1. Objectives
The objective of the prototype is to develop a system to store email messages’ header 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.

2. 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.

3. Prototype Architecture
3.1 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 SemanticLIFE project architecture 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.

**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.

**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.

---

1 The SemanticLIFE project of Institute of Software Technology, Vienna University of Technology.
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. Joseki is an ideal solution in context of this prototype because it allows user access RDF persistence remotely through HTTP GET protocol.

### 3.2 Workflow

There are 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 an e-mail server to get the messages, and then get the information from the messages’ headers. Consequently, the 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.

### 3.3 Implementation Results

#### 3.3.1 Technical Issues

Prototype implementation is oriented to open-source and it has been implemented using Java language and some open-source Java-based libraries and tools. Following is the chosen technologies:

- Ontology storage: using RDF and OWL
- RDF library: Jena Semantic Web Framework of HP Labs (analyzed above).
- E-Mail library: Java Mail and Java Bean Activation of Sun Microsystems.
- DBMS: MySQL
- Java IDE: Eclipse 3.0

#### 3.3.2 Email Ontology

```xml
<owl:Ontology rdf:about="" />
```
subclass of 'EmailMessage' class

Figure 2: E-Mail Ontology Visualization
3.3.3 Implemented Modules

In this part, we present functions of implemented modules in the prototype briefly. The system consists of:

- Module of parsing email messages to RDF (Mail2RDF): having 2 main sub-functions:
  - accessing a mailing server, getting email messages and some headers.
  - parsing email messages’ headers into RDF model, then storing in a RDF file.
- Module of persisting RDF model to RDBMS (PersistentRDBMS): persisting a RDF model to a MySQL database.
• Module of monitoring new messages (NMail2RDF) module:
  o monitoring a mailbox for new messages,
  o then parse them to RDF data and merge into existing RDF persistence.
• Query module (MailRDQL): querying information of email messages from RDF model or from a MySQL database using RDQL.
• Module of publishing a model to RDF Server (Joseki): publishing a model stored in a RDF XML file or in a persistent database to Joseki RDF server for remote access.
• Module of Remote Model Query (MailRDQLJoseki): making queries into remote models stored in Joseki RDF server using HTTP GET protocol (Joseki is a module of Jena Semantic Framework Toolkit).
• Reasoning Module (in progress): to create a inference layer for the system to support querying information from system in semantic way.

3.3.4 Disadvantages
- Prototype has not an UI because we want to build an UI which is not based on web (JSP) but an independent Java application. However we are not so familiar to develop a UI on Eclipse and JBuilder as well.
- Because of not having an UI, so prototype is not friendly with users.
- Security issue is not concerned is this prototype.

4. 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 SemanticLIFE project, however it is simple and should be improved in the next phases.

5. Future Work
We will continue to develop some features for this prototype as following ideas:
- Other data sources will be taken into account: calendar data (using iCal – refer to RFC 2445), browsed web pages, attached documents in emails.
  o Building ontology for them
  o Parsing data to RDF
- Focus on two work area in the SemanticLIFE project:
  o Storage Module
  o Querying module.
- Start to use Spring Framework for J2EE application and database issues as well.