf:surname ?y} =>
        { _:body tmp:variable [ tmp:content ?y;
                                tmp:location tmp:surname;
                                tmp:tag "font";
                                tmp:value "size=12";
                                tmp:value "color='red'"]
        }.

# schoolhomepage
{<http://www.chunleung.org/> foaf:schoolHomepage ?y} =>
        { _:body tmp:variable [ tmp:location tmp:schoolHomepage;
                                tmp:content "homepage";
                                tmp:a ?y]
        }.

#weblog
{<http://www.chunleung.org/> foaf:weblog ?y} =>
        { _:body tmp:variable [ tmp:location tmp:weblog;
                                tmp:content "weblog";
                                tmp:a ?y]
        }.

#homepage
{<http://www.chunleung.org/> foaf:homepage ?y} =>
        { _:body tmp:variable [ tmp:location tmp:homepage;
                               tmp:content "weblog";
                               tmp:a ?y]
        }.

#workplaceHomepage
{<http://www.chunleung.org/> foaf:workplaceHomepage ?y} =>
        { _:body tmp:variable [ tmp:location tmp:workplaceHomepage;
                                tmp:content "weblog";
                        tmp:a ?y]
        }.

#mbox_sha1sum
{<http://www.chunleung.org/> foaf:mbox_sha1sum ?y} =>
        { _:body tmp:variable [ tmp:location tmp:mbox_sha1sum;
                                tmp:content ?y]
        }.
```

```
#depiction
{<http://www.chunleung.org/> foaf:depiction ?y} =>
        { _:body tmp:variable [ tmp:location tmp:depiction;
                                tmp:image ?y]
        }.

# for multivalued tag, use template:list to print it out
{<http://www.chunleung.org/> foaf:nick ?y}=>
        {_:body tmp:list [ tmp:location tmp:nick;
                           tmp:content ?y]
        }.

#tables
_:doc tmp:hasTable [ tmp:name "table1";
                     tmp:location tmp:table1;
                     tmp:headers ( [tmp:name "col1";
                                    tmp:tag "a";
                                    tmp:content "see_also"]
                                    [tmp:name "col2";
                                    tmp:tag "a";
                                    tmp:content "name"] )
                    ].

{<http://www.chunleung.org/> foaf:knows ?y. ?y rdfs:seeAlso ?z. ?y foaf:name ?a}=>
        { tmp:table1 tmp:hasRow [ tmp:hasCell [ tmp:location "col1";
                                                tmp:content ?z];
                                  tmp:hasCell [ tmp:location "col2";
                                                tmp:content ?a]
                                ]
        }.
```

The HTML Output

==========

```
<html>
        <head>
                <title>Testing on HTML generator</title>
                <meta some meta data>
                <link rel="stylesheet"
                        href="http://rdfweb.org/ssi/main.c ss" type="text/css">
        </head>
        <body>
                Name: Johnny Leung<br>
                Full Name: <font size=12  color='yellow' >Chun</font>
                                <font size=12  color='red' >Leung</font><br>
                Weblog: <a href='http://SWEB-PRJ-7' >weblog</a><br>
                mbox: 0294fa59419cd2a52c0c88b8dae19d765521998b<br>
                homepage: <a href='http://SWEB-PRJ-7/index.html' >weblog</a><br>
                <b>Image:</b><br>
                <img src='http://SWEB-PRJ-7/cat.jpg'><br>
                wphp: <a href='http://www.hp.com/' >weblog</a><br>
                schhp: <a href='http://www.doc.ic.ac.uk/' >homepage</a> ><br>
                 Multilline -- Nick name: The fat John <br>
                 Multilline -- Nick name: @_@ <br>
                 Multilline -- Nick name: Johnnnnnny <br>
                 Multilline -- Nick name: look_yau <br>
                <br>Testing table...<br>
                <table border=1>
                        <tr>
                                <th><a>see_also</a></th>
                                <th><a>name</a></th>
                        </tr>
```

```
                    <tr>
                        <td>http://www.doc.ic.ac.uk/~cyl00/mei/</td>
                        <td>Mei Leung</td>
                    </tr>
                    <tr>
                        <td>http://www.doc.ic.ac.uk/~jmh00/</td>
                        <td>Jon Hayman</td>
                    </tr>
                    <tr>
                        <td>http://www.doc.ic.ac.uk/~jz00/</td>
                        <td>Jing Zhang</td>
                    </tr>
                    <tr>
                        <td>http://www.doc.ic.ac.uk/~whk00/about.xrdf</td>
                        <td>William Kong</td>
                    </tr>
                </table>
                <br>
        </body>
</html>
```