Ontology-based Virtual Query System for the SemanticLIFE Digital Memory Project

Hoang Huu Hanh, A Min Tjoa, Nguyen Manh Tho

Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria

{hanh, tjoa, tho}@ifs.tuwien.ac.at
Contents

- SemanticLIFE project brief overview
- Requirements of information retrieval in the SemanticLIFE system
- The Virtual Query System (VQS)
  - Goals
  - Design
  - State-of-the-Art of Implementation
SemanticLIFE: a Brief Overview

- The SemanticLIFE project is an attempt to come a step closer to Vanevar Bush’s vision of the Memex
- The SemanticLIFE was firstly presented at iiWAS ‘04
- The SemanticLIFE project aims at realizing a digital personal diary that records everything a person wants to be kept.
  - This notion defines also the boundaries of the project: do not deal with memory issues of the unconscious or procedural memories (e.g. how to open a bottle).
  - We are aiming at a tool which supports the long term memory by associating metadata with content and ontologies.
  - The possibility of adding annotations to all stored objects should enrich the potential use of such a ‘diary’.
SemanticLIFE: Architecture

Fig. 1. SemanticLIFE System Architecture
Ontology-based Virtual Query System

SemanticLIFE: Architecture (contd’)

- Data is fed into the system using a number of dedicated data acquisition modules (Data feeding modules)
- The data objects are passed on by a message handler to the analysis module.
  - Analysis contains a number of specific analysis plugins
- Processed information objects are forwarded by the message handler to the storage module for ontologically structured storage.
- A set of query processing and information visualization tools provides means for information exploration and report generation.
- Data sources are stored in SemanticLIFE in form of RDF, and their metadata. Each data sources have an ontology/schema.
- Query module uses these metadata and a global ontology of the system to give the user a better understanding of the system’s data before giving out requests.
SemanticLIFE is a modular architecture, and a work in progress. Following are developed modules:

- Logic Framework based in Java plug-in Framework
- Message Handler: a router for internal messages
- Data Feeder: a bunch of data feeding modules
  - Covered data feeds: Email, Contact, Calendar, Task, Note, Browsed web pages, Process motoring, Filesystem monitoring, File Uploader, and Chat sessions.
- Annotator: manual annotation on data
- Metastore: repository of SemanticLIFE with index mechanism
  - Kowari
  - Jena
- Query: the first prototype of VQS has been developed with basic functions
- Visualizer: a presentation layer for query results
The current similar systems (to SemanticLIFE) and even IR systems:

- **Focus on data integration:**
  - Gathering (distributed) data sources,
  - Integrating them (both data and schema)
  - Storing in scalable repositories

- **Try to deal with the ambiguity/imprecise of user’s queries in querying-time**
  - Accept the ambiguity/imprecise of the user’s requests at very early stage of querying process
  - Query processing will be taken on real data => dealing with a heavy transaction or traffic
The system is capable to work with ‘weaker’ search terminology

The system is possible to provide more powerful “imprecise searches”

Term “imprecise” has two meanings:

- the generated queries are to satisfy fuzzily defined information needs.
- the target of the query is specified but there is ambiguity in the query.

The Virtual Query System (VQS) has to solve these problems during query generation, by exploring the system’s database and ontology repository and then generate queries for a specific technology.
Query Module Workflow

Fig. 3. Query Module in SemanticLIFE context
Virtual Query System: Introduction

- The VQS is a query system to help the user:
  - Making as many unambiguous queries as possible
  - Quickly answering the user’s requests
  - Efficiently refining the queries (based on ontologies and virtual data) with the feed-back strategy

- The VQS
  - has a virtual data layer which contains metadata of real data sources
  - helps the user create virtual queries based on the virtual data and the user profile
  - analyzes the virtual query based on a common ontology mapped from local ontologies of data sources
  - generates sub queries from it for specific real data sources
  - aggregates the results from sub queries and represents to the user with regarding his profile
Virtual Query System: Architecture

**Virtual Query System Components**

- **Virtual Query Formulation**
- **Query Result Composition**
- **Sub Queries Formulation**
- **Meta-data sources**
- **Inference**
- **Services**
- **Ontology Repository**

**SemanticLIFE Repository & Metastore**

![Diagram](image)

**Fig. 4. The Virtual Query System Components**
Virtual Query System: Components

- Meta-data Sources Part
  - Acting as “virtual” data showed to the user
  - Collecting metadata from data in the system’s metadata store
  - Carrying out analysis and statistical processes to get necessary information about data sources
  - Storing them in this module in form of RDF(S)
  - Showing the virtual data to the user in order to help him/her make unambiguous requests.

- Ontology repository
  - contains the ontologies used in the VQS such as global ontology or user ontology
  - an ontology mapping mechanism will be applied to map the local ontologies of data sources in ontology repository into a global ontology
Sub-queries Formulation

- From the original virtual query of the user (using Virtual Query Language – VQL), this part will create the sub queries based on the global ontology for specific data sources.
- This part not only translates the virtual query directly to sub-queries for specific data sources but can ‘infer’ on the user’s request and created more sub-queries as well.
- After this process, the ‘real’ queries will be generated and based on a specific RDF query language.
Virtual Query System: Components (contd’)

- The VQS Services
  - Ontology mapping
  - Query caching
    - Semantic Query caching for RDF data
    - Evaluating virtual query
  - Inference and Query Refinement
    - Inference helps analyze the virtual queries
    - Step-by-step query refinement (+ user feedback)

- Virtual query user interface
  - Virtual query formulation
    - Forming virtual queries (in form of XML/RDF) against the virtual data
  - Query result composition
Fig. 5. The Virtual Query System Workflow Diagram
Virtual Query System: Implementation solutions

- **Meta-data sources part**
  - developed from scratch
  - Jena and DOM4J will be used for RDF and XML manipulation respectively.

- **Ontology representation**
  - SOFA is a promising candidate

- **Ontology mapping**
  - MAFRA and PROMPT will be weighed

- **Ontology-based inference**
  - RACER or Pellet (complicated) or SOFA (simple) will be used

- **Sub-queries formulation**
  - developed from scratch
  - SPARQL will be used for generated sub-queries

- **Query user interface: developed from scratch**
  - Virtual query formulation: template technique will be applied
  - Results composition: output standards must be obeyed current standards
State-of-the-Art of VQS Implementation

- Meta-data sources part: in progress
- **Ontology representation: done!**  
  - An Ontology has been built with OWL/Protégé
- Ontology mapping: in progress
- **Ontology-based inference: simple one: done!**  
  - With support of SOFA
- Sub-queries formulation: done!  
  - Proposing a new query language namely VQL  
  - Parsing to RDF query statements
- **Query user interface: done!**  
  - Interface for VQL/iTQL  
  - Results presentation in XML, HTML, Text, Tabular, and Graph
Conclusion

- Besides applying the existing the Semantic Web technologies
  - ontology, ontology mapping,
  - RDF techniques and
  - semantic query caching for RDF data

in the VQS with a new approach: front-end;

- The contributions of the VQS is a new idea about
  - reducing complexity of asking queries and also reducing number of ambiguous queries from the user
  - query processing based on a virtual data which contained metadata of real data, and user profile or behaviour.
Future Work

- The following issues will be focused
  - Improving VQL - Virtual query language
  - Building an ontology-based interactive interface based on VQL – I-VQS (“I” means “Interactive”)
  - User modeling for VQS
  - Semantic query caching for RDF data,
  - RDF query refinement
  - and query optimization
THAT’S ALL

Thank You for Your Attention!
Virtual Query Language (VQL)

Aims
- VQL helps clients making queries without knowledge of RDF query languages.
- VQL simplifies the communication between Query module and other parts. Since the components asking for information do not need to issues the RDF query statement, which is uneasy task for them. As well as this feature keeps the SemanticLIFE's components more independent.
- VQL enables the portability of the system.

Actually, the SemanticLIFE and VQS choose a specific RDF query language for the its back-end database. However, in the future, they could be shifted to another query language, so that this change does not effect other parts of the systems, especially query interface of the system database.
Virtual Query Language (VQL)

- **VQL Syntax**
  - The first part contains parameters of specifying needed-information which are defined in variable names, criteria values, and additional attributes.
  - The second part specifies the sources where the information will be extracted from.
    Obviously, the information need is defined in the first part must be related to the sources specified in this part as they are selected by the user from a well-defined ontology.
  - The constraints are defined in the third part of the document.
    Here the relations between sources, parameters are connected using the VQL operators. The operators and the interpretation of expressions in the VQL queries will be mentioned in the next sub section.
- **XML is used for the syntax of VQL**
Virtual Query Language (VQL)
Virtual Query Language (VQL)

- **VQL Query Types**
  - Data Query Type
  - VQL Schema Query Type
  - VQL iTQL Type 1
  - VQL iTQL Type 2

- **VQL Query Operators**
  - GetInstances operator
  - GetInstanceMetadata operator
  - GetRelatedData operator
  - GetLinks operator
  - GetFileContent operator
VQL Query Examples

- **Example 1:**

  Return time stamp of all records of “FileUpload” and “BrowsingSession” sent during specific date range

```
<query type="data">
  <params>
    <param show="1" name="s1:messageTimeStamp">2005-11-01</param>
    <param show="0" name="s2:messageTimeStamp">2005-11-31</param>
  </params>
  <sources>
    <source name="fileupload">FileUpload</source>
    <source name="browsingsession">BrowsingSession</source>
  </sources>
  <relations>
    <relation id="1" param="s1" source="">dt:gt</relation>
    <relation id="2" param="s2" source="">dt:lt</relation>
  </relations>
  <resultformat>xml</resultformat>
</query>
```
VQL Query Examples

Example 2:

Return all *metadata properties* of uploaded files during specific date range

```xml
<query type="data">
  <params>
    <param show="1" name="p0:messageTimeStamp">2005-11-01T00:00:00</param>
    <param show="0" name="p1:messageTimeStamp">2005-11-30T00:00:00</param>
    <param show="1" name="p2:METADATA"/>
  </params>
  <sources>
    <source name="fileupload">FileUpload</source>
  </sources>
  <relations>
    <relation id="1" param="p0" source="">dt:gt</relation>
    <relation id="2" param="p1" source="">dt:lt</relation>
  </relations>
  <resultformat>xml</resultformat>
</query>
```
VQL Query Examples

- Example 3:
  Return content of a uploaded file

```xml
<query type="data">
  <params>
    <param show="0" name="p0:filePath">
      c:/slifedata/kowari/slif/uploadedfiles/2005/11/28/CFP_RIVF06.txt
    </param>
    <param show="1" name="p2:CONTENT" />
  </params>
  <sources>
    <source name="fileupload">fileupload</source>
  </sources>
  <relations/>
  <resultformat>xml</resultformat>
</query>
```
VQL Query Examples

- Example 4: (VQL iTQL query types) containing an iTQL query

```xml
<query type="itql1">
  <statement>
    select $s $p $o
    from &lt;rmi://192.168.168.174/slife#BaseModel&gt;
    where $s $p $o;
  </statement>
  <resultformat>xml</resultformat>
</query>

<query type="itql2">
  <select>
    $s $p $o
  </select>
  <from>
    rmi://192.168.168.174/slife#BaseModel
  </from>
  <where>
    $s $p $o
  </where>
  <resultformat>text</resultformat>
</query>
```
Example 5: (VQL iTQL Type 1)

Return an uploaded file having “CFP” in its filename

```
select $Message $p $o
from <rmi://192.168.168.174/slife#BaseModel>
where $Message <slife:messageBody> $oo
    and $oo $p $o
    and $Message <slife:fileName> 'CFP*' in
        <rmi://192.168.168.174/slife#FTLiteralsModel>;
```

```
<query type="itql1">
    <statement>
        select $Message $p $o from
            &lt;rmi://192.168.168.174/slife#BaseModel&gt; where $Message
            &lt;slife:messageBody&gt; $oo and $oo $p $o and $Message
            &lt;slife:fileName&gt; 'CFP*' in
                &lt;rmi://192.168.168.174/slife#FTLiteralsModel&gt;;
    </statement>
    <resultformat>xml</resultformat>
</query>
```
Example 6: (VQL Schema Query)

Return *all subclasses* of a specific class

```sql
select $s
from <rmi://192.168.168.174/OntologyModel>
where $s <rdfs:subClassOf> <slifeont:FileUploadData>;
```

```xml
<query type="schema">
<statement>
  select $s from &lt;rmi://192.168.168.174/OntologyModel&gt;
  where $s &lt;rdfs:subClassOf&gt; &lt;slifeont:FileUploadData&gt; ;
</statement>
<resultformat>text</resultformat>
</query>
```
Query Client Screenshots

Startup Screen

<query type="data">
<params>
  <param show="1" name="p0:messageTimeStmp">2005-11-01T00:00:00Z</param>
  <param show="0" name="p1:messageTimeStmp">2005-11-30T00:00:00Z</param>
</params>
</query>
Query Client Screenshots

The image shows a user interface for a query client with a focus on query verification. The interface includes:

- A query editor with code snippets, including a query named "s1:METADATA".
- A query verification window indicating an error in the query syntax.
- A message stating that the query is not validated as a SLF4X query.
- Specific error messages indicating issues with attributes and element names.

The query verification error messages include:

- Error: Attribute 'type' must appear on element 'query'.
- Error: Invalid content was found starting with element 'params'. One of '{"statement"}' is expected.

Date: 20.6.2006

Title: Ontology-based Virtual Query System
Query Client Screenshots

Query Execution

Ontology-based Virtual Query System
That’s all. Thank you!

Contact:
HOANG HUU HANH
Institute of Software Technology and Interactive System
Vienna University of Technology
Favoritenstraße 9-11/188
A-1040 Vienna, AUSTRIA

E: hanh@ifs.tuwien.ac.at   T: +43.1.58801 18861   F: +43.1.58801 18899