A FRONT-END APPROACH FOR USER QUERY GENERATION AND INFORMATION RETRIEVAL IN THE SEMANTICLIFE FRAMEWORK

Hanh Huu Hoang, Tho Manh Nguyen, Amin Andjomshoaa, A Min Tjoa*

Abstract. Formulating unambiguous queries in the Semantic Web applications is a challenging task for users. This paper presents a new approach in guiding users to generate clear requests based on their common nature of querying for information. The approach known as the “front-end approach” gives users an overview about the system data through a “virtual data component” which stores the extracted metadata of the data storage sources in the form of an ontology. This approach reduces the ambiguities in users’ requests at very early stage; and allows the query refinement process to easily to fulfill users’s demands. Furthermore, the approach provide a powerful query engine, called “context-based querying”, that recommends the appropriate query patterns according to the user’s querying context. These features help the user in generating clear query more easier.

1. Introduction

The Semantic Web and ontologies have created a promising background for applying the intelligent techniques in information systems especially in Personal Information Management (PIM) systems. In PIM systems, the effectively information retrieval from a huge amount data of an individual is a challenging issue. The Virtual Query System (VQS) [7] of the SemanticLIFE framework is an approach of using semantic web techniques with an user-oriented method in order to tackle this challenge.

The SemanticLIFE project [1] is an effort to realize Vanevar Bush’s vision of Memex [3] by providing a general semantic PIM system. The SemanticLIFE system integrates a wide variety of data sources and stores them in an ontological repository. The SemanticLIFE’s user is supported in issuing imprecise queries to retrieve the rich semantic information from his/her personal data. However, users themselves often do not actually know or remember the specific qualities of what they are looking for, but have some awareness of other things related to the desired items [10]. The VQS supports users upon this nature when querying the information from a huge ontological repositories.

In this paper, we emphasize on the “front-end approach” with the help of system ontologies and data sources: how to collect the metadata of data storage sources and organize them; how to represent this “virtual data” to the user; and how it supports the user in generating unambiguous queries. The core of this approach is the Virtual Data Component which will answer these questions.

*Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria, email: {hanh, tho, andjomshoaa, amin}@ifs.tuwien.ac.at, tel: +43-1-58801-18861, fax: +43-1-58801-18899.
The VQS system, with the organization of the “virtual data component” as a context ontology, will guide its users go through the system by intelligently recommended query patterns based on the current query context, so-called query space, and that context ontology (the virtual data). In our “front-end” approach, this is the most crucial feature that is different to the current ontology-based query systems [5].

The remainder of this paper is organized as follows: the motivation for this research direction and the paper is mentioned in section 2. An overview of the SemanticLIFE project and the VQS is pointed out in section 3. The details of the “virtual data component” is described in section 4. The VQS interactive query generator is introduced in section 5. Finally, the paper is concluded with a sketch of the intended future work in section 6.

2. The State of The Art

There are two main approaches to reduce the difficulty in generating queries from user-side in Semantic Web applications [5]. The first trend is going to design a friendly and interactive query interfaces to guide users in generate the queries. The high-rated examples for this trend are GRQL [2] and SEWASIE [4].

GRQL - Graphical RQL - relies on the full power of the RDF/S data model for constructing on the fly queries expressed in RQL [8]. More precisely, a user can first graphically navigate through the individual RDF/S class and property definitions, then transparently generate the required RQL path expressions required to access the resources of interest. These expressions accurately capture the meaning of its navigation steps through the class (or property) subsumption and/or associations. Additionally, users can enrich the generated queries with filtering conditions on the attributes of the currently visited class by specifying the resource’s class(es) appearing in the query result.

Another graphical query generation interface, SEWASIE, is described in [4]. As a starting point, the user is provided some pre-prepared domain-specific patterns to choose from as a starting point, which he can then extend and customize. The refinements to the query can either be additional property constraints to the classes or a replacement of another compatible class in the pattern such as a subclass or superclass. This is performed through a clickable graphic visualization of the neighbourhood ontology of the currently selected class.

The second approach of reducing complexity is the effort in creating much lighter query languages than expressive RDF query languages. Following this trend, the approach in [11], known as GetData Query interface, is a typical example. GetData Query interface [11] of TAP\(^1\) expresses the need of a much lighter weight interface for constructing complex queries. The idea of GetData is to design a simple query interface which allows to present network accessible data as directed labeled graph. This approach provides a system which is very easy to build, support both type of users, data providers and data consumers.

Our approach, the VQS, supports the user in generating unambiguous requests by providing a “virtual data” layer to improve the user’s awareness of the stored information in the system. This virtual data are also used for answering the user’s simple tasks. VQS also has a lighter weight query language [6] to assist the users make queries in simple manner while still getting maximum of information.

\(^{1}\)TAP Infrastructure, http://tap.stanford.edu/.
3. **SemanticLIFE and Virtual Query System**

3.1. **The SemanticLIFE Framework**

The SemanticLIFE framework is developed on a highly modular architecture to store, manage and retrieve the lifetime’s information entities of individuals. It enables the acquisition and storage of data while giving annotations to email messages, browsed web pages, phone calls, images, contacts, life events and other resources. It also provides intuitive and effective search mechanism based upon the stored semantics, and the semantically enriched user interfaces according to the user’s needs. The ultimate goal of the project is to build a PIM system over a Human Lifetime using ontologies as a basis for the representation of its content.

The whole SemanticLIFE system has been designed as a set of interactive plug-ins that fit into the main application and this guarantees flexibility and extensibility of SemanticLIFE platform. Communication within the system is based on a service-oriented design with the advantage of its loosely coupled characteristics. To compose complex solutions and scenarios from atomic services which are offered by SemanticLIFE plug-ins, the Service Oriented Pipeline Architecture (SOPA)\(^2\) has been introduced. SOPA provides a paradigm to describe the system-wide service compositions and also external web services as pipelines. SOPA provides some mechanisms for orchestration of services and transformation of results.

![The Architecture of the SemanticLIFE Framework](image)

The SemanticLIFE’s system architecture overview is depicted in Figure 1. Data with user annotation is fed into the system using a number of dedicated plug-ins from variety of data sources such as Google Desktop\(^3\) captured data, communication logs, and other application’s metadata. The data objects are transferred to the analysis plug-in via the message handler. The analysis plug-in contains a number of specific plug-ins which provide the semantic mark-up by applying a bunch of feature extraction methods and indexing techniques in a cascaded manner. The semi-structured and semantically enriched information objects are then ontologically stored via the repository plug-in. In the SemanticLIFE system, data sources are stored in forms of RDF\(^4\) triples with their ontologies and metadata. This repository is called a metastore.

---

A set of query processing and information visualization tools provides the means for information exploration and report generation. The analysis module and metadata extraction capabilities make associations between the lifetime items/objects and the lifetime events based on user annotation, user profiles and the system ontologies.

### 3.2. The Virtual Query System

Formulating non-ambiguous queries is always a too demanding task to users as they do not have the awareness of the semantics of the stored information. The goal of the Virtual Query System (VQS) is to overcome this problem by providing an ontology-based virtual information view of the data available in the system. If the user can “be aware of” what is inside of the system he/she can clearly specify the queries on the “real” data stored in the repository.

The VQS system is primarily based-on the reduction of semantic ambiguities of the user query specifications at the very early stage of the retrieving process. The most important point in the VQS approach is that it provides an image of the real system data sources to the user. The user is aware of the data stored in the system when he/she generates the queries. As a result, the ambiguities in his/her requests will be much reduced.

![Virtual Query System](image)

**Figure 2. The Components and Architecture of the Virtual Query System**

Based on a common ontology mapped from the local data sources’ ontologies and the internal analysis which supports the inference, detecting ambiguity and fuzziness of user queries, the VQS refines the user’s queries and creates the “real” subqueries against the data sources in the metastore. In addition, a set of predefined query templates over the “virtual data” is available for the user according to his/her “context”. This makes the user more convenient in the manner of generating queries. Furthermore, the VQS could recommend the more relevant results to the user dependent on user’s experiences, user ontology reflection or user profile.

As depicted in Figure 2, with the support of virtual data component containing metadata of real data sources, the user can generate the virtual query against the virtual data. The VQS then analyzes the virtual query based on a common ontology mapped from local ontologies of data sources and user ontology. As the result, subqueries from initial query are generated for the specific underlined data sources. Finally, the results of subqueries are aggregated and represented to the user with regarding his profile.
4. The Virtual Data Component

4.1. The Goals

From the user perspective, the Virtual Data Component (VDC) plays an important role in the process of query generation in the VQS approach. The core of the VQS is the module containing the Metadata Storage Sources (MSS). This module acts as a "virtual" database that allows the user to be aware of the meaning of the stored data sources and to specify more precise queries. The VQS collects the metadata from the data sources within the SemanticLIFE metadata repository. An analysis process and a statistical computation are carried out on these metadata sources to get the semantic information which is then stored in the VDC and will be delivered to the user query generation interface. This component is also referred as an "image" of the system database in further query processing.

4.2. Metadata Storage Sources Collecting

In the SemanticLIFE metastore, there are different data sources’ ontologies exist along with the instances. The SemanticLIFE system manages a huge range of data from common personal information such as contacts, calendar, tasks, notes, documents, files, phone calls, instant messaging logs and so on; to general data such as maps and weather information [1].

Figure 3 presents the data feeds covered by the SemanticLIFE framework in the current prototype. The data feeds are about the information of an individual diary. From this ontology and the underlined instances, a VQS service extracts the metadata, performs some statistical computations and stores in the final ontology (so-called the Virtual Data component). This component is used in the Metadata Storage Sources to provide semantic information on the corresponding data sources to the users.

The VDC is a synthesis ontology which is merged from these variety data feeds ontologies. We use MAFRA ontologies merging framework with "semantic bridge concept" [9] as the mapping service to merge the ontologies. This merging process consists of aligning the schemes and merging the instances as well.

4.3. Context-based Support of the Virtual Data Component

The loosely-coupled organization of the metadata storage sources reflects the flexibility of the Virtual Query System as well as the SemanticLIFE framework. Based-on this ontology, the context-based query templates are also categorized according to the classes. We can apply the VQS or SemanticLIFE in different contexts by simply making changes of the corresponding metadata storage sources. The
metadata storage sources are constructed as an ontology namely metadata context or context ontology. By doing the taxonomy and reasoning on the classes and also on the instances, the metadata could be classified into the categories and the data are arranged into the relevant ontology dependent on the context that the SemanticLIFE framework is used for.

Figure 4 shows the constructed ontology by merging data sources’s schema and the taxonomized instances. The depicted ontology looks like a personal diary to record the daily activities of an individual who works in some projects. The extracted metadata will be fetched from system data feeds ontologies and put into this context ontology. For instance, the “Place” class is an abstract concept and formed from schemes and instances of “Contact”, annotations, map and so on.

![Figure 4. A Fragment of the Virtual Data Component Ontology](image)

Moreover, the VDC contains the statistic information for each data sources over the customized timelines such as total items, number of items for each data sources in last month/week or yesterday. This provides the user a very good awareness about his/her data in the system. As a result, the user can query on the necessary statistic information.

Virtual Data Component is the typical feature of our system compared with the other systems mentioned in Section 2.. The basic idea behind is quite simple: “If the user is aware of his/her data, then he/she could generate more unambiguous request”. As a result, on the one hand, this reduces the complexity of the query refinement process. On the other hand, with the virtual data component as a context query, the system could flexibly adapt with a new scenario (just by changing of the correspondent context ontology).

4.4. The Virtual Query Language

The Virtual Query Language (VQL) is the query language designed for the MSS component [6]. The VQL aims to be a simple query language which supports “semantic” queries on the ontological RDF-based storages. VQL is very easy to build and use and to be supported. In context of the VQS, the aims of VQL are as follow:

- VQL helps the clients making queries without the knowledge of RDF query languages. The user just gives basic parameters of request information in VQL, and would receive the expecting results.
- VQL assists users in navigating the system via semantic links/associations, categorized context queries provided in the powerful query operators based-on ontologies.

An example of VQL queries is depicted in Figure 5, in which, the VQL query is called "RELATED-WITH" query [6]. This query retrieves the related information from data sources of Email and Contact to the email address hta@gmx.at.

```xml
<query type="data">
  <params>
    <param show="0" name="p1:emailTo">hta@gmx.at</param>
    <param show="1" name="p2:RELATED-WITH"/>
  </params>
  <sources>
    <source name="email">Email</source>
    <source name="contact">Contact</source>
  </sources>
  <relations>
    <relation id="1" param="p1" source="email"/>
  </relations>
  <resultformat>xml</resultformat>
</query>
```

**Figure 5. An Example of a VQL Query**

The VQL is designed to help the user easily generate requests according to their nature: *minimum of words, maximum of results*. In order to fulfill this principle, the VQL defines the *virtual query operators* which allow the user to simplify the complex queries [6]. The operators are described as following:

*GetInstances Operator* is the common form of VQL data queries. The operator retrieves the appropriate information according to the criteria described in the parameters, sources and constraints of the query.

*GetInstanceMetadata Operator* assists the user easily retrieve all metadata properties and correspondent result instances. This query operator is very useful when the user does not know exactly what properties of data instances are.

*GetRelatedData Operator* provides the accessible related information to the current found information. In semantic web applications, particularly in the SemanticLIFE system, finding relevant or associated information plays an important role.

*GetLinks Operator* operates using the system’s ontology and RDF graph pattern traversal to find out the associations/links between the instances and the objects.

For instance, we query for a set of instances of emails, contacts and appointments. Normally, we receive these data separately and what we expect here is that the associations between the results provided additionally. The links are probably properties of email addresses, name of the persons, locations.

By providing these operators, VQL offers a powerful feature of navigating the system by browsing data source by data source, instances by instances based on found semantic associations.

---

5VQL samples, http://www.ifs.tuwien.ac.at/~hhhanh/VQL/samples/
5. Context-based Queries in the VQS

5.1. Query Templates

In the VQS, the query templates, so-called query patterns, are defined to assist the user in formulating unambiguous queries. The query templates contain the queries with the necessary parameters and the associated data sources. The templates are classified on the concepts of the VDC’s context ontology and the data sources they are involved. When the template is in use, the specified virtual query will be loaded and its variables are then replaced by the template’s parameters. The virtual query is continually edited by the user afterwards dependent on the interest of the user.

An example of the query template is depicted in Figure 6. That query template is about retrieving the locations of all web pages found by Google search engine and these web pages have been browsed by a given person in a period of time. The related sources for the retrieving process are mentioned in the part. The parameters are put in sub parts and these parameters could be changed or added by the user according the information of interest.

```
<template type="vql">
  <description>Finding a location of webpage browsed by a person using Google.com</description>
  <query name="webpersonse.vql">
    <param>location</param>
    <param>timeStamp</param>
    <param>person</param>
  </query>
  <sources>
    <source>Webpage</source>
    <source>Location</source>
    <source>Person</source>
  </sources>
</template>
```

Figure 6. A VQL Query Template Example

5.2. Query Map

With the associated data sources and the VDC’s context ontology, the query templates creates a query map to make the connection network amongst the templates and underlined resources. The query map is the query templates network which the nodes are the templates and the connections are the related concepts and their properties. According to the connections between the templates, when a query template is chosen for making the new queries, the system also recommend the linked templates.

Besides, when the user selects one or more properties to generate his/her query, the system could also recommend the relevant templates to the user based on the query map. The connections in the query map are used to determine which templates could be used.

This query map is a very useful feature of the system because it allows finding the related query templates more exactly and quickly.

---

5.3. Context-based Querying

The Virtual Data component also enhances the query process by the “context-based” querying feature, i.e. the query patterns will be proposed by the system according to the context where the user is in. However, a user’s query context not only contains all the queried objects and the querying concepts but they are also associated to each other based on the context ontology.

How could the VDC recommend the relevant templates to the user? During context-based information retrieval process, the VDC will do following steps:

1. Keeping track on the concepts queried by the user.
2. From these queried concepts, a context of the user’s querying process will be formed. The context is a graph of queried and querying concepts.
3. When the user asks for a new template from his/her querying context through an interactive interface, then a match of the query map in the virtual data component and the user’s querying context will be made
4. The query patterns/templates will be collected and offered the user.

For example, the context query being applied is about project management which contains the concepts of Project, Person, Document, Publication, Partner, Time, Location and so on. The user’s query context could be a graph of Person, Location, Web search for Project. In this case a query template such as “finding a person I have contacted in Vienna in a related project found by Google search engine” will be proposed.

This feature is applied in the VQS’s interactive interface, in which the user can right clicks on the results objects, instances or virtual data objects and the system will show dedicated templates based on his/her context.

6. Conclusion and Future Work

In this paper, we have presented a new approach for constructing unambiguous queries at very early stage of the query process by providing the user an image of the system data in a context ontology form. The metadata organization in the Virtual Data Component plays an vital role in our approach. The VQS not only supports the user in formulating the virtual queries over virtual data but also assists the user during the querying process with a concept “context-based querying”. The concept is a combination of the context ontology, the query templates and the query space of the user. Through this concept we have introduced the “query map” which is a way to organize the query templates for quick response to the demands.

As the next steps, all design aspects of the VQS and its components will be revised and improved. A benchmark tool will be developed to test the system with design alternatives and compare to similar systems for the accuracy, responsibility and performance. An interactive interface are under development using the VQL, and all mentioned concepts.
Acknowledgement

This work has been generously supported by ASEA-UNINET and the Austrian National Bank within the framework of the “ZELESSA: An Enabler for Real-time Business Intelligence” project - Grant No. 11806.

References


