

# An Architecture For RDF Storing And Querying For Email Messages

Hoang Huu Hanh and Nguyen Huu Tinh  
Institute of Software Technology  
Vienna University of Technology  
1040 Vienna, Austria  
{hhhanh, tinh}@ifs.tuwien.ac.at

## Abstract

*Mail messages are regarded as a main resource of personal data in modern life of a person. This architecture is designed for storing email messages of an individual into a RDF repository in ontological way in order to query efficiently in term of semantics.*

## 1. Introduction

Storing and managing personal data such as email message, documents become a challenge in terms of semantics and ontology in nowadays. In trends of supporting for people to store their data and easy to get them back in semantic and intelligent way, a system is designed using the Semantic Web and Java technologies to fulfill the demands for above mentioned things. In this paper we introduce a simple prototype with email messages as the main incoming resource of the system.

## 2. Objectives

The objective of the project and is to develop a personal information system for the management of one's personal data such as documents, messages (email messages, SMS/MMS messages, phone calls, notes...), multimedia in an ontological way. Subsequently, according to the user requests, the system should be possible to retrieve the data semantically.

The first version of prototype effort is to store email messages' heard information into RDF data (In-memory and Persistent Model Storage). Then, these data will be queried by user in simple way in term of non-semantic queries.

## 3. Scenario of Prototype

- a) Store all the messages from the mailboxes from the specific email servers (POP3/IMAP4) into the ontological store (RDF/RDFS), extracting a sample of metadata information about the message(s).
- b) Store some selected messages from new incoming messages the mailbox into the ontological store, extracting a sample of available metadata information about the message.
- c) All the trivial non-semantic queries should be possible.

## 4. Prototype Architecture

### 4.1 Description of the Components

**Interface/Analysis/Event-Handler:** this part of the architecture (see Fig. 1) is comprised from three separated parts in the architecture of the Semantic Web project architecture<sup>†</sup> including Interface, Event Handler and Analysis parts but with the simple functions. This part receive user's request of parsing email messages to RDF data or query information from in-memory RDF data or from database with RDF persistency.

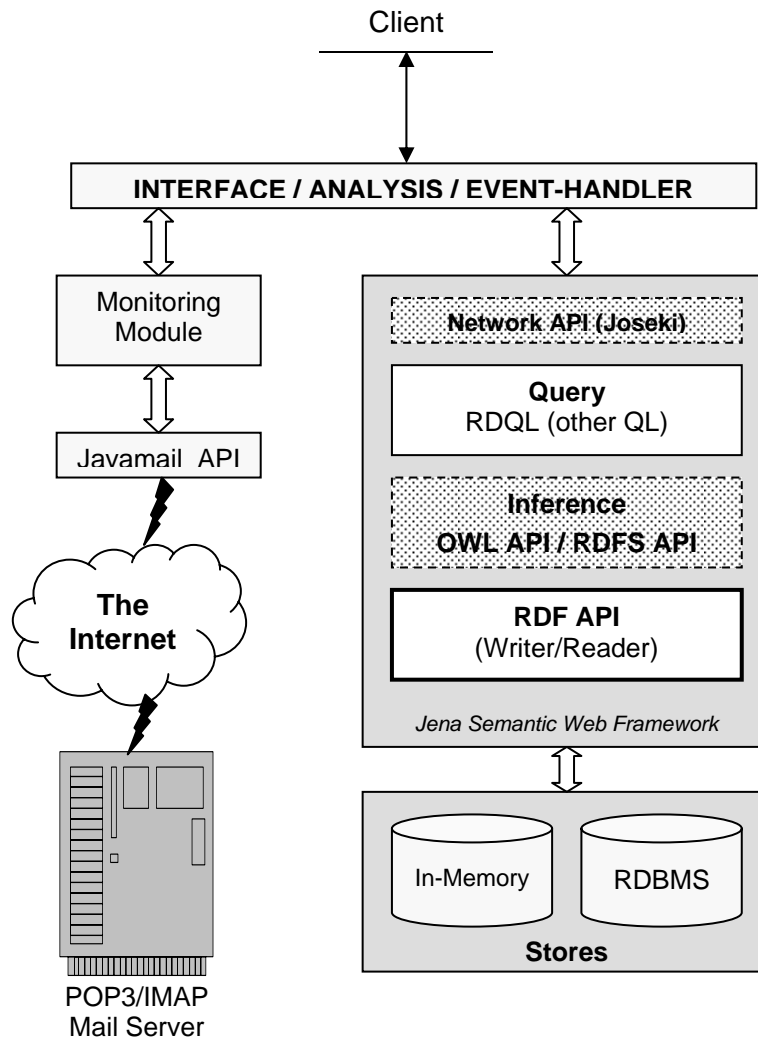


Figure 1: Prototype architecture for parsing email messages

**Monitoring Module:** this module takes in charge of accessing to email servers (POP3 or IMAP4) and parsing email messages, referring to RFC2822 message format and RFC2045 MIME Internet messaging, to RDF data. In addition, this part has been monitoring the servers for new incoming messages and parsing them selectively. This part uses Javamail API of Sun Microsystems to do these such work.

The kernel of this architecture is the Semantic Web framework. In our prototype, we choose Jena, a Semantic Web framework of HP Semantic Web Lab, as the framework for our system. We uses Jena version 2.1, the latest version of it.

<sup>†</sup> The Semantic Web project of Institute of Software Technology, Vienna University of Technology .

**Jena Semantic Framework:** is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference engine and supports relational database persistence.

**RDF API:** is the heart of the Jena architecture. RDF API supports the creation, manipulation and query of RDF graphs. The API also supports several different storage technologies. The Readers and Writers supports for different languages that developers can use to represent RDF graphs such as RDF/XML, n-triples, N3.

**Stores:** Jena contains two implementations of the API: One stores its data in main memory, another stores its data in a relational database (database persistence).

The Jena2 persistence subsystem implements an extension to the Jena Model class that provides persistence for models through use of a back-end database engine. The 'jena/db' module provides an implementation of the Jena model interface but with the ability to store and retrieve RDF statements using a database. It currently supports MySQL, PostgreSQL and Oracle for persistent storage and runs under Linux and WindowsXP using JDBC connections.

**Query:** RDQL is Jena's query language for RDF graphs. Designed to be familiar to many users, RDQL syntax is similar to SQL. RDQL has no built-in inference mechanisms. It relies on all inference being performed within the implementation of the RDF graph it is querying

**Inference:** The Jena2 inference subsystem is designed to allow inference engines to be plugged into Jena. Such engines are used to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. The primary use of this mechanism is to support the use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data and class descriptions. However, the machinery is designed to be quite general and, in particular, it includes a generic rule engine that can be used for many RDF processing or transformation tasks.

**Network API:** Jena supports a set of operations that applications can perform when they have direct access to an RDF database. Jena's network API allows an application to query and update a remote RDF database just as a browser accesses a Web server.

Joseki is a Java client and a server that implements the network API over HTTP. The server can be embedded in another application, run standalone, or run on a Web application server.

## **4.2 Workflow**

There two cases for workflow in this architecture. In the first case, users use the Monitoring module to request for parsing all existing messages or all new messages to RDF model. This module will access a e-mail server to get the messages, then get the information from the messages' headers. Consequently, these information will be stored into RDF model through RDF API in Jena framework. In addition, the system can persist the RDF model to RDBMS such as MySQL.

In the second one, when users make requests about the information in RDF database; they through the Interface, then RDQL statements will be generated and executed inside also RDF API is used. And the interface will generate and format the output results according to users' queries.

### 4.3 Program Demonstration

In this part, we present some code fragments of implemented modules for the prototype architecture. The fragments consist of :

- accessing a mailing server, getting email messages and some headers
- parsing email messages' headers into RDF model, then storing in a RDF file.
- persisting a RDF model to a MySQL database.
- querying information of email messages from RDF model or from a MySQL database using RDQL.

#### 4.3.1 Retrieving headers of an email message

- Fetching a message from the folder on a mailing system:

```
// Open folder INBOX
Folder fldr = store.getFolder("INBOX");
fldr.open(Folder.READ_WRITE);

// Get i-th message
Message m = fldr.getMessage(i);
```

- As per specifications of W3.org message identifiers for email messages would start with **“mid:”**.

```
// Get message ID
String msgId = m.getHeader("Message-ID")[0];
msgId = msgId.substring(1, msgId.length() - 1);
msgId = "mid:" + msgId;
```

- Retrieving some header information of a message:

```
// Get some headers
Date sdate = (m.getSentDate());
String from = ((InternetAddress)m.getFrom()[0]).getAddress();
String nfrom = ((InternetAddress)m.getFrom()[0]).getPersonal();
String to = ((InternetAddress)
            m.getRecipients(Message.RecipientType.TO)[0])
            .getAddress();
String nto = ((InternetAddress)
            m.getRecipients(Message.RecipientType.TO)[0])
            .getPersonal();
String subj = m.getSubject();
String mimeType = m.getContentType();
```

#### 4.3.2 Parsing an email message with basic information to RDF Model

- Creating an empty model with the properties for RDF database:

```
// Create an empty RDF graph (model)
Model model = ModelFactory.createDefaultModel();

// Create predicates of RDF model
Property _sn = model.createProperty(sMMURI, sSName);
Property _fr = model.createProperty(sMMURI, sFrom);
Property _rn = model.createProperty(sMMURI, sRName);
Property _to = model.createProperty(sMMURI, sTo);
Property _sb = model.createProperty(sMMURI, sSubject);
Property _mime = model.createProperty(sMMURI, sMimeType);
```

- Creating the resource and adding the properties according to the information of each email message into RDF graph:

```
// Create a new resource for each message
Resource msg = model.createResource(msgId);

// add the properties cascading style
msg.addProperty(_fr, from)
  .addProperty(_sn, ((nfrom!=null)?nfrom:"no-name"))
  .addProperty(_to, to)
  .addProperty(_rn, ((nto!=null)?nto:"no-name"))
  .addProperty(_sb, ((subj!=null)?subj:"no-subject"))
  .addProperty(DC.date, sdate)
  .addProperty(_mime, mimeType);
```

### 4.3.3. Persisting RDF Model to a MySQL database

- Preparing necessary parameters for database connection

```
// path of driver class
String className = "com.mysql.jdbc.Driver";
// URL of database server
String DB_URL = "jdbc:mysql://localhost/mailrdf";
String DB_USER = "root"; // database user id
String DB_PASSWD = ""; // database password
String DB = "MySQL"; // database type
```

- Loading database driver and making DB connection

```
// Load the Driver
Class.forName(className);

// Create database connection
IDBConnection conn =
    new DBConnection(DB_URL, DB_USER, DB_PASSWD, DB);

// Create a model in the database
ModelRDB m = ModelRDB.createModel(conn, "MailMessages");
```

- After create a variable typed ModelRDB, we can using it in the same way of Model-Factory as above to store RDF model into a MySQL database. The following picture (Fig. 2) shows the record set of resources which are stored in a MySQL database.

### 4.3.4. Query information from RDF data using RDQL

- Reading model from RDF file

```
Model model = ModelFactory.createDefaultModel();
model.read(new FileInputStream("messages.rdf"),
    "http://nowhere/", "RDF/XML-ABBREV");
```

- Or reading model from a persistent MySQL database

```
IDBConnection conn =
    new DBConnection(DB_URL, DB_USER, DB_PASSWD, DB);
ModelRDB model = ModelRDB.open(conn, " MailMessages");
```

- Generating RDQL statement:

```
String queryString = "SELECT ?x, ?name, ?from " +
    "WHERE (?x, <j.0:name>, ?name), (?x, <j.0:from>, ?from) " +
    "USING j.0 FOR <http://www.ifs.tuwien.ac.at/ontproj/1.0/>";
```

Subj	Prop	Obj
Uv::mid:003001c423d0\$bb07b230\$d09ba2cb@49PAS.	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/from:	Lv:0:hobac@gmx.net:
Uv::mid:003001c423d0\$bb07b230\$d09ba2cb@49PAS.	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/mime:	Lv:0:text/html; charset=ISO-8859-1:
Uv::mid:003001c423d0\$bb07b230\$d09ba2cb@49PAS.	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/recvname:	Lv:0:no-name:
Uv::mid:003001c423d0\$bb07b230\$d09ba2cb@49PAS.	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/sentname:	Lv:0:Nguyen Ho Bac:
Uv::mid:003001c423d0\$bb07b230\$d09ba2cb@49PAS.	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/subject:	Lv:0:Re: [sva] My son:
Uv::mid:003001c423d0\$bb07b230\$d09ba2cb@49PAS.	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/to:	Lv:0:sva@yahoogroups.com:
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://purl.org/dc/elements/1.1/date:	Lv:0:Mon Apr 19 15:09:20 CEST 2004:
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/from:	Lv:0:thluong@gmx.net:
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/mime:	Lv:0:multipart/mixed;
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/recvname:	Lv:0:Hanh Hoang Huu.
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/sentname:	Lv:0:Ha Thu Luong:
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/subject:	Lv:0:hinh:
Uv::mid:01de01c4260f\$87e2d010f0012fd5@SEN:	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/to:	Lv:0:hhanh@gmx.at:
Uv::mid:1082104211.407f999309336@www.ifs.tuwien.ac.at	Uv::http://purl.org/dc/elements/1.1/date:	Lv:0:Fri Apr 16 10:30:11 CEST 2004:
Uv::mid:1082104211.407f999309336@www.ifs.tuwien.ac.at	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/from:	Lv:0:Giap@ifs.tuwien.ac.at:
Uv::mid:1082104211.407f999309336@www.ifs.tuwien.ac.at	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/mime:	Lv:0:text/html; charset=ISO-8859-1:
Uv::mid:1082104211.407f999309336@www.ifs.tuwien.ac.at	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/recvname:	Lv:0:sva@yahoogroups.com:
Uv::mid:1082104211.407f999309336@www.ifs.tuwien.ac.at	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/sentname:	Lv:0:Giap Van Duong:
Uv::mid:1082104211.407f999309336@www.ifs.tuwien.ac.at	Uv::http://www.ifs.tuwien.ac.at/ontproj/1.0/subject:	Lv:0:Re: [sva] My son:

Figure 2: Database persistency of RDF Model for email messages

- Executing the query through RDF API:

```
Query query = new Query(queryString) ;
query.setSource(model) ;
QueryExecution qe = new QueryEngine(query) ;
```

- Getting the results and print them out:

```
QueryResults results = qe.exec() ;
for ( Iterator iter = results ; iter.hasNext() ; )
{
    ResultBinding res = (ResultBinding)iter.next() ;
    Object x = res.get("x") ;
    Object fname = res.get("name") ;
    Object email = res.get("from") ;
    System.out.println("x = " + x + "    from = ' " +
        fname + " ' has e-address <" + email + ">") ;
}
```

## 5. Conclusion

With regarding email messages as an incoming resource for the system, this architecture, with support of Jena framework and Javamail API, fulfills some main demands.

The above results are the first and core steps for us to go ahead with more complicated demands. This prototype architecture is designed according to the complete architecture of the Semantic Web project, however it is simple and should be improved in the next steps.

## 6. Future Work

We will continue to develop some features for this prototype as following ideas. Firstly, that is the creating an inference layer for this prototype architecture to support semantic

queries. With Jena, we can create the ontologies using RDFS or OWL and use them for reasoning.

Secondly, we continue to develop a module for monitoring and selective parsing new email messages to RDF data. This module helps a user can parse all new messages or some selected messages to RDF and add to a RDF model.

Last but not least, building a user interface for the system is very important. In this context, we will develop two separate parts: one part is a user's GUI of the system and another is an interface between ontology repository, based on Jena framework, and the "Interface/Analysis/Event-Handler" part in this system. This part is exactly the Network API in the system. We are considering to use Joseki as a solution for this task with web-based API calls (with HTTP/GET and POST).

## 7. References

- [1] Powers, S. Practical RDF: Solving Problems with the Resource Description Framework, *O'Reilly*, 2004.
- [2] Jena Semantic Web Framework, HP Semantic Web Lab (Aug. 2003), <http://jena.sourceforge.net>.
- [3] Javamail API, Sun Microsystems (2003) , <http://java.sun.com/products/javamail/>.
- [4] McBride, B. Jena: A semantic Web Toolkit, *IEEE Internet Computing*, 2002.
- [5] Loton, T. JavaMail Quick Start, Java World (2001), <http://www.javaworld.com>.
- [6] Singla, V. E-Mailing Through Java (2001), <http://www.vipan.com>.