CARA – A RDF Parser

Programming Project

Submitted to

Prof. Dr. Roland Schwänzl
Dept. of Computer Science & Mathematics,
University of Osnabrück,
Albrechtstr. 28, 49069 Osnabrück,
Germany

Submitted by

Krishnamoorthi. BM.

CONTENTS

1.0 CARA – An Introduction

1.1 AUDIENCE

If you want to know how you can use CARA for parsing RDF/XML to get the triples, then this documentation will be useful. A basic understanding of semantic web and web technologies is helpful before reading this, so you may want to start with those first if you don't have any background in them. It is more focused for the programmers who would like to implement this parser in their system.

1.2 About RDF

RDF --the Resource Description Framework-- is a universal format for data on the Web. Using a simple relational model, it allows structured and semi-structured data to be mixed, exported and shared across different applications. RDF data describe all sorts of things, and where XML schemas just describe documents, RDF and OWL schemas ("ontologies") talk about the actual things. This gives greater **re-use**. Where XML provides interoperability within one application (e.g. bank statements) using a given schema, RDF provides interoperability **across** applications (e.g. import your bank statements into your calendar).

RDF started as framework for metadata; providing interoperability between applications that exchange machine-understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources and as such provides the basic building blocks for supporting the Semantic Web. RDF metadata can be used in a variety of application areas; for example: in *resource discovery* to provide better search engine capabilities; in *cataloging* for describing the content and content relationships available at a particular Web site, page, or digital library; by *intelligent software agents*

to facilitate knowledge sharing and exchange; in *content rating*; in describing *collections* of pages that represent a single logical "document"; for describing *intellectual property rights* of Web pages, and in many others. RDF with *digital signatures* will be key to building the "Web of Trust" for *electronic commerce*, *collaboration*, and other applications.

1.3 CARA for RDF

CARA is a RDF parser. It parses the RDF/XML and generates triples. It follows the RDF Grammar given by W3C (World Wide Web Consortium). CARA was developed by Department of Mathematik / Informatik, University of Osnabrueck and was written in Perl. This was the part of CASHMERE project(http://www.iwi-iuk.org/cashmere/) headed by **apl.Prof.Dr. Roland Schwaenzl** and was funded by BMBF(Bundes ministerium für Bildung und Forschung)

1.4 History of CARA – In Brief

CARA was developed early by *Stefan Kokkelink*. This parser worked fine with the older version of RDF/XML (referenced in section 5). RDF/XML was recommended by W3C in 2004 and by this period there were lot of changes in the new version of RDF/XML. The changes in grammar are stated in section 4.1. CARA could not cope with these changes and some systems like **XHarvest** is more dependent on this parser. So, we were in the need for upgrading it to the new RDF/XML Grammar (Listed in following sections). It was upgraded by *Krishnamoorthi.BM* around April 2004 and is now available online in http://zoe.mathematik.uos.de/RDF/parser.html

1.5 RDF/XML and Perl

There seems to be a natural fit between Perl, a language known for its parsing and patter-matching capability, and RDF. Perl is free software which runs in many platforms. This program is free software; you can redistribute it and/or modify it under the same terms as Perl 5 itself. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the Perl 5 License schemes for more details.

1.6 Where the changes were made?

This section tells you more about the changes in the source code (with respect to the previous version). The variables and subroutines which were added or modified are discussed in detail.

The RDF Namespace: The RDF namespace is stored in a variable which is globally available in the parser module. [Note a variable is denoted with $string in Perl]

```
my $rdfns="http://www.w3.org/1999/02/22-rdf-syntax-ns#";
```

RDF Keywords: All the valid keywords of RDF is stored in a array (data-type in Perl denoted by @ symbol).

```
my @words=(     $rdfns."Description",     $rdfns."Bag",
                $rdfns."Seq",             $rdfns."Alt",
                $rdfns."ID",              $rdfns."about",
                $rdfns."type",            $rdfns."Statement",
                $rdfns."subject",         $rdfns."object",
                $rdfns."predicate",       $rdfns."Property",
                $rdfns."parseType",       $rdfns."resource",
                $rdfns."li",              $rdfns."value",
                $rdfns."nodeID",          $rdfns."datatype",
                $rdfns."List",            $rdfns."first",
                $rdfns."rest",            $rdfns."nil",
                $rdfns:"XMLLiteral"
);
```

The newly added keywords are $rdfns."nodeID", $rdfns."first", $rdfns."rest", $rdfns."datatype", $rdfns."nil" and $rdfns."XMLLiteral".

XML Base URI: If the "xml:base" is defined in the node as attribute then the value is stored in $baseuri variable.

```
my $baseuri;        # xml:base is stored here.   Global declaration.
```

XML Language definition: If xml:lang is defined in the namespace node then the value is stored in $xml_lang variable. This variable is global because when the xml:lang is defined in the root node, it will be affected to all the child nodes unless a empty string is given to avoid it. i.e., xml:lang="".

```
my $xml_lang;        # xml:lang is stored here.   Global declaration.
```

Child node language definition: Each child node in RDF can also be defined with xml:lang. If it is defined and if the xml:lang is defined in the root node then the former overrules the later. So this child node xml:lang definition is stored in a local variable $desclang.

```
my $desclang;
```

Preserving the Uniqueness of the IDs in RDF: To prevent the duplicates of rdf:ID the value of each rdf:ID is stored in a Hash so that we can find whether the ID has already declared.[data-type Hash is denoted with %]

```
my %IDs; # All IDs are stored here with URI and used for checking uniqueness.
```

rdf:li counter: Each rdf:li is represented with rdf:_n (n is a positive integer) in the output. The counter for this is represented as %li_count.

```
my %li_count; # it counter which counts number of li and represented as rdf:_n
```

RDF node validation: The RDF nodes are validated by checking whether it matches with the RDF keyword and whether it follows the RDF Grammar for a node. It is done by the subroutine "correctNode". The subroutine returns "1" if the node is properly defined and returns "undefined" when it is not properly defined. The old RDF terms are also checked here. A sample code snippet is shown below.

```
sub correctNode {
        my $node=shift;
        my $schema;
        my $name;
        if($node->getNodeType == ELEMENT_NODE){
                $schema=getElementSchema($node);
                return 1 unless $schema eq $rdfns;
.
.
.
        if($name eq $rdfns."aboutEach")
        {
        carp "Error Nr 04007: rdf:aboutEach element tag is not allowed here ",
 getPath($node);
         exit;
        }
.
.
.
}
```

Attribute rdf:about: The values of rdf:about has a special property. The value of this attribute is kept unescaped. If the value of the attribute is escaped in prior then it is kept undisturbed. The checking is done in the subroutine "urifromabout". A part of the code is shown below.

```
sub urifromabout {
        my $string=shift;
        return undef unless defined($string);
        my $result=URI->new_abs($string,$baseuri)->as_string;
        if($result =~ /$string$/){
                return $result;
        }
        else{
.. .. ..
```

In the above source the variable $result holds the URI and later it was checked for the escaped characters.

Attribute rdf:ID: The value of this attribute should follow NCName of XML Infoset. This is checked inside the subroutine "processDescriptionNode".

```
sub processDescriptionNode
{
undef $desclang;
my $xmlbase;
...
...
if(!(uri_escape($ID) =~ /^$XML::RegExp::NCName$/))
{
carp "Error Nr 06001: From XML rdf:ID $ID is not a legal[XML-NS] ...
}
...
```

Here the `XML::RegExp::NCName` is the module which checks the NCName validity.

Attribute rdf:datatype: The value of this attribute is stored in $child_datatype. In the subroutine "getCompleteAttributes" finds the datatype of the node. If it is not defined then the variable is set as undefined.

```
sub getCompleteAttributes {
        my $elementNode=shift;
        my $hash={};
        my $xmllang;
        my $child_datatype;
        foreach(@{$elementNode->getAttributeNodes}){
                if($_->getName eq "xml:lang"){
                        $xmllang=$_->getValue;
                        next;
                }elsif($_->getName eq "rdf:datatype"){
                        $child_datatype=$_->getValue;
                        next;
                }
                $hash->{getAttributeName($_)}=$_->getValue;
        }

        return ($hash,$xmllang,$child_datatype);
}
```

Attribute xml:lang : If this attribute is defined then the Literal is appended with a character "@" and the language code. For example if xml:lang="de" and the Literal is

"Schwänzl" then the appended string is "Schwänzl"@de . A code snippet from the "processPropAttr" subroutine is shown below

```
sub processPropAttr {
my ($source,$attr,$xmllang)=@_;
if (defined($xmllang) && ( $xmllang eq "")) { undef $xmllang; }
. . .
      if(defined($xmllang))
      {
            if($xmllang eq ""){
                  $value = "\'" . $value . "\'";
            }else{
                  $value = "\'" . $value . "\'@" . $xmllang;
            }
            }elsif(defined($desclang))
            {
                  $value = "\'" . $value . "\'@" . $desclang;
            }
            else
            {
                  $value = "'" . $value . "'";
            }
      ...
```

$value in the above snippet contains the value of the Literal. This value is returned to the caller.

The new ParseType – Collection, nodeID, XMLLiteral: This is checked in the subroutine "processPropertyElt". If parseType="Collection" is found as a attribute then the variable $parseCollection is set to "Collection".

```
sub processPropertyElt {
        my ($source,$node,$parent)=@_;
...
      elsif ($attr->{$rdfns."parseType"} eq "Collection"){
            $parseCollection="Collection";
      }
...
```

Later in the subroutine the Grammar for Collection parseType is applied. This routine also handles the rdf:nodeID, rdf:XMLLiteral

2.0 CARA Supports

There are some RDF APIs available. For example Jena RDF API written in Java, Redfoot RDF API written in Python, Redland RDF Application written in C. Our CARA is written in PERL and would be useful for Perl application programmers. CARA also passed in all the RDF Test cases. It supports

- Embedded RDF in HTML or XML
- Full Support of XML Schema Data Types (XSD)
- Full Unicode

2.1 Features in CARA

- Outputs
- Triples
- **JGraph*** object
- Graphical output
- Customized graphs
    - With different colors in Nodes, edges, background.
    - With gif and postscript file formats
    - With options to change font size
    - Orientations like Top to down or Left to Right
- User friendly error messages
- Avoids duplicate triples

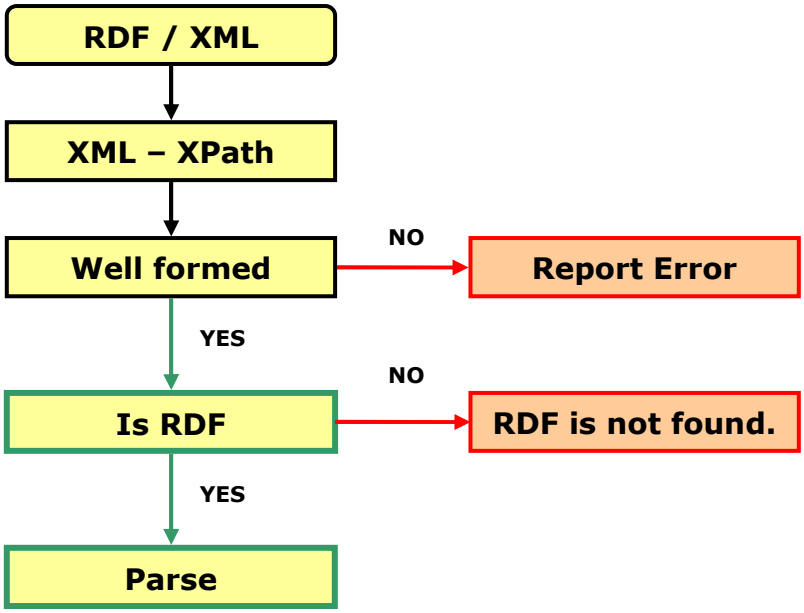
NOTE:

* JGraph is a Perl object created in runtime of CARA. This object stores all the nodes and edges of graph. This can be given in as input to Graph modules or triple modules for output.

2.2 Functionality

2.2.1 Overview

The block diagram below shows the functionality of CARA.

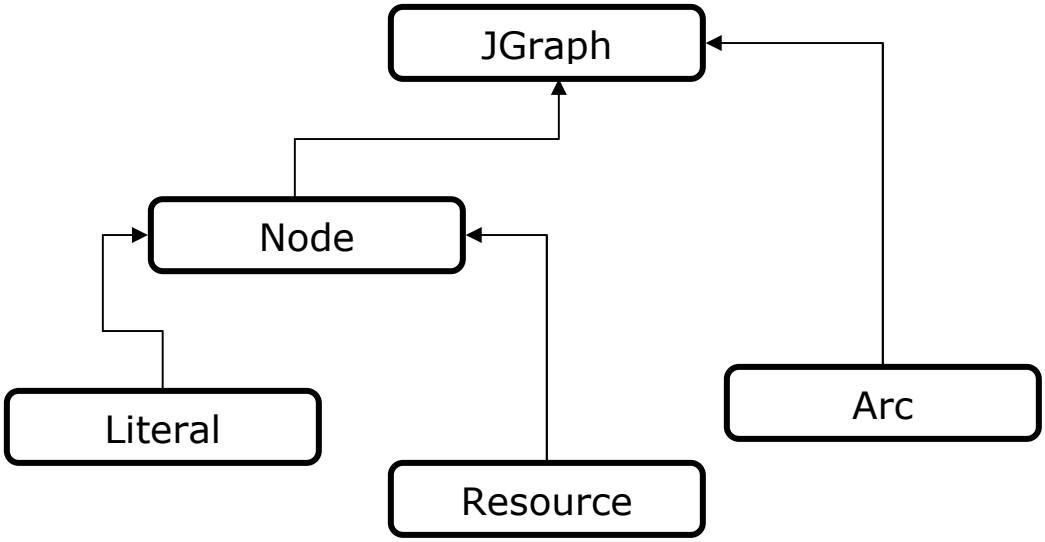


The input stream of RDF/XML is given in to XML XPath parser. This is the module available in CPAN (http://www.cpan.org). This Parser validates the given input if it is well formed. CARA takes the output of XPath if it is well formed and checks for the valid RDF entry. If RDF is embedded in the given input it starts the parse routine.

2.2.2 CARA Parser functionality

The parse subroutine of CARA takes in the output of XML XPath parser and checks in whether the given Parent node is of description node

(rdf:Description) or Container node (rdf:Bag, rdf:Seq or rdf:Alt) or Typed node (user defined). It follows the grammar of each separately as stated in section 4.2. For each triple the subject and object of the triple are stored in JGraph object as nodes. The node can be either a resource or literal. The predicate is stored in as arc. At last the Graph has the collection of nodes and arcs. The sample diagram below



The parser returns a JGraph object which can be further processed to take triples or to visualize in Graph using Graphviz (ref. In Sec 5)

2.3 A Sample RDF/XML with new syntax term

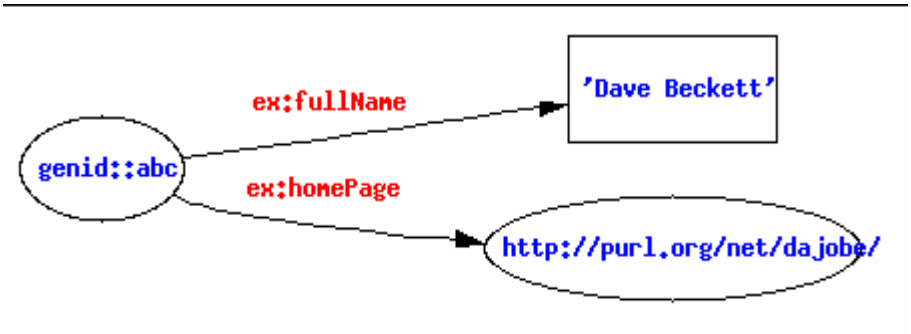
This sections deals with a sample RDF/XML of new syntax and how it differs with the old one, rather than how it is processed (Of course you can learn it in a RDF tutorial or in w3c syntax specification). A sample RDF containing rdf:nodeID as attribute. Please note, rdf:nodeID can be a attribute of a node element and cannot occur with either rdf:ID or rdf:about. The value of rdf:nodeID should also follow the rule of NCName of XML.

For the given RDF:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:ex="http://example.org/stuff/1.0/">
<rdf:Description rdf:nodeID="abc" ex:fullName="Dave Beckett">
    <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
  </rdf:Description>
</rdf:RDF>
```

The triples are

Subject	Predicate	Object
genid::abc	http://example.org/stuff/1.0/fullName	'Dave Beckett'
genid::abc	http://example.org/stuff/1.0/homePage	http://purl.org/net/dajobe/



3.0 DOCUMENTATION

CARA software was neatly documented and is available online. You can see the documentation in the web link http://zoe.mathematik.uos.de/RDF/carahelp

The methods in the parser are documented separately and it is given below.

3.1 Method Documentation

parse
Gets in the RDF/XML string as input and will pass it to the XML XPath parser as parameter and process it with RDF grammar and stores the triple in the Graph format and returns the $graph

findRDF
It takes in the XML XPath parsers' output as input and searches for the valid RDF declaration. If any declaration found then it will be stored in the @RDF array
beautyRDF
takes in the XML XPath parsers' output as input and reads every element, attributes, values, comments etc., and adds HTML color syntax before each to make it more clear for visualization. The output is stored in $colorRDF.
urifromID
It is to get the URI from the ID given in rdf:ID.
urifromabout
It is to get the URI from the rdf:about
getPath
Takes in node as input and returns the path where the node resides. for exampe rdf:RDF/rdf:Description/
correctNode
It is used to find whether the Node element is correct or not. For example rdf:RDF cannot appear inside RDF tag. Some rdf defined terms cannot be a node element. It returns boolean 1 if it is a correct node.
processRDFNode
Gets in RDF node element as parameter and sends each child for processing using processObject
processObject
Gets in node element as input. It calls processDescriptionNode when it finds $rdfns.Description, and calls processContainerNode when it finds $rdfns.Bag,Alt,Seq and calls TypedNode when it finds TypedNode.
getCompleteAttributes
This module gets in a node element as input. It collects all the attributes and values in a Hash and returns it. If it has xml:lang attribute then the value is stored in the $xmllang and when the datatype is defined then it is stored in $child_datatype variable
getAttributeSchema

returns the expanded namespaces for the given node. If not defined error 03000A is thrown.
getAttributeName
Returns the full attribute name with namespace expanded to it.
getElementSchema
returns the expanded namespace for the given element. if not found Error 03000b is thrown
getElementName
returns the element name
processPropAttr
gets in Source, Attribute, XMLLang (when defined) as input. creates triple for it.
processDescriptionNode
This routine is called when the Parser finds the node with Description tag. This is also used for user defined node because other than the tag everything is similar for the Typednodes. In this routine $desclang is to store the xml:lang defined as the attribute inside description tag. $ID is to store rdf:ID value, $about is to store rdf:about, $xmllang is to store the xml:lang for the particular node. $nodeID is to store the nodeID value. Error series thrown is 6
processPropertyElt
Gets in the source, node and parent as input. Process each element with RDF Grammar rules and produces the triples. Error series is 7
processContainerNode
This routine is called with the node is of rdf:Seq or rdf:Bag or rdf:Alt. Follows RDF Grammar rules for container nodes. Error series is 8.
processTypedNode
This routine will fire when userdefined node is found. It calls processDescriptionNode for further processing

3.2 ERROR Handling by CARA

CARA handles all errors which do not follow the RDF/XML Grammar. All errors are listed in a table (available in online documentation). For example if a user gives in rdf:abouts as input instead of rdf:about where rdf is the namespace of http://www.w3.org/1999/02/22-rdf-syntax-ns# then CARA reports a error with a error number 04011. It also gives the path where the error was occurred.

3.3 SAMPLE Error page.

Oops!!! CARA found some errors while parsing

Error Nr 04011: http://www.w3.org/1999/02/22-rdf-syntax-ns#abouts is not defined/rdf:RDF/rdf:Description/@rdf:abouts

Original RDF:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:abouts="http://example.org/item01">
 <ex:size rdf:datatype="http://www.w3.org/2001/XMLSchema#int">123</ex:size>
</rdf:Description>
</rdf:RDF>
```

4.0 Grammar Event Matching Notation

Each event is explained with an example with blue text

Grammar Event Matching Notation.	
Notation	**Meaning**
A == B	Event accessor A matches expression B.
If A equals B then a Boolean true is returned	
A != B	A is not equal to B.
If A does not equals B then a Boolean true is returned	

A | B | ...	The A, B, ... terms are alternatives.
If either A or B is true then true is returned	
A - B	The terms in A excluding all the terms in B.
All elements in A except the elements in B	
*	Zero or more of preceding term.
?	Zero or one of preceding term.
+	One or more of preceding term.
root(acc1 == value1, acc2 == value2, ...)	Match a Root Event with accessors.
start-element(acc1 == value1, acc2 == value2, ...) *children* end-element()	Match a sequence of Element Event with accessors, a possibly empty list of events as element content and an End Element Event.
attribute(acc1 == value1, acc2 == value2, ...)	Match an Attribute Event with accessors.
text()	Match a Text Event.

4.1 OLD and NEW RDF/XML GRAMMAR – Difference

In the old RDF/XML grammar, it was allowed to used syntaxes comprising rdf:aboutEach, rdf:aboutEachPrefix, rdf:bagID. These three were removed in the new RDF/XML grammar specification. It also allowed new syntaxes like rdf:datatype, rdf:nodeID, xml:lang, parsetype="Collection" etc., To know more about the old and new RDF/XML Grammar notations please refer to the links given in section 5.

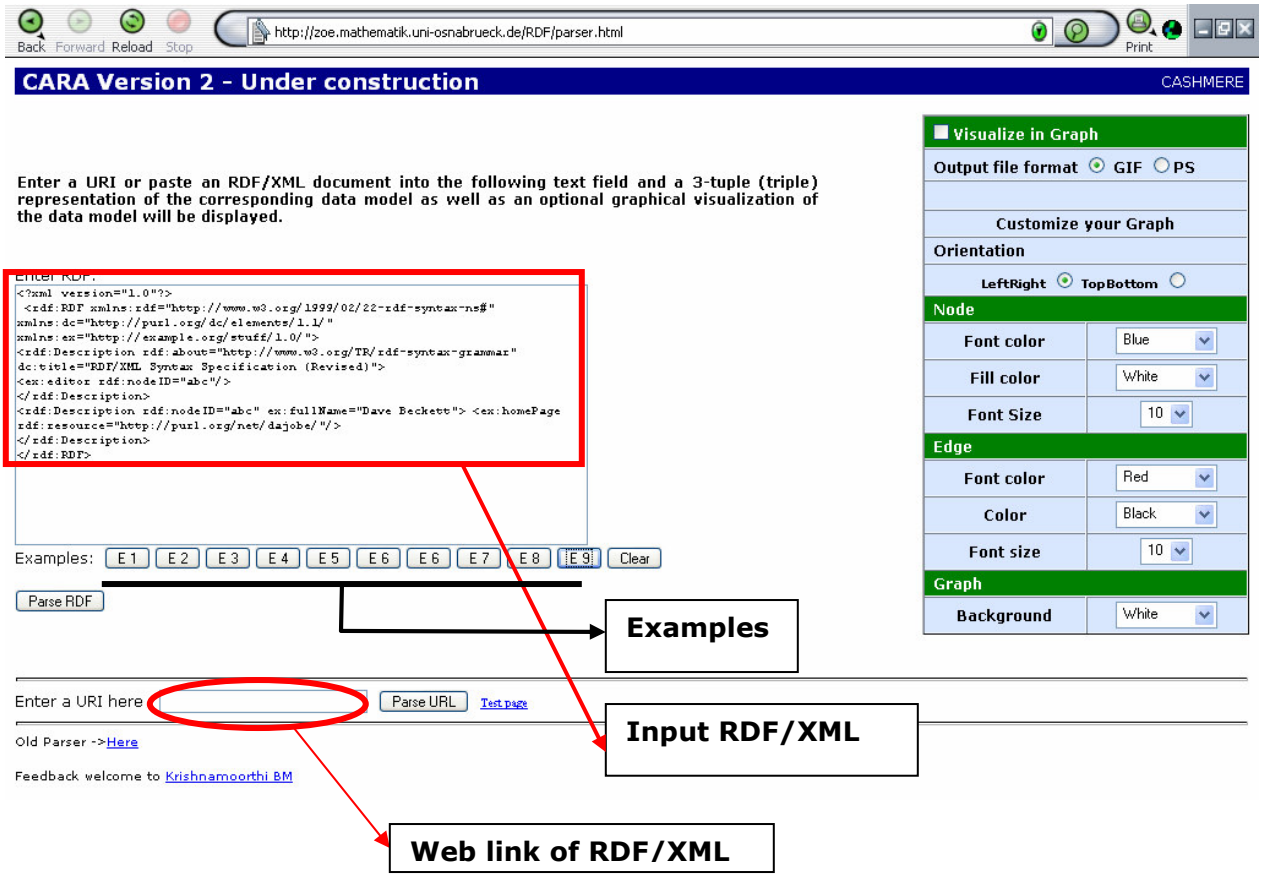
4.2 RDF/XML Grammar.

coreSyntaxTerms	rdf:RDF | rdf:ID | rdf:about | rdf:parseType | rdf:resource | rdf:nodeID | rdf:datatype
syntaxTerms	coreSyntaxTerms | rdf:Description | rdf:li
oldTerms	rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID
nodeElementURIs	anyURI - (coreSyntaxTerms | rdf:li | oldTerms)
propertyElementURIs	anyURI - (coreSyntaxTerms | rdf:Description | oldTerms)
propertyAttributeURIs	anyURI - (coreSyntaxTerms | rdf:Description | rdf:li | oldTerms)
doc	root(document-element == RDF, children == list(RDF))
RDF	start-element(URI == rdf:RDF, attributes == set()) nodeElementList end-element()

nodeElementList	ws* (nodeElement ws*)*
nodeElement	start-element(URI == nodeElementURIs attributes == set((idAttr | nodeIdAttr | aboutAttr)?, propertyAttr*)) propertyEltList end-element()
ws	A text event matching white space defined by [XML] definition *White Space*
propertyEltList	ws* (propertyElt ws*) *
propertyElt	resourcePropertyElt | literalPropertyElt | parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt | parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt | emptyPropertyElt
resourcePropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?)) ws* nodeElement ws* end-element()
literalPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, datatypeAttr?)) text() end-element()
parseTypeLiteralPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseLiteral)) literal end-element()
parseTypeResourcePropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseResource)) propertyEltList end-element()
parseTypeCollectionPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseCollection)) nodeElementList end-element()
parseTypeOtherPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseOther)) propertyEltList end-element()
emptyPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, (resourceAttr | nodeIdAttr)?, propertyAttr*)) end-element()
idAttr	attribute(URI == rdf:ID, string-value == rdf-id)
nodeIdAttr	attribute(URI == rdf:nodeID, string-value == rdf-id)
aboutAttr	attribute(URI == rdf:about, string-value == URI-reference)
propertyAttr	attribute(URI == propertyAttributeURIs, string-value == anyString)
resourceAttr	attribute(URI == rdf:resource, string-value == URI-

	reference)
datatypeAttr	attribute(URI == rdf:datatype, string-value == URI-reference)
parseLiteral	attribute(URI == rdf:parseType, string-value == "Literal")
parseResource	attribute(URI == rdf:parseType, string-value == "Resource")
parseCollection	attribute(URI == rdf:parseType, string-value == "Collection")
parseOther	attribute(URI == rdf:parseType, string-value == anyString - ("Resource" | "Literal" | "Collection"))
URI-reference	An RDF URI reference.
literal	Any XML element content that is allowed according to [XML] definition
rdf-id	An attribute ·string-value· matching any legal [XML-NS] token NCName

4.3 Web Interface of CARA



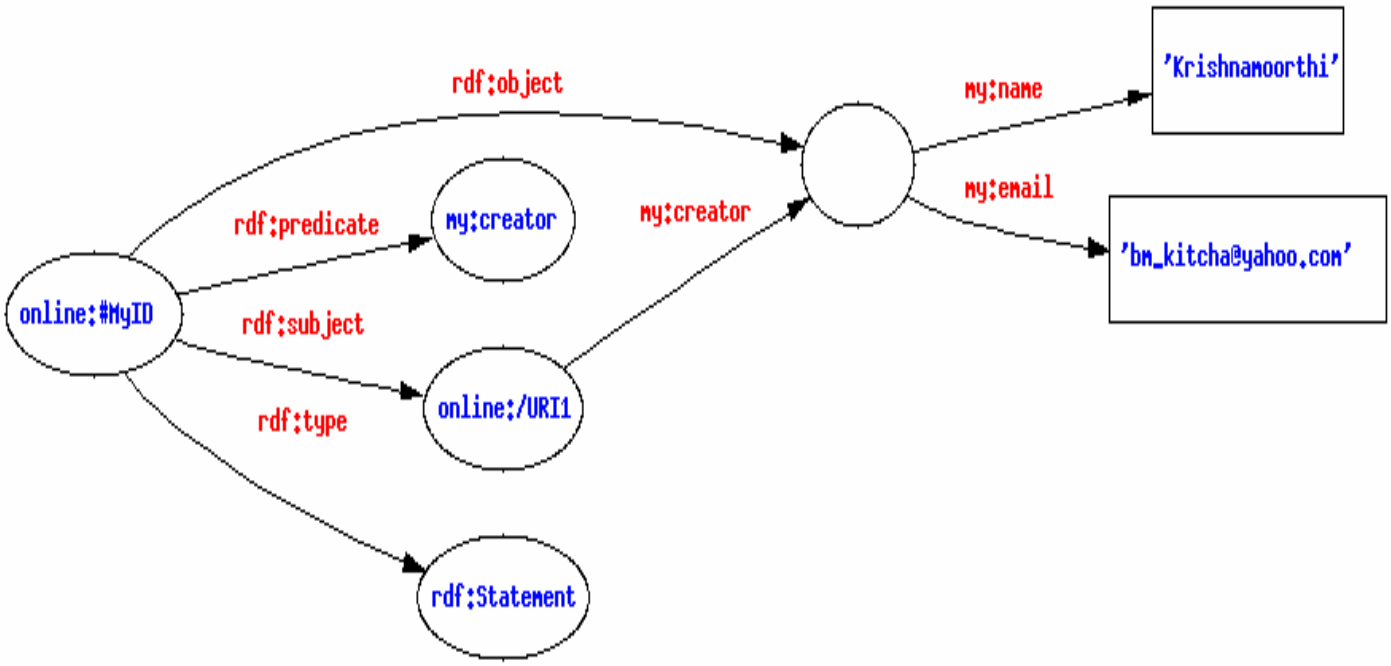
The above figure shows the web interface of CARA. This interface is available online in http://zoe.mathematik.uos.de/RDF/parser.html. It has an advantage of parsing both the text RDF/XML input and to parse the web link of RDF file.

4.4 SAMPLE OUTPUT

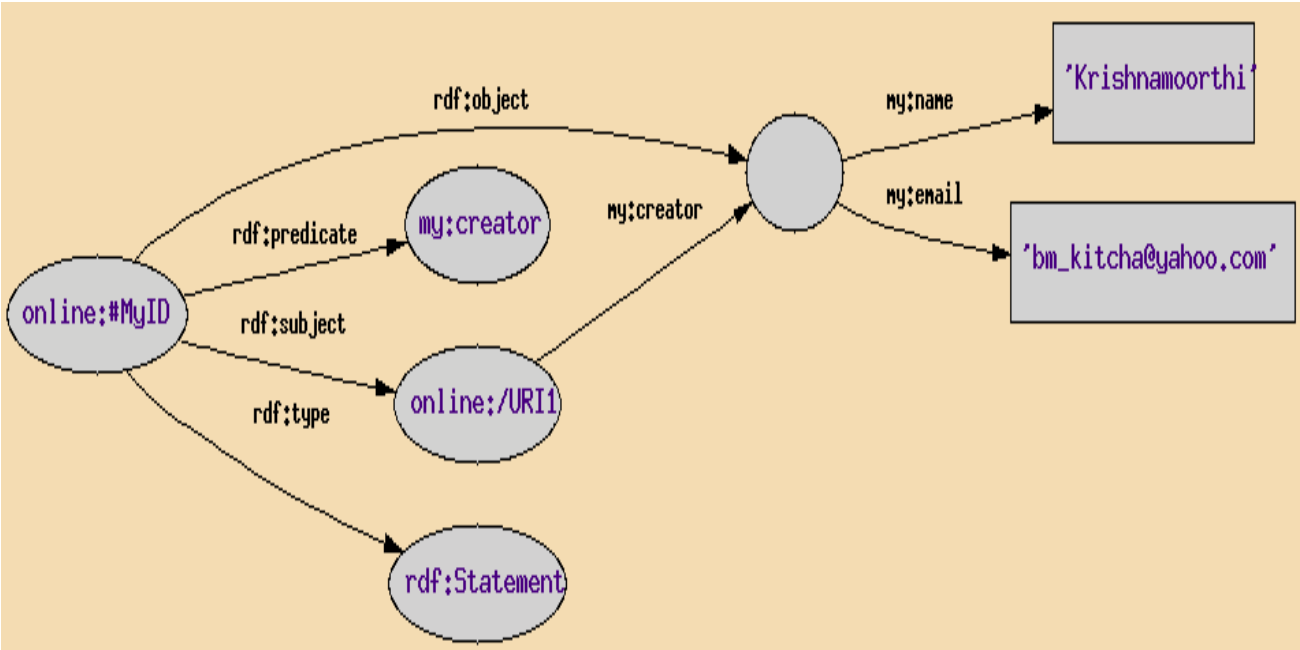
A sample RDF/XML is shown below.

```
<?xml version="1.0" ?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:my="http://mynamespace#">
<Description about="URI1">
        <my:creator rdf:ID="MyID" my:name="Krishnamoorthi"
                my:email="bm_kitcha@yahoo.com"/>
 </Description>
</RDF>
```

When this is given as input to CARA, then it produces the graph as shown below. In this graph the oval shaped nodes denotes the resources, the boxes denotes the literals, the empty node is the blank node.



One can also customize the graph by choosing different colors, different font types etc., A sample for the customized graph is shown below.



The triple output for the above RDF/XML is

Subject	Predicate	Object
online:#MyID	http://www.w3.org/1999/02/22-rdf-syntax-ns#object	genid:1
online:#MyID	http://www.w3.org/1999/02/22-rdf-syntax-ns#subject	online:/URI1
online:/URI1	http://mynamespace#creator	genid:1
genid:1	http://mynamespace#email	'bm_kitcha@yahoo.com'
online:#MyID	http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate	http://mynamespace#creator
genid:1	http://mynamespace#name	'Krishnamoorthi'
online:#MyID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement

5.0 References

CPAN – http://www.cpan.org

Graphviz - http://www.graphviz.org/

OLD RDF/XML Syntax specification:

http://www.w3.org/TR/2001/WD-rdf-syntax-grammar-20010906/#section-Grammar

Online Documentation of CARA:

http://zoe.mathematik.uos.de/RDF/carahelp

PERL – http://www.perl.com

RDF APIs:

http://www.w3.org/RDF/ (Search in for Developer Resources)

RDF/XML Syntax specification:

http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

RDF Test Cases:

http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/

W3C – http://www.w3c.org

XML NCName - http://www.w3.org/TR/REC-xml-names/#NT-NCName