



TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DISSERTATION

Semantic Web Information Retrieval

A Semantic Web Based Virtual Query System for Supporting User Query Formulation and Information Retrieval in The SemanticLIFE Personal Digital Memory Framework

Ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von

o.Univ.-Prof. Dipl.-Ing. Dr.techn. A Min Tjoa
Institut für Softwaretechnik und Interaktive Systeme
Technische Universität Wien

und

Univ.-Prof. Dipl.-Ing. Dr.techn. Roland Wagner
Institut für Anwendungsorientierte Wissensverarbeitung
Johannes Kepler Universität Linz

eingereicht an der

Technische Universität Wien

Fakultät für Technische Naturwissenschaften und Informatik

von

HOANG HUU HANH

Matrikelnummer: 0327076

Wiesberggasse 9/33

A-1160 Wien, Österreich

Wien, im Feb 2007

Zusammenfassung

Im Jahre 1945, erfand Vannevar Bush die Idee von Memex, einer “Vorrichtung, in der alle Bücher, Aufzeichnungen und Unterhaltungen einer Person gespeichert und mechanisiert werden, damit man mit sehr hoher Geschwindigkeit und Flexibilität beraten werden kann. Diese Vorrichtung ist eine vergrößerte und vertrauliche Ergänzung zum Gedächtnis” [Bus45]. Das SemanticLIFE Projekt ist ein Versuch, an Vannevar Bush’s Anblick einen Schritt näher zu kommen. Bis zuletzt können wir eine rasante Verbreitung neuer Projekte beobachten, die einige der Ziele Bush’s erfinderischen Ideen anstreben. Diese Verbreitung wird hauptsächlich durch die schnell wachsende Vorrantreibung der technologischen Entwicklung verursacht, die neue Potentiale für die Realisierung öffnet. Ein Indikator für die Reduktion der Diskrepanz zwischen den Visionen von Memex und deren Realisierung ist die Ansage vom britischen Computing Research Committee (<http://www.ukcrc.org.uk/>) als eine der sieben größten Herausforderungen fürs Computing Research [FR03], “Gedächtnisse fürs Leben - Mit Informationen über die menschliche Lebenszeit hinaus zurechtzukommen”. Das ACM Seminar über sukzessives Archivieren und Wiederherstellen von persönlichen Erfahrungen (<http://research.microsoft.com/CARPE2004/>) ist ein anderes Indiz dafür, daß die Zeit um solche Systeme einzuführen reif ist.

In Richtung dieser Ziele, persönliche Informationen des gesamten Leben einer Person abzuspeichern und wiederherzustellen, ziehen Forscher auch andauernde Speicherung und Abrufung aller Daten von persönlichen Erlebnissen, unter anderem auch E-Mails, Kontaktinformationen, Treffen, besuchte Webseiten, Dokumente, Logdaten von Instant-Messaging Programmen, Telefongesprächen usw. in Betracht. Die herausfordernde Aufgaben sind, wie man nützliches Wissen von dieser mit Informationen überfüllter Bibliothek extrahiert; und wie man dieses Wissen effektiv verwendet. Das *virtuelle Abfrage System* (*Virtual Query System*) der SemanticLIFE, das in dieser Dissertation vollständig dargestellt wird, ist eine Annäherung an die erwähnten Aufgaben und stellt in einer erfinderischen Art und Weise die benutzerorientierte Abfrage-Formulierung, ein sogenanntes “front-end” zur Verfügung. Mit Unterstützung des virtuellen Frage Systems, wird dem SemanticLIFE-Benutzer bei der Ausgabe von eindeutigen Fragen geholfen, um die reichen semantischen Informationen von seinen historischen persönlichen Daten wiederherzustellen.

Bei der Entwicklung von ähnlichen PIM Systemen wird der Fokus sehr oft auf backend gesetzt, wie zum Beispiel auf die Erfassung von allen Datenquellen, Integration und Speicherung deren in großen Datendepots. Zu diesem Zweck ist es notwendig, die Ontologies der verschiedenen Datenquellen in ein allgemeines Ontology des Systems abzubilden. Jedoch werden Benutzer innerhalb des Systems hinsichtlich der gespeicherten

Informationen mit mangelndem Wissen konfrontiert und würden vieldeutige Anfragen formulieren, so müssten viele Hindernisse überwunden werden, bevor das System die verlangten Resultate liefern kann.

Meine Dissertation strebt ein Design von innovativen Eigenschaften unseres virtuellen Abfrage-Systems an. Dieses Abfragesystem basiert auf eine “front-end” Annäherung, die dem User erlaubt, angestrebte Informationen von sehr großen Datendepots in einer effizienten Art und Weise zu extrahieren. Die Konzeption dieses Abfrage-Systems, das grundsätzlich auf Reduktion von semantischen Mehrdeutigkeiten von Benutzerabfragenspezifikationen im sehr frühen Stadium des Informationsextraktionsprozesses basiert und den User durch die Verwendung von einem Satz von Abfrage-Schablonen im Abfrageprozess führt, die sowohl auf Benutzers Abfragen basieren wie auch auf virtuelle Informationen, heisst “Context Ontology”. Diese Annäherung integriert viele Forschungsbemühungen aus den Bereichen der semantischen Netze, Abfrageoptimierung, Zwischenspeicherung von semantischen Abfragen für RDF, der Schlussfolgerung, des Ontology-Diagramms und der Benutzerinteraktion.

Abstract

Back in 1945, Vannevar Bush coined the idea of Memex as “a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory” [Bus45]. The SemanticLIFE project is an attempt to come a step closer to Vannevar Bush’s vision. Recently we can observe a mushrooming of new projects aiming at some of the goals of Bush’s innovative ideas. This is mainly caused by the racy technological development which opens new realization potentials. An indicator for the narrowing of the discrepancy between the visions of Memex versus its realization is the announcement of “Memories for life - Managing information over a human lifetime” as one of the seven Grand Challenges for Computing Research [FR03] by the UK Computing Research Committee (<http://www.ukcrc.org.uk/>). The ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (<http://research.microsoft.com/CARPE2004/>) is another indicator for the maturity of our present time to implement such systems.

Towards the goals of personal information storage and retrieval of all one’s data throughout a lifetime, researchers consider continuous archival and retrieval of all media relating to personal experiences including emails, contacts, appointments, web browsing history, documents, IM logs, phone calls, etc. The challenging issues are how to extract useful knowledge from this rich library of information; and how to use this knowledge effectively. The SemanticLIFE’s *Virtual Query System*, which is presented thorough this thesis, is an approach for mentioned issues and by providing an innovative way of user-oriented query formulation, so-called “front-end” approach, to the SemanticLIFE Personal Information Management (PIM) system. With support of the Virtual Query System, the SemanticLIFE user is supported in issuing unambiguous queries to retrieve the rich semantic information from his/her historical personal data.

Most often when developing similar PIM systems, researches focus on back-end issues, i.e. capturing all data sources, integrate and then store them in huge repositories. For this purpose it is necessary to map the ontologies of the various data sources into a common ontology of the system. However, users are confronted with the lack of knowledge concerning the stored information inside the system, and they would formulate ambiguous requests, so that many barriers have to be overcome before the system could deliver the demanded results.

My thesis is aiming at a design of the innovative features of our Virtual Query System. This query system is based on a front-end approach allowing the user to retrieve the information of interest from huge ontology-based repositories in an efficient

way. The conception of this query system which is primarily based on the reduction of semantic ambiguities of user query specifications at the very early stage of the retrieving process; and continually guide the user in query process using a set of query templates based-on the user's querying context as well as the *virtual information*, say *context ontology*. This approach integrates many research efforts from the area of the Semantic Web, query refinement, semantic query caching for RDF data, inference, ontology mapping, and user interaction.

Acknowledgement

Pursuing a PhD program is an enjoyable trying progress where dissertation is a significant checkpoint's deliverable. The work in this dissertation has been conducted within three and half years studying at the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. However, without help, support, and encouragement of several persons I would never have been able to finish this work.

First of all, I would like to express my sincere gratitude to my supervisor, Prof. A Min Tjoa for his inspiring and encouraging way to guide me to a deeper understanding of knowledge work, and his valuable guidance during the whole work with this dissertation. He showed me different ways to approach a research problem, broaden my view to investigate the topic, and the need to be persistent to accomplish any goals. More important, I deeply indebted for his spiritual supports which help me passing any difficulties to reach the target.

I would like to thank Prof. Roland Wagner at Johannes Kepler University of Linz, my second supervisor for his guidance and comments to my work.

Thanks to all my colleagues at the Institute of Software Technology and Interactive Systems for providing a good working collaboration atmosphere as well as further debates and discussion in many topics. Especially, my obligation is given to all members of the SemanticLIFE Team from the day one, Andi Rauber, Alexander Schatten, Monika Lanzberger, Amin Andjomshoaa, Shuab Karim, Shah Khusro, Khabib Mustofa, Khalid Latif, Ahmed Mansoor and Nguyen Manh Tho. I also would like to send my appreciation to Mrs. Maria Schweikert, and Ing. Michael Schadler for their support in providing me the perfect working environment.

Last but not least, I am greatly indebted to my devoted wife Hanh-Tien and my lovely daughter Thieu-Anh for supporting me with love and spiritual understanding. Their love and support without any complaint or regret has enabled me to complete this dissertation.

This work has been financed out of the means of the Austrian Development Cooperation in framework of the North-South Dialog Scholarship Program (Grant No. 1186-4/EZA/2002), and generously supported by ASEA-UNINET and the Austrian National Bank within the frameworks of the project "Application of Semantic Web Concepts for Business Intelligence Information Systems" (Grant No. 11284).

Vienna, February 2007
Hoang Huu Hanh

Contents

List of Figures	x
List of Tables	xi
List of Listings	xii
Abbreviations	xiii
1 Introduction	1
1.1 The Web of Today	1
1.2 Knowledge Access and The Semantic Web	3
1.2.1 Limitations of Current Search Technology	3
1.2.2 Role of Semantic Technology	4
1.3 ‘Memories for Life’ and SemanticLIFE	5
1.4 A Brief Introduction to Our Approach	6
1.5 Thesis Outline	7
2 Semantic Web Fundamentals	9
2.1 The Semantic Web	9
2.2 The Goals of the Semantic Web	10
2.3 Describing Data with RDF	10
2.3.1 RDF	10
2.3.2 Terminology	11
2.3.3 Identifying Resources	12
2.4 Ontologies	12
2.4.1 Ontologies	12
2.4.2 Ontology Languages	13
2.5 Summary	14
3 SemanticLIFE: The Personal Digital Memory Project	15
3.1 Motivation	15
3.2 Related Work	16
3.3 ‘SemanticLIFE’	18
3.4 System Architecture	20
3.4.1 Design Principles	20

3.4.2	Internal Communication	21
3.4.3	Data Acquisition	21
3.4.4	Analysis Module	22
3.4.5	Storage and Indexing Module	23
3.4.6	Search and Query Functionality	24
3.4.7	Extensibility and Persistence of Information and Semantics	25
3.4.8	Client Interaction	25
3.5	Summary	26
4	Ontology-based Query Systems: The State of The Art	27
4.1	Introduction	27
4.2	Classification Criteria	27
4.3	Search Strategies with Ontology Support	28
4.3.1	Ontology-Enhanced Keyword Search	28
4.3.2	Ontology-based Multi-facet Search	29
4.3.3	Native Ontology-based Search	30
4.4	Query Formulation	32
4.5	Query Refinement	34
4.5.1	Query Ambiguity Discovering	34
4.5.2	Query Refinement	35
4.5.3	Ranking and User Interaction	37
4.5.4	Inference	37
4.6	Methodologies in Common	38
4.6.1	Role of Ontologies	38
4.6.2	Query Refinement	38
4.6.3	Keyword-Concept Mapping	39
4.6.4	Graph Patterns	39
4.6.5	Inference	39
4.6.6	Fuzzy Concepts, Relations, and Logics	40
4.7	Discussion and Summary	40
5	The Virtual Query System	42
5.1	Motivation	42
5.2	Virtual Query System Enabled SemanticLIFE	43
5.2.1	Information Retrieval in the SemanticLIFE Framework	43
5.2.2	Aims of The Virtual Query System	44
5.3	The Virtual Query System	44
5.3.1	The Architecture	44
5.3.2	The Virtual Query System Workflow	47
5.4	Summary	48
6	The Virtual Query Language	50
6.1	Introduction	50

6.2	VQL - The VQS's Virtual Query Language	51
6.2.1	The Goals of the VQL	51
6.2.2	The Syntax of VQL	52
6.2.3	Operators and Expression in VQL	55
6.2.4	The VQL Query Types	56
6.2.5	VQL Query Results Format	58
6.2.6	Well-formed and Validated VQL Queries	61
6.3	Query Operators of VQL	62
6.3.1	GetInstances Operator	63
6.3.2	GetInstanceMetadata Operator	63
6.3.3	GetRelatedData Operator	64
6.3.4	GetLinks Operator	64
6.3.5	GetFileContent Operator	66
6.4	VQL Parser: Query Languages Mapping	67
6.4.1	Expression Mapping	67
6.4.2	Syntax Mapping	68
6.4.3	Semantic Mapping	69
6.5	Summary	70
7	An Innovative Query Formulation	71
7.1	Introduction	71
7.2	The Virtual Data Component	72
7.2.1	The Goals	72
7.2.2	Metadata Storage Sources Collecting	72
7.2.3	Context-based Support of the VDC	73
7.3	Context-based Query Formulation in the VQS	75
7.3.1	The VQL Query Template	75
7.3.2	VQS User Context	77
7.3.3	VQS Query Map	78
7.3.4	Context-based Querying	78
7.3.5	Context-based Query Results Representation	79
7.4	Semantic Navigation with the VQS	80
7.4.1	VQS Semantic Traces	80
7.4.2	Context-based Navigation	80
7.5	Summary	81
8	Implementation Results	82
8.1	The SemanticLIFE Infrastructure	82
8.1.1	A Short History	82
8.1.2	Current Architecture	84
8.2	SemanticLIFE's SOPA	85
8.2.1	SOPA – The Service-Oriented Plug-in Architecture	85
8.2.2	The SemanticLIFE's Service Bus	85

8.2.3	Services Pipeline	88
8.3	The Virtual Query System: Specifications	89
8.3.1	VQS Workflow	90
8.3.2	The Used Techniques	91
8.3.3	The VQS Plugins	92
8.4	The Virtual Data Component	96
8.5	The Context-based Querying Feature	97
8.5.1	VQL Query Templates	97
8.5.2	The Context-based Querying	98
8.6	Application Scenarios	99
8.6.1	Personalized Project Management Scenario	99
8.6.2	The Personal E-Government services	100
8.7	Summary	101
9	Conclusion	102
9.1	The SemanticLIFE Framework	102
9.2	The Virtual Query System	103
9.3	Future Work	104
	Bibliography	106
	Curriculum Vitae	114

List of Figures

1.1	MyLifeBits Store Diagram	6
2.1	An RDF triple example	11
2.2	An Example of a University Hierarchy	13
3.1	The Architecture of the SemanticLIFE Framework	21
4.1	A Phase of GRDL - ICS-Forth [ACK04]	32
4.2	SEWASIE - graphical query formulation	33
5.1	The VQS Enabled SemanticLIFE Information Retrieval	43
5.2	The Components of the Virtual Query System	45
5.3	The Workflow of the Virtual Query System components	48
6.1	The schema for general VQL queries	52
6.2	The VQL query is validated.	61
6.3	The VQL query is not validated.	62
6.4	The Expression Tree.	68
7.1	A Fragment of the SemanticLIFE's Datafeeds Ontology	72
7.2	A Example of the Virtual Data Component Ontology	73
7.3	The schema of the VQL query template	76
7.4	An Example of Context-based Querying	79
7.5	The Process of Returning Query Results in the VQS	79
8.1	The first architecture design of the SemanticLIFE framework	83
8.2	The New Architecture of the SemanticLIFE Framework	84
8.3	An query service extension in the SemanticLIFE framework	87
8.4	Service transparency in the SOPA architecture	89
8.5	VQS Workflow in its UML Sequence Diagram	90
8.6	The Declaration of the Query Execution Plugin	93
8.7	The Declaration of the VQS Components Plugin	94
8.8	The Declaration of the Query Interface Plugin	94
8.9	Project Management Context Ontology Diagram	96
8.10	The Graphical User Interface of the Virtual Data Component	97
8.11	UML Sequence Diagram of the VQS Context-based Query	98

List of Tables

4.1	Summary of Ontology-based Query Systems	40
-----	---------------------------------------------------	----

List of Listings

6.1	An Example of a XML-based VQL Query	53
6.2	An example of VQL SCHEMA query	57
6.3	An example of the VQL RDF query type	57
6.4	The XML VQL Query Result	58
6.5	The TEXT VQL Query Result	59
6.6	The JASON VQL Query Result	60
6.7	The VQL RDF Query Schema	61
6.8	An Example of the VQL GetInstanceMetadata Query Operator .	63
6.9	An Example of the VQL GetRelatedData Query Operator	64
6.10	An Example of the VQL GetLinks Query Operator	65
6.11	An Example of the VQL GetFileContent Query Operator	66
7.1	A VQL Query Template Example	77
8.1	The business services extension-point schema	86
8.2	Abridged version of a service extension description	87
8.3	Calling a service plugged into the Services Bus	87
8.4	A Simple Pipeline	88
8.5	The Query UI Plugin Listing	95

Abbreviations

ACM	Association for Computer Machinery
AI	Artificial Intelligence
API	Application Programming Interface
B2B	Business To Business
B2C	Business To Customer
DAML	DARPA Agent Markup Language
DBMS	Database Management System
IEEE	Institute of Electrical and Electronics Engineers
IEEE CS	IEEE Computer Society
JAR	Java Archive
J2EE	Java Platform 2 - Enterprise Edition
J2SE	Java Platform 2 - Standard Edition
JDBC	Java Database Connectivity
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RDQL	RDF Query Language (Jena)
RQL	RDF Query Language (Sesame)
RuleML	Rule Markup Language
RUP	Rational Unified Process
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	Simple Protocol And RDF Query Language
SWRL	Semantic Web Rule Language
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	eXtension Markup Language
XPath	XML Path Language
XQuery	XML Query Language
XSD	XML Schema Definition
XTM	XML Topics Maps

1 Introduction

1.1 The Web of Today

The World Wide Web (WWW or Web) has changed the way people communicate with each other and the way business is conducted. It lies at the heart of a revolution that is currently transforming the world toward a knowledge economy and, more broadly speaking, to a knowledge society [AvH04]. This development has changed the way we think of computers, i.e. they were originally used for computing; nowadays, their foremost use is currently for information processing and text processing.

The today's WWW content is mostly suitable for human beings. Typical current uses of the WWW involve people's seeking and using information, searching for and getting in touch with other people, reviewing catalogs of online stores and filling out product order forms [AvH04].

These activities are not particularly well supported by software tools. Apart from the existence of links that establish connections between documents, the main valuable utilities are search engines. The keyword-based search engines, such as Yahoo!¹ and Google², are the main tools for using today's Web. It is clear that WWW would not have been the huge success as it has without the search engines. However, there are still problems associated with their use according [AvH04]:

- High recall, low precision.
- Low or no recall.
- Results are highly sensitive to vocabulary.
- Results are single WWW pages.

¹ Yahoo!, <http://www.yahoo.com/>

² Google, <http://www.google.com/>

Currently, the main obstacle to provide better support to Web users is that the meaning of Web content is not machine-accessible. Of course, there are tools that can retrieve texts, split them into parts, check the spelling. However when it comes to interpreting sentences and extracting useful information for users, the capabilities of current software are still very limited [AvH04]. For instance, it is simply difficult to differentiate the meaning of two following sentences

I am a student of computer science.

and

I am a student of computer science, you may think.

With the current text processing method over the Web pages at present, how can this problem be solved or improved at least? One solution is to use the content as it is represented today and to develop increasingly sophisticated techniques based on artificial intelligence and computational linguistics. This approach has been followed for some time now, but despite some advances the task still appears too ambitious [AvH04].

An alternative approach also according to [AvH04] is to represent WWW content in a form that is more easily *machine-processable* and to use intelligent techniques to take advantage of these representations. This idea refers to the plan of revolutionizing the WWW as the *Semantic Web* initiative. It is important to understand that the Semantic Web will not be a new global information highway parallel to the existing World Wide Web; instead it will gradually be evolved of the existing WWW [AvH04].

The Semantic Web is spread out by the World Wide Web Consortium (W3C). The main factor of the Semantic Web initiative is Tim Berners-Lee, the person who is famous at the invention the World Wide Web in the late 1980s. He expects from this initiative the realization of his original vision of the Web, a vision where the meaning of information played a far more important role than it does in today's WWW [BL98, BL99, BLHL01].

The Semantic Web's development has been pushed forward with a lot of industry motivation such as Oracle with the Semantic Technologies Center³ and HP Labs with the Semantic Web Research Programme⁴; and governments are investing heavily in this area. The U.S. government has established the DARPA Agent Markup Language (DAML) Project, and the Semantic Web is among the key action lines of the European Union's Sixth Framework Programme⁵.

³ http://www.oracle.com/technology/tech/semantic_technologies/

⁴ <http://www.hpl.hp.com/semweb/>

⁵ <http://ec.europa.eu/research/fp6/>

1.2 Knowledge Access and The Semantic Web

1.2.1 Limitations of Current Search Technology

Query Formulation

In general, when specifying a search, users enter a small number of terms in the query. Yet the query describes the information need, and is commonly based on the words that people expect to occur in the types of document they seek. This causes a fundamental problem, in which not all documents use the same words to refer to the same concept. Therefore, not all the documents that discuss the concept will be retrieved by a simple keyword-based search.

Furthermore, query terms may have multiple meanings (polysemy). As conventional search engines cannot interpret the sense of the user's search. The ambiguity of the query leads to the retrieval of irrelevant information. Despite the problems of query ambiguity can be overcome to some degree by careful choice of additional query terms, there is many people may not be prepared to do this [DSe06].

Lack of Semantics

Conversing to the problem of polysemy is the fact that conventional search engines that match query terms against a keyword-based index will fail to match relevant information. This happens if the keywords used in the query are different from those used in the index, despite having the same meaning (index term synonymy). Although this problem can be overcome to some extent through thesaurus-based expansion of the query [BRA05, WR05], the resultant increased level of document recall may result in the search engine returning too many results for the user to be able to process realistically [DSe06].

In addition to an inability to handle synonymy and polysemy, conventional search engines are unaware of any other semantic links between concepts. For example, considering the following query:

'football club' Europe 'David Gill' director

In this example, the user might require documents concerning a football club in Europe, a person called David Gill, and a board appointment. However, we note that a document containing the following sentence would not be returned using conventional search techniques:

'At a meeting in September 2003, the board of Manchester United PLC appointed David Gill as CEO'

In order to be able to return this document, the search engine would need to be aware of the following semantic relations:

‘Manchester United PLC is a company, which possesses the football club called Manchester United FC; Manchester is located in the UK, which is a part of Europe; A CEO is a kind of director’.

Lack of Context

Many search engines fail to take into consideration aspects of the user’s context to help disambiguate their queries. User context would include information such as a person’s role, department, experience, interests, project work and so on.

The real practices have been illustrated this matter. Any person working in a particular business searching for information is presented with numerous irrelevant results if they simply enter few keywords for his/her query. More relevant results are only returned if users modify their query to include further search terms to indicate the part of the business in which they work. According to [iPr06], users are in general unwilling to do this.

1.2.2 Role of Semantic Technology

Semantic technologies have the potential to offer solutions to many of the limitations described above, by providing enhanced knowledge access based on the exploitation of machine-processable metadata. Central to the vision of the Semantic Web are ontologies. These facilitate knowledge sharing and reuse between agents, be they human or artificial. They offer this capability by providing a consensual and formal conceptualization of a given domain. Information can then be annotated with respect to an ontology. This leads to distributed, heterogeneous information sources being unified through a machine-processable common domain model (ontology).

Search engines solely based on conventional information retrieval techniques tend to offer high recall but lower precision. The user is faced with too many results and many results that are irrelevant due to failures concerning polysemy and synonymy.

The use of ontologies and associated metadata can allow the user to more precisely express their queries thus avoiding the problems identified above. Users can choose ontological concepts to define their query or select from a set of returned concepts following a search in order to refine their query. They can specify queries over the metadata and indeed combine these with full text queries if desired.

Furthermore, the use of semantic web technology offers the prospect of a more fundamental change to knowledge access. Current technology supports a process wherein the user attempts to frame an information need by specifying a query in the form of either a set of keywords or a piece of natural language text. Having submitted a query, the user is then presented with a ranked list of documents of relevance to the query. However, this is only a partial response to the user's actual requirement which is for information rather than lists of documents.

It is suggested here, therefore, that the future of search engines lies in supporting more of the information management process, as opposed to seeking incremental and modest improvements to relevance ranking of documents. In this approach, software supports more of the process of analyzing relevant documents rather than merely listing them and leaving the rest of the information analysis task to the user. Users also need information relevant to their interests and to their current context. They need not to find just documents, but sections and information entities within documents and even digests of information created from multiple documents.

1.3 'Memories for Life' and SemanticLIFE

People are capturing and storing an ever-increasing amount of information about themselves, including emails, web browsing histories, digital images, and audio recordings. This tsunami of data presents numerous challenges to computer science, including: how to physically store such "digital memories" over decades; how to protect privacy, especially when data such as photos may involve more than one person; how to extract useful knowledge from this rich library of information; how to use this knowledge effectively, for example in knowledge-based systems; and how to effectively present memories and knowledge to different kinds of users. The unifying grand challenge is to manage this data, these digital memories, for the benefit of human life and for a lifetime [FR03].

The idea of Bush's article became very popular and, nowadays, that "big" idea is not something impossible because many systems are geared towards its realization. Two of those applications are Haystack⁶ and MyLifebits⁷ [GBL03b, GBL⁺02b]. Haystack was developed with the aims to make users comfortable in organizing various information sources and make relationships of information transparent in the way that make the most sense to them. It provides easier management of email, schedule (calendar), contact person, photo, browsing and relationship exploration. MyLifebits is a system close to the main idea of Memex. The system provides features for managing information coming from almost most

⁶ <http://haystack.lcs.mit.edu/>

⁷ <http://research.microsoft.com/barc/mediapresence/MyLifeBits.aspx>

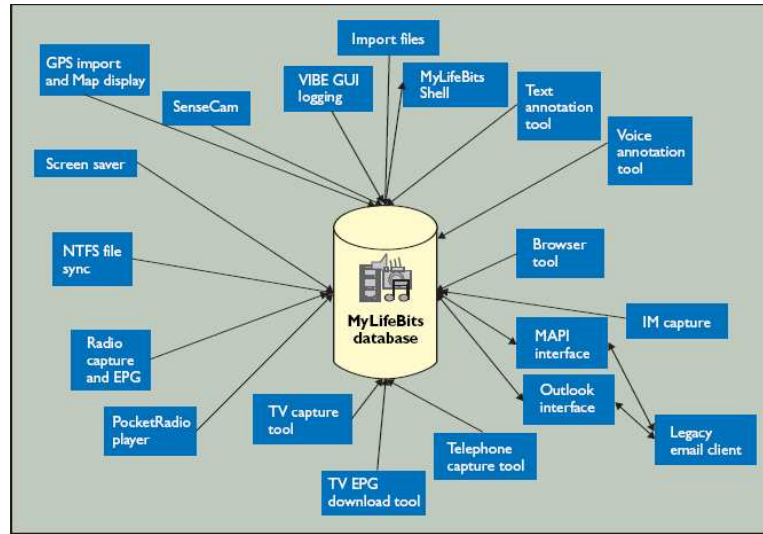


Figure 1.1: MyLifeBits Store Diagram

common data sources of current modern life. It claims that currently it can hold information of 25 item types including: web browsing, chat session, contacts, documents, email, events, photos, music, and video (Figure 1.1). Items can be linked implicitly via time property (enable visualization by timeline) or explicitly using *typed-link* [GLB06].

The efficient information retrieval in the mentioned systems is a vital issue for their success. Details of these problems are discussed in Chapter 4.

1.4 A Brief Introduction to Our Approach

Our approach originates from the user-side manner in trying to formulate unambiguous requests as early as possible during the querying process. The principle of the approach follows “*better known, clearer request*” and “*customizing than generating*”. If users are aware of what information they possess, they could ask precise queries against their stored data. This helps the query refinement process of the system by eliminating ambiguities at a very early stage of the query process.

Secondly, the user’s nature of asking question is that he/she often does not know what he/she is looking for; but he/she remembers some concepts about related information he/she is looking for. This leads us to a way of querying (browsing/navigating) the system by using re-defined query templates (patterns) based-on his/her querying context. This would help the user not to be embarrassed in a new phase of query formulation.

The difficulty of query formulation appears not only in the initial phase but it continues in the query process or query refinement. During the query process, the user is asked for new requests using the new knowledge to get the information of interest. In order to ease the user from thinking of new constraints of their queries; we propose a new way based on the users' nature: i.e. prefer to customize the query patterns to make new queries. We trace the context of the user's query process and recommend the user the appropriate query patterns matching up his/her query context.

These approaches resulted in our query system the *Virtual Query System* (VQS) for the SemanticLIFE framework with a query language called the Virtual Query Language for the VQS; and with a new way of query formulation entitled *pattern- and context-based querying process*. The details of our approaches and related research issues are discussed in the next chapters in this thesis.

1.5 Thesis Outline

The thesis structure is organized as follows:

Part I. Fundamentals and Motivation

In this part, we give a brief overview of the Semantic Web, the background concepts and other related work, current ontology-based systems. This part includes the following chapters:

Chapter 1 gives a general overview of the topic and related issues that covers the dissertation. It also give an introduction to the approach of our research.

Chapter 2 describes the concepts and motivation of the Semantic Web. This chapter highlights technologies that are useful for research of ontology-based querying.

Chapter 3 introduces the SemanticLIFE Personal Digital Memory project with the motivation and its general infrastructure; with our approach which is the base for developing a query system with new innovative features in simplifying the complex query formulations.

Chapter 4 presents a state-of-the-art of the current ontology-based query systems similar to our approach. This survey is based on an deep investigation of techniques and research directions of current approaches.

Part II. Virtual Query System

Chapter 5 describes a design overview of our Virtual Query System (VQS) with the innovative features such as context ontology, i.e. query and context- and query pattern-based querying.

Chapter 6 introduces a query language for the VQS in the attempt to support users in generating easy queries. The language is called Virtual Query Language.

Chapter 7 gives an introduction of an innovative approach in query formulation: the context-based querying process and several related techniques from our approach such as query templates (so-called named queries or query patterns), query map and user query context are described.

Chapter 8 After establishing the theoretical foundation, this chapter deals with the implementation and evaluation of the VQS and related issues. We first describe the realization of the platform independent and scalable SemanticLIFE infrastructure along with facilities, before we introduce the design and implementation of the VQS query processing architecture. In addition, Chapter 8 also demonstrates the practical benefits of the VQS by outlining three application scenarios.

Part III. Conclusion and Outlook

Chapter 9 presents a conclusion about what we have achieved, i.e. the SemanticLIFE framework and the VQS approach. Future work is sketched for further steps ahead.

2 Semantic Web Fundamentals

2.1 The Semantic Web

The dream of a more powerful web has brought the idea of attaching semantic to web pages in some way. The Semantic Web idea was then introduced and popularized by Tim Berners-Lee, Ora Lassila and Jim Hendler in their article “The Semantic Web” [BLHL01] in 2001. The term *Semantics Web* itself was actually already mentioned a few years before Tim Berners-Lee did, but active responses and efforts by many researches boomed only after the publication of [BLHL01]. It is stated in the article that:

“The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”

The word *semantic* implies meaning or as defined in WordNet, “relating to the study of meaning and changes of meaning”. The term *semantic* in the Semantic Web shows that not only people but also computers can discover the meaning of data on the Web. Nevertheless, most tasks are done by people such as inferring most meaning on the Web, reading web pages and the labels of hyperlinks, and writing specialized software to work with the data. The phrase *Semantic Web* is the vision in which all computers and people can access data throughout the World Wide Web to achieve useful goals for users.

Software is already used to satisfy user requirements on the Web; however the difference relies on the “use” word. We, not computers, directly do tasks such as surfing the Web, do business on websites, read the labels on hyperlinks, and select links. Hence, it would be quicker and more convenient if a process proceeded by itself could be launched. This is also the main objective of the Semantic Web – to turn such capabilities into wider use.

2.2 The Goals of the Semantic Web

The Semantic Web and Semantic Web technologies provide us a promising way to managing information and processes. The fundamental principle of which is the creation and use of semantic metadata.

For information, metadata can exist at two levels. Firstly, they may describe a document, e.g. a web page; or part of a document, e.g. a paragraph. On the other hand, they may describe entities within the document. In any case, metadata is a semantic nature, i.e. it tells us about the content of a document (e.g. its subject matter, or relationship to other documents) or about an entity within the document. In contrary, the metadata on today's Web, encoded in HTML, purely describes the format in which the information should be presented: using HTML, we can specify that a given string should be displayed in bold or red font but we cannot specify that the string denotes a specific meaning.

After analyzing of disadvantages of today's Web and contrary advantages of semantic metadata, [DSe06] goes to a conclusion that using semantics we can improve the way information is presented. At its simplest, instead of a search providing a linear list of results, the results can be aggregated by meaning. For example, a search for 'Eclipse' can provide documents clustered according to whether they are about a software development platform, a universal event, or different subjects all together. However, we can go further than this by using semantics to merge information from all relevant documents, removing redundancy, and summarizing where appropriate. Relationships between key entities in the documents can be represented. Supporting all this is the ability to reason, that is to draw inferences from the existing knowledge to create new knowledge.

The use of semantic metadata is also crucial for integrating information from heterogeneous sources, whether within one organization or across organizations [DSe06]. Typically, different schemas are used to describe and classify information, and different terminologies can be used within the information. By creating mappings between the different schemas, it is possible to create a unified view and to achieve interoperability between the processes using the information.

2.3 Describing Data with RDF

2.3.1 RDF

The Resource Description Framework (RDF) is an extremely flexible technology, capable of addressing a wide variety of problems. RDF has different aspects of

specification with our own interpretations of what it is and what it is good for. In the role of describing data and metadata, RDF (and other potential languages for the role) includes the following capabilities:

- Able to describe most kinds of data that will be available
- Able to describe the structural design of data sets
- Able to describe relationships between bits of data

Toward these ends, RDF uses a simple data model. Basically, there are entities called *resources*, and there are *statements* that can be made about those resources. A single statement links two resources. These statements are like simple sentences that have a **subject-verb-object** structure, such as “*Tien lives in Vienna*”. ‘*Tien*’ is the subject, ‘*lives in*’ is the verb, and ‘*Vienna*’ is the object. In short, it represents a data model. Naturally, complications arise as the model is adapted to practical applications, but fundamentally RDF is about simple statements that describe information about specific subjects.

2.3.2 Terminology

In RDF, a statement is represented by a *triple*: the subject of a statement is in fact called the *subject*, the equivalent of a verb is called the *predicate*, and the remaining part is called the *object*. Other terms are also in common use: *property* instead of predicate, and *value* instead of object (because many RDF statements assign property values to their subjects).

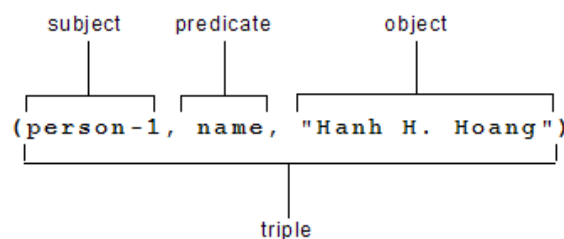


Figure 2.1: An RDF triple example

Figure 2.1 depicts the structure of an RDF triple. The value of a property can be a simple value, like an ordinary number or a string of characters such as “*Hanh H. Hoang*” in the diagram. Such values are called *literals*. The value of a property can be either a literal or another resource, as appropriate. RDF has a way of indicating whether a literal value has a data type, meaning that it is intended to be, for example, an integer or a chunk of XML. A literal cannot be the subject of a statement.

A collection of RDF data has no standard name. It is sometimes called an *RDF store*, an *RDF data set*, a *knowledge base*, or even a *database*.

2.3.3 Identifying Resources

To be widely usable over the Web, RDF needs to be able to identify the entities it describes in a standard, widely used manner so that a host of systems on the Web can also refer to them. Older systems, such as conventional databases, have no standard way to identify their (equivalent of) subjects across systems and networks.

To identify resources—that is, what RDF makes statements about—RDF uses *URI* (Uniform Resource Identifier) *references*. A URI can be used to identify a concept, a tangible thing that cannot be downloaded, or a chunk of data that can be retrieved over a network. A URI reference is a URI plus optional characters, such as the so-called *fragment identifier* (the part that follows the # sign after a URI, if any). RDF has rules about how to construct related URIs so that they can be used conveniently when RDF data is exchanged.

2.4 Ontologies

2.4.1 Ontologies

The core of all Semantic Web applications is the use of ontologies. There are several definitions of an ontology, however the following definition is commonly agreed by the community: ‘*An ontology is an explicit and formal specification of a conceptualization of a domain of interest*’ [Gru93]. This definition stresses two key points: that the conceptualization is formal and hence permits reasoning by computer; and that a practical ontology is designed for some particular domain of interest.

In general, an ontology formally describes a domain of interest; and typically, an ontology consists of a finite list of terms and the relationships between these terms [Pas05]. The *terms* denote important *concepts* (*classes* of objects) of the domain.

For example, staff members, students, courses, lecture theaters, and disciplines are some important concepts in a university context. The *relationships* typically include hierarchies of classes. A hierarchy specifies a class *C* to be a

subclass of another class C' if every object in C is also included in C' . For example, all faculty are staff members. Figure 2.2 shows a hierarchy for the university domain [Pas05].

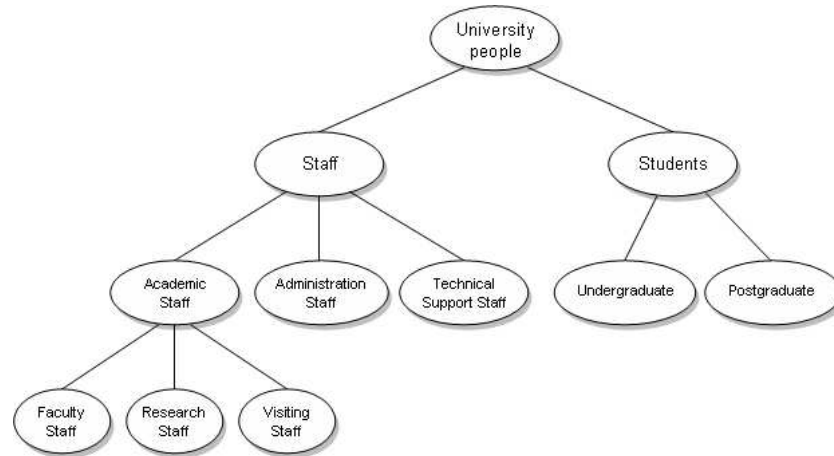


Figure 2.2: An Example of a University Hierarchy

In addition, according to [Pas05], apart from subclass relationships, ontologies may include information:

- properties (e.g. X teaches Y)
- value restrictions (e.g. only faculty members can teach courses)
- disjointness statements (e.g. faculty and general staff are disjoint)
- specification of logical relationships between objects (e.g. every department must include at least ten faculty members)

In short, an ontology consists of concepts (also known as classes), relations (properties), instances and axioms. Hence a more concise definition of an ontology is as a 4-tuple $\langle C, R, I, A \rangle$, where C is a set of concepts, R a set of relations, I a set of instances and A a set of axioms [SS04].

2.4.2 Ontology Languages

Many efforts in over the world (almost in Europe and the U.S.A.) on defining ontology languages, e.g. RDF/RDFS and DAML+OIL, has now converged under the W3C, to introduce a Web Ontology Language, OWL¹ [DSe06].

OWL is built on the Resource Description Framework (RDF) which is essentially a data modeling language, also defined by the W3C. The OWL language

¹ Web Ontology Language, <http://www.w3.org/2004/OWL/>

provides mechanisms for creating all the components of an ontology: concepts, instances, properties (or relations) and axioms. Two sorts of properties can be defined: object properties and datatype properties. Object properties relate instances to instances. Datatype properties relate instances to datatype values, for example text strings or numbers. Concepts can have super and subconcepts, thus providing a mechanism for subsumption reasoning and inheritance of properties. Finally, axioms are used to provide information about classes and properties, for example to specify the equivalence of two classes or the range of a property.

Indeed, OWL has three types: OWL Lite, OWL DL (Description Logic) and OWL Full. OWL Lite provides a limited feature set, though decent for many applications. OWL DL, a superset of OWL Lite, is based on a form of first order logic known as Description Logic. OWL Full, a superset of OWL DL, removes some restrictions from OWL DL but at the price of introducing problems of computational tractability. Practically, OWL Lite is very good enough for many tasks.

2.5 Summary

To conclude this chapter, we borrow words from [Pas05] to summarize the promising features of the Semantic Web technology and the views derived from.

“The Semantic Web is not an integrated technology. It is a concept of how computers, people, and the Web can work together more effectively than is possible now. Because it is visionary, it has no one definition. In fact, we saw a staggering array of notions such as the machine-readable-data view, the intelligent agents view, and many more in” [Pas05]. Basically, all the views assume that computers will be able to process data that today is mainly accessible to people and use this data to perform tasks that help people.

3 SemanticLIFE: The Personal Digital Memory Project

3.1 Motivation

“A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory” - Vannevar Bush in [Bus45]

The ‘SemanticLIFE’ project, which is introduced in this chapter, is an attempt to come a step closer to Vannevar Bush’s vision of the Memex from the year 1945. Recently we can observe a mushrooming of new projects aiming at some of the goals of Bush’s innovative ideas. This is mainly caused by the racy technological development which opens new large realization potentials. An indicator for the narrowing of the discrepancy between the visions of Bush’s memex versus its realization is the announcement of ‘Memories for Life’¹ – Managing information over a human lifetime as one of the seven Grand Challenges for Computing Research by the UK Computing Research Committee [FR03]. The announcement of the First ACM Workshop on Continuous Archival and Retrieval of Personal Experiences in October 2004 is another indicator for the maturity of our present time to implement such systems.

Another significant development which narrows the gap toward the realization of Memex-like systems is the advent of the Semantic Web initiatives. As proposed by Tim Berners-Lee [BLHL01] the realization of Semantic Web would narrow this gap by the use of domain specific ontologies and their reuse.

The goal of our project is to build a Personal Information Management (PIM) system over a Human Lifetime using ontologies for the representation of semantics, let say the *Personal Digital Memory*. Basic ontological infrastructure re-

¹ <http://www.memoriesforlife.org/>

quired for applications development on top of the current generation of the Web are very well described in [FHLW03]. 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 our project - we 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’. The project as a whole covers aspects of Human Computer Interfaces, Databases, Data Mining, Information Retrieval, Security, Device Engineering, etc.

The idea of Bush’s article became very popular and, nowadays, that challenging idea becomes feasible because many systems are geared toward its realization. There have been many applications in the same track such as MyLifeBits from Microsoft, Haystack from Massachusetts Institute of Technology (MIT), e-Person from HP Labs Semantic Web Research, LifeStream of Yale University. Among them, Haystack and MyLifeBits are still being developed until now and some results have been archived. Details of these work will be presented in the related work section.

3.2 Related Work

In the area of digital memories research, a lot of work has already been carried out in some major projects. In this section we highlight some of their significant features.

MyLifeBits (Microsoft Research)

MyLifeBits² is a system closer to the real idea of Memex [GBL⁺02b, GBL03b]. The system provides features for managing information coming from almost most common data source of current modern life. It claims that currently it can hold information of 25 item types including: web browsing, chat session, contacts, documents, email, events, photos, music, and video [GLB06].

MyLifeBits is a system for storing all of one’s lifetime data on a PC. The guiding principles are: (a) collections and search must replace hierarchy for organization, (b) multiple visualizations, (c) easy annotations (d) the authoring tool should support reuse of external references [GBL⁺02a]. As an experiment, G. Bell has captured all his articles, books, cards, etc, and stored them digitally. He is now paperless, and is beginning to capture phone calls, instant message

² MyLifeBits, <http://research.microsoft.com/barc/mediapresence/MyLifeBits.aspx>

(IM) logs, television, and radio [GBL03a]. They have successfully incorporated multiple annotation types, and creation of stories which are helpful for the short term memory.

They are still trying to explore features such as versioning, document similarity ranking and faceted classification. Until now, they were more concerned with functionality, but now the future work is related with user interface (UI), advanced visualization techniques, data mining for search, new capture mode and devices, shared usage, security, privacy and social issues.

Haystack (MIT)

Haystack³ was developed with the aims to make users easy in organizing various information sources and make relationships of information easily and in the way that make the most sense to them. It provides easier management of email, schedule (calendar), contact person, photo, browsing and relationship exploration.

Haystack uses ontologies for information management [HKQ02]. Its ultimate goal is to provide high-quality retrieval. Primarily it is designed as a single machine single user tool so as to give a psychological illusion of privacy and security. The guiding design principles are: (a) generic handling of all types of information, (b) flexibility to define additional information types by the user, (c) ability to define the interaction objects and associated operations directly by the user and (d) ability to delegate certain information processing tasks to the agents.

Haystack has a typical three tier architecture [AKS99], i.e., a database layer, service layer and client layer. It also provides some implementation at the Trust layer of Semantic Web by using reification mechanism for RDF storage, and identifier strings as digital signatures during storage of RDF statements. The future developments include ontology conversion, enhanced query mechanisms using machine learning tools to improve retrieval, provide better interface for hybrid search, recommender system based upon user's interests.

e-Person (HP)

Developed by HP, an ePerson is a personal representative on the net that is trusted by a user to store personal information, and make it available under appropriate controls for shared working environments. HP's approach is focused on three principals, i.e., social filtering of information by the users themselves,

³ Haystack, <http://haystack.lcs.mit.edu/>

structured knowledge in terms of ontologies mutually agreed upon by the communities, and person-centric instead of being corporate-centric in terms of ownership, vocabularies and scaling [BCDD02].

The ePerson infrastructure is designed as a series of layers, i.e. transport layer (TCP/UDP and Jabber transports), knowledge-base access layer (remote access to RDF stores and services), Structure layer (modeling of RDF vocabularies using DAML), Knowledge sources layer (provides specific knowledge services such as classification servers, importing profiles from the history server and a discovery server), and Applications layer (reusable UI components, viewing tools for knowledge based access during development and the SnippetManager application itself).

Lifestreams (Yale University)

Lifestreams [FG96], is an academic project at Yale University. It is a personal store that uses a simple organizational metaphor, a time-ordered stream of documents combined with several powerful operators that replaces many conventional computer constructs (such as named files, directories, and explicit storage). Their work on the client side includes an X-Window client, command line interface and a PDA client.

The motivating ideas were, (a) storage should be transparent, (b) directories are inadequate as an organizing device, (c) archiving should be automatic, (d) the system should provide sophisticated logic for summarizing or compressing a large group of related documents on one screen, (e) reminding should be made more convenient, and (f) personal data should be accessible anywhere and compatibility should be automatic.

3.3 ‘SemanticLIFE’

Living systems have different characteristics like self-regulation of processes, reproduction and growth [Nic96]. Nevertheless, the relevant characteristics could be envisioned in a semantic way in personal knowledge management. Ontologies of personal life items grow and reproduce new ones with processes and services. These ontologies include information about our life objects such as documents, persons, places, organizations, events and tasks.

In the physical world, entities are usually interconnected, either by physical or by semantic means; in the latter case, the semantic meaning is added by human interaction (in an abstract sense) with the physical world. Life items in the system proposed in this thesis can be understood as information entities (in

some cases they are representations of such physical entities) stored according to ontologies in a semantic database, which are connected to other information entities according to their semantic meaning. Also ontologies ‘live’ in a way, as they develop and modify permanently during the system- and user- lifetime.

Current (Web-) technologies are highly efficient in processing data for human reception; that is, the transformation from data to information, the ‘generation of meaning’ is up to the human. A great deal of effort has already been made, and work is still going on to represent semantics explicitly on the Web. This is required to give computer systems the capability to enhance preprocessing of huge amounts of data for the user. It becomes more important as the ‘awareness radius’ of the contemporary knowledge worker and consumer is continuously increasing. This results from the observation, that users do not limit their information search to specific data repositories, like searching for an address or an event in a calendar any longer. The availability of databases under common or similar interfaces (like web-pages) creates the demand to express more complex queries demanding information aggregated from many different systems using different semantic concepts.

The proposed PIM Systems can significantly contribute in overcoming the common inherent human problems such as limited short term memory, memory loss, forgetfulness, high complexity of data etc. Therefore, it is useful for the system to be able to define and capture the user’s life-related events and takes or triggers appropriate action(s) for it. This process involves the following sub-processes:

1. Capture events and associated information
2. Process action associated with events (e.g., in the sense of an active database system)
3. Extract Metadata from the event, or allow the user to enrich the data manually with semantic meaning
4. Store the data including semantic context as ontology in an efficient manner
5. Allow the user to query the data or support the user directly or via associated applications and tools with context-sensitive information or action

The typical usage of such system can be illustrated with following example: Consider scientists, who work in a specific domain. They might be interested to get into contact with other researchers in the scientific community that (1) share the same interests or have similar problems (2) are publishing in similar conferences and (3) were recently active in the specific field of research (4) and speak a common language. The result of such a query could be the web pages and email addresses of the researchers coming into question.

It is clear that such problems can only be solved by querying a multitude of information resources like web pages, conference journals, scientific databases, email repositories, newsgroups and the like. Moreover, the system needs to ‘understand’ that entities differently labeled are identical in a semantic sense and also need to be able to ‘understand’ and solve specific issues like the fact, that some results are only valid in a specific interval of time or in a specific language and so on.

Additionally as described in [DHNS03], the system must be able to adjust to new user features derived from user interactions with the system or from the information being fed. Thus each user may have individual views and navigational possibilities for working with the system. From the technology perspective, new technologies emerge and older ones fade out. If a system has a too tight coupling with some technology, it may become obsolete with the change in technology. A layered approach that provides some extent of separation from the technology is more suitable, making the overall structure still working if there is a change in the technology or even in case of replacement by the newer ones.

3.4 System Architecture

3.4.1 Design Principles

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. As depicted in Figure 3.1, there is a bundle of components that form the complete SemanticLIFE framework such as the Data Feed Plugin using Google Desktop⁴ as input line to gather the personal data, the VQS Plugin is the query system of the system, and the Pipeline Plugin to compose the collective service calls.

⁴ Google Desktop, <http://desktop.google.com/>.

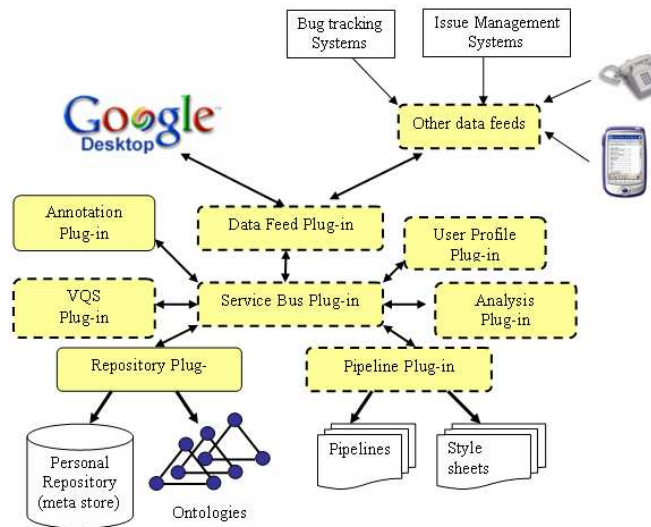


Figure 3.1: The Architecture of the SemanticLIFE Framework

3.4.2 Internal Communication

Due to the SOA-based and loosely coupling design, components of the SemanticLIFE system are plugins offering the services in form of the service extensions of the main and commonly service namely *Service Bus*. Service Bus performs service calls from and to plugins using the registered service extensions on it.

To compose complex solutions and scenarios from atomic services which are offered by SemanticLIFE plug-ins, the Service Oriented Pipeline Architecture (SOPA)⁵ 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.

3.4.3 Data Acquisition

Currently web-applications and frameworks are not designed to deal with semantic issues as described in the previous section, as mainly human users are analyzing and interpreting the data themselves. However, this makes the process of solving more complex problems which requires aggregated information from various sources time consuming, inconsistent, unreliable and hence inconvenient. So the need for a well-defined interface to data repositories on the web as well as personal data stores is undeniable [GC03].

⁵ JAX Innovation Award 2006 Proposal, <http://www.jax-award.com/>.

Therefore, all information entities associated with one's lifespan must be stored in an ontological way according to some already established metadata frameworks such as RDF and Topic Maps, to facilitate the semantic queries, life trails and processing of life events. Information items can be of various kinds such as documents, emails, images, audio or video streams.

Hence the first step in creating a 'SemanticLIFE' repository is to implement a powerful data acquisition module. Basically three different types of data sources are distinguished:

- Data acquired automatically and stored in a semantic data store
- Data acquired or enriched manually by the user
- External data sources that are invoked when needed, and are not imported into the semantic data store

The third type is, in the strict sense, not a data acquisition step. But it is important to understand, that there are data repositories that are not reasonable to import into the system, e.g., typically because they are external, fast changing, contain huge amounts of data and are highly structured by definition. Examples could be literature databases, enterprise information systems, company databases, web-search engines and so on. These external sources are invoked on demand (query) and a fitting ontological representation is generated by a plug-in defined in the system (see Figure 3.1).

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 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 triples with their ontologies and metadata. This repository is called a metastore.

3.4.4 Analysis Module

In order to efficiently extract the meta-information, the incoming messages are sent to the analysis interface in the first step. This is basically a plug-in mechanism that allows adding various analysis modules to pre-process messages of certain types. In these analysis steps meta-data is generated and added to the message. Depending on the message type, more than one analysis module might be invoked in processing a particular message. But it is important to understand

at that point, that no data is removed during these analysis steps. This is desired to guarantee that no original data is lost or modified, and the history of changes is preserved! Moreover it allows re-processing data already in the semantic store in the case, that more powerful analysis modules are available in the future.

As mentioned above, incoming messages may contain the information items, placed in a nested manner. Additionally, an information item can have some other information items attached with it, like an email with attachments. Also, an information item can have other information items embedded within, like images, audio or video clips within an HTML or PDF file. Consequently, all the information items have to be analyzed in a nested way for the extraction of meta-information available in their respective headers.

Of course, currently automatic information extraction is limited to messages or data sources that provide certain machine-readable structures. For example, it is still extremely difficult to extract structured information (description) about the content of a picture or a movie. Making manual annotations to information items being fed into the system will ultimately improve its quality. This should act as a complement to content analysis and automatic metadata extraction described earlier.

After all analysis steps are finished and the generated metadata is added to the message subsequently, this message is sent back to the message queue, which forwards it to the storage module.

3.4.5 Storage and Indexing Module

The Storage is responsible for extracting the XML-based metadata information from the message and writes it into the semantic data repository. Upon reception of a message from the Event Handler, the following steps are performed:

- Parsing of the message and extraction of metadata.
- Invoke the appropriate plug-in for writing to specific ontology framework, via Connector Interface.
- Updating indexes.

A basic consideration regarding the storage is, that the size of storage media is continuously increasing, which makes it possible to keep all data that were entered into the system. This is a very basic feature of the system, as it allows keeping a personal history from the data as well as from the semantic viewpoint.

Besides annotating entities with metadata, building connections with related entities is a crucial feature of the storage module. Such relationships can be built automatically or with human intervention leading to the concept of weak and

strong links respectively. The weak link creation could be carried out periodically or based upon some related events. For example, email objects can be related based on the senders, the receivers or automatically by the subjects. But also relations based on some criteria given by the user are needed. Later, the user can retrieve information along with other related information. Then it would be possible to retrieve a trail of documents, messages and pictures [GBL⁺02b].

The connections stored in the system should also have the ability of refreshing or updating itself in an automatic or semi-automatic manner. The updates occur as a result of changes in instance values, class hierarchy such as relations of classes and subclasses, kinds of properties in ontology and rules in the knowledge base.

3.4.6 Search and Query Functionality

Structured database systems like relational or object oriented database systems usually provide query mechanisms that allow powerful queries on highly structured data. As mentioned above, it is difficult to define highly structured queries (e.g., in a language like SQL) when a multitude of information systems are addressed or the information is only semi-structured. Hence the system must be capable to work with ‘weaker’ search terminology that has to be transformed into more specific queries by the system.

As the data is already stored semantically enriched (or the metadata is added ‘on the fly’ invoking external data sources respectively), it is possible to provide more powerful ‘imprecise searches’, that go far beyond ‘simple’ full-text indices and return information to the user in more meaningful, rich and intuitive ways. But the term ‘imprecise’ has two meanings: firstly, the generated queries are about undefined targets. Secondly, the target of the query is specified but there is ambiguity in the query. Therefore, the system has to solve these problems during query generation, by exploring the systems database and ontology repository and generate queries for a specific technology.

Finally these specific queries could be ontological query languages like SPARQL, RQL or RDQL for RDF. For nested and distributed sub-queries, many joins and complex subqueries would require ordering, re-ordering by the Query Optimization module, so that an optimal query execution plan is generated. Also, suitable search algorithms for non-ontological searches are to be used. Later, the generated queries could also be stored for possible reuse for system optimization.

The next step is query post processing: The received query output needs analysis and ranking to derive more precise results matched to user’s requests and preferences. Analysis and ranking would rely on specific calculations in terms of semantic distances and reasoning steps to give more concrete and accurate

results. To perform these tasks, the system has to refer to the set of rules and user preferences. For this purpose data from internal or external data sources are also taken into account. As the system keeps the complete history of entered data post processing needs to filter data according to the desired time-span for the problem domain.

After analysis and ranking of the query results, the system would aggregate the results before sending the result back. The processed query results can be presented to the user through presentation application that prepares the search result, e.g. for rendering in a web-browser. The query user interface needs to be designed in a way that the user is able to write and refine the queries in an iterative manner, which is on the back-end supported by the query engine of the server.

3.4.7 Extensibility and Persistence of Information and Semantics

The core of the system is based on its analysis and metadata extraction capabilities. New data sources emerge by the time and need to be treated by the system. It becomes a time consuming task to add support for newer data sources if the system has tight coupling with its components. A light weight asynchronous messaging based solution is required where new modules could be just plugged into the system without any change in the existing code.

Furthermore, openness is an extremely important issue considering systems that are designed for a lifetime acquisition of information and metadata [McB02]. Hence the ‘SemanticLIFE’ project is developed as an open source application, which uses various open source frameworks. Moreover for each module it is evaluated if open standards exist for the representation of data, semantics and data exchange.

3.4.8 Client Interaction

The messaging interface provides a standardized communication mechanism for various types of clients as described above. In the current prototypical implementations, the system is oriented on individual users. However, it is clear, that future (server) versions need to be implemented as multiuser systems.

This is an encouraging task, but the arising problems are mainly ‘typical’ problems of multiuser server systems. As the focus of our system is treatment of semantically enriched information, only a brief discussion of these issues is given here.

First of all user management and authentication requirements need to be implemented into the server system. As a consequence each message has to keep the information of the user (who performs a query for example). All sensitive processing units (mainly the storage and search module) then have to filter the messages or query results according to the user rights and roles defined for the user. This will be most probably an ontology by itself and will be stored in the server.

This user module might additionally maintain profiles of various types, such as users, devices, applications, tasks and interaction objects. This will help in presenting the results to the user in a way which is more personalized, accessible and fits the input/output capabilities of the device being used (thin, fat clients, etc.).

3.5 Summary

As SemanticLIFE is an evolving research project, we are not aiming at finalizing a commercial product. In the contrary, SemanticLIFE will be continuously improved in an evolutionary manner.

As many standards and tools are still under heavy development, our approach was and is to develop multiple prototypes with a similar functionality using different technologies. In this ‘evolutionary’ approach the best strategy will succeed and implemented into the “final” system.

As soon as technology and standard decisions converge, different team members will focus on specific parts of the system including multi-user and security issues, user interface design (particularly concerning queries and ontology editing) and developing testing schemes to explore the limits of the server approach, particularly concerning the amount of data, query performance and size of the ontology.

4 Ontology-based Query Systems: The State of The Art

4.1 Introduction

This chapter is aiming at presenting a survey of main approaches in current ontology-based search/query systems and researches. This survey based on an investigation of approximately 40 important publications on *ontology-based search/-query systems*. Furthermore, the approaches are analyzed in this chapter are very recently high-profiled. Nevertheless, in this chapter, we are not covering the topic of semantic ‘search engines’ but we focus on ‘query systems’ or ‘query modules’ in current systems or frameworks which are based on ontology and the Semantic Web technology.

From the data gathered from this survey, relevant research directions in ontology-based querying were identified based on similarities of research goals. Besides research directions, we also analyze the literature for the common methodology and gives a short discussion about mentioned issues.

4.2 Classification Criteria

The main issues of our investigation presented in the next sections are described according to the following well-considered criteria:

Search strategies with support of ontology. Here, we are going to analyze three basic approaches of ontology-based search/query systems in generating queries: the traditional keyword search with augmenting of ontology, multi-facet (view-based) search, and native ontology-based search/query.

Query Formulation. We survey the approaches dealing with the query formulation problem, including the issues of proposing a new query language, complex constraint queries, problem solving, and query user interface as well.

Query refinement. With this criterion, we analyze how ontology-based query systems deal with the problems of ambiguity solving, ranking, and inference during the query refinement process. We also distinguish approaches in the current query refinement techniques.

User's interaction. The user's interaction criterion is discussed along two aspects: query user interface in query formulation, and user's interaction during query refinement and answering processes.

Roles of ontologies in the approaches. Roles of ontologies are investigated on the aspects of reasoning/inference, query refinement, result ranking, query user interface are analyzed in details.

The term “search” and “query” are interpreted similarly in this thesis somehow. However, they are not totally identical in some other contexts. In the scope of this thesis “search” is used for a general purpose, and “query” is used for the query process of a system.

4.3 Search Strategies with Ontology Support

4.3.1 Ontology-Enhanced Keyword Search

In the early research of the semantic-enabled search, efforts deal with enhancing traditional keyword search with semantic techniques. In this approach, ontological techniques are used in several ways to enhance keyword search and increase precision.

Firstly, there are many query expansion applied in keyword search use the thesaurus ontology navigation in query expansion such as [MTT98], [MM00], [BRA05], and [WR05]. Particularly, using WordNet ontology¹ which defines sets of synonym and metonym is the common method in these systems. These systems function following the same basic line: first, the keywords are located in the ontology, then other concepts are located through graph traversal, after which terms related to those concepts are used to either expand or constrain the search. For example, in [MTT98] and [MM00] query terms are expanded to their synonyms and metonyms by boolean operators.

¹ Wordnet 2.0, <http://wordnet.princeton.edu/>

An algorithm is presented in [RSA04] for finding relevant information to a query via text search. Firstly, the text search technique is used into a document collection. Secondly, a process of RDF graph traversal is started from the annotations of those documents to find related concepts such as the document's author and the project the document belongs to. The RDF traversal is used to identify the interested, similar and linked concepts.

Meanwhile, the main goal of the OntoDoc [CMN04] system is to allow users to query their own personal digital libraries in an ontology-based fashion. OntoDoc uses a reference ontology to represent a conceptual model of the digital library domain, distinguishing between text, image and graph regions of a document. OntoDoc offers the user two options to query his/her digital library: 1) free text searching using WordNet as the same as above approaches; or 2) through composition of semantic expressions by browsing ontology.

A simpler usage of enhancing keyword search results is undertaken in the TAP's interface [GMM03]. Here, a traditional keyword search targeted at a document database is enhanced by matching the keywords against concepts in an RDF repository. The the found documents are then returned along with matched concepts. In the case that several concepts match the keyword, the user can select the appropriate concept to constrain the search. Here, the idea is not to expand search terms, but to clarify the documents if appropriate to concepts.

Another search system in the same line is the CIRI search system [AJS⁺04]. The search is carried out through an ontology browser to select concepts for constraining the search. The actual search is performed through keywords associated to selected concepts. In CIRI, the user selects concepts directly from the ontology visualization instead of mapping the keywords into the underlying ontology to constrain the search.

4.3.2 Ontology-based Multi-facet Search

There is a powerful search paradigm which is a combination of the multi-facet search [MHS05] and ontology-based search [HSV03] called ontology-based multi-facet search. This is a union of the IR community and the ontology technique. This approach is reflected in search mechanisms of the Ontogator [HSV03], OntoViews [MHSV04]-based portals and SWED portals [RSC04]. In a ontology-based multi-facet search, the distinct views are created via ontology containing the hierarchy and class relationships.

In Ontogator, a search component of OntoViews, the underlying domain ontologies are mapped into facets and facilitate a multi-facet search. After finding information of interest by the multi-facet search, Ontogator uses the domain ontology together with annotation data to recommend the user related results.

These relevant results are not yet available in the multi-facet search phase, which then can be delivered by the recommendation system of Ontogator by semantic navigations.

In some versions of OntoViews [MHSV04], authors define a concept called semantic auto-completion which uses the keyword search as an introduction to the ontology navigation. The idea is that the main interface of the portal opens with a keyword field. The keywords, however, are linked to ontological classes in the different views instead of directly to information items [HSV03]. As the result, semantic disambiguation can be made. Finally, the search performs as a multi-facet search. Once the search has proceeded to the point where at least a single interesting instance is found, additional information can be retrieved via browsing. This process is similar to the browsing of Web pages through the hyperlinks. However, here the items are resources and the links between them are defined by their relations.

In short, the idea behind this approach is that the user can start generating his/her queries from the views associated to the underlying ontologies which is most natural to the user.

4.3.3 Native Ontology-based Search

Parallel with the above approaches, there are the methods for writing down information with its ontology. Here, some research efforts are based on the assumption of concepts, instances and relationships, and deal with the task of efficiently finding instances of the core semantic web data types.

With the help of ontology technique, OntoLogger [SGS03] builds a query mechanism by recording the user's behaviors in an ontology and recall it. OntoLogger, similarly to its general version- the Library Agent [Sto03b], is a query system based on usage analysis in the ontology-based information portals. Its query mechanism is based on usage-data in form of an ontology, so-called semantic log file. The structure of the ontology reflects the users' needs. By using this, OntoLogger supports the user in fine-tuning of his initial query. Moreover, during the refinement process, the system also uses this log ontology for ranking query results and refinements according to the user's needs.

A similar approach to [AJS⁺04] but with enhanced services (reasoning, catalog, etc.) and more dimensions of data with respect to searching geographic information, is taken in the GeoShare project [HSR⁺04]. GeoShare uses ontologies for describing vocabularies and catalogs as well as search mechanisms for keywords to capture more meaning. During the search process, the user narrows his/her search space's size by selecting specific domain (thematic, spatial, or temporal model); then he/she picks the appropriate concepts from these models

and application ontologies, covering all available concepts, to define the concrete query. After that, he/she can parameterize his query to concertize the retrieval process. In the sequel, the system processes the query and transforms it into the ontology language for the terminological part, where the system looks for equivalent concepts and subconcepts. After processing the query, the system composes a ranked list of relevant information providers based on the weightings of a specific reasoning process.

Semantic web data consist of ontological and instance data. The actual data the user is interested in are entities belonging to a class, but the domain knowledge and relationships are described primarily as class relationships in the ontology, and this is exemplified in the SHOE search system [HH00]. In SHOE, the user is firstly provided a visualization of the ontology, and he/she can choose the class of instances he/she is looking for. The possible relationships or properties associated with the class are then searched, and the user constrains the instances by applying keyword filters to the various instance properties. A similar approach is also applied in some versions of the SEAL portal [MSS⁺01] and Ontobroker [DEFS98].

However, there are some differences between these systems in their usage ontologies. In SHOE, providers of information can introduce arbitrary extensions to a given ontology. Furthermore, no central provider index is defined. In contrast, Ontobroker relies on the notion of an Ontogroup [DEFS98] and domain specific ontology defining a group of web users that agree on an ontology for a given subject. In addition, the Ontogroup is stored in a provider index.

Knowledge Sifter [KCD⁺04] is another approach using ontologies and is based on agent technique. The agents of this system help users to perform the query process. For example, the Ontology Agent provides a conceptual model for the domain by an OWL schema specification. A Web Service Agent accepts a user query which has been refined by consulting Ontology Agent and decomposed by the Query Formulation Agent. The Integration Agent is responsible for compiling the sub-query results from various sources, ranking them according user preferences, as supplied by Preferences Agent.

In a further effort of the above approaches, the authors of Haystack [HKQ02] based their user interface paradigm almost completely on browsing from resource to resource [QHK03]. This is affirmed by scientific results of a search behavior research [TAAK04] that actually most human beings seek information via a process they call “orienteering”, rather than carefully formulating a query that precisely defines the a desired information target. Users often prefer to start from a familiar location, or a vague search, and “home in” on the desired information through a series of associative steps [KBH⁺05]. The browsing functionality can be similarly found in the approach of OntoViews-based portals between individual information items.

4.4 Query Formulation

Formulating complex queries is the problem of finding a group of objects of certain types which are connected by certain relationships. For instance, formulate the query “Find all publications published in IEEE proceedings from 2000 to 2003 about ‘ontology-based ontology’, cited by recent publications in 2005”. Where “publications”, “IEEE proceedings”, “2000, 2003, and “2005” are ontological class restrictions on nodes and “published in”, “cited by”, and “time restriction” are the required connecting arcs in the pattern.

In the Semantic Web context, these such patterns are easy to formalize and to query. However, they remain uneasy for the users. Therefore, a number of approaches of the research on formulating complex queries have been developed on the user-oriented methods which are mainly GUI interfaces for creating such query patterns as visually as possible.

In an effort of this line, [ACK04] presents GRQL, a graphical user interface for building *graph pattern queries* that is based on ontology navigation. Firstly, a class in the ontology is selected as a starting point and all properties of the class are shown for expansion. Clicking on a property expands the graph pattern which contains that property, and moves selection to the range class defined for that property. An example by [ACK04], clicking the “creates property” in an Artist class creates the pattern "Artist \rightarrow creates \rightarrow Artifact", and moves the focus to the Artifact class, showing the properties for that class for further path expansion. In addition to lengthening the path, other operations can be performed on the query pattern.


Navigation	RQL query	Path
	<pre>select x2 from {x1}creates{x2;Painting }, Sculpture{x2}</pre>	Artist \rightarrow creates \rightarrow Painting Sculpture

Figure 4.1: A Phase of GRDL - ICS-Forth [ACK04]

The pattern can be bound to concern only some subclasses of a class, e.g. by binding **Artifact** to “Painting or Sculptures” in the previous example to "Artist \rightarrow creates \rightarrow Painting or Sculpture" [ACK04]. In a similar way, property restriction definitions can be bound into subproperties. More complex queries can be formulated by visiting a node created earlier and branching the expression there, creating patterns such as the one visually depicted in Figure 4.1.

Another graphical query generation interface, SEWASIE, is described in [CMF⁺04]. Herewith, the user is given some pre-prepared domain-specific patterns to choose from as a starting point. From this point, the user can 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 sub or superclass. This is performed through a clickable graphic visualization of the ontology neighborhood of the currently selected class, as shown in Figure 4.2.

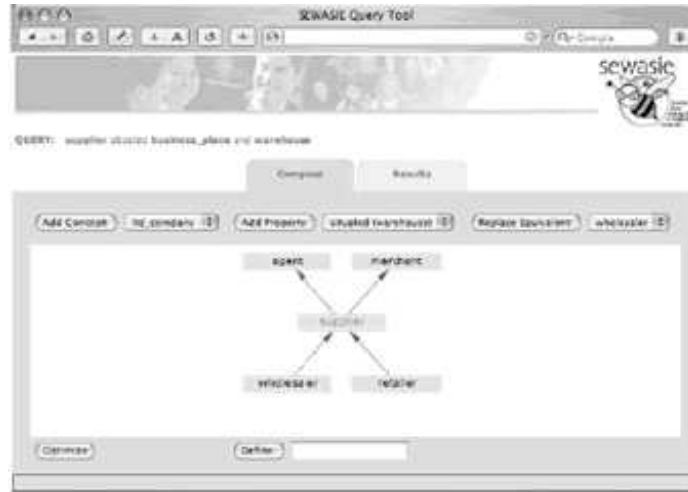


Figure 4.2: SEWASIE - graphical query formulation

In a further effort of reducing the complexity of query formulation, the approach of GetData Query interface [GM03] used in “Semantic Search” [GMM03] expresses the need of a much lighter weight interface for constructing complex queries. The reason is that the current query languages for RDF, DAML, and more generally for semi-structured data provide very expressive mechanisms that are aimed at making it easy to express complex queries; however, the queries require a lot of computational resources to process. The idea of GetData is to design a simple query interface which enables to network accessible data presented as directed labeled graph. This approach should provide a system which is very easy to build, support both type of users, data providers and data consumers. In a similar way, the contribution of [GHP01] is a high profiled example.

The multi-facet search portals mentioned in previous section can also be regarded of as user interfaces for generating a very constrained subset of complex graph patterns. In a simple case, the query is formulated as searching for an information with particular properties. Nevertheless, in a wider sense the definitions of how the objects are mapped to the views can be arbitrarily complex and involved graph navigation.

4.5 Query Refinement

Query refinement is an important part of a query process, particularly in the process of resolving problems of disambiguation from users' queries in the Semantic Web applications. A bulk of approaches for this issue has been developed. The varieties of these approaches are being to be discussed as follow; and we start with current approaches.

4.5.1 Query Ambiguity Discovering

[CTC02] have shown that the precision of information query relies strongly on the clarity of the query which a user posts to the system. When the query is formulated in an ambiguous manner, one can expect that a high percentage of irrelevant information can be retrieved, independently of the mechanism which is used for querying. Therefore, [Sto03b] argues that the query disambiguation should be the initial step in searching or querying for information in information systems. In another word, the determination of an ambiguity in a query, as well as the sources of such an ambiguity, is the prerequisite for the efficient information searching.

Word sense disambiguation of the terms in the input query and words in the documents have shown to be useful for improving both precision and recall of an information retrieval (IR) system. In the approaches of [MTT98], [MM00] and [BRA05], lexical relations from WordNet are used for query expansion, but without treating the query ambiguity.

The Librarian Agent [Sto03b] examines the query ambiguity in two factors: the structure of the query and the content of the knowledge repository. Regarding ambiguities in the query structure, two issues are defined: structural ambiguity in which the structure of a user's query is analyzed regarding the underlying ontology; and semantic ambiguity. Here the terms from a user's query is analyzed regarding the relations existing in the underlying ontology. Another factor of query ambiguity is the content of the knowledge repository. The ambiguity of a query posted in a knowledge repository is obviously repository-dependent. For instance, a user post a query of "World Cup" with his intend of retrieving the competitors in 2006 Soccer World Cup against the collection of news articles in which the articles about Chess World Cup Tournament are predominant, it is simply impossible for the system to return soccer articles consistently ranked higher then related to chess. In order to overcome this difficulty, [Sto03b] introduces a 'Response factor' for taking into account the specificities of knowledge repository content in determining the ambiguity of a query. The Response factor of a query is the measure to know how the terms from that query cluster the

resources in the underlying knowledge repository. The Response factor also describes the probability that a knowledge resource relevant for the query will not be relevant for the one of the non-empty subsets of that query.

In recent activities, this group have introduced another approach for the process of detecting query ambiguity and refinement [Sto03a]. In this process, firstly, potential ambiguities (i.e. misinterpretations) of the initial query are discovered and assessed (Ambiguity-Discovery phase). Next, these ambiguities are interpreted regarding the user's information need, in order to estimate the effects of an ambiguity on the fulfillment of the user's goals (Ambiguity-Interpretation phase). And finally, the recommendations for refining the given query are ranked according to their relevance for fulfilling the user's information need and according to the possibility to disambiguate the meaning of the query (Query Refinement phase). In that way, the user is provided with a list of relevant query refinements ordered according to their capabilities to decrease the number of irrelevant results or/and to increase the number of relevant results.

4.5.2 Query Refinement

The approach of [Sto03b] for query refinement tries to simulate and reflect the refinement model which a human librarian (or a shop assistant) uses in her daily work. It means that we use three sources of information in suggesting query refinement: 1) the structure of the underlying ontology, 2) the content of the knowledge repository and 3) the users' behaviors (how users refine their queries on their own). Since the first two sources are used for measuring the ambiguity of a query, the query refinements based on them are treated cooperatively as the ambiguity-driven query refinement.

In the ambiguity-driven query refinement approach, the ambiguity parameters presented in the previous section are combined and presented to the user in case she wants to make a refinement of the initial query. Each of the ambiguity parameters has its role in quantifying ambiguity. For each of the parameters, query term(s) that affect the ambiguity most importantly are determined. In that way, the user receives the most specific suggestions.

[Sto03a] presents another query refinement approach called information-need driven query refinement. This approach is a formalized one; it is based on 1) the definition of an order between queries, in order to create a map of the query neighborhood query map (i.e. query space), and 2) the characterization of the query ambiguity, in order to control the navigation in the query space compass. The query refinement process is then realized as the movement through the query's neighborhood in order to change the ambiguity of that query.

Similarly, in [Sto03a] an extension of the Query Management component in the Librarian Agent framework, presents a comprehensive approach for the refinement of ontology-based queries, which is founded on the incrementally and interactively tailoring of a query to the current information needs of a user. These needs are implicitly and on-line elicited by analyzing the user's behavior during the searching process. The gap between a user's need and his query is quantified by measuring several types of query ambiguities. Consequently, in the refinement process a user is provided with a ranked list of refinements, which should lead to a significant decrease of these ambiguities. Moreover, by exploiting the ontology background, the approach supports the detection of "similar" results that should help a user to satisfy his information need.

The third source for making the query's refinement recommendations in [Sto03b] mentioned above requires an analysis of the users' activities in an ontology-based application. That is also the approach of many query refinement mechanisms and OntoLogger [SGS03] is a one of them. OntoLogger is based on the log-ontology (usage-data) and analyzes the user's behavior in order to guide the user in refinement process. By doing this, the refinement process will support a user in fine-tuning of his/her initial query. Thereafter, it ranks the received resources according to their relevance of the user's query, and finally, the system relaxes the user's query such that its best approximation can be found.

In a similar manner to OntoLogger, OntoViews with its query mechanism Ontologator, deals with the query refinement based on the domain ontology and user annotation on data. The Recommendation system of Ontogator utilizes the domain ontology together with annotated data and recommendation rules to recommend the user to view other related information which maybe missed by his initial query. This process is known as the semantic browsing function. Through this kind of system, the user can refine his/her queries by selecting related information that suits his needs.

Query refinement in Knowledge Sifter is an aggregation of query expansion (Query Formulation Agent), which is also used in [MTT98], [MM00] and [BRA05], and recommendation system (Integration Agent) techniques. The Query Formulation Agent consults the Ontology Agent to refine or generalize the query based on the semantic median provided by the available ontology services. Besides, the Integration Agent is responsible for compiling the sub-query results from various sources, ranking them according user preferences, as supplied by Preferences Agent, for such attributes as: 1) The authoritativeness of a source which is indicated by a weight assigned to that source; and 2) The weight associated with a term comprising a query. Furthermore, the Integration Agent calculates each result's similarity by normalizing the total similarity value; ranks these results according to their similarity values and return the ranking information to the User Agent for display.

On the methodology side, the query refinement in GeoShare is similar to discussed approaches. However its implementation differs in some aspects. GeoShare's query refinement is based on its reasoning service and data annotation. The user, after given guidance to give an initial query on the basis of the data model and the underlying ontology, receives a ranked list of results or refinements which are outcome of processes of data retrieval and reasoning on data annotation. At this stage, the user can select 'information providers' according to his needs.

4.5.3 Ranking and User Interaction

In processes of query refinement and answering of the current ontology-based systems, the roles of ranking and the interaction of the user are very important. Ranking mentioned on surveyed papers included refinement ranking and as well as results ranking. This task is almost a general requirement for query refinement in semantic web applications, because of the nature of imprecise querying or searching. Ranking is almost always based on domain ontology, user ontology, user's profile or preference, and annotation data as well. All discussed ontology-based query systems in this chapter use this kind of technique.

In addition, user interaction in a query mechanism can be considered in many contexts such as formulating queries, query refinement and answering processes. However, relying on the power of the query refinement, the well-designed ontologies (such as domain ontology, user ontology), and the approach of the systems, user's interaction is required much or less. For example, in end-user based systems such as Ontogator, OntoLogger, query refinement based on user-data analysis and user ontology; so that the intervention of the user during the refinement process is less. In contrast, in other systems, the user interaction might be more. GeoShare [HSR⁺04] and Knowledge Sifter are examples.

4.5.4 Inference

Inference or reasoning is a challenging task in semantic web applications. GeoShare is one among the very few actual applications currently make use of inference based on OWL, Description Logics or actually any other rule systems. It uses OWL and Racer as inference engine for the system. Many of others that do, such as [FHH03], could have also been developed using simpler graph patterns.

4.6 Methodologies in Common

While surveying the field of ontology-based querying research, some common methodologies can be determined. The knowledge and understanding of these common methods as well as how they are used in the various actual approaches are of great importance for future ontology-based search/query methodologies.

4.6.1 Role of Ontologies

In the regarded systems, ontologies are very crucial and play a key-role. Ontologies appears from the starting (query formulation) until the end (query answering) of querying processes. We can conclude the roles of ontology as following:

1. Providing a pre-defined set of terms for exchanging information between users and systems.
2. Providing knowledge for systems to infer information which is relevant to user's requests.
3. Filtering and classifying information.
4. Indexing information gathered and classified for presentation.

4.6.2 Query Refinement

All query refinement methods which are ontology-based approaches aimed at disambiguation the posted user's queries. In the IR community, generally, we can see two directions of modifying queries or query results to the needs of users: query expansion and recommendation systems. Query expansion is aimed at supporting the users to make a better formulated query, i.e. it attempts to improve retrieval effectiveness by replacing or adding extra terms into the initial query.

The interactive query expansion supports such an expansion task by suggesting candidate expansion terms to the users based on some indexes or concept hierarchies. Recommendation systems try to recommend items similar to those a given user has liked in the past, or identify users whose tastes are similar to those of the given user. More and more ontology-based query refinement techniques are formalized, and more complex and more effective approaches have been introduced, that are [SGS03] (usage-based), [Sto03b] (ambiguity-driven), [Sto03a] (information-need driven) and [Sto03a] (step-by-step).

4.6.3 Keyword-Concept Mapping

Mapping between keywords and ontological concepts is a method commonly used in discussed ontology-based query systems. Huge research efforts are specifically achieved on the issue of combining searching through textual material with searching through formally defined information (for example in [HSV03] and OntoWebs-based systems).

The natural language is the form that is most familiarly to human beings. Mapping patterns to sentences can give the user a clearer awareness of the represented knowledge, and again the user may be more comfortable in formulating his/her queries as natural language sentences. In this case keywords provide an entry-point for a quick way of finding information. Keywords and other restrictions can be easily specified for the text search fields in comparison with the ontology navigating to identify the concepts and graph patterns to be used as query constraints.

4.6.4 Graph Patterns

Graph patterns are an important concept in semantic web search methods which is used in different functions. Firstly, because of the way the RDF data model is organized, graph patterns are often used to formulate complex constraint queries.

In some systems, such as OntoViews, general RDF path patterns are also used to link interested resources to each other, or to formulate patterns for identifying interested connecting paths between named resources [AS03]. Additionally, in results presentation, the parameters fetching information are also often considered as simple graph patterns.

4.6.5 Inference

Obviously inference on the semantic web must be regarded as a very complex problem. The fact that the Semantic Web is designed to work under the open world assumption, whereas most well explored logics operate only on the base of a closed world assumption, builds a fundamental difficulty. Also, the vision of the semantic web which comprises a large amount of data, constitutes a problem for most current inference algorithms. GeoShare is one among the very few actual applications which currently use inference based on OWL. Meanwhile, many of others that do, such as [ACK04], could have also been developed using simpler graph patterns.

Systems	Search Methods		Approaches		Query Refinement	Inference	Query Formulation		Using annotated data
	Enhanced-Keyword	Ontology-based	Front-End	Back-end			Query UI-based	Ontology-based	
<i>Ontologator</i>	✓		✓	✓				✓	
<i>Ontologer</i>		✓	✓		✓			✓	
<i>KnowledgeSifter</i>		✓	✓		✓				
<i>OntoViews</i>	✓	✓	✓	✓	✓			✓	✓
<i>OntoDoc</i>	✓	✓		✓				✓	
<i>GeoShare</i>		✓		✓	✓	✓	✓	✓	
<i>Ontobroker</i>		✓		✓			✓	✓	✓
<i>SHOE</i>		✓		✓			✓	✓	
<i>SEAL</i>		✓		✓				✓	
<i>Haystack</i>		✓		✓	✓		✓	✓	
<i>TAP</i>	✓		✓				✓		✓

Table 4.1: Summary of Ontology-based Query Systems

4.6.6 Fuzzy Concepts, Relations, and Logics

In the research of enhancing the traditional search with ontology techniques, there is a requirement for formalisms allowing the combining fuzzy annotations based on text search with the semantic annotations. As a result, a number of formalizations and experimentations with fuzzy logics, fuzzy relations and fuzzy concepts have been undertaken in that field ([ZZY⁺05] is herefore an excellent example).

Fuzzy logics are, however, not only useful in combining text search with ontologies. In [SDA04], authors use fuzzy qualifiers to constrain the complex queries, and in [Par04] user profiling is used for measuring the relevance of an ontological relation.

4.7 Discussion and Summary

A number of common patterns can be detected in the approaches described in this thesis. On the technical level, it can be concluded that in the working context of an RDF model, quite many of the used common methodologies are of general nature.

Usually complex constraint queries are focused on models where individuals and classes are the interesting information items; we can observe relations which are present as equal partners in all the graph pattern, path and logic formalisms. After the deduction of a result set by using complex constraints, there are strong tendencies to use graph traversal algorithms to locate additional result items. While fuzzy logic formalisms and fuzzy concepts allow us to combine keyword search results as equal partners in complex constraint querying.

Besides, the ontology-based query refinement, which includes ranking issue and user-interaction, can be recognized as innovative approach for improvement of query precision and helping users clarify their queries from ambiguous initial ones. The query refinement has been started very early along with the query process in semantic web application, which uses simple expansion algorithms. The current approaches have proved their power with effective refinement strategies based on ontologies. The only approach which does not neatly wrap into the others is inference-based problem solving. Inference in general builds a much greater challenge for the most usual cases of ontology-based query systems.

A summary of the discussed ontology-based query systems according the common criteria is presented in the Table 4.1.

5 The Virtual Query System

5.1 Motivation

Towards the goals of personal information storage and retrieval of all data throughout a lifetime, researchers consider continuous archival and retrieval of all media relating to personal experiences including emails, contacts, appointments, web browsing, documents, phone calls, etc. The challenging issues are how to extract useful knowledge from this rich library of information; and how to use this knowledge effectively [FR03].

The SemanticLIFE [AHK⁺04] user is supported in issuing imprecise queries to retrieve the rich semantic information from his/her historical personal data. For example, consider scientists, who work in a specific domain. They might be interested to get into contact with other researchers in the scientific community that (1) share the same interests or have similar problems (2) are publishing in similar conferences and (3) were recently active in the specific field of research (4) and speak a common language. The result of such a query could be the web pages and email addresses of the researchers coming into question.

Most often when developing similar PIM systems, researches focus on back-end issues, i.e. capturing all data sources, integrate and then store them in huge repositories. For this purpose it is necessary to map the ontologies of the various data sources into a common ontology of the system. However, users are confronted with the lack of knowledge concerning the stored information inside the system, and they would formulate ambiguous requests, so that many barriers have to be overcome before the system could deliver the demanded results.

This chapter is aiming at an introduction of the innovative features of our ***Virtual Query System*** design. This query system is based on a front-end approach allowing the user to retrieve information from huge ontology-based repositories in an efficient way. The conception of this query system which is primarily based on the reduction of semantic ambiguities of user query specifications at the very early stage of the retrieving process; and continually guide the user in query

process using a set of query templates. This approach integrates many research efforts from the area of the Semantic Web, query refinement, semantic query caching for RDF data, inference, ontology mapping, and user interaction.

5.2 Virtual Query System Enabled SemanticLIFE

5.2.1 Information Retrieval in the SemanticLIFE Framework

In SemanticLIFE, it is uneasy to define highly structured queries when a multitude of information systems are addressed or the information is only semi-structured. Hence the system must be capable to support user-queries which are formulated with an *imprecise search* terminology by automatically transforming them into more specific queries.

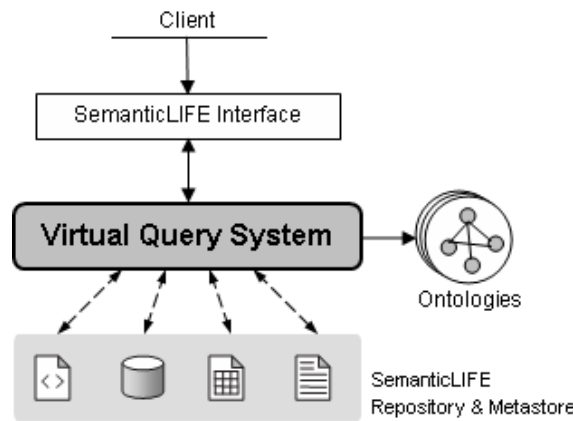


Figure 5.1: The VQS Enabled SemanticLIFE Information Retrieval

In the SemanticLIFE metastore, the data is already stored in a semantically enriched manner, it provides more powerful imprecise searches. Here, the term “imprecise” has two meanings: firstly, the query processing system has to satisfy imprecisely defined user information needs. Secondly, the target of the user-query is well specified but there are ambiguities in processing the query because of the heterogeneity of the different data sources. Therefore, the system solves these problems during query generation by exploring the system database and ontology. A part of the query module uses system metadata and ontology to provide the user a better awareness of stored data.

The querying process in SemanticLIFE depicted in Figure 5.1 is supported by our Virtual Query System. This mediator system is not only capable to deal with the discussed issues above but also reduces the imprecision of the user’s requests by offering the user an overview of the relevant stored information and query

templates during his/her information retrieval process. As a result, the user will significantly specify more precise queries on information and data stored in the system.

5.2.2 Aims of The Virtual Query System

Formulating unambiguous queries is always a demanding task to users as they do not have the overview on the semantics of stored data. The principle of the Virtual Query System (VQS) is to provide an ontology-based virtual information view of the system data, which we could call *virtual information*. Hereby, “virtual information” are the metadata extracted from the SemanticLIFE metastore and delivered to the user after a well-organizing process. The user can clearly specify queries on the “real” data in the repositories when he/she can “be aware of” what is inside of the system.

The VQS also provides predefined query patterns which will be matched with users’ query space and the matching ones then will be recommended to the users. In addition, based on a common ontology and the internal analysis (inference, detecting ambiguity and fuzziness of user queries), the VQS refines the user’s queries and generates “real” queries against data sources in the metastore. Finally, relied on user’s experiences, reflected in user ontology or user profile, the VQS recommends the most related results to the user.

In short, we can summarize the main goals of the VQS which are different to the similar approaches as follow:

1. Enabling users to be aware of the semantics of data stored in their SemanticLIFE systems.
2. Supporting users in query formulation not only in the initial phase by delivering the “virtual information”, but also during the information retrieval with pre-defined query patterns based-on their query context or querying space.
3. Presenting the results based-on the user’s profile in an interactive manner which is integrated with above features.

5.3 The Virtual Query System

5.3.1 The Architecture

The Virtual Query System (VQS) consists of six modules, presented in figure 5.2, to deal with the challenging task of a complete Semantic Web query system.

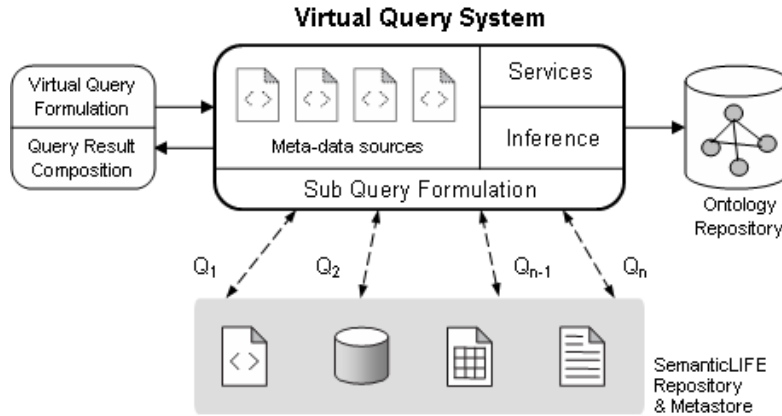


Figure 5.2: The Components of the Virtual Query System

The Virtual Data Component

The *Virtual Data Component* (VDC) contains the *metadata* of storage sources. This VQS crucial module acts as a *virtual information layer* to be delivered to the user. It enables the user to be aware of the semantics of the data sources stored and to specify more precise queries.

The VQS collects metadata from data sources in the metadata repository of the SemanticLIFE system. An analysis process is carried out on these meta-data sources to get the semantic information. Then the processed information is stored in this module as a “context ontology” and will be delivered. The features of VQS are very similar to those of a recommender system. Furthermore, this part is also referred as an *image* of the system database in further query processing, so-called the *context-based querying* which is discussed in detail in Chapter 7 and in [HAT06a, HNAT06]. A query languages has been developed for querying the virtual data from this component (more details in [HT06b]).

This component is the main different point of our system in compared with similar approaches discussed in [HT06a]. The rational behind the idea of this approach is that when users are aware of their data then they could formulate more unambiguous requests. This ultimately leads to the reduction of the query refinement process complexity. Additionally, this virtual data component plays as a *context ontology*. This makes the SemanticLIFE system very flexible as the system can adapt to a new scenario by simply changing the context ontology.

This component also enhances the query process by the “context-based” querying feature. With this feature, the query patterns will be proposed by the system based-on the *context* where the user is in. The details of this VDC feature and VDC itself are discussed in Chapter 7.

The Ontology Repository

The second part of the system is the ontology repository which builds up the core of the VQS. The repository contains the ontologies used in the VQS system such as the global ontology or inference ontologies. According to [CSC04], an ontology-driven approach to data integration relies on the alignment of the concepts of a global ontology that describe the domain, with the concepts of the ontologies that describe the data in the local databases. Once the alignment between the global ontology and each of the local ontologies is established, users can potentially query hundreds of databases using a single query that hides the underlying heterogeneity.

Sub-Query Formulation

Sub-queries formulation is another essential part of the VQS. From the user's initial virtual query, this part parses it into the sub queries (Q_i in the Figure 5.2) based on the global ontology for specific data sources. This module does not only transform the virtual query to sub-queries for specific data sources but additionally perform inference on the user's request in order to create more possible sub-queries afterward. After this process, the "real" queries (RDF queries) for each the data sources to be generated.

The VQS Services

Ontology Mapping. Mapping service is a mechanism to map local ontologies into a global one. This service deals with new data sources added with their respective ontologies, so that these ontologies are mapped or integrated to the global ontology. In our approach, we do not reinvest to develop a new ontology mapping framework. We use the MAFRA framework [MMSV02] for our mapping tasks.

Query Caching. This service improves the performance of the VQS by caching the queries in a period of time. We distinguish two kinds of caching mechanisms: query caching and result caching. The first addresses the process of generating sub-queries; and the second covers the caching of query results. Both caching types will use the semantic query caching methodology proposed in [Stu04].

Inference. The ontology-based inference service provides a basis for the deduction process on the relationships (rules) of concepts of the ontologies specified. Inference tasks are performed on the foundation of the ontologies and the data described by them. This service helps the system to analyze and evaluate the user's virtual queries in the process of generating sub-queries based-on the inference ontology.

Query Refinement

The VQS's query refinement is another important service for our query processing. This is the interactive way (semi-automated) for the VQS dealing with user's ambiguous queries, which is based on incrementally and interactively (step-by-step) tailoring a query to the current information needs of a user [SSS04]. This query refinement service of the VQS is a semi-automated process: in the refinement process, the user is provided with a ranked list of refinements, which leads to a decrease of some of these ambiguities. In another hand, by exploiting the user's profile, the ontology background, and as well as user's annotation on data, this VQS service supports finding "similar" results.

The Virtual Query User Interface

The Virtual Query User Interface (VQUI) delivers the virtual data to the user and helps the user to define virtual queries. A set of query patterns is offered to the user. If these patterns do not match the demands, the user can use a query-by-example tool alternatively to write the virtual queries. The VQUI also acts as query results composition which performs the integration and aggregation of the sub-query results and show to the user.

5.3.2 The Virtual Query System Workflow

The workflow of the VQS is illustrated in the Figure 5.3. In the figure, the dashed arrows denote the internal activities; normal arrows are used for the interaction between the user and the VQS. The external query queue is used for receiving queries from the user and to return the results. Inside of the system, an internal query queue will be used to receive processed sub-queries and to deliver results.

The following steps are performed in a chronological order: At the very early stage when the VQS is installed into the whole system of SemanticLIFE, action (1) will be performed. This action collects meta-data of the data sources from the meta-store, performs necessary statistical computation and stores the results

into the Meta-data sources. Parallel, the global ontology is mapped from local ontologies using the VQS ontology mapping service.

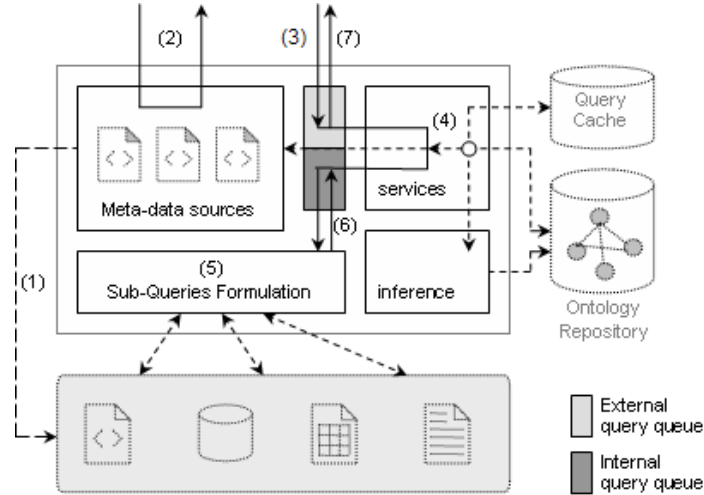


Figure 5.3: The Workflow of the Virtual Query System components

As also depicted in Figure 5.1, the user directly retrieves the information by specifying his/her queries to the VQS with help of the query interface. The VQS gets user input through the virtual queries. Using this interface, the user can be confronted with intentional information of the meta-data sources, so that we can avoid to a certain degree ambiguous requests (2).

In the next step, the formulated query, so-called virtual query, is sent to the VQS (3) and from this stage, the virtual query is evaluated on the foundation of the ontology-based services of the VQS (4). The VQS query caching service is now in charge of the subsequent processing of the virtual query: firstly, it looks up in to the query cache, if there is a match, the result is quickly returned to the user. In the second case if the virtual query does not match, an analysis and evaluation process will be undertaken (4) and the virtual query with the analyzed information will be sent to the Sub-queries formulation (5). As a result, sub-queries for each specific real data source will be generated. Finally, query results will be passed to the user ((6) and (7)). They are aggregated in the Query Result Composition part before the delivery to him/her in a suitable form.

5.4 Summary

In this chapter we have introduced the Virtual Query System used for querying tasks in the SemanticLIFE framework. The VQS is an approach of building a complete Semantic Web query system based on a “front-end” approach.

Besides applying current Semantic Web technologies known in the area such as ontology mapping, user annotation and semantic query caching for RDF data, we have designed a query system which aims at a significant complexity reduction in formulating semantic meaningful queries and at the same time aims at a considerable reduction of the number of ambiguous user queries. The details of the innovative features of the VQS is presented in the next chapters.

6 The Virtual Query Language

6.1 Introduction

Making unambiguous queries in the Semantic Web applications is a challenging task for users. The Virtual Query System (VQS) of the SemanticLIFE framework is an attempt to overcome this challenge. The VQS is a *front-end* approach for user-oriented information retrieval [HTN06, HAT06b].

The issue is on the user-side, where users are required to make queries for their information of interest. The current query languages for RDF, DAML and more generally for semi-structured data provide very expressive mechanisms which are suitable for back-end querying mechanisms. To users, who are inexperienced with them, these query languages are too complicated to use. Additionally, the communication of components of the SemanticLIFE system with the VQS requires the facility to transfer their requests without knowledge of the RDF query language. Another important point is the portability of the system, i.e. if the system is bound to specific RDF query language, we could have problems when shifting to another one.

In the effort of coming over the above tackles, there is an approach of creating much lighter query languages than current expressive RDF query languages. According to [HT06a], the approaches of [GHP01] and the GetData Query interface [GM03] are high-rate examples. [GHP01] describes an a simple expressive constraint language for Semantic Web applications. The framework defines a ‘Constraint Interchange Format’ in form of RDF for the language, allowing each constraint to be defined as a resource in its own right. Meanwhile, the approach of the GetData Query interface of TAP¹ 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 enables to network accessible data presented as directed labeled graph.

¹ TAP Infrastructure, <http://tap.stanford.edu/>.

Continuing this trend, we present in this chapter an effective and lighter weight query language namely *Virtual Query Language* (VQL). This language is used by the VQS for information querying in the SemanticLIFE system. The VQL is a much lighter weight language than RDF query languages; but it offers interesting features to complete the tasks of information querying in the Semantic Web applications. In addition, the VQL is designed to assist the users make queries in simple manner and simplify the communication between components of the SemanticLIFE system with the query module - the VQS.

6.2 VQL - The VQS's Virtual Query Language

6.2.1 The Goals of the VQL

A number query languages have been developed for the Semantic Web data such as data in form of RDF (RQL [KAC⁺02], RDQL [Sea04], SPARQL [PS05], and iTQL [WGA05]). Why do we need yet another query language?

All these query languages provide very expressive mechanisms that are aimed at making it easy to express complex queries. Unfortunately, with such expressive query languages, it is not easy to construct queries to average users as well as to ask abstract information. What we need is a much lighter weight query language that is easier to use. A simple lightweight query system would be complementary to more complete query languages mentioned above. VQL is intended to be a simple query language with a support in a “semantic” manner for users' queries. In the context of the VQS and the SemanticLIFE system, we can see the aims of VQL as follows:

- VQL helps clients making queries without knowledge of RDF query languages. The user just gives basic parameters of needed information in VQL queries, and would receive the expecting results.
- VQL assists users in navigating the system via semantic links or associations provided in the powerful query operators based on ontologies.
- VQL simplifies the communication between the Query module and other parts. With VQL the components asking for information do not need to create RDF query statements which are uneasy tasks for them. Additionally, this feature keeps the SemanticLIFE's components more independent as they do not bind to a specific RDF query language.
- VQL enables the portability of the system. Actually, the SemanticLIFE and VQS choose a specific RDF query language for its back-end database. However, they probably could be shifted to another query language, so

that this change does not effect other parts of the systems, especially the interface of the system database.

6.2.2 The Syntax of VQL

Query Document Syntax

Definition 6.1. A *VQL query document* has four parts: parameters, data sources, constraints (relations), and query results format which are described in the schema depicted in Figure 6.1.

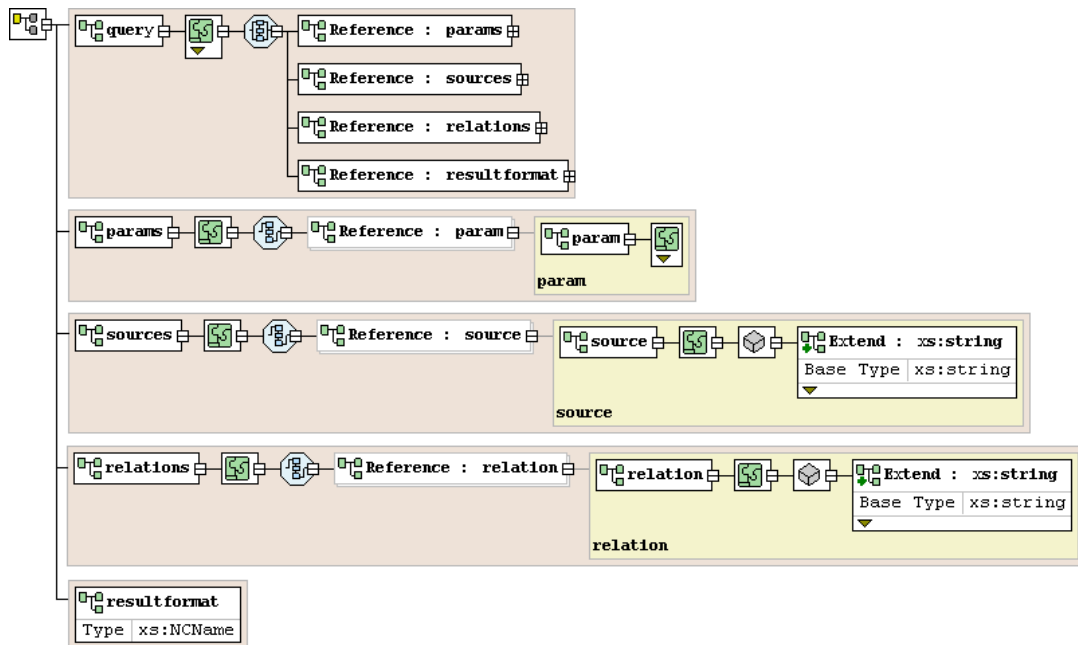


Figure 6.1: The schema for general VQL queries

The first part contains parameters of specifying the information of interest. A *parameter* consists of a variable name, the criteria value, and additional attributes for sorting or eliminating unneeded information from the results.

The second part—*sources*—is used for specifying the sources where the information will be referred to extract from. Obviously, the information need to be defined in the first part must be related to the sources specified in this part.

Thirdly, the *constraints* of the query—*relations*—are defined in the third part of the document. Here the relations between sources, parameters are combined using the VQL operators. Finally, the format of query results is identified in the fourth part. VQL supports four query results formats that are XML, text, RDF

graph, and serialized objects of query result sets. This provides flexibility for clients to process the query results.

XML-based Format

A standard format for information exchange, a easy-to-use and familiar-to-clients format, a widely accepted standard, and a flexible and open format are the requirements for the VQL query document. We have considered some alternatives and decided to choose XML as the format for VQL query syntax. A XML-based VQL query is structured as follows (Listing 6.1):

Listing 6.1: An Example of a XML-based VQL Query

```

1 <query type="data">
2   <params>
3     <param show="1" name="s1:messageTimeStamp">2005-11-01</param>
4     <param show="0" name="s2:messageTimeStamp">2005-11-31</param>
5   </params>
6   <sources>
7     <source name="fileupload">FileUpload</source>
8     <source name="browsingession">BrowsingSession</source>
9   </sources>
10  <relations>
11    <relation id="1" param="s1" source="">dt:gt</relation>
12    <relation id="2" param="s2" source="">dt:lt</relation>
13  </relations>
14  <resultformat>xml</resultformat>
15</query>

```

- **The top level** or the body of query is the `<query>` element. Here, the query type must be specified `type` attribute. The reserved terms used for this attribute are "data", "schema", "rdf1" and "rdf2" for different VQL query types (Section 6.2.4). For example, `<query type="data">` is a VQL data query.

- **The second level** contains required sub-elements. Depending on the type of a VQL query the elements are used respectively: for the data query, elements of `<params>`, `<sources>`, and `<relations>` are used once for each. These elements have their children specified in the third level. Listing 6.1 is an example.

For the schema query or the VQL RDF type 1 query, we use only one `<statement>` element which contains a RDF query statement (Figure 6.1). For the "rdf2" query, elements of `<select>`, `<from>`, `<where>`, `<orderby>`, `<limit>` and `<offset>` are used in the same way, where last three are optional as described

in Listing 6.3. Moreover at this level, we must identify the query results format in the `<resultformat>` element as "xml", "text", "rdf", or "object".

- **The third level** elements are only applied for the VQL data queries. The elements are children of `<params>`, `<sources>`, and `<relations>`; and the tags consequently are `<param>`, `<source>`, and `<relation>` with their own attributes.

- `<param>` *element*: each parameter is identified by this element with required attributes: `show` and `name`. While `show` is set to 1 or 0 that means the result of this parameter is shown in the result sets or not; `name` has two parts: a *variable* and the *meta – information* which are put together with ":", i.e. `variable:metainfo`. Besides, this element has two optional attributes known as `order="1"/"0"` and `exclude="string"`. The `order` attribute is used for sorting, while `exclude` is for excluding some information from query results. The element is enclosed with an optional value for filtering.

Example 6.2.1. A parameter is defined as follow:

```
<param show="1" name="v1:emailTo">tuwien.ac.at</param>
```

The parameter in this example is dened for extracting information of email addresses sent to recipients in domain of `tuwien.ac.at`. The variable `v1` contains the meta-information `emailTo`.

- `<source>` *element*: names of concerned sources for users' requests will be put here. This element has only one attribute `name` which is an internal name of data sources. This internal name is assigned automatically by the system.

Example 6.2.2. A data source is defined as follow:

```
<source name="at.ac.tuwien.slife.feed.email">Email</source>
```

Here, the source is specified using two names, the internal name, which is hidden to users, is put in `name` attribute; and the external name is optional.

- `<relation>` *element*: contains a constraint of the VQL query. The required attributes of each constraint are: `id="number"`, `param="variable"`, `source="source-name"`, and optional `or="id"`. The `id` is assigned a number, *variable* in the related `<param>` is used in this `param` attribute. Attribute `source` is identified with *source – name* or left empty in the case of only one data source specified; and `or` is assigned by `id` of another `<relation>` in order to make an OR expression. The operator for the constraint is put as value of the `<relation>` element.

Example 6.2.3. A constraint is defined as follow:

```
<relation param="v1" source="Email">str:match</relation>
```

This is a constraint to form an expression for the query. The expression means to get the email addresses from `Email` data source where emails are from `tuwien.ac.at` domain. The operator `str:match` is a pattern comparison.

6.2.3 Operators and Expression in VQL

The SemanticLIFE's back-end is organized by using ontologies and RDF enhancing with data typing and a powerful index mechanism. Hence, the supported operators in VQL inherit these features and reflect them in its operators.

Logic Operators

VQL's logic operators consist of AND, OR and NOT. While NOT is defined in each `<param>` element by using the `exclude` attribute, AND and OR operators are identified in `<relation>` items. OR is defined by the `"or"` attribute, e.g. `or="2"` means that the current constraint will be combined with the constraint number "2" using logic OR operator. AND is implied in a relation without `"or"` attribute.

Comparison Operators

The comparison operators are used in `<relation>` part of VQL queries. These operators are presented as follow:

- equal** An equal comparison in form of triples which is used by leaving the `<relation>` item empty.
- gt** The operator means **GREATER-THAN** which is used for comparing values of basic data types such as String, numbers.
- lt** Similarly to the previous one, the operator means **LESS-THAN**.
- dt:gt** This operator is used for comparing the date/time value **AFTER** a point of time. Value format of this type confronts XML Schema (XSD)² data types, i.e. `'DD-MM-YYTHH:MM:SSZ'` where `'T'` is the delimiter and `'Z'` is the time-zone.
- dt:lt** Similarly to the **dt:lt** operator; but it means for the date/time value **BEFORE** a point of time.
- str:match** This is the pattern matching operator for strings. This comparison operator uses common pattern expression.
- ft:match** This is the powerful operator relying the full-text index applied in SemanticLIFE's metastore. It is used for searching the full-text data such as content of a file or email attachments, or stored WWW pages.

² XML Schema, <http://www.w3.org/XML/Schema>

Expressions

In order to formulate the expressions for the query criteria, VQL uses "or" attribute in `<relation>` elements as boolean *OR* operator to combine these constraints first, and then it uses boolean *AND* to combine into the final expression. The sequence of `<relation>` elements are important in combining expressions.

6.2.4 The VQL Query Types

VQL Query Types

Definition 6.2. *VQL query types* are different forms of VQL query documents used for special purposes of information retrieval in the VQS.

There are three VQL query types: *data query type*, *schema query type* and the *embedded query types*. The VQL query types conform the main syntax of the VQL query document but having some particular part for each types.

Data Query Type

The VQL data query type is commonly used for information querying. A query of this type consists of the four parts as discussed above. In order to inform VQL parser to process the query as a VQL data query, we identify "data" term in the query: `<query type="data">`.

An example for this query type is shown in Listing 6.1: the query retrieves messages' time-stamp of files uploaded and browsed web pages in the SemanticLIFE repository in 'November 2005'.

From this query type, we formulate deductive queries for special operations in semantically information retrieval. These operations help users easily get information of interest and obey the principle of VQL design: "*ask minimum words, get maximum information*". The deductive queries, so-called *VQL Query Operators*, are described in details in section 6.3.

Schema Query Type

The syntax of this query type consists of two parts: the first one is the `<statement>` element containing a RDF query statement; and the second part is used to set the query results format. Necessarily, "schema" must be identified in the query body. A schema query example is illustrated in Listing 6.2.

Listing 6.2: An example of VQL SCHEMA query

```

1 <query type="schema">
2   <statement>
3     select ?Message ?p ?o
4     from &lt;rmi://192.168.168.174/sliffe#BaseModel&gt;;
5     where ?Message &lt;rmi://192.168.168.174/sliffe:messageBody&gt;; ?oo
6       and ?oo ?p ?o
7       and ?Message &lt;rmi://192.168.168.174/sliffe:fileName&gt;; 'CFP*' in
8       &lt;rmi://192.168.168.174/sliffe#
9       FTLiteralsModel&gt;;
10   </statement>
11   <resultformat>xml</resultformat>
12 </query>

```

However, the question is that how does the user create RDF query statements? Actually, the schema or ontology queries are offered to clients in form query templates or programmatic VQL API.

Embedded Query Types

Similarly, with *embedded query types* RDF query statements are wrapped in VQL query documents. We distinguish two ways of embedding the RDF statements: firstly, a whole statement is embedded; while their parts is embedded separately in the second type.

Listing 6.3: An example of the VQL RDF query type

```

1 <query type="rdf2">
2   <select>$s $p $o</select>
3   <from>rmi://192.168.168.174/sliffe#BaseModel</from>
4   <where>$s $p $o</where>
5   <resultformat>text</resultformat>
6 </query>

```

The format of the first embedded query type, so-called VQL RDF query type 1, is similar to the schema query described Listing 6.2, where the term "rdf1" is used instead of "schema" in the <query> tag. Meanwhile, in VQL RDF query type 2, each parts of RDF query statement, such as **select** and **from**, are put in respective query parts.

Listing 6.3 is an example of VQL RDF query of type 2, in which "rdf2" is specified in <query> tag. As depicted in the figure, the expressions of clauses of the RDF query statement are filled in respective elements of the VQL query.

6.2.5 VQL Query Results Format

In order to increase the flexibility in query results processing, VQL provides four query results formats that are XML format, text format, RDF graph, and serialized query results. In a VQL query, we identify the query results format in the `<resultformat>` element at the second level (see Figure 6.1).

Query Results XML Format

The VQL query results XML format is similar to a W3C's format for SPARQL XML query results presented in [Bec05]. This XML format for VQL query results has two main parts: the first one is the list of the query's variables which actually are properties mentioned in the query and additional information such as the message's URI variable and the data source type of the resource. The second part contains found items, i.e. values of these variables. A VQL query result is described in Listing 6.4 is an example.

Listing 6.4: The XML VQL Query Result

```
1 <answer>
2   <query>
3     <variables>
4       <Datasource/>
5       <fileName/>
6       <fileLastModified/>
7     </variables>
8     <solution>
9       <Datasource resource="http://www.slife.at/DS#FileUpload"/>
10      <fileName>14_07_04_Butler.pdf</fileName>
11      <fileLastModified>2004-07-14T15:10:29+02:00
12      </fileLastModified>
13    </solution>
14    ...
15  </query>
16 </answer>
```

In order to receive the query result in the XML format, the "xml" term should be put in the result-format element of the query document: `<resultformat>xml</resultformat>`.

Query Results Text Format

Some applications have the trend of avoiding using any XML parser to simplify the query results processing. Therefore, the text format of query results is an

alternative solution for them.

The text format of VQL query results is structured as follows: firstly, a summary of the result is presented, including a number of returned rows and a number of the query's variables; and they are presented as `Rows = num_of_rows` and `Vars = num_of_vars`. In the second part, the variable's values of each item are then paired in form of `VAR_NAME = VALUE`. These pairs are connected by semicolons, and one row is for each items. A simulated text result of a query is presented as follow (Listing 6.5):

Listing 6.5: The TEXT VQL Query Result

```

1 Rows = 2
2 Vars = 3
3 [ VAR1 = MSG_URI1; VAR2 = VAL21; VAR3 = VAL31 ]
4 [ VAR1 = MSG_URI2; VAR2 = VAL22; VAR3 = VAL32 ]

```

The VQL query results will be transformed into the text format, if the `"text"` term must be identified in the result-format element of the query document:
`<resultformat>text</resultformat>`

Query Results Jason Format

Jason is the intentionally misspell of JSON³ and used for a query result format. JSON is a light-weight data-interchange format. It is easy for humans to read and write, for machines to generate and parse. JSON is based on a subset of the Javascript programming language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

In order to request the VQL returns the query result in form of JSON format, the `"json"` term must be put in the result-format element of the query document as `<resultformat>jason</resultformat>`. The following listing is an example of query results in form of Jason format (Listing 6.6).

Query Results RDF Format

The RDF-graph format of query results is designed for semantic web client applications preferring semantic enriched data. Since the RDF format is the standard

³ JavaScript Object Notation, <http://www.json.org/>

Listing 6.6: The JASON VQL Query Result

```
1 {"head":  
2   { "vars": ["a" ,"b" ] },  
3   "results": {  
4     "distinct": false,  
5     "ordered": false,  
6     "bindings": [  
7       {  
8         "a": { "type": "uri" , "value": "http://www...." } ,  
9         "b": { "type": "uri" , "value": "http://www..." }  
10      } ,  
11      ... ] }  
12 }
```

for this purpose, the query results will be transformed to RDF graphs before returning them to the clients. RDF/XML is used as default RDF format of these query results. Obviously, the clients must have abilities to process RDF-graphed query results by using amongst RDF parsers such as Java-based semantic web frameworks, Jena⁴ or JRDF⁵.

In order to request the VQL returns the query result in form of RDF graphs, the "rdf" term must be put in the result-format element of the query document as `<resultformat>rdf</resultformat>`.

Serialized Query Results Object

Concerning communication of the inside components of the SemanticLIFE system, sometimes we would like to use query results object without format transformation. In this case, the VQL must satisfy the demands by support of serializing the results object using the Java serialization technique and sending the serialized object back to the asking component.

In order to request the VQL returning the query result in form of serialized objects, the "object" term must be identified in the result-format element of the query document as `<resultformat>object</resultformat>`. Nevertheless, this format is API-dependent, therefore the requirement is that the asking component must use the exactly same RDF API to what VQL parser uses.

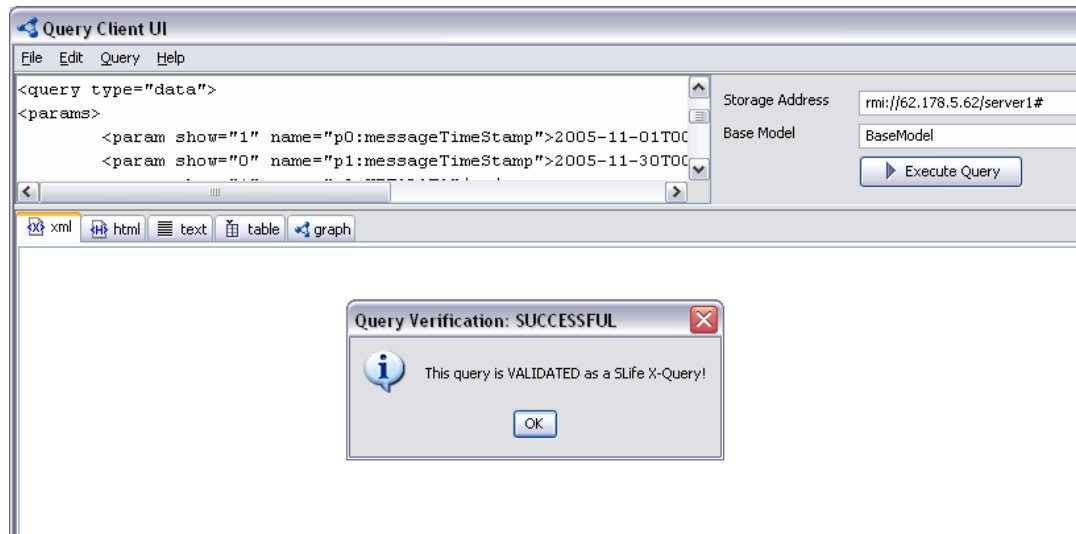


Figure 6.2: The VQL query is validated.

6.2.6 Well-formed and Validated VQL Queries

Checking the validity of the generated VQL queries plays an important role in order to help the user dealing with the virtual queries formulation. As first mentioned on the section 6.2.2 (Figure 6.1), we have XML schemas for validating the VQL queries. Actually, there are XML schemas to checking the well-formedness for the VQL query types. Users can choose to verify their VQL queries before executing them; or letting the VQS do itself before carrying out any VQL queries.

Listing 6.7: The VQL RDF Query Schema

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3     elementFormDefault="qualified">
4     <xs:element name="query">
5         <xs:complexType>
6             <xs:sequence>
7                 <xs:element ref="statement"/>
8                 <xs:element ref="resultformat"/>
9             </xs:sequence>
10            <xs:attribute name="type" use="required" type="xs:NCName"/>
11        </xs:complexType>
12    </xs:element>
13    <xs:element name="statement" type="xs:string"/>
14    <xs:element name="resultformat" type="xs:NCName"/>
15 </xs:schema>

```

In general, in VQL we have two formats for VQL queries which are VQL data queries and VQL RDF-embedded queries. Hereafter, we have created two

⁴ Jena Semantic Web Framework, <http://jena.sourceforge.net/>

⁵ Java RDF, <http://jrdf.sourceforge.net/>

XML schemas for each: the first schema is described in Figure 6.1 for the data queries; and the second one, whose XML code is presented in Listing 6.7, is for the embedded queries.

VQS uses these schema for validating every VQL queries before executing them. If the VQL query is not validated and well-formed, the system will guide the user to correct it by point out where the errors are as in Figure 6.3. In case the user did not specify the type of his VQL query, the system would consequently check out of two VQL schemas.

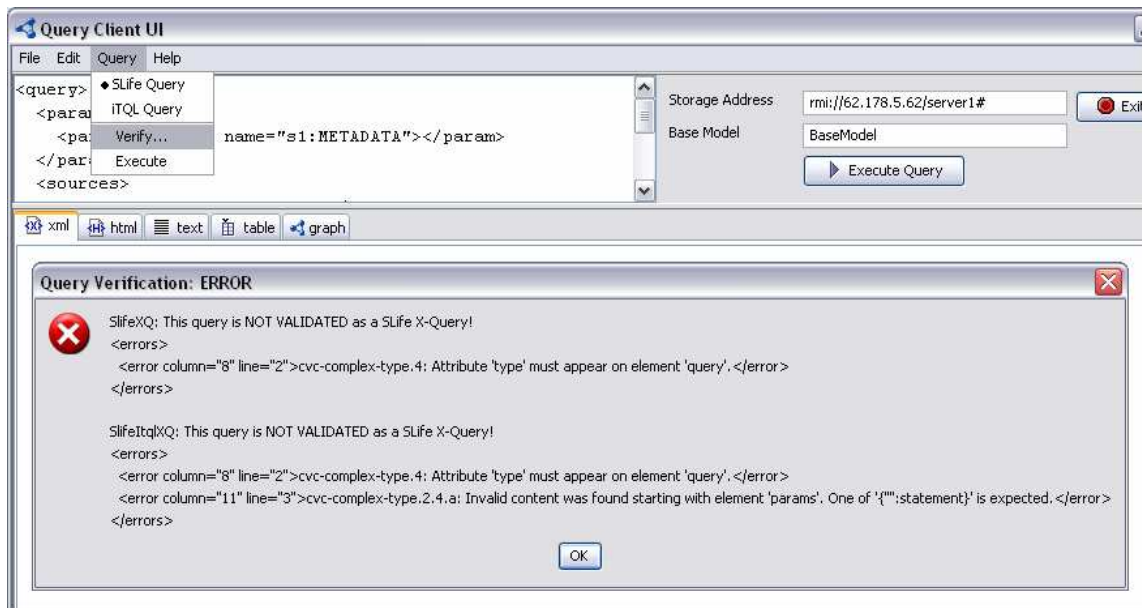


Figure 6.3: The VQL query is not validated.

If the query is form-valid, the system could carry out it continuously. Figure 6.2 shows an example of validating a well-formed VQL query, the information will be shown to the user.

6.3 Query Operators of VQL

In this section, we present *VQL Operatoes* which are deductive queries from the data query type for special operations for making complex queries in simpler manner which helps users “*minimum of words, maximum of information*”. This conforms the principle of building user-centered applications as in [GS03].

6.3.1 GetInstances Operator

GetInstances operator is the common form of VQL data queries. The operator retrieves appropriate information according the criteria described in parameters, sources, constraints of the query. The properties with **show** attribute set to "1" will be involved in the query results.

Listing 6.1 is an example of *GetInstances* operator. As depicted, the query is about retrieving the message's time-stamp from 01/11/2005 to 30/11/2005 of uploaded files and browsed WWW pages. The operators in the constraint part, "dt:gt" and "dt:lt", are combined using boolean 'AND'.

6.3.2 GetInstanceMetadata Operator

This query operator assists the user easily retrieve all metadata properties and their values of resulting instances. This query operator is very useful when the user does not care or not know exactly what properties of data instances are. The case could happen when he/she makes request; or he/she would like to get all metadata of these data items by the simplest way.

Listing 6.8: An Example of the VQL GetInstanceMetadata Query Operator

```

1 <query type="data">
2   <params>
3     <param show="1" name="p0:messageTimeStamp">2005-11-01</param>
4     <param show="0" name="p1:messageTimeStamp">2005-11-30</param>
5     <param show="1" name="p2:METADATA"/>
6   </params>
7   <sources>
8     <source name="fileupload">FileUpload</source>
9   </sources>
10  <relations>
11    <relation id="1" param="p0" source="">dt:gt</relation>
12    <relation id="2" param="p1" source="">dt:lt</relation>
13  </relations>
14  <resultformat>xml</resultformat>
15 </query>

```

In order to make a *GetInstanceMetadata* operator, we must put one parameter in the query document with the reserved string "METADATA". The other parameters could be used for the criteria of the query to filter the query results. The rest of the query document is similar to a normal data query. Listing 6.8 describes a example of this operator. In the example, the query is about getting the *metadata* and their values of uploaded files which are sent from 01/11/2005 to 30/11/2005, as well as the timestamps of these files.

6.3.3 GetRelatedData Operator

In semantic web applications, particularly in the SemanticLIFE system, finding relevant or associated information plays an important role. When we make a query to search for a specific piece of information, we also would like to see associated information to what we found. For example, when we are looking for an email message with a given email address, we also want to see the linked data to this email such as the contact having this email address, appointments of the person in the email, web pages browsed by this person. Obviously, this operator shows the VQL power and obeys the principle of “*minimum words, maximum information*” for users.

Listing 6.9: An Example of the VQL GetRelatedData Query Operator

```

1 <query type="data">
2   <params>
3     <param name="p1:emailTo" show="0">hta@gmx.at</param>
4     <param name="p2:RELATED-WITH" show="1" />
5   </params>
6   <sources>
7     <source name="email">Email</source>
8     <source name="contact">Contact</source>
9   </sources>
10  <relations>
11    <relation id="1" param="p1" source="email" />
12  </relations>
13  <resultformat>xml</resultformat>
14 </query>

```

In order to make a request of this operator, we must identify a parameter containing the a reserved word "RELATED-WITH" in the query document. The `<sources>` element is used to limit a range of data sources in searching associated information as presented in follows (Listing 6.9).

In this example, the query asks for instances of **Email** data source containing a specific email address, e.g. `hta@gmx.at`; and from found messages, the related information in the appreciate data sources, which are identified in `<sources>` of the query, will be located as well.

6.3.4 GetLinks Operator

This query operator operates by using the system’s ontology and RDF graph pattern traversal. The operator aims at finding out the associations/links between instances and objects. For instance, we are querying for a set of instances

of emails, contacts and appointments, and normally, we receive these data separately. However, what we are expecting here is that the links between instances (as well as objects) provided additionally. The links are probably properties of email addresses, name of the persons, locations, and so on.

GetLinks operator helps us to fulfill this expectation. This operator is similar to *GetInstanceMetadata* in the way of exploiting the metastore. While *GetInstanceMetadata* operator tries to get related instances based on analysis of a given link or information and the ontologies, *GetLinks* extracts the associations in instances or objects. In order to make a *GetLinks* operator, the reserved word "SLINKS" (*semantic links*) must be identified in one `<param>` element.

Listing 6.10: An Example of the VQL *GetLinks* Query Operator

```

1 <query type="data">
2   <params>
3     <param show="1" name="p1:emailTo">hta@gmx.at</param>
4     <param show="1" name="p2:conName">Hoang Thieu Anh</param>
5     <param show="1" name="p3:SLINKS" />
6   </params>
7   <sources>
8     <source name="email">Email</source>
9     <source name="contact">Contact</source>
10    <source name="calendar">Calendar</source>
11  </sources>
12  <relations>
13    <relation id="1" param="p1" source="email" or="2" />
14    <relation id="2" param="p2" source="contact" />
15  </relations>
16  <resultformat>xml</resultformat>
17 </query>

```

We distinguish the detecting links between instances and objects as following: if sources are specified in the query document without any parameters except "SLINKS", then the links will be detected between objects. Otherwise, if some parameters are shown, the links are implied for instances' associations. An example of *GetLinks* query operator is presented in Listing 6.10. The query will return the associations between instances having the given receiver's email and the contact name in three data sources **Email**, **Contact**, and **Calendar**.

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.

6.3.5 GetFileContent Operator

The SemanticLIFE system covers a large range of data sources, from personal data such as contacts, appointments, emails to files stored in his/her computer, e.g. the office documents, PDF files, media files. Therefore, a query operator to get the contents of these files is very necessary.

Normally, to carry out this task, we must define two parameters in the query document, the first one is the file path got from the previous query; and the second one is defined with reserved word "CONTENT". In the `<source>` element of the query, a data source is identified as a reference; and the constraint part is often left empty. The query is described in Listing 6.11 is an example of *GetFileContent* operator, where a content of the file having name "CFP_WISM_06.pdf" with its full path will be extracted from the metastore.

Listing 6.11: An Example of the VQL GetFileContent Query Operator

```

1 <query type="data">
2   <params>
3     <param show="0" name="p0:filePath">
4       c:/slifedata/uploadedfiles/2005/11/28/CFP_WISM06.txt
5     </param>
6     <param show="1" name="p2:CONTENT"/>
7   </params>
8   <sources>
9     <source name="fileupload">fileupload</source>
10  </sources>
11  <relations/>
12  <resultformat>xml</resultformat>
13 </query>

```

The files have been uploaded to SemanticLIFE metastore through a Message Handler [AHK⁺04], and they are coded using BASE64 data encoding⁶. The metastore will decode and stores them in the file system, then it carries out a full-text index on these files for faster and more accurate retrieving later on. After executing the *GetFileContent* query, clients receive a BASE64-encoded string encapsulated in a XML document. For the user, what he/she needs is the 'real' contents of the file, and it is the VQS interface's turn to do the job that decodes the content and shows it to the user [HTN06].

⁶ RFC 3548, <http://www.faqs.org/rfcs/rfc3548.html>.

6.4 VQL Parser: Query Languages Mapping

The VQL parser translates the VQL query to correspondent RDF query statements; accesses the metastore to get results, then transforms them to the appropriate format and sends the processed results back to the user. In this section, we present the first stage of a query process in the VQS [HTN06] using VQL that is the mechanism to map VQL queries to RDF queries. The mappings consist of three phases: expressions mapping, syntax mapping; and semantic mapping.

6.4.1 Expression Mapping

Expressions in VQL queries are likely to be mathematical, while the “expressions” in RDF queries are in form of the triples. Therefore, an accurate expression mapping from normal expressions in VQL queries to triple expressions in RDF queries is crucial. VQL carries out this task as following steps:

1. Forming mathematical expressions from elements of a VQL query.
2. Representing the final aggregated expression into an expression tree.
3. From the expression tree, we traverse and formulate triple expressions for RDF query with referring to ontologies and related sources in the VQL query.

Example 6.4.1. Looking back at Listing 6.1 as an example, an expression will be formed from described query’s parts as follows (here *msgTS* is used instead of *messageTimeStamp* for shorter representation):

$$(msgTS \geq \#01/11/2005\#) \cap (msgTS \leq \#30/11/2005\#)$$

From this expression, we can create the expression tree as depicted in Figure 6.4. Using this tree, we generate the triple expressions for the RDF query by traversing the tree. The generated triple expressions are described below in RDF query language.

```
(<slife:msgTS> >= '2005-11-01T00:00:00Z') and
(<slife:msgTS> <= '2005-11-30T00:00:00Z')
```

where, "slife" is the namespace for the ontology and data schema of SemanticLIFE metastore.

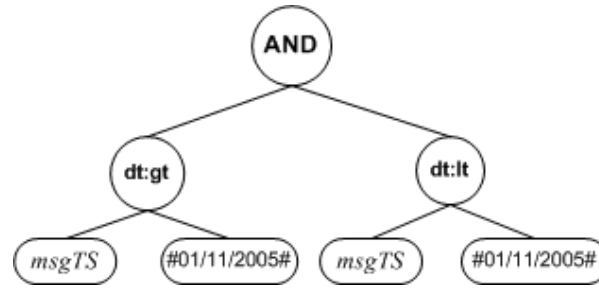


Figure 6.4: The Expression Tree.

Furthermore, the data sources are taken into account during mapping expressions. The specified data sources in the VQL query document (in **sources** part and **<relation>** elements) are used for either identifying the expression applied on them or generating correspondent queries for them. Hence from a VQL query, more than one RDF query are probably generated.

6.4.2 Syntax Mapping

Syntax mapping takes care the issue of translating from VQL query syntax to a RDF query language syntax. VQL queries are actually interpreted as **SELECT** statements of RDF query languages. The **SELECT** statement contains three required clauses **select**, **from**, **where**, and optional clauses such as **orderby**, and **filter**. The syntax mapping will parse VQL query's parts to the clauses of RDF **SELECT** statement(s).

First of all, the **select** clause of the RDF query will be filled by **<params>** parts of the VQL query. The parameters set to "1" will be put as variables of the **select** clause, and unshown parameters (set to "0") are used for forming the criteria only. Additionally, some extra variables will be added in the clause to get the message's URI and the name of data sources. Secondly, the **from** clause will be generated automatically by VQL parser. For the time being, all data sources are stored in one huge metastore with a unique network address. And this network address plugged access protocol will be placed in **from** clause. Thirdly, generated triple expressions will be used for the **where** clause. As discussed, the process of generating the triple expression combines all three parts of the VQL query - **params**, **sources**, and **relations** - along a further analysis by adding more expressions to clarify the criteria.

Last but not least, we have two optional attributes, **order** and **exclude**, for each parameter. If the **order** is set to "1", then the variable will be added into **orderby** clause. And if an **exclude** is specified, an expression for filtering will be added into the **where** clause.

6.4.3 Semantic Mapping

The semantic mapping takes part in both mapping tasks above to resolve semantic ambiguity problems. This is the vital and decisive mapping task of the VQL. The semantic mapping is going to solve the following concerns during query generating process from the user's initial VQL query:

- Disambiguating query items
- Resolving semantic conflicts

Disambiguating query items

This feature ambiguities inside itself. Coping with ambiguous items in the VQL query made by a user is a decisive step in parsing it later on. Here the ambiguity could be in terminological manner, i.e. requested data properties are not clear. For example, a "Name" property in a query is ambiguous because the query parser can not identify which "name" will be extracted, contact name or name of email sender/receiver.

Clarifying these properties could be done by using data source specified in the query and the ontologies of the system. Based on the data sources, the appreciate properties in the ontology will be detected and used instead of ambiguous items. For instance, concerning the "Name" property is described above, this mapping task must rely on the related source described either in source or constraints elements, e.g. **Contact**; after on, based on ontologies, appreciate properties will be located such as `contactFirstName`, `contactLastName`.

Resolving semantic conflicts

In a further disambiguation users' queries, especially when the user asks for information using an ambiguous entity over many data sources. The issue is that how to identify user's "intended" properties for which data sources. For example, "Name" property is constrained with **Email** and **Contact**. If the "Name" properties is constrained with a source identified in a `<relation>` element, then the issue is similar to the discussion above. Otherwise, the query has to:

- get all related properties in system ontology based on specified data sources. In this case, there are probably more than one query generated; or
- suggest the most related properties from ontology based on a *semantic similarity* of properties in the same query. For instance, if other properties are major requesting for **Contact** items, so that the "names" of **Contact** object would be suggested.

Obviously, the strategies could be combined as a aggregated solution in resolving this problem.

Furthermore, all discussions above are mainly focused on VQL data query; however these solutions of The semantic mapping is applied for all types of the VQL query. Generally, all user-entered queries should be check for implied semantic problems as well as the syntax of them. All these problems could be limited by using a query generation interface of the VQS which is discussed details in another context; and the system library (API) as well. Nevertheless, because of the openness of the VQS and VQL, these issues must be taken care.

6.5 Summary

In this chapter we have presented the Virtual Query Language, a design of a query language aiming at a significant complexity reduction in formulating semantic meaningful queries.

Along with the VQS, this query language design helps VQS's users comfortably work with the virtual information. With support of the VQL, the users only care about the concepts (information) of interest when they formulate the requests. The VQL also introduces the special operators deducted from the VQL query data type that support users in making complex queries in a simpler manner, as well as the operators are also used by the VQS itself for further analysis in the information retrieval process.

In addition, in this chapter we also presents the parsing mechanism for VQL. The VQL parser can deal with the syntactic mapping from VQL queries to the RDF queries. It can cope with the semantic mapping: disambiguating the concepts and semantic conflicts .

7 An Innovative Query Formulation

7.1 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 SemanticLIFE's VQS is our approach of using semantic web techniques with an user-oriented method in order to tackle this challenge.

In the VQS-enhanced SemanticLIFE, the user is supported in issuing imprecise queries to retrieve the rich semantic information from his/her historical 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 [HT06a]. The VQS supports users in this nature when querying the information from the huge ontological repository effectively not only in the initial phase with offered 'virtual information' but during the query process with the 'context-based' querying features.

In this chapter, 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 into the form of an ontology; how to represent the "virtual information" to the user for consuming; and how it is used to support the user in generating unambiguous queries. The core of this approach is the *Virtual Data Component* or so-called the *Metadata Storage Sources* which will answer these questions.

The VQS system, with the organization of the 'Virtual Data Component' as a context ontology, will guide its users to go through the system by intelligently recommended query patterns based on the current query context, so-called *query map*, and that context ontology (the virtual information). In our "front-end" approach, this is the most crucial feature that is different to the current ontology-based query systems [HT06a].

7.2 The Virtual Data Component

7.2.1 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 VQC is the module containing the Metadata Storage Sources (MSS). This module acts as a *virtual* information layer allowing the user to *be aware of* the meaning of the stored data sources and he/she can then specify more precise queries as the result.

The VDC harvests the metadata of 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.

7.2.2 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 [AHK⁺04].

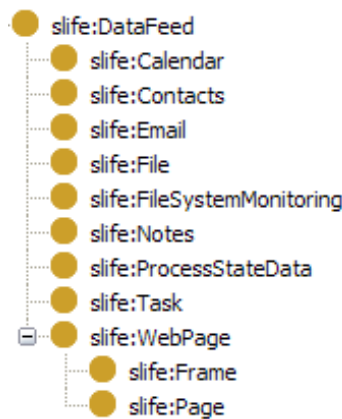


Figure 7.1: A Fragment of the SemanticLIFE’s Datafeeds Ontology

Figure 7.1 presents the data feeds covered by the SemanticLIFE framework in the current prototype. The data feeds are about the personal data of an

individual's diary. From this ontology and the underlined instances, a VDC service is called to extract the metadata, perform some statistical computation and store the information in a new ontology, called the *context ontology*. This ontology is used in the VDC to provide semantic information on the corresponding data sources to the users.

As mentioned, the core of the VDC is a synthesis ontology which is formed from these variety datafeeds ontologies. This task has been done by using MAFRA ontologies merging framework, with “semantic bridge concept” [MMSV02], as the mapping service to merge the ontologies. The process consists of aligning the schemes and merging the instances as well.

7.2.3 Context-based Support of the VDC

The loosely-coupled organization of the metadata storage sources (an ontology of the ‘virtual information’) reflects the flexibility of the Virtual Query System as well as the SemanticLIFE system. Based-on this ontology, the context-based query templates are also categorized according to the concepts. We can apply the VQS or the SemanticLIFE system in different contexts by simply making changes of this ontology.

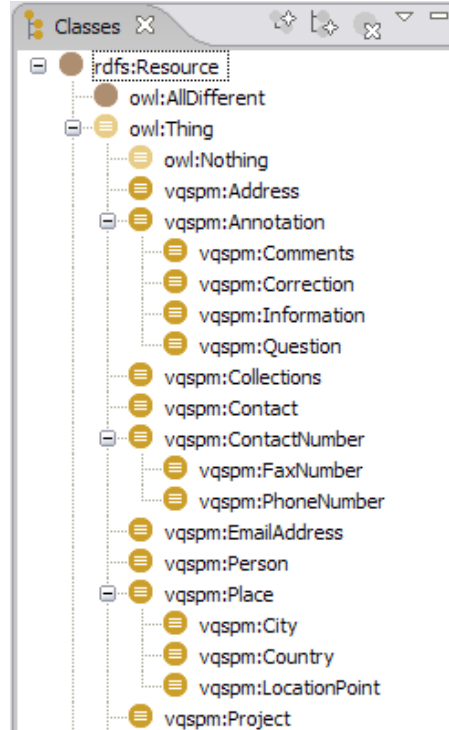


Figure 7.2: A Example of the Virtual Data Component Ontology

The metadata storage sources are constructed as an ontology namely *context metadata* or *context ontology*. By doing the taxonomy and reasoning on the concepts 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 7.2 shows the ontology constructed by merging data sources' schemes and the instances. The ontology in the figure is an ontology for a personal diary recording the daily activities of an individual who works in some projects. The extracted metadata will be fetched from system datafeeds ontologies and put into the VDC's context ontology and conformed to its hierarchy. For example, the "Place" class is an abstract concept and formed from classes "Contact", annotations, maps and their instances. This make the semantic information becomes more contextually, and we call this process is the *concept contextualization*.

Definition 7.1. A *concept contextualization*, Con , in VQS is a transformation of concept (class) C of system ontology, O_1 , to the context ontology, O_2 . The relationships between C and other concepts in O_2 will be reformed.

$$Con : \langle C, O_1 \rangle \longmapsto \langle C, O_2 \rangle$$

Hence, the context ontology could be redefined based-on the concept contextualization as following:

Definition 7.2. The *concept ontology*, CO , of the VDC is:

$$CO = \bigcup_{i=1}^n Con_i$$

where, n is the number of concepts.

The metadata in the VDC's context ontology ('context ontology' in short) is still associated to the 'real' metadata in the system ontology. The metadata in the new context is used for the VQS's purposed in rendering information for presentation and reasoning during the querying process.

Additionally, the VDC contains the summary information for each data sources over the customized time-lines such as total items, number of items for each data sources. 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.

The Virtual Data Component is the typical feature of our system compared with the other systems mentioned in Chapter 4. The basic idea behind is quite

simple: *If the user is aware of his/her data, then he/she could generate more unambiguous request.* In the one hand, this reduces the complexity of the query refinement process. And in another hand, with the VDC as a context ontology, the system could flexibly adapt to a new scenario by changing the correspondent context ontology.

7.3 Context-based Query Formulation in the VQS

In the VQS, the user is not only supported in the query formulation by the ‘virtual’ information, but also during his/her querying process by a load of features introduced as follows.

7.3.1 The VQL Query Template

Definition 7.3. A *VQL Query Template* (VQL-QT) is an abstract query pattern, which is attached with a specific VQL query, containing the concepts and resources for the user querying process.

VQL-QTs are classified on these concepts and resources so that the appropriate template will be recommended to the user based his/her querying context. VQL-QTs help the user generating clear requests by only replacing the values into the selected query pattern.

VQL Query Template Syntax

In the VQS, the VQL query templates, so-called *VQL query patterns* (VQL-QPs), are defined to assist the user in formulating unambiguous queries. The query templates contain the VQL queries with the necessary parameters and the associated data sources. A VQL-QT or VQL-QP mainly consists of:

- A VQL query in form of a *query file name*.
- *Parameters* containing the values from the user’s input .
- *Resources* involved in the querying process.

and two optional parts:

- *Description* of the VQL-QT which is used for display to users.
- *Query result format* to specify the dedicated format of the results.

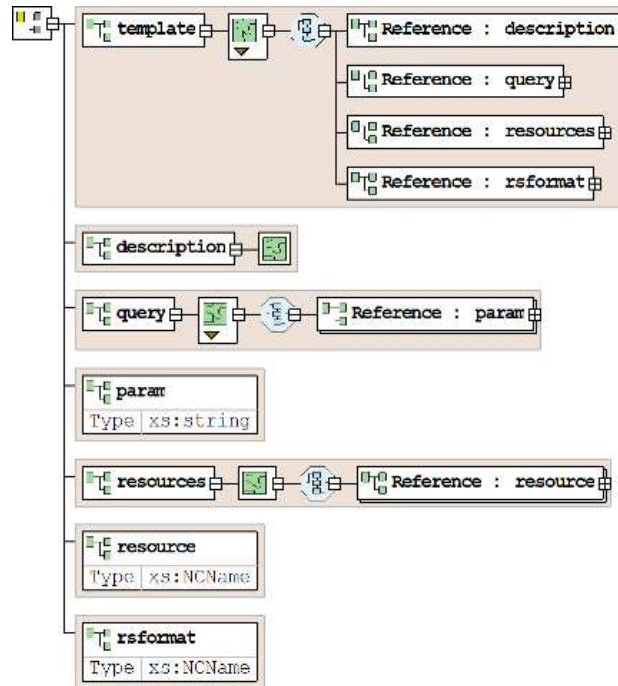


Figure 7.3: The schema of the VQL query template

The structure of a VQL query template is illustrated in Figure 7.3, in which the components of the VQL query template are shown in a hierarchical tree.

Listing 7.1 is an example of a VQL query template. That VQL query template is about retrieving the locations of all web pages found by Google search engine¹ 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 `<resources>` part. The parameters are put in sub parts `<param>` and these parameters could be changed or added by the user according the information of interest.

VQL Query Template in Use

The VQL query templates are classified on the concepts of the VDC's context ontology as well as their involved data sources—so-called *resources*—they are involved. The concepts of the VDC's context ontology are reflected in parameters of the VQL-QPs. Based on this classification, the VQS will match the VQL-QTs with the user query context and the appropriate VQL-QP will be delivered to the user as a recommendation.

¹ Google, <http://www.google.com/>

Listing 7.1: A VQL Query Template Example

```

1 <?xml version="1.0"?>
2 <template type="vql">
3   <description>Finding a location of webpage browsed by
4     a person using Googl search engine</description>
5   <query name="webpersonse.vql">
6     <param>?location</param>
7     <param>?tiemStamp</param>
8     <param>?person</param>
9   </query>
10  <resources>
11    <resource>Webpage</resource>
12    <resource>Location</resource>
13    <resource>Person</resource>
14  </resources>
15  <rsformat>xml</rsformat>
16 </template>

```

When the template is in use, the attached VQL query will be loaded and its variables are then replaced by the template's parameters. The VQL query could be continually edited by the user afterward dependent on the user's interest.

Furthermore, during the VQS user's querying process, a new VQL-QT could be also created. The new VQL-QTs creation can be carried out in two ways: firstly, it comes from an analysis based-on the querying context of the user. Secondly, from the editing of an existing VQL-QT which the user needs to save for later use in form of another VQL-QT.

7.3.2 VQS User Context

Definition 7.4. The *VQS User Context* (VQS-UC) is a set of requested concepts from the context ontology linked to the system ontologies and associated resources, as well as the properties are in action.

Let call U is a VQS-UC, we have:

$$U = \langle C, R, P \rangle$$

where C is the set of the underlying concepts, R is a set of associated resources, and P is a set of queried properties.

In general, in order to formulate a VQS-UC, an analysis process is carried out based on the objects in the querying process. The analysis process counts on the following metadata:

- Querying concepts, their associated resources (based-on the system ontologies) and the querying properties.
- The detected semantic links (obtained from VQL's *GetLinks* operator) would help in term of finding the relationships of the information.
- And the query results the last execution of the user's querying process.

A VQS-UC is used to keep the user's querying concepts and querying space that is about how the concepts and query results associated. From that, the new knowledge will be deducted for ideas of the further requests.

7.3.3 VQS Query Map

Definition 7.5. A *VQS Query Map* (VQS-QM) is a network of VQL query templates, in which the nodes are the VQL query templates and the connections are the related concepts and their properties. $M = \langle T, C, P \rangle$ is a VQS-QM, where T is the set of VQL-QTs, C is the set of concepts of the underlying resources, and P is the set of querying properties.

Generally, with the associated data sources and the VDC's context ontology, the VQL query templates creates a *query map* to make the connection network among the templates and underlined resources.

According to the connections between the templates, when a VQL-QT is chosen for making the new queries, the system also recommend the linked VQL-QTs. 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.

7.3.4 Context-based Querying

The Virtual Data Component also enhances the query process by a "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 to the user.

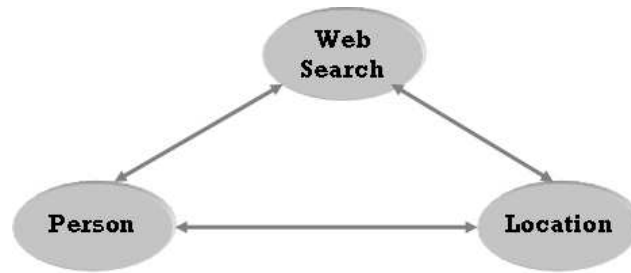


Figure 7.4: An Example of Context-based Querying

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* as depicted in Figure 7.4. 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.

7.3.5 Context-based Query Results Representation

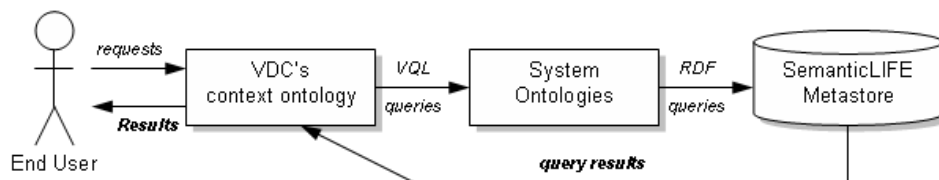


Figure 7.5: The Process of Returning Query Results in the VQS

The query results back to the VQS according to the schema the SemanticLIFE metastore. Therefore, for the user's information consuming, the results is put into

the “context” of the query concerning the VDC’s ontology before presenting to the user as depicted in Figure 7.5.

As described in the figure, the results from the SemanticLIFE metastore are based on the system ontologies; therefore, via the VDC, the query results are *contextualized* with the VDC’s context ontology. Moreover, based-on the VDC’s ontology, the related information associated to the relationships within its hierarchy is also presented as the recommendation to the user.

7.4 Semantic Navigation with the VQS

7.4.1 VQS Semantic Traces

The traces are to keep tracing on successful querying sessions. VQS keeps traces in form of query pipelines with annotation and classification, i.e. every trace is attached with pipelines, used resources and annotation of each pipeline.

Definition 7.6. A *trace* is an extended pipeline with the user’s annotation and the context applied. Let call T is a trace, we have:

$$T = \{P_i, CT_i\}_{i=1-n}$$

where T is an ordered set with P_i is a the i^{th} query pipeline and CT_i is the attached i^{th} context, which is a finite set of m concepts:

$$CT = \{C_k\}_{k=1-m}$$

where C_k is a concept.

Based-on these traces, the users could resume and restore last querying sessions. Moreover the traces could be used as the guidelines for the new querying processes as the recommendations.

7.4.2 Context-based Navigation

The VQS is aiming at not only information retrieval aspects, but in inventing a new way of query and process the query results for the next round. With helps of the VQS-UC, VQS-QTs, VQS semantic traces, the feature of system navigation by semantics, *semantic navigation*, is realized.

Initially, the VQS’s user would generate a new query based-on the virtual information of the VDC. The user could also select from the commonly used

VQL-QTs and customize according to his/her interest. After this initial phase, the user enters the process of retrieving information, the system helps him/her for the next query formulation. This could be done based on many VQS's features such as the user's querying context, his/her query map, the query patterns in form of the VQL-QTs, as well as the query results, especially the results of the VQL operators which is rich of semantics.

In addition to the semantic traces, the VQS supports the user go through the system by navigating source by source, concept by concept which would entertain the user by suggestions and offered query patterns associated with his/her querying space (query context and query map).

7.5 Summary

In this chapter, we have presented two main points: the Virtual Data Component (VDC) organization; and a new approach for query formulation during the user's querying process. The VQS activities basically rely on the way how the metadata organized and reflected in the VDC of the VQS. The VDC plays a vital role in our approach. It is the base for supporting the innovative query formulations of the VQS.

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.

In aggregation of the innovative features, the new way of support users in exploiting the system—semantic navigation—is also discussed with helps of the semantic traces, which are used to keep users' querying session, and such stuffs like VQL-QT, VQS-UC, query map.

8 Implementation Results

8.1 The SemanticLIFE Infrastructure

The SemanticLIFE framework has been built on a very high-loosely coupling architecture: plugin architecture. Each component acts as an independent plugin with its commonly-agreed interface about input and output data. This architecture design deliberates its components from their development and implementation.

8.1.1 A Short History

The general design of the SemanticLIFE framework has not been changed so much from “the day one”. However there are some changes in the details of technologies applied and approaches in building components. Figure 8.1 shows the first architecture design of the SemanticLIFE framework.

The SemanticLIFE framework is built based-on the Java Plugin Framework (JPF)¹, an opensource platform for developing plugin applications. The enabler of the system is the *Message Handler* which processed every messages come through the SemanticLIFE system, i.e. incoming, outgoing and the internal exchange messages. The Message Handler component is a web service providing the service call for all other components. Every SemanticLIFE’s component (plugin) is registered into the system’s JPF, and can send information to each other by calling a service of the Message Handler by mentioning source and destination components.

In the first and second prototypes, our SemanticLIFE framework had the following components successful developed:

- Datafeed modules

¹ <http://jpf.sourceforge.net/>

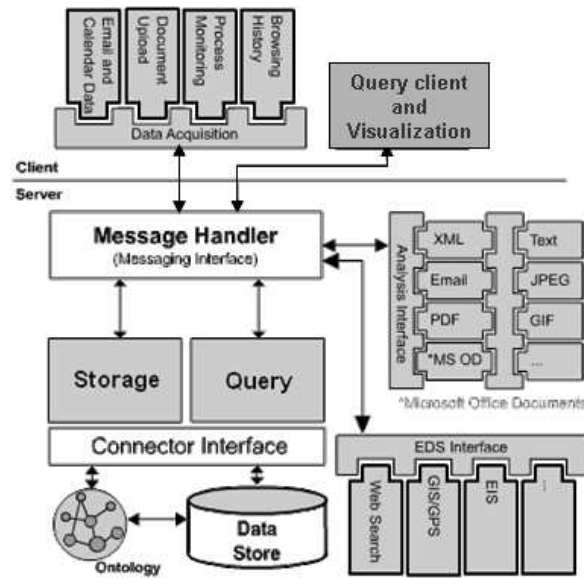


Figure 8.1: The first architecture design of the SemanticLIFE framework

- Message Handler
- Simple Analysis Module
- Storage Module
- Search/Query Module
- Simple Presentation Module (Visualization)

Among the above modules, the Datafeed modules have been changed significantly. From the beginning of our project, a set of datafeed modules were built to deal with different data sources, such as emails, personal data (appointments, tasks, notes), documents, web browsing sessions, chat logs with efficient synchronization mechanism to detect changes from the data sources. For example, the Datafeed module for personal Outlook data had been developed with a synchronization mechanism—SyncML²—reflected in its Java implementation, Sync4j³; and it worked well with all Outlook data types.

However, the problems we have to deal with these datafeed modules are caused by the versioning issue, i.e. these applications must be kept updated for the changes of data sources' versions. This leads to another approach that is not so much version-dependent.

² <http://www.snycml.org/>

³ <http://www.sync4j.org/>

8.1.2 Current Architecture

JPF is an excellent platform for plugin-based application development; however when we started a new prototype for the SemanticLIFE with several new requirements and higher demanding tasks we decided to choose the Eclipse platform with its innovative features. The new architecture of the SemanticLIFE framework is depicted in Figure 8.2.

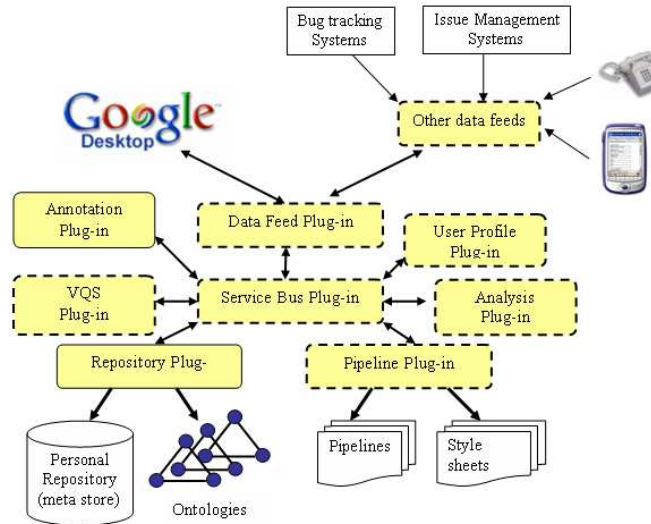


Figure 8.2: The New Architecture of the SemanticLIFE Framework

The main changes from the first two prototypes are:

1. Conform SOA software architecture.
2. Components are plugins offering services.
3. The Message Handler is now replaced by the *Service Bus* for registering and invoking the services.
4. SemanticLIFE framework is now based-on the Eclipse platform to benefit its powerful features such as PDE⁴, RCP⁵, ECP⁶, GMF⁷ and SWT⁸.
5. A new architecture is developed for the service transparency and collective services invocation called *Service Oriented Pipeline Architecture* (SOPA).
6. The Datafeed modules are replaced by the GoogleDesktop plugin in order to collecting data from the personal storage without considering the data source versioning.

⁴ Plug-in Development Environment, <http://www.eclipse.org/pde/>

⁵ Rich Client Platform, <http://www.eclipse.org/rcp/>

⁶ Eclipse Communication Framework, <http://www.eclipse.org/ecf/>

⁷ Graphical Modeling Framework, <http://www.eclipse.org/gmf/>

⁸ The Standard Widget Toolkit, <http://www.eclipse.org/swt/>

The data acquisition module—Datafeed module—is developed with help of Google Desktop tool. The Google Desktop is becoming a ‘middleware’ for upper level applications to access the data on computers with the powerful index mechanism and a wide range data cover and monitoring such as files, documents, personal data stored in Outlook.

8.2 SemanticLIFE’s SOPA

In this section, we introduce an service-oriented architecture which is the heart of the SemanticLIFE architecture. This part is a contribution mainly from [And06] firstly designed for specific tasks inside the SemanticLIFE framework. Nevertheless, it can be applied in a wide range of SOA’s applications.

8.2.1 SOPA – The Service-Oriented Plug-in Architecture

Services and pipelines are grounding components of our proposed Service-Oriented Pipeline Architecture (SOPA). The services may range from GUI services (a.k.a. visual plug-ins) to business processing components (Web Services). On the other hand, pipelines describe the composition of these services to fulfill particular tasks. $\langle S, P \rangle$ is a *SOPA* where the services S could be GUI services S_v , internal web services S_w , and external web services S_x i.e. $S = \langle S_v, S_w, S_x \rangle$. The pipelines P orchestrate different business services (S_w and S_x) and apply transformations T to render the results back to the user or other services i.e. $P = \langle S_w, S_x, T \rangle$.

8.2.2 The SemanticLIFE’s Service Bus

Based-on Eclipse PDE and its *extension point* mechanism, we have built a solution for our integrating and publishing web services in the SemanticLIFE framework. One part of our proposed solution is *Services Bus* offering the extension point for service developers to publish their standard Java classes as web services. The standard extension point mechanism of Eclipse facilitate visual configuration of extensions with the extension provider. During the application start-up, the Services Bus loads all the connected services and automatically deploys them using embedded Jetty and Apache AXIS⁹. The deployment scripts are created on the fly from the service description. Thus developers can, at the same time, benefit from Eclipse RCP and Java web services in the similar and coherent

⁹ Apache Axis, <http://xml.apache.org/axis/>

mechanism. Lastly, the Services Bus uses the standard WSDD¹⁰ and WSDL¹¹ conventions for the service configuration. Different aspects of Services Bus plug-in are discussed in detail below.

Plug-n-Play Web Services

The rational behind the development of Services Bus is to achieve the vision of plug-n-play web services using plug-in and extension mechanism of Eclipse platform. First of all an extension-point was configured by following the service specification and deployment standards such as WSDL and WSDD. An abridged version of the extension point schema is depicted in Listing 8.1.

Listing 8.1: The business services extension-point schema

```

1 <schema targetNamespace="at.slife.sbus">
2   <element name="service">
3     <complexType>
4       <sequence>
5         <element ref="operation" minOccurs="1" maxOccurs="unbounded"/>
6       </sequence>
7       <attribute name="name" type="string" use="required"/>
8       <attribute name="class" type="string" use="required"/>
9       <annotation>
10        <appInfo><meta.attribute kind="java"/></appInfo>
11      </annotation>
12    </complexType>
13  </element>
14
15  <element name="operation">
16    <complexType>
17      <sequence>
18        <element ref="parameter" minOccurs="1" maxOccurs="unbounded"/>
19      </sequence>
20      <attribute name="name" type="string" use="required"/>
21      <attribute name="returnType" use="required"/>
22    </complexType>
23  </element>
24
25  <element name="parameter">
26    <complexType>
27      <attribute name="type" use="required"/>
28      <attribute name="name" type="string"/>
29    </complexType>
30  </element>
31 </schema>
32
```

Importantly the Services Bus exposes this extension point whereas the web services developers consume it to publish standard Java classes as services (see Figure 8.3 and Listing 8.2). Thus the web services could be developed and maintained analogous to other Eclipse plug-ins. The Services Bus on the other hand

¹⁰ Apache Axis WSDD, <http://xml.apache.org/axis/wsdd/>

¹¹ Web Services Description Language, <http://www.w3.org/TR/wsdl/>

reads configuration details of all the connected services during application start-up. It then *automatically creates the WSDD based deployment script* and uses embedded Jetty and Apache AXIS to complete the task.

Listing 8.2: Abridged version of a service extension description

```

1 <extension point="at.slife.sbus.services">
2 <service class="at.slife.query.QueryService"
3     name="at.slife.query">
4     <operation name="runQuery" returns="string">
5         <parameter name="queryStatement" type="string"/>
6         <parameter name="resultFormat" type="string"/>
7     </operation>
8 </service>
9 </extension>

```

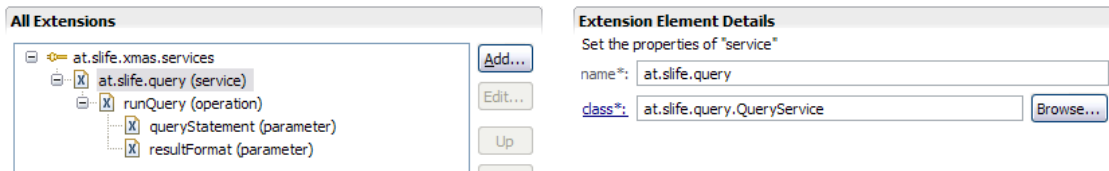


Figure 8.3: An query service extension in the SemanticLIFE framework

Service Call Transparency

The Services Bus can be seen as the door to the SOPA system. It is responsible for routing the service call requests to the actual connected service. An important functionality of Services Bus is to deliver a lever of abstraction between system services which greatly improves the flexibility of SOPA systems. Thus it provides a uniform access layer and transparency to internal and external services.

Listing 8.3: Calling a service plugged into the Services Bus

```

1 Object [] params = ...
2 Call client = new Call("at.slife.query");
3 Object result = client.invoke("runQuery", params);

```

The services plugged into the Services Bus could be called using either the utility classes provided by the Services Bus (see Listing 8.3) or by using Apache AXIS. The former shares the same naming conventions for class and method names of the later, and both type of calls return exactly the same results. Additionally for local services specifying only their name is sufficient but calling external web services requires providing complete end-point URI.

8.2.3 Services Pipeline

A *Pipeline* in SOPA terminology is a uniquely named set of service-calls and intermediate transformations. The pipeline plug-in enables the SOPA systems to realize scenarios based on the basic services and the pipelines. The pipeline idea has been inspired from Apache Cocoon¹² which is a web development framework built around the concepts of separation of concerns and component-based web development. Cocoon implements these concepts around the notion of ‘component pipelines’, each component in the pipeline specializing on a particular operation. This makes it possible to use a LEGOTM-like approach in building web solutions, hooking together components into pipelines without any required programming.

The pipelines and their corresponding structure are defined using an XML structure that specifies the pipeline components and relevant transformations. Listing 8.4 shows the basic structure of a typical pipeline:

Listing 8.4: A Simple Pipeline

```
1 <pipeline name="square">
2 <parameters>
3   <parameter name="num" type="xsd:double"/>
4 </parameters>
5 <call service="org.example.arithmetics" operation="multiply"/>
6   <parameter>{num}</parameter>
7   <parameter>{num}</parameter>
8 </call>
9 <transform method="xml" stylesheet="result.xsl"/>
10</pipeline>
```

As shown above a pipeline is identified by its name (line 1). Each pipeline may receive some input parameters that might be used anywhere inside the pipeline’s scope. Lines 2 to 4 show the parameter section and definition of a parameter called "num". The most interesting part of a pipeline which distinguishes our approach from other such solutions is the service-call part. At line 5 the "multiply" operation of the service "org.example.arithmetics" is requested. The operation call can consume parameters of the pipeline. It is important to mention that the services in a SOPA system are not limited to those provided by other plug-ins but also include pipelines and external Web Services (distinguished by complete end-point URI). We will discuss this issue in depth afterwards.

The results returned by the services may be transformed during the execution of a pipeline. This feature let the results be transformed and converted to required format. The transformation is performed by applying an XSLT transformation

¹² <http://cocoon.apache.org/>

to the current pipeline results. The pipeline plug-in keeps the results internally and finally at serialization phase the results are rendered in required format. The supported serialization formats are TEXT, XML, HTML, and XSWT¹³.

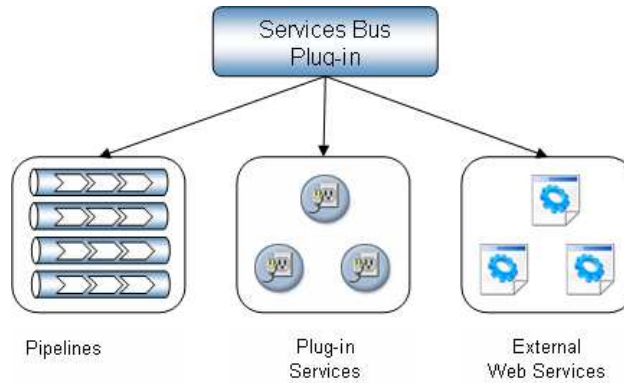


Figure 8.4: Service transparency in the SOPA architecture

As explained in the previous section, the available services in the SOPA environment are routed via the Services Bus plug-in; i.e. all services will be requested from Services Bus which is responsible for finding and then invoke the corresponding service to do the task. This feature provides a service transparency in the whole SOPA environment.

As stated earlier the services in SOPA are not limited to plug-in exposed services but optionally may include pipelines and external Web Services. As a result the SOPA system brings the service orchestration scenarios to a new horizon. The business scenarios developed under eclipse programming framework can combine resources coming from internal or external components via a single service routing plug-in (Services Bus plug-in). Figure 8.4 depicts the service transparency.

8.3 The Virtual Query System: Specifications

Based-on the SemanticLIFE infrastructure with the basics of SOPA and Service Bus features, the VQS module and its components have been developed as plug-ins of the whole SemanticLIFE framework. The VQS is a integration plugin containing all its component plugins with service extensions.

¹³ <http://eclipsewiki.editme.com/XSWT/> or <http://xswt.sourceforge.net/>

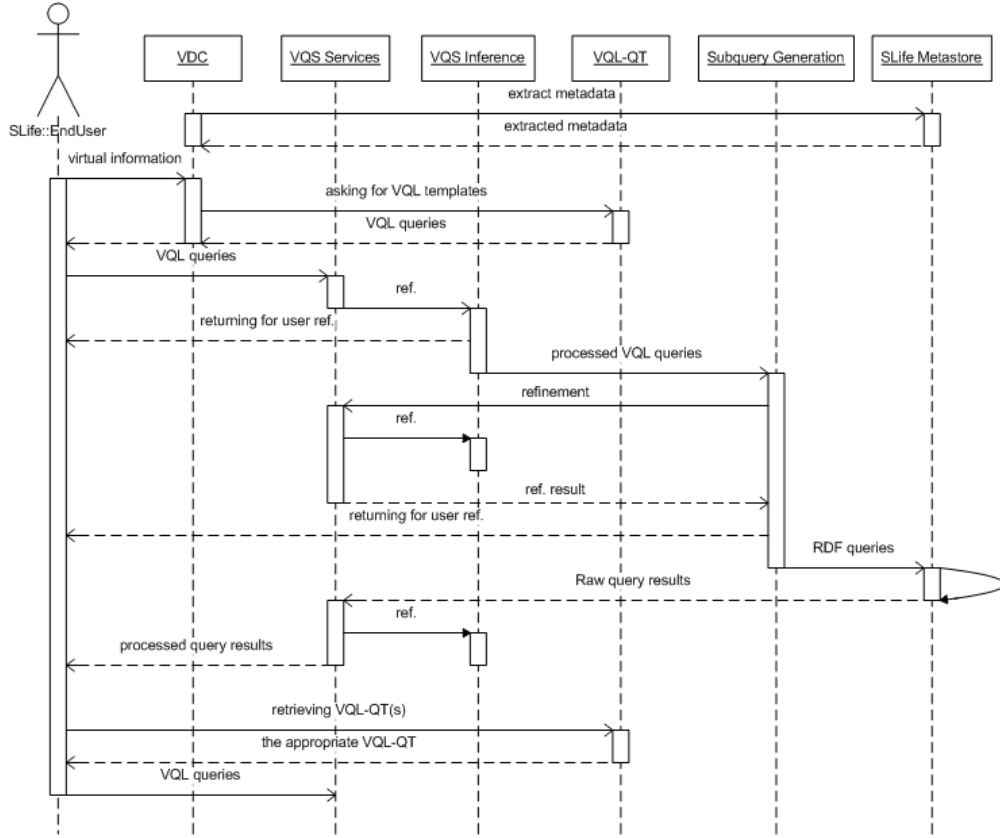


Figure 8.5: VQS Workflow in its UML Sequence Diagram

8.3.1 VQS Workflow

The detailed workflow of the VQS is described in the UML sequence diagram depicted in Figure 8.5. Here, at first, a process of fetching metadata from the SemanticLIFE metastore and organized then into the VDC of the VQS as the context ontology. This process only run once and it would be updated if the ontologies of metastore is changed.

The overview of the VQS workflow has been mentioned in Section 5.3.2, where we only highlight the other important points. Firstly, as described in the Figure 8.5, in the phase of query formulation, the user could use the VQL query templates which are pre-defined query patterns of the VQS. This helps the users in the first steps. The VQL-QTs could be retrieved directly by the user with(out) referring to the selective information of the VDC. Moreover, after the initial phase, the VQL-QTs would be offered automatically based-on the user query context.

Next, the VQL queries delivered to the subqueries generation phase are semantically clarified based on the system ontologies. The query refinement is

performed semi-automatically: referring to the VQS services to detect the ambiguities during this phase; the generated subqueries are updated by the user if necessary as a feedback to finalize the execution of the RDF queries. Finally, the query results are again “contextualized” by VQS services before representing to the user.

8.3.2 The Used Techniques

In the VQS implementation, we have taken the SemanticLIFE architecture as the baseline and inherited the innovative features of the Eclipse platform. We have used the Java open-source frameworks in form of the Semantic Web applications, the ontology engineering and the web services.

The Eclipse Platform

The Eclipse platform gives a technical background not only for developing the basic SemanticLIFE architecture, but also its modules as plugins using Eclipse PDE. In the development of the VQS and its components, the help of PDE, RCP, SWT is very valuable. Particularly, in the Eclipse-based VQS GUI design, we have used WindowBuilder Pro¹⁴, a product of Instantiations¹⁵.

WindowBuilder Pro is a powerful and easy to use two-way Java GUI designer based on the Eclipse SWT technology. It is composed of the SWT Designer and the Swing Designer and is a very easy to create Java GUI applications without spending a lot of time writing code to display simple forms.

The Semantic Web Framework

The Semantic Web framework is mainly used in our development is the Jena Semantic Web Framework¹⁶ of HP Labs. Jena is a leading Java framework in the area of building Semantic Web applications. Jena provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Research¹⁷.

With the new releases (since version 2.4), Jena has integrated the Lucene index engine¹⁸ to provide a powerful full-text searching feature for the SPARQL

¹⁴ <http://www.windowbuilderpro.com/>

¹⁵ <http://www.instantiations.com/>

¹⁶ <http://jena.sourceforge.net/>

¹⁷ <http://www.hpl.hp.com/semweb/>

¹⁸ <http://lucene.apache.org/>

query language. This help to increase the precision in searching documents in the SemanticLIFE framework.

The Ontology Mapping Framework

MAFRA – Mapping FRamework – is a conceptual description of the ontology mapping process. Ontology mapping is the process where semantic relations are defined between two ontologies at the conceptual level which in turn are applied at data level transforming source ontology instances into target ontology instances.

The MAFRA Toolkit¹⁹ implements a specific architecture for MAFRA. The architecture of the system is based on the notion of Service which represents not only the system transformation capabilities, but also the expertise in the manipulation of specific semantic relations.

The Web Services Framework

Finally, for the backbone of the whole system, which connects the services extensions offered, invokes the services or related tasks, the Apache Web Services frameworks²⁰ have been used in our SemanticLIFE development. The VQS service development inherits from the baseline system for its service extension development.

The XML parser

A XML parser is an essential part in our approach and development. Our VQL queries, VQL query templates, query pipelines and traces are coded in XML-format. With the enhanced features, we choose DOM4J²¹ as our XML parser in the VQS development.

8.3.3 The VQS Plugins

The VQS implementation has been developed with three plugins: the core query execution unit, the VQS components and the VQS query GUI plugins.

¹⁹ <http://mafra-toolkit.sourceforge.net/>

²⁰ <http://ws.apache.org/>

²¹ <http://www.dom4j.org/>

Query Execution Plugin

This plugin is the lowest component in the VQS architecture as it works directly with the back-end database. The query execution unit contains internal service extensions such as: the query execution, the query results transformation (e.g. transforming the query results to the specific format such as XML, JSON, RDF graphs, or formatted text); and an aggregated query invocation that consists of the query execution and results transformations. The declaration of this plugin into the SemanticLIFE master service is shown in the Figure 8.6.

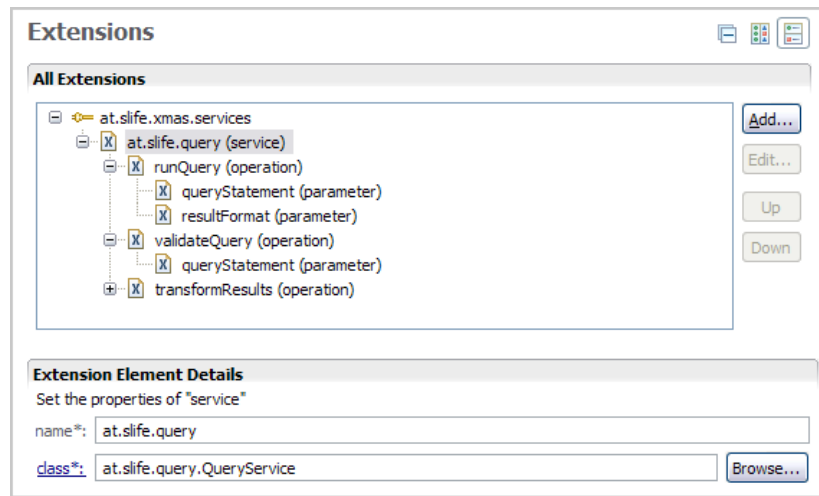


Figure 8.6: The Declaration of the Query Execution Plugin

VQS Components Plugin

This plugin is the main implementation of our query system which contains the main features of the VQS. The services are mostly for internal use of the VQS querying process; however, some service extensions are offered to other external uses such as getting the ‘virtual’ information, retrieving the VQS-UC, executing VQL queries, and getting the VQL-QT(s). The declaration of the VQS components plugin into the SemanticLIFE framework is shown in Figure 8.7.

VQS User Interface Plugin

This plugin is the top layer in the VQS architecture as it works interactively with the SemanticLIFE’s users. The query interface plugin consists of functional windows (views) built on Eclipse RCP ViewParts and SWT widgets. These components are then organized in a separate Eclipse perspective associated with the

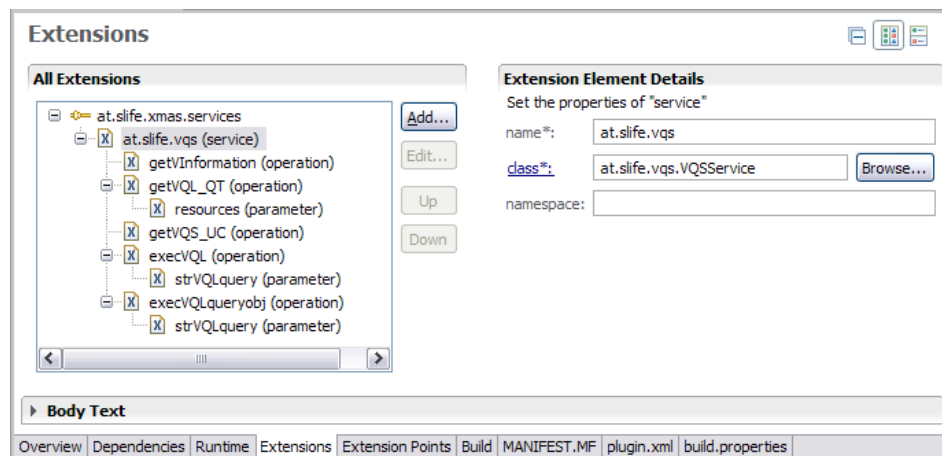


Figure 8.7: The Declaration of the VQS Components Plugin

main SemanticLIFE perspective. The declaration of the query interface plugin into the ‘backbone’ of the SemanticLIFE framework is shown in the Figure 8.8.

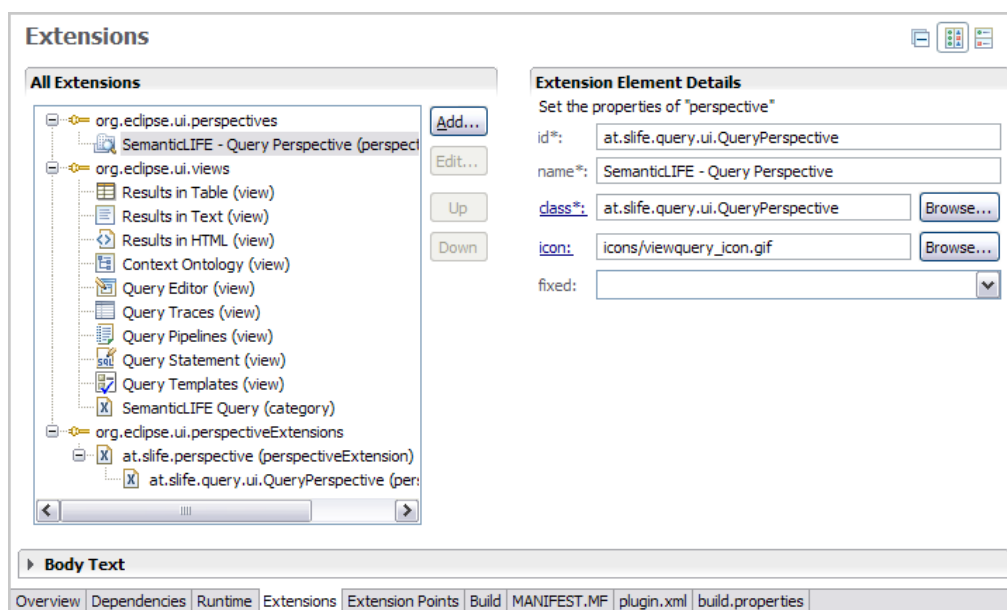


Figure 8.8: The Declaration of the Query Interface Plugin

With the above declarations, each plugin needs an interface to “plug” (register) into the SemanticLIFE plugin infrastructure; so that when the main application runs, all registered plugins will automatically be loaded. The following listing is an example for registering the VQS query UI plugin:

Listing 8.5: The Query UI Plugin Listing

```

1 public class QueryUIPlugin extends AbstractUIPlugin
2 {
3     private static final String PLUGIN_ID = "at.slife.query.ui";
4
5     //The shared instance.
6     private static QueryUIPlugin plugin;
7
8     public QueryUIPlugin() {
9         plugin = this;
10    }
11
12    public static InputStream getResource(String res)
13        throws IOException {
14        Path path = new Path(res);
15        URL url = Platform.find(plugin.getBundle(), path);
16        return url.openStream();
17    }
18
19    public void start(BundleContext context)
20        throws Exception {
21        super.start(context);
22    }
23
24    public void stop(BundleContext context)
25        throws Exception {
26        super.stop(context);
27        plugin = null;
28    }
29
30    /**
31     * Returns the shared instance.
32     */
33    public static QueryUIPlugin getDefault() {
34        return plugin;
35    }
36
37    public static ImageDescriptor getImageDescriptor(String path) {
38        return AbstractUIPlugin.
39            imageDescriptorFromPlugin(PLUGIN_ID, path);
40    }
41
42    protected void info(String message) {
43        plugin.getLog().log(new Status(IStatus.INFO, PLUGIN_ID, 0,
44            message, null));
45    }
46
47    protected void error(String message, Exception exp) {
48        plugin.getLog().log(new Status(IStatus.ERROR, PLUGIN_ID, 0,
49            message, exp));
50    }
51 }

```

8.4 The Virtual Data Component

As mentioned in the previous chapter, the VDC contains the virtual information stored in a context ontology. The context ontology can be customized by the usage of the system and the user. For example, the system could have been used in use case of personal project management system, therefore the VDC's context ontology would reflect the scenario as depicted in Figure 8.9.

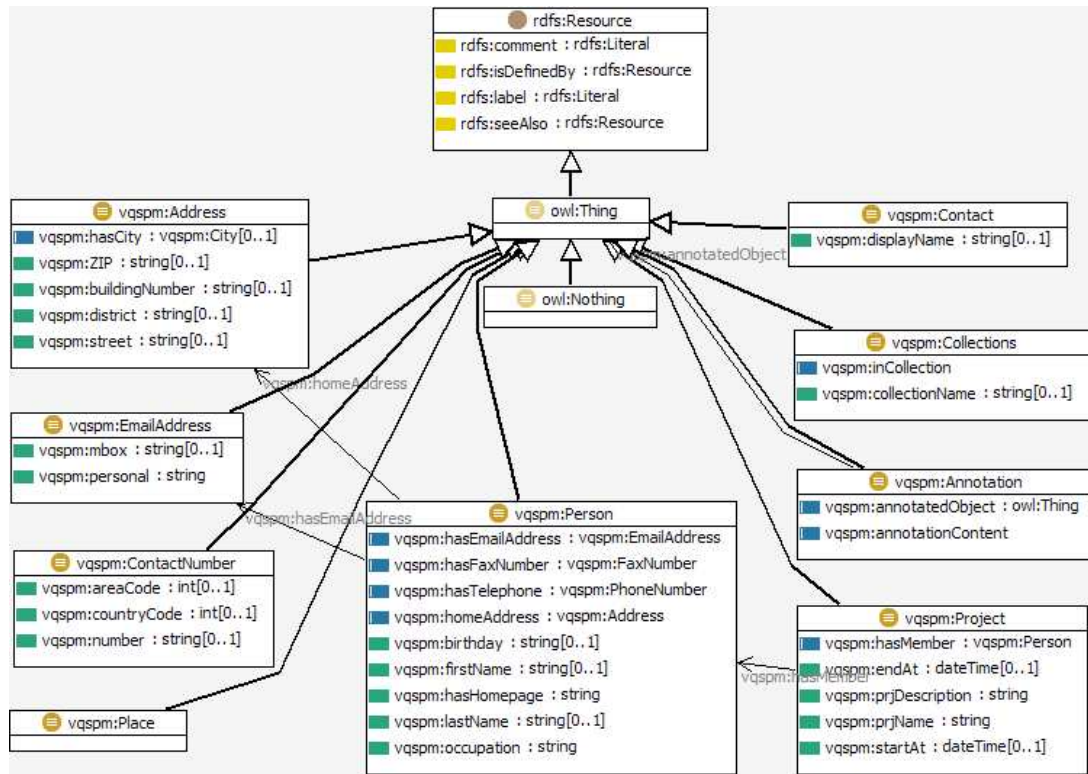


Figure 8.9: Project Management Context Ontology Diagram

The VDC interface display the virtual information aggregated from the data of SemanticLIFE metastore with reference to the context ontology and the system ontology. Figure 8.10 is a screenshot of the VQS interface containing the VDC component.

In the Figure 8.10, the VDC component is reflected in two views, the first one is the left-upper views which contains the context ontology; and the second one is the middle view with tab entitled “*Query Home*”. This view shows the virtual information extracted from the SemanticLIFE metastore in form of the summary information of the personal data. In this ‘home’ window, there are two tabs for presenting the virtual information: the basic and advanced level. The basic view is just for basic information of the user’s personal data; and the advanced view

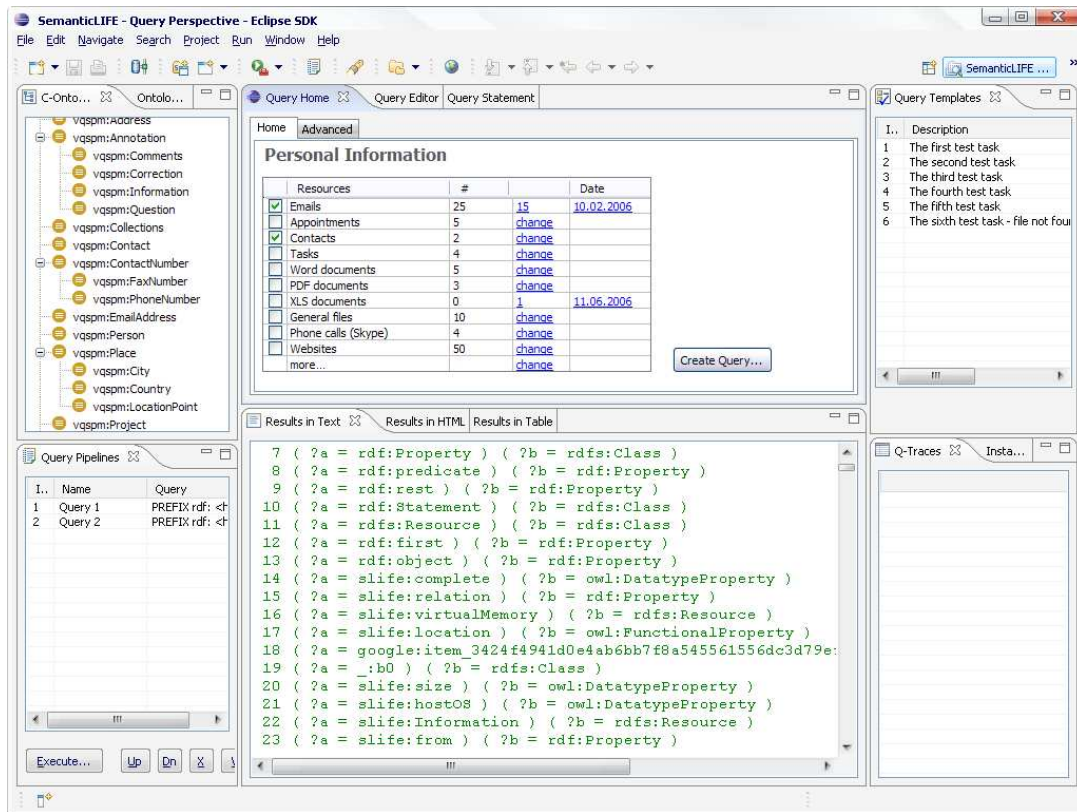


Figure 8.10: The Graphical User Interface of the Virtual Data Component

presents additional information such as time-based statistical information, last ten items stored; last ten queries executed.

Other modules of the VQS are also depicted in Figure 8.10. The VQL-QT(s) is listed in the right-upper view, and the traces are kept in the right-lower window. The middle-lower part is used for results presentation. Finally, the query pipelines are shown in the left-lower view of the main interface window.

8.5 The Context-based Querying Feature

8.5.1 VQL Query Templates

As the VQL-QTs are coded in XML that means any party could create and edit the VQL-QTs outside the system. This could lead to syntactical errors during the query template processing. Therefore, we have built a validity checking mechanism for VQL-QTs based-on the its XSD schema.

There are two cases of performing this check: firstly, when the VQS system starts, all VQL-QTs are checked for their validity. Secondly, this checking procedure will be called upon a VQL-QT is loaded for using.

8.5.2 The Context-based Querying

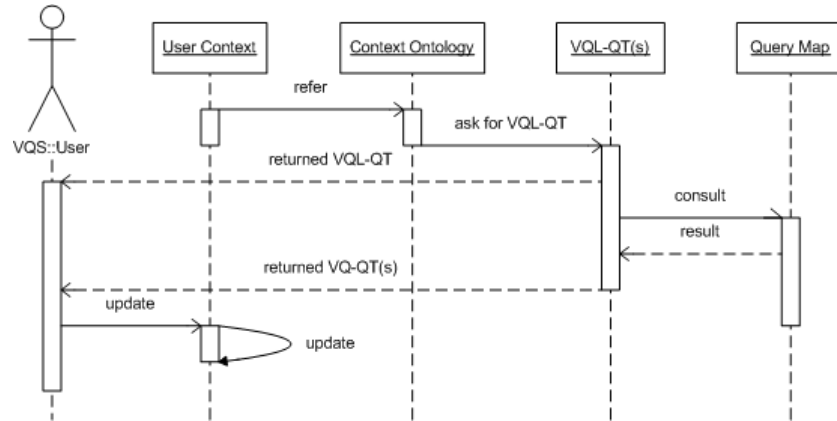


Figure 8.11: UML Sequence Diagram of the VQS Context-based Query

Figure 8.11 shows the sequence diagram of the VQS context-based querying feature. We make it clear that this feature would be initialized after the first query is successfully executed. Firstly, based on the VQS-UC the necessary resources for the next query are obtained with respect to the context ontology; VQS then retrieves the VQL-QT repository to get the appropriate VQL-QT upon the resources.

The VQL-QT is then returned to the user for editing. Continuously, a procedure of the VQS would consult to the current query map and recommend the relevant VQL-QT(s) to the user.

After execution the VQL query from the selected VQL-QT, the system will update the VQS-UC for the next round of the querying process. This process will reflect in the change of the VQL-QT view in the VQS GUI main window. The updated VQL-QT view contains only the VQL-QT(s) for the current VQS-UC. Nevertheless, there is an option to see all VQL-QTs at this time.

During the querying process, the user could save temporarily the queries in form of “traces” shown in the “Traces” window in the main GUI screen. In addition, the user can save the traces into permanent query pipelines for later use because the traces will be removed for the next start of the system. He/she could also add, change or delete the inappropriate VQL-QT(s) to improve his/her query effectiveness.

8.6 Application Scenarios

After having introduced the VQS approach for the ontology-enhanced querying of SemanticLIFE, we want to illustrate in this chapter by means of two application scenarios how our approach may be practically applied and what benefits it offers.

The first scenario is about the personalized project management (Section 8.6.1) and the second scenario addresses the domain of e-Government in term of personal use (Section 8.6.2). These two application scenarios have in common that they require an infrastructure providing means for user-supported querying. Moreover, as mentioned before, the ultimate goal of the SemanticLIFE framework is aiming at building a personal digital memory supporting the PIM systems. Therefore all the scenarios taken into consideration must reflect their personal usage perspective.

8.6.1 Personalized Project Management Scenario

Scenario Description

A scientist works for several projects at the same time with different roles. Everyday he/she has a lot of work to do such as reading and sending a lot of emails to projects' members, making many phone calls, processing digital documents, browsing the websites for his tasks, reading the paper/articles, and coding at sometimes. Generally, he/she often makes annotation on what he has done.

He/she wants the SemanticLIFE system to organize his/her large repository in a semantic way in order to retrieve effectively. Searching information in a "mountain" of information of working projects is a challenging task for him/her. He/she would like to have the support through the SemanticLIFE's query mechanism, the VQS, to simplify the task of information retrieval.

VQS-contextualization

In order to fit the scenario into the SemanticLIFE, especially the information retrieval task of the VQS, we can divide it into two main parts.

Firstly, the SemanticLIFE will deal with the issues related to storing data semantically with the user's annotation. A range of covered data could be email messages, contacts (persons), calendar for storing appointments, tasks for keeping projects' tasks and progresses, digital documents, call logs, IM logs, web

browsing sessions, external publication databases. All these data are kept into the SemanticLIFE metastore with an underline ontologies.

Secondly, the main focus of the scenario is about the information retrieval in the context of the SemanticLIFE's VQS. This task is divided into two subparts: first, the query formulation should be simple to the user so that he/she could have the supportive feeling during his/her information retrieval sessions. The second point is the capabilities of the VQS in returning the aggregated, exact results as his/her interest as well as associated information that may be helpful for the user. It means the VQL-QTs are oriented to the subject of scenario; and the query results must be based on the context ontology for suggesting the information as much relevant as possible.

The context ontology for this scenario is described in Figure 8.9. Additionally, the demonstration for this application use case is presented in following link: <http://www.ifs.tuwien.ac.at/~hhanh/vqs/scenario1/>.

8.6.2 The Personal E–Government services

Scenario Description

This scenario addresses the role of the VQS in dealing the user's problems during an individual's life relating e-government services. The personal matters of e-government services could be housing concerns, health-care/insurance services, children schooling, justice matters and so on. With these, the user must store a lot of his/her personal data during his/her life, and the range of data is quite diversified.

In this scenario, we do not discuss about the issue of storing with annotated information and ontological enrichment, we only mention the information retrieval aspect to support the user in seeking the information of interest effectively.

VQS-contextualization

In this scenario, the user has to deal with a lot of data from many government services. The range of this data could consist of digital documents (Word/PDF documents, digital forms, scanned documents in forms of images or PDF files, and medical records and so on), contact persons, appointments, emails, logged phone calls (if any), digital map, web browsing sessions or website URLs, etc. The SemanticLIFE could help the user in organizing this data and store them in its metastore with the semantic enrichment.

The most important issue relies on the VQS with the context ontology that can customize and orient all tasks to the user's needs. In order to customize the VQS for this scenario, the following matters must be prepared:

- The context ontology reflecting the personal e-gov services which mainly focuses on organizing the personal document library.
- Extracting the metadata from SemanticLIFE metastore into the context query is also the main concern.
- The set of VQL query templates for the proposed services.
- The presentations of VDC and the query results.

The context ontology and the demonstration for this application scenario is presented in following link:
<http://www.ifs.tuwien.ac.at/~hhhanh/vqs/scenario2/>.

8.7 Summary

The SemanticLIFE Framework is a collective work of a team, and it is still going on with the exploration of new research directions.

In this chapter, we have presented the implementation contribution of the query mechanism to the whole picture of SemanticLIFE. Details of used technologies, frameworks and the system design as well as two scenarios are described. Additionally, the Service Bus, which is backbone of the SemanticLIFE framework, is presented along the SOPA architecture. SOPA is an innovative feature used by the VQS to organize the collective query invocation, and keep the queried sessions for later use.

9 Conclusion

In this thesis we have introduced the SemanticLIFE framework as a personal information management system to organize the personal data of humans and discover the implicit knowledge with help of the Semantic Web technologies. In the aspect of information retrieval in the SemanticLIFE system, we have presented an user-oriented approach—*front-end* approach—as an effort to help SemanticLIFE’s users in formulating the unambiguous queries as well as to simply the way of creating complex queries. In this approach, an innovative method of making queries based-on the metadata of the system data and the context of the user and query space has been discussed. As the result, a query language—the Virtual Query Language—has been developed, and the most importance is that the Virtual Query System with many new features has been successfully implemented.

9.1 The SemanticLIFE Framework

Inspired by the ‘Memex’ vision of Vannevar Bush in his article ‘As May We Think’ [Bus45] and the emerging of the Semantic Web technologies, the SemanticLIFE project is aimed at realizing the *digital memories* for managing the personal lifetime information and events.

The ideas of Bush’s became very popular nowadays with many research and applications are geared to its realization. The distinguished vision of the SemanticLIFE project is the ideas about *capturing and digitalizing life-events, life-information of individuals by exploiting the metadata from personal data; annotating and associating the extracted metadata and semantic enriched data for users’ information retrieval*. The important issues of the SemanticLIFE PIM are: (1) how the personal data are to be discovered, organized and associated in answering the questions from the users; (2) to enable the users to early querying for information of interest when they use the system.

The development of the SemanticLIFE framework is a joint-work of all members of SemanticLIFE-Team. The SemanticLIFE framework implementation has not completely and ideally finished due to the evolutionary character of a basic research project. There are some components still need to be completed and improved. Nevertheless, the whole system is now functioning with the basic and important components.

Firstly, the data acquisition module is developed with help of Google Desktop tool as ‘middleware’ for upper level applications to access the data on computers with the powerful index mechanism and a wide range data cover and monitoring such as files, documents, personal data stored in Outlook. The data are collected and semantically enriched by RDFizing and annotating with ‘Annotation’ component before storing into the SemanticLIFE metastore.

Secondly, a new architecture of the system has been developed based-on Eclipse platform with new invention of the SOPA with Pipeline service and Service Bus—a replacement of former Message Handler. With this architecture, the components are registered into the system as plugins with their operations configured as service extensions. A set of plugins for information retrieval are known as *query* and *query UI* plugins are also developed with many new features such as their own query language, new way of formulating queries, query templates.

And finally, a presentation and visualization tool is also successfully developed. It allows users to customize their views of information and data personally and flexibly. The results are personally consulted with user profiles when rendering for presentation.

9.2 The Virtual Query System

In the main part of this thesis, we have focused on our query system—the Virtual Query System—for the SemanticLIFE framework. Starting from the idea of users’ awareness about their stored data which could help in asking for information, our front-approach offers many features based-on the well-organization of the collected metadata from the SemanticLIFE metastore. These metadata are organized as the *virtual information* and stored in an ontology—the *context ontology*.

We have also investigated the similar approaches in current ontology-based query systems which have been discussed in details in Chapter 4. The survey give us motivation for building another ontology-based query system and the research directions which help to add a new approach into in on-building query system of the SemanticLIFE. As presented through three chapter (Chapter 5 – Chapter 7), the VQS has three distinguished parts: the Context Ontology, so-called the

‘virtual information’, the Virtual Query Language (VQL) and its parser, and the innovative context-based querying feature.

The second contribution of VQS is its query language—the Virtual Query Language. This language is aiming at a light-weight query language instead of RDF query languages. It enables the users and SemanticLIFE’s components to deliberate from knowledge and binding to a specific RDF query language. VQL is a XML-based language, so that we can formulate queries by any means of editing or even create on the fly with the help of VQL’s interface. A schema-based validation has been built for well-formed checking. In addition, the VQL parser is successfully developed for syntactically and semantically mapping VQL queries to the RDF queries used in the metastore. Furthermore, in our user-oriented approach for VQL, we define VQL Operators that simplify the complex queries formulation from the user’s side. The operators are about getting the related information from the initial request, retrieve the metadata information of data sources, the semantic associations of requested information, and the documents encoded and uploaded into the metastore.

With VQS, the SemanticLIFE’s user can not only be supported in the query formulation at the initial phase of querying process, but also in getting help from the VQS during the process. Based-on the user context of querying and the VQS’s context ontology, the VQS analyzes the user’s query space—known as VQS Query Map—for the new knowledge, and trying to match with the pre-defined query patterns—known as VQL Query Templates—and then recommend the most appropriate query patterns to the user. We have considered this feature as an innovative approach to support users in complex query formulation.

As a part of the final product, an interactive query mechanism for VQS approach implementation and its programmatic interface based-on the SemanticLIFE’s architecture and SOPA have been developed and tested for some applications scenarios. The results are very much promising.

Last but not least, by outlining two attractive application scenarios, i.e. user-oriented querying mechanism for the personal project management, and an information retrieval environment for the personal use in manner of the e-Government service, we have highlighted the benefits of using the VQS infrastructure for an attractive information retrieval.

9.3 Future Work

In our future work, we will focus on the realization of the e-Learning and e-Government scenarios. Additionally, we plan to extend these two scenarios by integrating the features and functionality described in the third scenario, i.e. the

context-based query system. By integrating the underlying workflow ontology, we want to demonstrate the benefits of seamlessly integrating multiple ontology instances—possibly originating from different domains—within one VQS application.

Based on real-world data gathered from this use case, we will carry out further experiments for performance evaluation, in particular to achieve a more detailed analysis and understanding of the effects of the various factors on query performance. We will use the application-specific data as starting point for the development of a VQS query optimizer.

Furthermore, our future work will focus on the improvement of the VQL and its parser, as well as the development of an ontology for VQL queries and VQL query templates. These issues are to improve the preciseness during the process of matching the user’s context and query map’s knowledge.

And finally, we plan to take a focus on user modeling research for applying them into the VQS’s approach to improve the personal aspect of the SemanticLIFE’s aims. This would enable the VQS to work towards a more user-oriented future.

Bibliography

- [ACK04] Nikos Athanasis, Vassilis Christophides, and Dimitris Kotzinos. Generating on the fly queries for the semantic web: The ics-forth graphical rql interface (grql). In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of International Semantic Web Conference - LNCS 3298*, pages 486–501. Springer, November 2004.
- [AHK⁺04] Mansoor Ahmed, Hanh H Hoang, Shuaib Karim, Shah Khusro, Monika Lanzenberger, Khalid Latif, Elka Michlmayr, Khabib Mustofa, Tinh H Nguyen, Andreas Rauber, Alexander Schatten, Tho M Nguyen, and A Min Tjoa. Semanticlife - a framework for managing information of a human lifetime. In *Proceedings of the 6th International Conference on Information Integration and Web-based Applications and Services*, September 2004.
- [AJS⁺04] Eija Airio, Kalervo Jarvelin, Pirkko Saatsi, Jaana Kekalainen, and Sari Suomela. Ciri - an ontology-based query interface for text retrieval. In *Proceedings of 11th Finnish Artificial Intelligence*, Helsinki, 2004.
- [AKS99] Eytan Adar, David Karger, and Lynn Andrea Stein. Haystack: Per-user information environments. In *Proceedings of the 8th International Conference on Information and Knowledge Management.*, 1999.
- [And06] Amin Andjomshoaa. Service-oriented pipeline architecture. JAX Innovation Award Proposal, 2006. <http://jax-award.de/>.
- [AS03] Kemafor Anyanwu and Amit Sheth. Ρ-queries: enabling querying for semantic associations on the semantic web. In *Proceedings of the 12th international conference on World Wide Web*, pages 690–699, New York, NY, USA, 2003. ACM Press.
- [AvH04] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.
- [BCDD02] Dave Banks, Steve Cayzer, Ian Dickinson, and Reynolds Dave. The ePerson Snippet Manager: A Semantic Web Application. Technical report, HP Laboratories Bristol, 2002.

- [Bec05] Dave Beckett. Sparql query results xml format. Technical report, W3C Working Draft, August 2005.
- [BL98] Tim Berners-Lee. Semantic web road map. September 1998.
- [BL99] Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper Sans Francisco, October 1999.
- [BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284(5):34–43, May 2001.
- [BRA05] Davide Buscaldi, Paolo Rosso, and Emilio Sanchis Arnal. A wordnet-based query expansion method for geographical information retrieval. Technical report, CLEF Workshop, August 2005.
- [Bus45] Vannevar Bush. As we may think. *The Atlantic*, 176(1):101–108, July 1945.
- [CMF⁺04] Tiziana Catarci, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, and Sergio Tessaris. An ontology based visual tool for query formulation support. In Robert Meersman and Zahir Tari, editors, *Proceedings of OTM 2003 Workshops - LNCS 2889*, pages 32–33. Springer, 2004.
- [CMN04] Luigi Cinque, Alessio Malizia, and Roberto Navigli. Ontodoc: An ontology-based query system for digital libraries. In *Proceedings of the 17th International Conference on Pattern Recognition - Volume 2*, pages 671–674, Washington, DC, USA, 2004. IEEE Computer Society.
- [CSC04] Isabel F. Cruz, William Sunna, and Anjli Chaudhry. Ontology alignment for real-world applications. In *Proceedings of The National Conference on Digital Government - DG.O 2004*, 2004.
- [CTC02] Steve Cronen-Townsend and W. Bruce Croft. Quantifying query ambiguity. In *Proceedings of the Conference on Human Language Technology*, pages 94–98, 2002.
- [DEFS98] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *Proceedings of the IFIP TC2/WG 2.6 Eighth Working Conference on Database Semantics- Semantic Issues in Multimedia Systems*, pages 351–369, Deventer, The Netherlands, 1998. Kluwer, B.V.
- [DHNS03] Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. Towards the adaptive semantic web. In *Proceedings of the International*

- Workshop on Principles and Practice of Semantic Web Reasoning*, pages 51–68. LNCS-2901, Springer-Verlag, 2003.
- [DSe06] John Davies, Rudi Studer, and Paul Warren (eds.). *Semantic Web Technologies – Trends and Research in Ontology-based Systems*. Wiley, 2006.
- [FG96] Eric Freeman and David Gelernter. Lifestreams: A storage model for personal data. In *ACM SIGMOD Record, Bulletin 25,1*, pages 80–86, March 1996.
- [FHH03] Richard Fikes, Pat Hayes, and Ian Horrocks. Owl-ql: A language for deductive query answering on the semantic web. Technical Report KSL 03-14, Stanford University, Stanford, CA, 2003.
- [FHLW03] Dieter Fensel, James Hendler, Henry Lieberman, and Wolfgang Wahlster. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2003.
- [FR03] Andrew Fitzgibbon and Ehud Reiter. “memories for life” - managing information over a human lifetime. Technical report, Grand Challenges in Computing Workshop, UK Computing Research Committee, May 2003.
- [GBL⁺02a] Jim Gemmel, Gordon Bell, Roger Lueder, Steven Drucker, and Curtis Wong. Mylifebits: Fulfilling the memex vision. In *ACM Multimedia '02*, pages 235–238, December 2002.
- [GBL⁺02b] Jim Gemmell, Gordon Bell, Roger Lueder, Steven Drucker, and Curtis Wong. Mylifebits: Fulfilling the memex vision. In *ACM Multimedia*, December 2002.
- [GBL03a] Jim Gemmel, Gordon Bell, and Roger Lueder. Mylifebits: Living with a lifetime store. In *ATR Workshop on Ubiquitous Experience Media*, September 2003.
- [GBL03b] Jim Gemmell, Gordon Bell, and Roger Lueder. The mylifebits lifetime store. In *ACM SIGMM Workshop on Experiential Telepresence 2003*, November 2003.
- [GC03] Vladimir Geroimenko and Chaomei Chen. *Visualizing the Semantic Web. XML-based Internet and Information Visualization*. Springer, 2003.
- [GHP01] Peter Gray, Kit Hui, and Alun Preece. An expressive constraint language for semantic web applications. in preece,. In *Proceedings IJCAI-01 Workshop on E-Business and the Intelligent Web*, pages 46–53, 2001.

- [GLB06] Jim Gemmell, Roger Lueder, and Gordon Bell. Mylifebits: Personal database for everything. *Communication of The ACM*, 49(1):88–95, January 2006.
- [GM03] R. Guha and Rob McCool. Tap: a semantic web platform. *International Journal on Computer and Telecommunications Networking*, 42(5):557–577, August 2003.
- [GMM03] R. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the 12th international conference on World Wide Web*, pages 700–709, New York, NY, USA, 2003. ACM Press.
- [Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
- [GS03] Elena Garcia and Miguel-Angel Sicilia. User interface tactics in ontology-based information seeking. *Psychology Journal*, 1(3):242–255, 2003.
- [HAT06a] Hanh Huu Hoang, Amin Andjomshoaa, and A Min Tjoa. Towards a new approach for information retrieval in the semanticlife digital memory framework. In *Proceedings of the 6th IEEE/WIC/ACM International Conference on Web Intelligence*, Hong Kong, December 2006. In press.
- [HAT06b] Hanh Huu Hoang, Amin Andjomshoaa, and A Min Tjoa. Vqs: An ontology-based query system for the semanticlife digital memory project. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *Proceedings of the 2th IFIF WG 2.14 & 4.12 International Workshop on Web Semantics, LNCS 4278*, pages 1796–1805, Montpellier, France, November 2006. Springer.
- [HH00] Jeff Heflin and James Hendler. Searching the web with shoe. In *AAAI Workshop*, pages 35–40. AAAI Press, 2000.
- [HKQ02] David Huynh, David Karger, and Dennis Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Proceedings of International Workshop on the Semantic Web*, 7 2002.
- [HNAT06] Hanh Huu Hoang, Tho Manh Nguyen, Amin Andjomshoaa, and A Min Tjoa. A front-end approach for user query generation and information retrieval in the semanticlife framework. In *Proceedings of the 8th International Conference on Information Integration and Web-based Applications and Services*, Yogyakarta-Indonesia, December 2006. OCG Book Series. In press.
- [HSR⁺04] Sebastian Hübner, Rainer Spittel, Rainer, Ubbo Visser, and Thomas J. Vögele. Ontology-based search for interactive digital maps. *IEEE Intelligent Systems*, 19(3):80–86, 2004.

- [HSV03] Eero Hyvönen, Sampsa Saarela, and Kim Viljanen. Ontogator: Combining view- and ontology-based search with semantic browsing. In *Proceedings of XML-Finland Conference: Open standards, XML and the Public Sector*, October 2003.
- [HT06a] Hanh Huu Hoang and A Min Tjoa. The state of the art of ontology-based query systems: A comparison of current approaches. In *Proceedings of the International Conference on Computing and Informatics*, June 2006.
- [HT06b] Hanh Huu Hoang and A Min Tjoa. The virtual query language for information retrieval in the semanticlife framework. In *Proceedings of the International Workshop on Web Information Systems Modeling - CAiSE 06*, pages 1062–1076, June 2006.
- [HTN06] Hanh Huu Hoang, A Min Tjoa, and Tho Manh Nguyen. Ontology-based virtual query system for the semanticlife digital memory project. In *Proceedings of the 4th International Conference on Computer Sciences*, February 2006.
- [iPr06] iProspect. Search engine user behavior study. iProspect White Papers, April 2006.
- [JSS00] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing Management*, 36(2):207–227, 2000.
- [KAC⁺02] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. Rql - a declarative query language for rdf. In *Proceedings of the Eleventh International World Wide Web Conference*, pages 592–603. ACM Press, August 2002.
- [KBH⁺05] David R. Karger, Karun Bakshi, David Huynh, Dennis Quan, and Vineet Sinha. Haystack: A general purpose information management tool for end users of semistructured data. In *Proceedings of 2nd Biennial Conference Innovative Data Systems Research*, pages 13–26, January 2005.
- [KCD⁺04] L. Kerschberg, M. Chowdhury, A. Damiano, H. Jeong, S. Mitchell, J. Si, and S. Smith. Knowledge sifter: ontology-driven search over heterogeneous databases. In *Proceedings of 16th International Conference on Scientific and Statistical Database Management*, June 2004.
- [McB02] Brian McBride. Four steps towards the widespread adoption of a semantic web. In *Proceedings of the 1st International Semantic Web Conference*, 2002.
- [MHS05] Eetu Mäkelä, Eero Hyvönen, and Teemu Sidoroff. View-based user interfaces for information retrieval on the semantic web. In *Pro-*

- ceedings of End User Semantic Web Interaction Workshop*, Alway, Ireland, 2005. CEUR-WS.
- [MHSV04] Eetu Mäkelä, Eero Hyvönen, Samppa Saarela, and Kim Viljanen. Ontoviews - a tool for creating semantic web portals. In *Proceedings of the 3rd International Semantic Web Conference*, Hiroshima, Japan, 2004. Springer.
- [MM00] Dan I. Moldovan and Rada Mihalcea. Using wordnet and lexical operators to improve internet searches. *IEEE Internet Computing*, 4(1):34–43, 2000.
- [MMSV02] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra - an ontology mapping framework in the semantic web. In *Proceedings of the 12th International Workshop on Knowledge Transformation*, July 2002.
- [MSS⁺01] Alexander Maedche, Steffen Staab, Nenad Stojanovic, Rudi Studer, and York Sure. Seal - a framework for developing semantic web portals. In *Proceedings of the 18th British National Conference on Databases*, pages 1–22, London, UK, 2001. Springer-Verlag.
- [MTT98] Rila Mandala, Takenobu Tokunaga, and Hozumi Tanaka. The use of WordNet in information retrieval. In Sanda Harabagiu, editor, *Proceedings of the Conference on Use of WordNet in Natural Language Processing Systems*, pages 31–37, Somerset, New Jersey, 1998. Association for Computational Linguistics.
- [Nic96] J. M. Nicolau. On thoughts about the brain. *Brain Processes, Theories and Models*, pages 71–77, 1996.
- [Par04] David Parry. A fuzzy ontology for medical document retrieval. In *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 121–126, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.
- [Pas05] Thomas B. Passin. *Explorer's Guide to the Semantic Web*. Manning, 2005.
- [PS05] Eric Prud'hommeaux and Andreas Seaborne. Sparql query language for rdf. W3C Working Draft, November 2005.
- [QHK03] Dennis Quan, David Huynh, and David R. Karger. Haystack: A platform for authoring end user semantic web applications. In *Proceedings of the 12th International World Wide Web Conference*, pages 738–753, May 2003.
- [RSA04] Cristiano Rocha, Daniel Schwabe, and Marcus Poggi Aragao. A hybrid approach for searching in the semantic web. In *WWW '04*:

- Proceedings of the 13th international conference on World Wide Web*, pages 374–383, New York, NY, USA, 2004. ACM Press.
- [RSC04] Dave Reynolds, Paul Shabajee, and Steve Cayzer. Semantic information portals. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 290–291, New York, NY, USA, 2004. ACM Press.
- [SDA04] Shailendra Singh, Lipika Dey, and Muhammad Abulaish. A framework for extending fuzzy description logic to ontology based document processing. In *Proceedings of 2nd International Atlantic Web Intelligence Conference - LNAI 3034*, pages 95–104, Berlin - Heidelberg, May 2004. Springer.
- [Sea04] Andy Seaborne. Rdfql - a query language for rdf. Member submission, W3C, 2004.
- [SGS03] Nenad Stojanovic, Jorge Gonzalez, and Ljiljana Stojanovic. Ontologer: a system for usage-driven management of ontology-based information portals. In *K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture*, pages 172–179, New York, NY, USA, 2003. ACM Press.
- [SS04] Steffen Staab and Rudi Studer. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.
- [SSS04] N. Stojanovic, R. Studer, and L. Stojanovic. An approach for step-by-step query refinement in the ontology-based information retrieval. In *Procceedings of the IEEE International Conference on Web Intelligence (WI'04)*, pages 36–43, 2004.
- [Sto03a] Nenad Stojanovic. Information-need driven query refinement. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence*, pages 388–395, Washington, DC, USA, 2003. IEEE Computer Society.
- [Sto03b] Nenad Stojanovic. On the role of a librarian agent in ontology-based knowledge management systems. *Journal of Universal Computer Science*, 9(7):697–718, July 2003.
- [Stu04] H. Stuckenschmidt. Similarity-based query caching. In *Proceedings of the 6th International Confonference on Flexible Query Answering Systems*, 2004.
- [TAAK04] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 415–422, New York, NY, USA, 2004. ACM Press.

-
- [WGA05] David Wood, Paul Gearon, and Tom Adams. Kowari: A platform for semantic web storage and analysis. In *Proceedings of the 14th International WWW Conference*, 2005.
- [WR05] Dennis Wollersheim and J. Wenny Rahayu. Ontology based query expansion framework for use in medical information systems. *International Journal of Web Information Systems*, 1(2):101–115, March 2005.
- [ZYZ⁺05] Lei Zhang, Yong Yu, Jian Zhou, ChenXi Lin, and Yin Yang. An enhanced model for searching in semantic portals. In *Proceedings of the 14th international conference on World Wide Web*, pages 453–462, New York, NY, USA, 2005. ACM Press.

Curriculum Vitae

Full Name: HOANG HUU HANH
Address in Austria: Wiesberggasse 9/33
A-1160 Vienna, Austria
Address in Vietnam: 15/2 Nguyen Su Street
Hue City, Vietnam
Born: 13 April 1974
in Hue City, Vietnam.
Nationality: Vietnamese.
Marital Status: Married.

Education

Since 2003 PhD Studies of Computer Science at
Vienna University of Technology, Austria.
1999 – 2001 Master Studies of Computer Science at
Hanoi University of Technology, Vietnam.
Thesis: Performance Evaluation of TCP/IP Computer Network.
1992 – 1996 Bachelor Studies of Mathematics & Informatics at
Hue Pedagogy University, Vietnam.
Thesis: Vietnamese Text Data Compression.
1989 – 1992 Gia-Hoi High School, Hue City, Vietnam.
1980 – 1989 Nguyen-Du Elementary and Secondary School,
Hue City, Vietnam.

Working Experience

Since 2003: Research Assistant at Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria.
Research areas: Software Engineering, Semantic Web, Ontologies, Knowledge Management.

1996 – 2003: Lecturer at Hue University of Sciences, Vietnam.

Teaching areas: DataBases and DBMS, Software Engineering, Computer Networks, Web Development, Programming Languages.

February–March 2003: Visiting Lecturer at Hue-Aptech Computer Education (Quang-Tri branch), Vietnam.

Teaching subjects: DBMS, Web Development, Computer Networks.

1997 – 2002: Being involved in a number of projects:

- National Project (as Project Assistant): Employee Management System in the “Building the Management Software System at Hue University of Sciences” project, 2003. (Grant No. NN03.09).
- Ministerial Project (as Project Assistant): “Computerizing the University Management at Hue University of Sciences”, 1996. (Grant No. B96.TD.01).
- University Projects (as Project Manager):
 - Vietnamese Data Compression, 1997.
 - Building the Inter-Network at Hue University of Sciences, 1999.
 - A Research Survey on XML Technology, 2001.
- External Project (as Project Manager): “Building the Internal Network for Information Exchange at Department of Education – Thua-Thien-Hue Province”, 2000.

Publications

Invited Talk

Information Retrieval in the SemanticLIFE Personal Digital Memory Framework (Hoang Huu Hanh, Nguyen Manh Tho), invited talk at CISE 2006, The 18th International Conference on Computer and Information Science and Engineering, Vienna, Austria, December 16-18, 2006.

Conference and Workshop Papers

1. **A Front-End Approach for User Query Generation and Information Retrieval in the SemanticLIFE Framework** (Hoang Huu Hanh, Nguyen Manh Tho, Amin Andjomshoaa, A Min Tjoa), full paper accepted at iiWAS 2006, The 8th International Conference on Information Integration and Web-based Applications and Services, Yogyakarta, Indonesia, December, 2006.

2. **Towards a New Approach for Information Retrieval in the SemanticLIFE Digital Memory Framework** (Hoang Huu Hanh, Amin Andjomshoaa, A Min Tjoa), short paper accepted at WI 2006, The 6th IEEE/WIC/ACM International Conference on Web Intelligence, Hong Kong, December, 2006.
3. **VQS - An Ontology-based Query System for the SemanticLIFE Digital Memory Project** (Hoang Huu Hanh, Amin Andjomshoaa, A Min Tjoa), full paper accepted at SWWS 2006, The 2nd IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics, LNCS 4278, pp. 1796-1805, Montpellier, France, November 1-3, 2006.
4. **The Virtual Query Language for Information Retrieval in the SemanticLIFE Framework** (Hoang Huu Hanh, A Min Tjoa), full paper accepted at WISM 2006, The International Workshop on Web Information Systems Modeling, pp. 1065-1076, Luxembourg, June 5-7, 2006.
5. **The State of the Art of Ontology-based Query Systems: A Comparison of Existing Approaches** (Hoang Huu Hanh, A Min Tjoa), full paper accepted at ICOCI 2006, The IEEE International Conference on Computing and Informatics, Kuala Lumpur, Malaysia, June 6-8, 2006.
6. **Ontology-based Virtual Query Systems for the SemanticLIFE Digital Memory Project** (Hoang Huu Hanh, A Min Tjoa, Nguyen Manh Tho), short paper accepted at RIVF 2006, The 4th IEEE International Conference on Computer Sciences, HoChiMinh City, Vietnam, February 12-16, 2006.
7. **'SemanticLIFE' - A Framework for Managing Information of A Human Lifetime** (Ahmed M., Hoang Huu Hanh, Karim M.S., Khusro S., Lanzenberger M., Latif K., Michlmayr E., Mustofa K., Nguyen H.T., Rauber A., Schatten A., Tho M.N. and Tjoa A. M.), full paper accepted at iiWAS 2004, The 6th International Conference on Information Integration and Web-based Applications and Services, Jakarta, Indonesia, September 27-29, 2004.

Books

1. **Programming Language C** (Le Manh Thanh, Hoang Huu Hanh, Truong Cong Tuan), Education Publishing House, Vietnam, 1998. (Vietnamese)
2. **Building Websites with FrontPage** (Le Manh Thanh, Hoang Huu Hanh), Education Publishing House, Vietnam, 2000. (Vietnamese)