SERSE

An Agent-based System for Scalable Search on the Semantic Web

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree
of Doctor in Philosophy

By

Ian Blacoe

Department of Computer Science

March 2009s

ORIGINAL COPY TIGHTLY BOUND

Contents

To Loredana and my family.

List of Figures

Abstract

The Semantic Web is envisioned as an open, distributed environment in which heterogeneous, distributed and dynamic knowledge can be shared and utilised by software. This software includes knowledge-based applications and autonomous software agents that act proactively to achieve their goals. The Semantic Web is intended to be an extension of the current World Wide Web, in which the existing human-understandable information is supplemented by machine-readable metadata – to enable this information to be manipulated by software. In this environment knowledge is represented in a machine-processable way by use of ontologies and their extensions, defining the vocabulary and specific entities that can be used in metadata descriptions. This knowledge is then linked to relevant resources through a process of annotation, which is usually performed in a (semi-) automatic manner – attaching suitable metadata representations to existing information resources. These metadata representations can then be utilised by software to gather and aggregate knowledge about available resources, and to perform further processing and reasoning over this knowledge.

One of the key functions of this knowledge representation and metadata annotation is to enable access to information based upon the *meaning* of the information and of the search terms, rather than simply on the presence of keywords within texts. That is, the retrieval of information resources is based upon the semantics of their metadata annotations, as defined in the underlying ontologies. In this thesis we propose the use of Semantic Web technologies for the provision of semantics-based search for information resources, and within this context we describe two main research threads – scalable and robust indexing of semantically annotated resources; and the application of semantic search techniques within real commercial environments.

The first research thread is predicated upon the characteristics of the Semantic Web, its potential size, and the expected quantity of metadata that may be generated. The amount of information in digital form that is available for semantic annotation, and the emphasis on openess and distribution in the Semantic Web, means that there is a requirement for scalable and robust mechanisms supporting the search for

and location of these annotated resources. As indicated in the original vision, the Sémantic Web environment is seen as being well suited to autonomous agents.

In this thesis, we present the design, implementation and evaluation of SERSE – SEmantic Routing SýstEm – a distributed, multiagent system, composed of task-specialised agents, that provides robust and efficient indexing and aggregation of annotated information resources from diverse and heterogeneous sources. The agents composing SERSE apply ontological knowledge to search for and retrieve semantically annotated resources, by maintaining a distributed *semantic index* that stores the annotation metadata, and a *semantic overlay network* that inter-connects the distributed index elements. Efficient retrieval of annotated resources is then made possible through the *semantic routing* mechanism, which enables identification and location of the agents that index the resources requested by a user query, without relying on a central index or on message broadcasts. Furthermore, the semantic similarity algorithm[1] that underlies the semantic routing is able to generate links between entities from different, heterogeneous ontologies, thus enabling SERSE to support information aggregation and semantic browsing of retrieved information. The multiagent system permits the distribution of the metadata indexes, and it is this distribution, along with a number of basic autonomic system features, that underlies the scalability and fault-tolerance of the system.

In the second research thread we examine the application of semantic annotation and search techniques within realistic commercial contexts, in order to examine the benefits offered by semantics-based searching over information sources when applied to specific business problems. We examine the use of ontologies and metadata to represent specific and restricted business-knowledge domains, with the aim of enabling an enhanced, semantics-based search facility over the company information that can be annotated with this metadata. The QuestSemantics system is presented as a proof-of-concept solution to identification and retrieval of commercially useful information based upon a relatively lightweight knowledge model – by focussing upon carefully delimited domains of knowledge within the two different commercial test-cases undertaken. In this system the knowledge representations are utilised in two key processes: firstly, the annotation component analyses the selected information resources and then automatically generates suitable metadata annotations to describe them; secondly, the stored metadata is made searchable through a specialised interface, which provides access to the resources based on the encoded business knowledge. QuestSemantics is intended to demonstrate the effectiveness of semantics-based resource annotation and retrieval to address knowledge-based business tasks.

[1] Based upon ontology alignment techniques.

Acknowledgements

I would like to thank my supervisors, Dr. Valentina Tamma and Prof. Mike Wooldridge for their instruction and guidance in the research areas of Multi-Agent Systems and the Semantic Web, and for their support throughout my PhD research. This work has been made possible thanks to the financial support of the Esperonto (IST-2001-34373) project, and the University of Liverpool Academic Enterprise Fund. I would like to thank all the people involved in these projects that have contributed towards this work.

I would also like to thank all my friends and colleagues at the University of Liverpool. I especially wish to thank all the past and present members of the Semantic Web Lab at Liverpool: Loredana Laera, Luigi Iannone, Ignazio Palmisano, Paul Doran and Ben Lithgow Smith. I thoroughly enjoyed my time at Liverpool University, professionally and socially, and it was the people around me that made this possible.

I want to thank both my family and Lori's family – my parents Peter and Diane and Lori's parents Angelo and Giovanna; sisters and brothers Helen and Chris, Gennaro and Kiwa, Paola and Renzo; my nieces Amelie, Elena and Greta, and my nephew Logan. I am very grateful to them for the love and support they have all given to me.

Finally, but most importantly, I would like to thank Lori – for your love, support and patience throughout the last five years. Without your help and encouragement this thesis would never have been completed. Ti amo tanto stupenda mia.

Part I

Background and Context

Chapter 1

Introduction

"Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information on it."

- Samuel Johnson (1709 - 1784)

The primary goal of this thesis is to examine the application of semantic web technologies in order to provide enhanced search facilities over semantically annotated information resources. As the main thread of the thesis, we present an agent-based approach to managing a distributed knowledge-base of annotation metadata, in order to support a robust and scalable indexing system for the search and location of semantic web resources. Development of such distributed, scalable and flexible approaches to the process of resource location is seen as an important goal for the Semantic Web, to leverage the encoded knowledge in order to better answer queries over annotated resources. As a secondary thread we present further developments in the application of Semantic Web technologies to information searches, and evaluate these developments in the context of two commercial test-cases.

In this chapter we provide an overview of our work. We begin with an extended outline of the background and motivation for this thesis in section 1.1, and continue with a description of the main assumptions with regard to our scenario in section 1.2. We summarize the main goals and contributions of this thesis in section 1.3, and in section 1.4 we present a guide to the structure of this thesis.

1.1 Background and Motivation

Information on the World Wide Web (WWW) is primarily intended to be read and processed by humans, and is not information that can be readily manipulated by computers. The intelligence applied in search tasks is supplied by the user, with limited support from software [154]. The WWW is based on the use of visual markup languages (such as HTML) that are intended to facilitate the presentation of information

for human users. Current keyword based search engines present limitations, in that they cannot fully capture the richness intrinsic in natural language; as indicated by the obvious problems that synonymy and polysemy raise for the search task. Enhancing search engines with lexicons such as WordNet [108] can help to relieve this problem, but this is not sufficient to identify and resolve more complicated types of ambiguity. Furthermore, keyword-based search engines make little provision for the formulation of very precise queries, particularly those that make use of relationships between entities.

The Semantic Web (SW) is an evolution of the WWW in which knowledge is also expressed in a machine understandable way. The aim is to extend the web to become a container of interconnected, machine-processable information and knowledge, rather than just a collection of documents. The Semantic Web primarily aims to share knowledge from distributed, dynamic, and heterogeneous sources, whose content is expressed in a machine-readable format by means of languages such as RDF [36] and OWL [106], in a similar way to that in which information is shared on the WWW. The SW promises to add value to the web, without requiring any fundamental changes to the infrastructure that is currently in place. The added value is created by enriching information resources with annotations that reference *ontologies* – explicit and sharable, machine-readable representations of the conceptualisation abstracting a phenomenon [142].

Ontologies [142] are crucial in providing *meaning* to web resources. An ontology models the entities that are used to describe the content of a web resource, and, most importantly, the logical relations existing between these entities. Using such a model, an explicit representation can be created of the information contained in relevant web documents through *annotation.* Annotation is normally achieved by using or creating metadata items (as instances of concepts from the ontology) to represent specific entities recognised in the resources, and then linking this metadata to the resource as its description.

Semantically annotated resources can, therefore, be located on the basis of their metadata annotations, leveraging the knowledge encoded in the ontologies (used for the annotation) to express precise queries for resources matching complex, knowledge-based constraints. Such semantics-based retrieval of resources requires a robust system for indexing and search for the resources and their metadata, which supports the processing of metadata queries to efficiently retrieve the required resources. In order to handle the vast scale of resource and metadata information potentially available on the Semantic Web, any such system must be inherently scalable, decentralised, and amenable to distribution over the WWW. Furthermore, the SW has the potential to make large volumes of knowledge more readily available, through the combination of different information items from diverse sources, thus making the provision of scalable, decentralised indexing and search mechanisms even more important to its success [151]. In

recent years, many tools have been developed for managing traditional knowledge sources, but such approaches usually imply a static environment in which control is centralised. This type of approach does not promise to scale well to the Semantic Web, which is an open, dynamic, and often chaotic environment. Searching for information on the SW is arguably more complex than searching the current web – not only must a SW search system deal with a large number of distributed, heterogeneous resources, but the metadata annotating these resources may reference many different ontologies. Distributed, decentralised systems are thought to be a better alternative for scalability [113] than traditional, centralised systems. Their architecture is characterised by system components each with equal roles and the capability to exchange knowledge and services directly with each other. Peer-to-peer technology (P2P) such as Edutella [113] or Morpheus[1] is a possible answer to this quest for decentralisation. P2P systems are networks of peers with equal roles and capabilities, and recently peer-based management systems have been proposed, which exploit P2P technology for sharing and retrieving huge amounts of data [70]. However, most P2P approaches are oriented towards file sharing – utilising only lightweight categorisations and/or ad-hoc keyword tagging as file annotations, rather than at the management of semantically enriched content for a wide range of resources utilising complex, shareable knowledge models for resource annotation as provided by the Semantic Web. However, it would be possible for the information indexed within existing P2P file-sharing systems to be 'upgraded' to Semantic Web standards by provision of mappings between existing categorisations and suitable ontologies.

Autonomous software agents [170] are central to the vision of the Semantic Web described in [151], and the agent paradigm seems to offer equally good prospects for the management of semantically annotated content. This vision is based on the provision of sustainable support to distributed and decentralised knowledge processing and management, with agents able to provide *persistent* and *reliable* knowledge-based services. Agents can use the machine-readable representations to gather and aggregate knowledge, as well as to reason in order to manage inconsistencies, and to infer new facts. Agents are able to process the knowledge expressed in these semantic markup languages, and can thus offer services which make use of this knowledge, including search and retrieval services. Agents are intended to operate as individual, autonomous entities, exhibiting goal-directed behaviour that permits them to perform complex tasks without user intervention. When such agents are engineered as a multiagent system, they are able to communicate in order to collaborate over specific tasks, and thus provide self-organising, distributed processing for the task at hand. There exist a number of widely used platforms for agent-oriented programming, which offer standardised communication protocols and management mechanisms, and have well-automated discovery mechanisms for advertising and locating resources within an open framework (for instance, Jade [9]). There are a number of other features of multiagent systems

[1] http://www.morpheus.com

that make them particularly useful in the provision of service-based support for semantic web tools and applications, such as pre-existing work on automated discovery mechanisms for locating services within an open framework, established trust and reputation frameworks, and expressive communication languages and protocols [148]. Furthermore, multiagent systems can be constructed to exhibit features of autonomic computing, enabling the creation of robust, fault-tolerant and self-managing systems to provide services to users and other software systems [86]. Autonomic computing is an emerging branch of software engineering, promoting the design and implementation of self-managing systems, in which independent system components provide monitoring and management services to other components, in addition to their functional tasks, in order to achieve system-level self-monitoring and self-mangement. Many autonomic systems are constructed as modular systems, consisting of several interacting, autonomous components that, in turn, comprise large numbers of interacting, autonomous, self-governing components at the next level down [86]. This type of behaviour is intended to make it easier to manage the complexity and scalability of complex distributed systems. The combination of these features make multiagent systems particularly well suited to manage large, heterogenous, and distributed knowledge bases – as might be found on the Semantic Web.

1.2 Application Scenarios

The primary application scenario is intended to enable an examination of the the issues involved in the provision of a scalable and robust resource indexing and search system for the Semantic Web. This research has sought to manage the complexity inherent in such a system through the integration of different technologies such as ontologies and metadata, multi-agent systems, and peer-to-peer approaches. This integration resulted in the design of SERSE (SEmantic Routing SystEm), a multi-agent system providing distributed indexing of and semantic search over Semantic Web resources. In SERSE, agents are organised in a multiagent system according to a *semantic overlay*, where the interconnections between agents are determined by the semantic proximity of the ontological definitions known to the agents. SERSE provides *robust* and *reliable* indexing, retrieval and aggregation of semantically annotated knowledge sources. SERSE is composed of specialised agents that collectively maintain a distributed index of annotated resources, and autonomously discover and maintain links between themselves, based on the semantics of the information they index.

The SERSE system was originally built as a component of the Esperonto project (IST-2001-34373). It is intended to function as a scalable[2], open and dynamic index of semantic web resources that can be queried for semantically annotated resources using semantically specified queries. The system is built as a distributed collection of indexes that store pointers to web resources that are characterised by instances

[2]A measure of how well a hardware or software system can adapt to increased demands.

of concepts. The aim of Esperonto was to provide a set of tools for performing the transition from the traditional web to the semantic web [147]. In addition to SERSE (from the University of Liverpool), the other partners of the Esperonto project provided a number a components performing fundamental Semantic Web tasks, as depicted in Figure 1.1:

- Ontology server - providing facilities for the creation, editing and storage of ontologies.
- Annotation system - to support the semi-automatic annotation of resources with metadata.
- Multilingual services - to support the use of multiple human languages in ontologies and metadata.

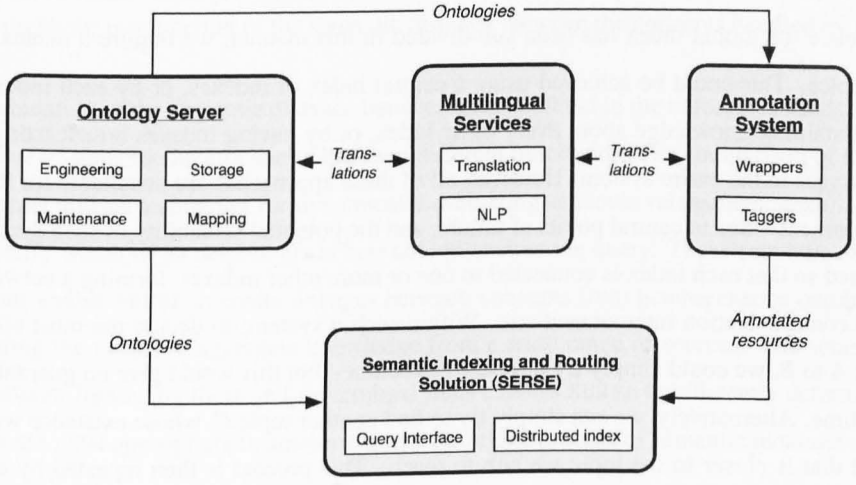


Figure 1.1: Esperonto system components.

The part of the project in which SERSE was developed was concerned with exploring the creation of an infrastructure of semantic indexes and semantic routers for the SW, following a decentralised peer-to-peer approach. The goal was to enable the automatic aggregation of distributed, semantically annotated web resources regarding the same domain by means of domain specific semantic indexes. These semantic indexes were to be used to route user queries, improving the results of current key-word based search engines and ontology-based search engines. The aggregation of SW content in semantic indexes is important to improve scalability and to ensure that those applications on top of the SW can aggregate content in order to provide value-added services. The semantic indexes implemented in this system are generated dynamically using both ontological information and annotated documents. Scalability is also addressed by coordinating semantic indexes, which behave as active agents whose knowledge includes the topics they can handle. In order to operate within an open and dynamic environment such as the SW the indexes need to exist in a distributed, decentralised and scalable architecture. Peer-to-peer systems exhibit these qualities and are, therefore, seen as useful paradigm on which to base the index infrastructure.

In examining the required features for the intended distributed semantic index, we identified a number of desiderata for searching the Semantic Web. The design choices we made in SERSE are motivated by some considerations and requirements for a (multi-agent) system that is able to efficiently navigate and search the Semantic Web. Searching the SW is arguably more complex than searching the current web, due to the inherent complexity in the manipulation of 'knowledge level' metadata. We believe that one way to overcome this increased complexity is to take the view of the SW as being composed of elements of 'fragmented knowledge': each fragment represents a specific topic, that is described by a concept or a group of similar concepts[3]. This enables a very large and complex knowledge index to be managed as a set of simple, independent indexes that each describe a specific subset of the knowledge. However, once the global index has been sub-divided in this manner, we require a means to navigate between topics. This could be achieved using a central index of indexes, or by each index within the system maintaining knowledge about every other index, or by having indexes broadcasting communication messages to the entire system. However, all of these approaches are unsuitable for the Semantic Web environment – due to central points of failure, and the potential scalability of such systems. SERSE was designed so that each index is connected to one or more other indexes, forming a network of index nodes and communication interconnections. Within such a system, to decide the most efficient route from topic A to B, we could simply try a random direction – but this would give no guarantee to find B in a finite time. Alternatively, we can simply try to find another topic C, whose existence we are certain about, and that is *closer* to the topic we aim to reach. This process is then repeated by each successive recipient of the message, breaking down the communication between two nodes that unaware of each other into a succession of comminations between linked nodes that successively move the message closer to its intended target. By using this approach we are sure to reach the right fragment of knowledge in a limited time (the sum of the times needed to reach each topic between A and B) – providing a suitable mechanism to avoid message cycles has been implemented, as within SERSE.

Based upon the various required features, the general architecture of SERSE is as follows:

- Annotated resources are indexed on the basis of ontological concepts, so resources will be indexed together with those characterised by the same instance and by other instances of the same concept. They are dynamically constructed using the annotations contained in the web resources and the definitions in ontologies of the annotation terms used. The indexes are handled by autonomous agents that 'know' what topics they can handle (i.e., find content for) based upon their own ontologies.

[3]Such similar concepts are likely to be defined within heterogeneous ontologies that describe overlapping domains of knowledge.

- The agents are linked together into *semantic neighbourhoods* on the basis of the ontological concepts they handle, i.e., concepts that are significantly related are explicitly connected to each other, thus forming a network of overlapping groupings of closely related concepts. This enables queries to follow semantic linkages between concepts, and to aggregate resources annotated by related concepts.

- Semantic queries for resources are formulated in the search interface and then sent into the network of interconnected agents. Any query received by an agent for a concept that is not handled by that agent is then semantically routed to a 'neighbouring' agent. In this context the closeness of a neighbour is a function of the semantic distance between the concepts handled by the agents.

- Determination of the semantic distance between terms defined in the ontologies needs to consider the issue of semantic heterogeneity between different ontologies. The comparison of these ontology terms utilises techniques for dynamically evaluating semantic relatedness, enabling an agent to identify which of its neighbours is best able to answer the query. These semantics-based connections enable SERSE to create linkages between concepts from heterogeneous ontologies, thus providing the means to aggregate knowledge from a wide range of sources. The semantic overlay network formed by these self-determined links enables SERSE to efficiently determine which agents are most appropriate to answer a query through a process of semantic message routing.

The system is also intended to be robust[4], using autonomic system techniques to preserve index knowledge and to adjust the agent inter-connections when one or more agents within the system are unavailable. Most other approaches to storing and querying semantic web meta-data largely rely on a centralised repository (e.g., Sesame).

SERSE was designed to function in concert with the other tools produced by the Esperonto project, and as such depends upon some of the other tools for essential support functions. The first of these dependencies is on the Ontology Server, that supports the construction and maintenance of the ontologies that SERSE uses – however, it is not relied upon for access to the ontologies themselves, as they are accessed directly by SERSE. The second dependency is upon the Annotation System, that provides more fundamental support, by performing the annotation of information resources on the basis of the available ontologies and their extensions. SERSE then receives notifications of newly acquired and annotated resources from the Annotation system, and it is these annotations that are then stored, indexed and made searchable using the distributed index structure.

[4]Does not break down easily or is not wholly affected by a single component failure, and either recovers quickly from or holds up well under exceptional circumstances.

The second application scenario considered within this research regards the application of Semantic Web technologies to the provision of enhanced search facilities over specified information sources in real commercial test-case scenarios. The SERSE approach to robust and scalable indexing addresses issues related to the volume and interconnectedness of Semantic Web metadata annotations. In this additional investigation we have examined the practical application of semantic annotation and search technologies to problems of information overload and precise information access. One result of this investigation was the QuestSemantics system, which is designed to provide an integrated semantic annotation and search facility over business information, and is tailored to the needs of the system's end-user. The QuestSemantics system consists of two principal components: the annotation component, which automatically discovers suitable web resources and annotates them with relevant metadata; and the search component, providing a specialised search interface to enable the specification of precise semantic queries to be executed over the generated metadata. Both the annotation and search functions are based upon the use of a task and domain specific ontology and extension, that represents the relevant business knowledge.

1.3 Goals and Contributions

The primary objective of my research is to develop an approach for the search and location of Semantic Web resources that addresses many of the problems involved in other approaches. These problems include a lack of robustness and fault tolerance due to reliance on a centralised index of content, limited use of the available information describing the semantics of the indexed resources, and poor scalability due to use of broadcast messages for communication between the nodes of distributed indexes. This research is based upon the use of a multiagent system architecture to distribute an index of semantically annotated web resources, and to link these indexes on the basis of the semantics of their content. We investigate to what extent this use of an agent architecture contributes towards dealing with the inherent complexities of the Semantic Web, and seek to answer the following questions:

- Can this approach make use of the available semantic information (from the semantically annotated resources and the referenced ontologies) to index the web resources and to answer semantically specified queries for such resources?

- Does the distribution of a content index, sub-dividing it on basis of the semantics of its content, provide a more scalable solution than a centralised index?

- To what extent does the application of agent technologies facilitate the provision of such a distributed content index? Specifically, do the concepts of agent autonomy and social ability provide means to dynamically manage such a distributed index?

- Does the index distribution and use of agents provide a more robust and fault-tolerant indexing system than existing centralised indexes?

The secondary objective of this research is to investigate and assess the applicability and added-value of semantics-based search, particularly in the context of information overload problems, when searching over business information. As part of this objective we have investigated the means to provide automatic semantic annotation of selected information sources, based upon the knowledge encoded in the used ontology and its extension. Furthermore, we have performed an evaluation of the application of semantic search technologies in real-world commercial test-cases, to determine both the added-value of such technologies to business, and some of the barriers to the large-scale adoption of these technologies. In the development and deployment of this QuestSemantics system, a number of important lessons were learnt regarding the applicability and utility of semantic annotation and search in business environments. For example:

1. The knowledge elicitation process, to obtain and encode the specialist business knowledge required for the task, remains a significant barrier to the adoption of knowledge-based systems – as it requires a significant expenditure of time and effort from a client before any benefit can be realised.

2. The knowledge modeling constructions available in the Semantic Web languages are readily applicable to a wide range of business domains, particularly as much of the knowledge can be expressed within categorisation hierarchies.

3. The triples-based Semantic Web query languages are not readily utilised by business users, and a graphical user interface to simplify the definition of knowledge-based queries is required to make Semantic Web search available as a viable business tool.

Furthermore, the direct involvement of the commercial end-users in the development and evaluation of QuestSemantics has revealed many practical considerations regarding the adoption and use of semantic web technologies by end-users. The QuestSemantics system and the various points raised by the development and deployment process are discussed in detail in Chapter 7.

1.4 Thesis Structure

This thesis is divided into four parts, totaling eight chapters and two appendices. Part I presents the background and context of our work, surveying the main contributory research areas that underlie the work in this thesis. Part II presents the design and implementation of the agent-based system, along with an extensive experimental evaluation of its operation. Part III presents the adaptation and extension

of the techniques presented in Part II that were undertaken to investigate the application of semantic indexing techniques to real-world issues. Finally, Part IV presents a synopsis of the main results and contributions of this thesis, and concludes with a summary of future work.

A more detailed description of the structure and content of this thesis is as follows:

Chapter 1 – Provides the thesis introduction, and outlines the background, motivation and goals of this work.

Chapter 2 – Presents background material on autonomous agents and multiagent systems. The chapter provides an overview of the field, and describes the primary features of agents and multiagent systems. It further discusses the topics of agent communication and agent development platforms, with particular reference to the FIPA standardisation efforts. The chapter then moves on to present in more detail the features of information agents, and the roles such agents play in multiagent information systems. Finally, the chapter briefly presents two associated topics: (agent-based) Digital Libraries and peer-to-peer systems.

Chapter 3 – Presents background material on the Semantic Web. The chapter presents an overview of the various technologies involved in the Semantic Web effort, and focusses on the role of ontologies and metadata annotations within this environment. It provides a detailed examination of the interoperability problems associated with the use of multiple, heterogeneous ontologies, and discusses how such heterogeneity can be addressed in order to generate semantic connections between entities from different ontologies.

Chapter 4 – Introduces the design of the SERSE system, describing the system's conceptual architecture, together with a detailed presentation of the system components. In doing so, the chapter reviews the related work that has directly contributed to the design of the system, and describes the main algorithms underlying the system functions. In particular, the chapter addresses the design and function of the algorithm for calculating the semantic proximity between ontological entities, that is fundamental to the operation of SERSE

Chapter 5 – Presents the details of the implementation of SERSE. This chapter describes how the agent development platform and Semantic Web language processing toolkit utilised in the implementation are applied in practice. We then examine the operation of SERSE, describing the main functionalities provided by the system and presenting the manner in which they operate together to provide scalable and

robust semantic indexing.

Chapter 6 – Provides a detailed experimental evaluation of the SERSE system. The set of experiments presented are intended to evaluate the performance of SERSE in each of the main functional scenarios. A particular focus of the evaluation regards the performance of the semantic proximity metric (described in Chapter 4), which is compared to estimations of semantic distance obtained from state-of-the-art ontology alignment tools.

Chapter 7 – Presents the QuestSemantics system, a development of many of the technologies and ideas within SERSE, which provides an integrated semantic annotation and search system. The chapter then goes on to investigate the application of this semantic search system to commercial test-cases in which there was a requirement for enhanced search facilities over business information.

Chapter 8 – Provides some brief conclusions on the work presented, and indicates a number of potential directions for further development and enhancement of the semantic search tools presented.

The thesis also includes some additional material in appendices, that supplements the material presented in the main thesis:

Appendix A – Provides an overview of the messages exchanged by SERSE, both within the agent system and between the agents and external systems.

Appendix B – Provides a summary of the ontology and knowledge-base developed for the QuestSemantics system in the context of the North West Aerospace Alliance search application.

The work presented in this thesis has, in part, been previously published or accepted for publication, as follows:

- V. Tamma, M. Wooldridge, I. Blacoe, and A. Persidis. *Retrieval of Scientific data in Esperonto.* In Proceedings of the 3rd International Semantic Web Conference (ISWC 2003) Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, Sanibel Island, Florida, October 2003.

- V. Tamma, I. Blacoe, B. Lithgow Smith and M. Wooldridge. *SERSE: searching for digital content in Esperonto.* In Proceedings of the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004), Whittlebury Hall, UK, October 2004.

- V. Tamma, I. Blacoe, B. Lithgow Smith and M. Wooldridge. *SERSE: Searching for Semantic Web Content.* In Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004, Valencia, Spain, August 2004.

- V. Tamma, I. Blacoe, B. Lithgow Smith, and M. Wooldridge. *SERSE: surfing the Semantic Web with Esperonto.* In Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004) Workshop on Challenges in the Coordination of Large Scale Multi-Agent Systems, New York, USA, August 2004.

- V. Tamma, I. Blacoe, B. Lithgow Smith, and M. Wooldridge. *Introducing Autonomic Behaviour in Semantic Web Agents.* In Proceedings of the 4th International Semantic Web Conference (ISWC 2005), Galway, Ireland, November 2005.

Chapter 2

Intelligent Agents and Multiagent Systems

In this chapter the fundamental principles behind software agents and multiagent systems are introduced. It is not intended to provide a comprehensive review of this wide-ranging research area, but rather to focus on those aspects that are relevant to this thesis. The purpose of this chapter is to present the key points of agent technologies that underlie the design and function of SERSE *– such as agent autonomy and social ability. The chapter also introduces multi-agent information systems, in which agent technologies are utilised to manage distributed and heterogeneous information. In addition the chapter briefly reviews the main features of peer-to-peer and digital library systems, that can be implemented using agent frameworks, and have contributed to the architecture of* SERSE.

Section 2.1 presents a definition of agency and an overview of the main characteristics of software agents. This continues into a presentation of agent societies and multiagent systems in section 2.2, along with an overview of agent communication techniques, FIPA standardisation efforts, and available agent systems. Section 2.3 then focusses on the application and specific roles of agents in specialised information systems such as SERSE, *where agents are used to manage the heterogeneity and distribution of information. The closely related approaches utilised in Digital Libraries are then briefly reviewed in Section 2.3.1, and Section 2.4 presents the main ideas underlying peer-to-peer (P2P) systems, their association with multiagent systems, and their use as distributed content indexing systems. Finally, Section 2.5 presents a summary of the main ideas and approaches drawn from these research areas that have contributed to the design of* SERSE.

2.1 Autonomous Agents

The term 'agent' has a number of different interpretations with AI research, dependent on the particular context of their conceptualisation. However, a number of fundamental features can be abstracted from these various viewpoints that characterise the properties of intelligent software agents [125]:

1. **They are situated in a dynamic environment.**
   Agents operate both within and upon an environment. This may be the real world for a robot, the WWW for network agents [166], or a software environment such as UNIX for softbots [47]. This is in contrast to the theorem provers and expert systems of early AI research, that were not aware of their environment.

2. **They can have only partial information about their environment.**
   Agents are able to perceive their environment, but the information obtained can only be a limited view, and may be erroneous. Based upon this, the agent is only able able to make, at best, limited predictions about the future.

3. **They can act to make changes to the environment.**
   Agents have only limited control over their environment, and can undertake actions to alter it;

4. **They have potentially conflicting tasks they must perform.**
   In many circumstances an agent cannot achieve all of the tasks that have been allocated to it, because they are in conflict. In these situations an agent determine some subset of the tasks that are all achievable, and commit to realizing them.

5. **They have many different possible courses of action available.**
   An agent will typically have a number of ways in which it can achieve a task or tasks. The agent must select those actions that it believes will achieve the required task(s) in the 'best' possible way, and then commit to performing them.

6. **They are required to make decisions in a timely fashion.**
   Agents do not have unbounded computational resources, and the environment is dynamic and changes in real time. Therefore, agents must make the best (i.e., most rational) decisions possible, with respect to the resources (e.g., information, time, computational power) available to them [130].

Based upon these requirements for intelligent software agents, we can identify a number of key properties that an individual agent should possess [170]:

- **autonomy**: agents have control over their actions and internal state, and are able to operate without the direct intervention of any other entity;
- **reactivity**: agents can perceive their environment, and can respond to changes occurring within it in a timely fashion;
- **pro-activeness**: agents exhibit goal-directed behaviour, and are able to take the initiative to achieve these goals, rather than simply reacting to their environment;
- **social ability**: agents are able to communicate and cooperate with other entities, humans or agents, in order to achieve their tasks.

In addition to these properties, that are widely accepted as essential agent features, a number of other properties have been identified that may have a role in the characterisation of intelligent agents:

- **rationality**: an agent will act in order to achieve its goals, and will not act in a way that will prevent its goals being achieved, according to its knowledge and beliefs at the time [131];
- **veracity**: agents are assumed to be honest, and will not knowingly communicate false information [60];
- **mobility**: agents are able to move around within an electronic network, i.e., the sofware and knowledge comprising an agent can be physically transported between different host systems [166];
- **benevolence**: agents are assumed to share common goals, and so every agent will always attempt to do what is asked of it [128];
- **learning**: agents can adapt themselves in order to fit their environment.

2.2 Multiagent Systems

The design of individual agents has been the primary focus of traditional AI research, which until recently did not consider the issues of agent societies – also known as MultiAgent Systems. This area has been the main focus of research in in the subfield of AI – Distributed Artificial Intelligence (DAI) [77, 16, 61]. Therefore, the main issues in DAI are those of organisation, co-ordination and co-operation [61]. A multiagent system can be defined as a loosely coupled network of intelligent agents that are able to work together in order to tackle problems that are beyond the capabilities of any individual agent. The characteristics of multiagent systems can be summaried as [144]:

- individual agents within the system have a limited viewpoint, and have insufficient information or capability to solve the problem addressed by the system as a whole.
- the system does not have a control component, and the overall behaviour is dependent on agent interactions and social rules within the system.
- the information and resources required to complete the system's tasks are decentralised and distributed over the system.

Examining the spectrum of systems composed of more than one intelligent agent, we can distinguish two main classes of system [171]:

1. **Distributed problem solving systems** – the agents within the system have been specifically designed to co-operate in order to achieve certain goals. All agents are known a priori, can be assumed to be benevolent towards each other, and can be trusted.
2. **Open multiagent systems** – the agents are heterogeneous in design, tasks, goals, etc., and are able to dynamically enter and leave the system at will. In this case there are many other factors for agents to take into account: arrival of unknown agents; self-intertested behaviour; competition; etc.

In either class of multiagent system, the main thrust of the research regards the problem of managing co-ordination between agents in order for them to jointly take action and solve problems [16]. Therefore, when considering agent societies there are additional architectural issues to address:

- the organisational structure(s) employed in the agent system, e.g. hierarchy, anarchy, etc.
- the organisational interactions supporting these structures.
- the organisational and environmental rules that regulate the structures and interactions.

As is often the case within DAI research, there are differing approaches to solving these architectural issues. In the multiagent system field we can distinguish two main threads, fixed-design and emergent approaches. Fixed-design approaches seek to pre-determine the organisational structure and interactions in order to obtain a system that conforms to an application specification. In contrast, emergent approaches seek to create mechanisms that exert an influence upon the independent component agents, in order to generate a configuration that achieves the application objectives.

2.2.1 Agent Communication

In order for agents within multiagent systems to be able to interoperate, co-operate, etc. it is necessary for them to be able to communicate with each other. Developments towards effective agent communication began by the application of speech act theory to communications between planning agents [28]. Speech act theory considers communication as action, i.e., communications are modeled as actions that alter the mental state of the participants [5].

Following this early work on agent communication using speech act theory, a number of agent communication languages (ACLs) were developed. The US-based Knowledge Sharing Effort (KSE)[1] pursued the development of protocols and languages for the exchange of represented knowledge between autonomous information systems. There were two key results of this project: the Knowledge Interaction Format [112], and the Knowledge Query and Manipulation Language [56]. The Knowledge Interaction Format (KIF) was developed to enable the representation of a 'domain of discourse', and is based on first-order logic. KIF is intended to form the content of an ACL message, allowing agents to express properties of and relationships between entities in a domain. To enable this representation, KIF contains the usual constructs of classical logic (boolean connectives, universal and existential quantifiers, etc.), and provides a basic vocabulary of objects (numbers, characters and strings), basic functions and relations for these objects. The Knowledge Query and Manipulation Language (KQML) defines an envelope format for ACL messages. KQML enables an agent to express the illocutionary force of a message, by encapsulating a specified speech act performative. KQML defines a standard message format, and number of parameters that can be set to define the message, e.g. the performative, the message content (to be expressed in KIF), agent IDs for sender and recipient, etc.

However, despite early success, there were a number of problems with KQML, that subsequently led to the development of a new, but closely related, language from the Foundation for Intelligent Physical Agents[2] (described in Section 2.2.2):

- The performative set was not fixed – meaning different implementations of the language could not interoperate.
- Transport mechanisms were not fixed – again hindering interoperation.
- The language semantics of KQML were not fully defined – meaning that it could not be determined if an agent was using the language 'properly'.

[1] http://www-ksl.stanford.edu/knowledge-sharing/
[2] http://www.fipa.org

- A class of performatives (commisives) were missing – meaning an agent could not make a commitment to another, which is vital for coordination.
- The performative set was too big and rather ad hoc.

2.2.2 FIPA Standardisation

The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 to foster standardisation in agent-based systems. FIPA specifications are divided into five categories: Applications, Abstract Architecture, Agent Communication, Agent Management and Agent Message Transport. These specifications are shown in Figure 2.1, and are detailed further below, with the exception of Applications that are examples of systems constructed using the standard architecture, communication, etc..

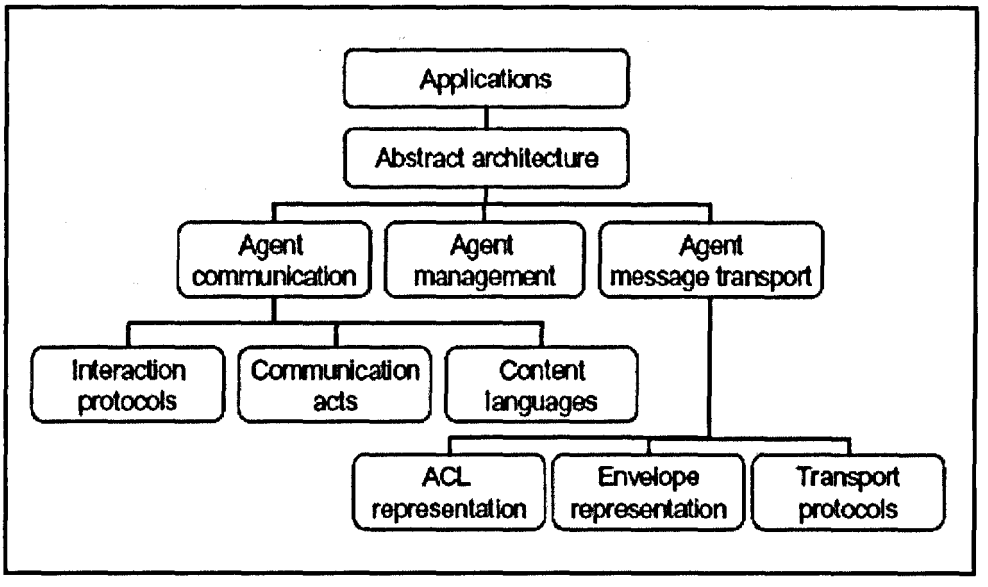


Figure 2.1: FIPA standards.

Abstract Architecture

The Abstract Architecture is intended to promote interoperability and reusability. The specification defines the key elements of agent systems, and how these elements interact. Furthermore, it distinguishes the readily abstractable elements, such as ACL, message transport, directory services, etc., and the more implementation specific elements, such as agent management and mobility services.

Agent Communication

Specialised communication techniques are required to structure and control the interactions within agent

systems. The FIPA agent communication specifications address these issues, covering the communication language, and predefined libraries of communicative act types, interaction protocols and content languages.

The FIPA ACL[3], similarly to KQML, defines an envelope language for messages, defining 22 performatives (such as inform and request) to fix the intended interpretation of messages, but does not require a specific message content language. The syntax of FIPA ACL messages is also similar to KQML, with the same general structure and attributes. The performatives in this ACL differ from those in KQML, and have been given a formal semantics based on the 'speech acts as rational action' approach [27, 19], using the formal Semantic Language (SL). The performatives are defined in terms of SL formulas, representing the constraints on beliefs, desires, actions, etc. that the sender must conform to in order to correctly use the performative, and that represent the rational effect of the performative (though conforming to these effects is not mandatory). Several FIPA implementations have been developed, and examples of these are described in Section 2.2.3 below. The ACL Communicative Act Specifications defines a library of the 22 FIPA communicative acts and their requirements, and the ACL Message Structure Specification describes the grammatical structure of the language.

In addition to the ACL specification, the FIPA standards for agent communication include interaction protocol definitions to give structure to agents' message-based interactions. The Interaction Protocol Library defines a set of interaction protocols: Request, Query, Request When, Contract Net, Iterated Contract Net, English Auction, Dutch Auction, Brokering, Recruiting, Subscribe, and Propose. Furthermore, the ACL defined requires use of a content language to express the 'body' of a message. The Content Language Library provides a generic description of the requirements for a FIPA content language, and specifies the following content languages: SL Content Language, CCL Content Language, KIF Content Language, RDF Content Language. Within the content language employed for a message, agents require a means to unambiguously express information about entities and relations within a domain of discourse – requiring the parties to a communication to have an agreed terminology about the domain. A widely accepted means to do this is through ontologies, that formally define the terms and relations present within a conceptualisation [142]. Ontologies are described in Section 3.3.

2.2.3 Agent Platforms

Once a degree of standardization has been achieved, though efforts like FIPA, practical commercial and industrial use of agent-based systems requires implemented tools and platforms. The are a wide variety

[3] http://www.fipa.org/repository/aclspecs.html

of both academic and commercial agent platforms available [158], which differ significantly in terms of architecture and implementation. In the following, we give an example selection of the major publicly available platforms that implement the FIPA specifications.

AgentCities project

Whilst not an agent platform in its own right, Agentcities[4] provided an open, dynamic agent environment in which the FIPA specifications were evaluated and refined. Agentcities developed an open agent environment to allow the deployment of agents, enable communication between agents on the basis of standard protocols, agent languages and ontologies, enable them to dynamically discover one another through directories, and support interactions between them to establish coordination relationships. Agentcites main objectives were development of:

- a realistic, decentralised, open distributed system enabling high-level peer-to-peer interoperability between agents on multiple platforms,
- a rich trading environment through agent-based business services, enabling business transactions between agents in the system to support the dynamic composition of services,
- and agent-based applications by deploying multiple agents offering diverse services.

Agentcities is comprised of a number of agent platforms that each form a node within the network, and agents can directly interact with others on any platform. By use of the communication models, semantic frameworks, shared ontologies, etc., agents on different platforms can interact to provide end-user applications. The Agentcities distribution model has three levels:

- Network level: Agentcities platforms interoperate and exchange basic communications at the infrastructure level.
- Service composition level: The services provide an open test bed for running services, by hosting business components, including their service and behaviour descriptions.
- Semantic interoperability level: Agentcities as a test bed for communication in an open environment that can dynamically host business components, and perform service discovery and invocation.

Agent Development Kit

The Agent Development Kit (ADK) is developed by Tryllian[5]. ADK is a mobile component-based

[4] http://www.agentcities.org
[5] http://www.tryllian.com/

development platform that enables the construction of reliable and scalable applications – featuring dynamic tasking, a JXTA-based P2P architecture with XML message-based communication supporting FIPA and SOAP, JNDI directory services, and uses a reliable, lightweight runtime environment based on Java. ADK supports the creation of large-scale distributed solutions that operate regardless of location, environment or protocol, enables an adaptive, dynamic response to environmental changes.

FIPA-OS

FIPA-OS[6] was the first Open Source implementation of the FIPA standards, supporting all the main standarised elements, and now also supports the majority of the FIPA experimental specifications. A further development of the platform, FIPA-OS 2, is a component-based toolkit implemented in Java, and there is also a "small-footprint" version aimed at PDAs and smart-phones.

JADE

The Java Agent Development Environment [9] (JADE) is a Java-based agent platform developed under a grant from the European Commission (IST-1999-10211). JADE agents exist within a 'container', hosted on a 'platform', that forms the agent's home environment. The 'main container' on a platform contains agents that provide the particular functions of a JADE platform – Agent Management System, Agent Communication Channel, Directory Facilitator, etc. JADE also provides support for FIPA ACL, content language ontologies, multiple MTPs, agent mobility, etc. The JADE platform and the methodologies for developing agent-based systems upon it are described further in Section 2.2.4.

Java Agent Services

Java Agent Services (JAS) project[7] was developed by a number of software industry vendors, and has defined a standard specification and API for the deployment of commercial-grade agent platform-service infrastructures. The project consists of a Java API for deploying open platform architectures, that support the plug-in of third-party platform service technology, and provides interfaces for message creation, encoding and transport, and directory and naming services. The plug-in based design is intended to ensure that JAS deployments are unaffected by changes in underlying implementations and technologies.

LEAP

The Lightweight Extensible Agent Platform[8] (IST-1999-10211), known as LEAP, is a development and run-time environment for agents. LEAP is an integrated agent development environment, that can generate agent applications in the ZEUS environment and execute them on JADE-dervived run-time

[6] http://fipa-os.sourceforge.net/
[7] http://www.java-agent.org/
[8] http://leap.crm-paris.com/

environments. In this way, LEAP benefits from the advanced design-time features of Zeus and the lightweight and extensible properties of JADE. LEAP is intended to have a small 'footprint' enabling an agent contain to run on almost any device, and has been implemented on wide variety of devices (computers, PDA and mobile phones), and over various communication mechanisms (TCP/IP, WAP).

ZEUS

ZEUS[9] is an open-source toolkit for constructing collaborative, multi-agent applications. Zeus provides support for generic agent functionality and, in particular offers sophisticated support for the planning and scheduling agent actions. The toolkit provides a graphical environment for agent construction and behaviour modification. ZEUS implements FIPA ACL for communication, and uses a TCP/IP sockets-based message transport.

We now return to the JADE agent platform introduced above to provide a more in-depth description of the platform, its components and functions.

2.2.4 The JADE Platform

JADE is a middleware environment, developed by TILab[10], for the development of distributed multi-agent applications based upon the FIPA Abstract Architecture and the employing the FIPA Agent Communication Language (both of which are described in more detail in Section 2.2.2). JADE is composed of two main components: the run-time environment providing the services required by deployed agents; and the API libraries needed to develop agents within this environment. The run-time environment consists of one or more agent 'containers' that together comprise an agent 'platform' – that provides a homogeneous middleware layer over the lower-level tiers (operating system, network, hardware, etc.). The platform and container services enable agents to perform a number of vital tasks:

- Dynamically discover other agents, and communicate with them on a peer-to-peer basis.
- Register the services it offers, and search for services offered by other agents.
- Manage its state and life-cycle.

The host platform assigns each agent deployed on it a unique name – a Globally Unique IDentification (GUID) – by assigning a unique local name and qualifying it with the unique platform ID. As described in the previous section on FIPA ACL, agents communicate by exchanging asynchronous

[9]http://www.labs.bt.com/project/agents.htm
[10]TILab is a research division of Telecom Italia S.p.A.

messages. This is a model of communication that is very widely accepted for distributed and loosely-coupled communications, i.e., between heterogeneous entities that have no a priori knowledge of each other.

The architecture of the run-time environment is that of the FIPA Abstract Architecture, consisting of an Agent Management System, a Directory Facilitator, and a Message Transport System. These components and the relationships are depicted in Figure 2.2, and described in more detail as follows:

- The Agent Management System (AMS) has supervisory control over access to and use of the Agent Platform. The AMS is implemented as an agent on the JADE platform, and only one AMS can exist on any platform. The AMS maintains a directory of the identifiers (AID) and state of all agents on the platform, each of which has registered with the AMS and been allocated an AID. The AMS uses this information to provide a yellow-pages look-up service and life-cycle management services.

- The Directory Facilitator (DF) provides the default yellow-pages service in the platform. The DF is implemented as an agent on the platform, an offers a number of services to the other agents that enable them to register descriptions of the services they offer, and search for specific services from each other. A JADE system may support multiple DFs, each offering a yellow-pages service for a specific domain or set of agents. Furthermore, DFs on the same or different agent platforms can be federated into a DF network that acts as a global facilitation service.

- The Message Transport System (MTS) – also known as the Agent Communication Channel (ACC) – provides facilities for the exchange of all messages within the platform, and for messages to and from other platforms. Unlike the AMS and DF, the MTS is not implemented as an agent on the platform, but is implemented as a set of middleware services.

On the launch of a JADE platform, the AMS and DF are initialised on the platform, and the MTS activated to enable inter-agent communication. The AMS and DF are hosted in the platform 'main container', and subsequently other agent containers may be created within the platform and connect to this main container. Platforms may be distributed over different host machines, each running one or more containers. Each of these hosts execute a single Java Virtual Machine (JVM), which each provide a complete run-time environment for a set of JADE agents, executing concurrently within separate execution threads.

FIPA-compliant agent environments in general, and JADE in particular, offer a number of features that were useful for the implementation of the SERSE system design presented in the previous chapter.

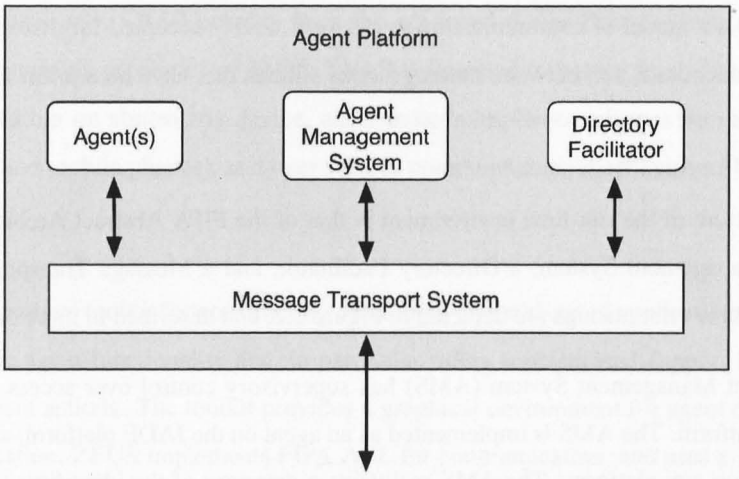


Figure 2.2: JADE platform architecture.

Firstly, the FIPA specifications enable end-to-end interoperability between agents on different agent platforms, that may be implemented using different development environments. Secondly, JADE simplifies the development of distributed applications composed of autonomous entities that need to communicate in order to function as a system, by provision of a software framework that masks the complexity of the distributed architecture. This allows application developers to focus on the application rather than on framework and middleware issues, and, in addition, a number of debugging tools are included to assist in the development of multiagent systems. Thirdly, JADE supports the development of systems that require coordination and negotiation between agents, through provision of software libraries implementing peer-to-peer communication and agent interaction protocols. Fourthly, agents control their own thread of execution and, so can be programmed to execute actions on the basis of a goal and (internal and environmental) state changes – without requiring human intervention. Finally, peer-to-peer architectures are, in general, more efficient multi-party solutions than client-server architectures – removing the server as a potential bottleneck and/or point of failure of the whole system. This is because agents can both provide and consume services, and so remove any need for a distinction between the two. JADE agents allow clients to communicate with each-other without the intervention of a central server.

Agent Behaviours

An agent must be able to perform several, possibly concurrent, tasks in response to different external and internal events. The different ways in which an agent can act are governed by the *behaviours* that an agent is equipped with – within an agent system the execution of a behaviour may involve a sequence of actions, both internal and external to the agent itself. When designing multiagent systems within JADE (and within FIPA-based systems in general) all the actions that the intended agent will be able to take must be encoded into a set of behaviours. This set of behaviours may be arranged hierarchically, with

some behaviours representing low-level or 'atomic' actions of the agent, and other behaviours making use of one or more of these simpler behaviours to provide a higher-level or more complex agent behaviour. Such structuring of agent behaviours bears some similarity with modularisation of programme code, as both intend to avoid replication of functionality (and consequent developer effort). This behaviour model provides support for the execution of multiple, parallel and concurrent agent activities.

Within the JADE middleware environment, every agent consists of a single execution thread and all its possible actions are modelled and implemented as `Behaviour` objects. Developers implementing a specific agent behaviour, define a sub-class of one of the library `Behaviours`, then instatiate this object and add the behaviour to the agents task queue. The generic `Agent` class provides methods to add and remove behaviours from the task queue, allowing the agent to manage its own task list and add behaviours and sub-behaviours as required. The `Agent` class provides a behaviour scheduling process, that performs a round-robin, non-preemptive scheduling policy among all behaviours in the task queue, and then executes the selected behaviour until it releases execution control. A behaviour can relinquish control before completing, in which case it is added to the task queue and re-scheduled, or it can *block* execution whilst waiting for a message to be received. This behaviour model is intended to enable the behaviours to operate as cooperative execution threads. However, there is no equivalent to the thread stack – so all elements of the agent's state must be managed in variables of the behaviour and the agent itself, as in finite-state machines. This approach can become problematic when dealing with complex tasks, therefore, JADE provides behaviour composition methods that allow complex behaviours to utilise more simple one (as described above). The JADE development library provides a set of 'skeleton' behaviours for composing sub-behaviours, that each apply different execution policies to the sub-behaviours.

We now turn our attention to agent-based systems that may be implemented using one of the aforementioned agent platforms. Specifically, we will now examine the tasks that agents can achieve and the roles that agent can play in Information Retrieval systems.

2.3 Information Agents

The amount, diversity and complexity of online information resources is overwhelming manual browsing and centralised search system approaches. Furthermore, the diversity of information systems means that there is a need for interoperation between them, so that isolated systems can "exchange information and services with other programs and thereby solve problems that cannot be solved alone" [62].

In the same way that they are fundamental to agent communication, ontologies are a vital technology for information systems. Ontologies provide the basis for handling information resource heterogeneity – enabling semantics-based integration of diverse sources. The use of semantics in information systems is intended to complement existing techniques for information navigation and retrieval [107]. Mena and colleagues in [107] propose a division of information systems approaches into three types:

- Classical keyword-based and navigational approaches that do not use semantic representations, often leading to poor quality answers.

- Global shared ontology approaches that enable complex queries and indexes, supporting high quality answers. Such systems are only achievable where prior terminological agreement can be reached.

- Loosely coupled approaches that use multiple ontologies, where interoperability is achieved by ontology mappings. These systems are more complex than shared ontology systems and the quality of answers is lower, but they are more scalable and extensible.

Development of loosely-couped, ontology-based approaches to distributed information systems found many points of correspondence with developments in multiagent systems – in particular the emphasis upon distributed reasoning, and the use of explicit knowledge representations. It could be seen that the combination of these technologies offered the possibility of providing a common means to access independent information sources, and, further, to offer common access to all types of resources and services. The application of agents and multiagent systems offer further means to handle the information and software heterogeneity by using agents to 'wrap' information systems and resources. Such 'intelligent information agents' are intended to interact with each other, and with users to provide active assistance in the location and organisation of information [101]. The benefits of using agents within a heterogeneous information environment are based on their ability to interoperate through common protocols, whilst exerting local control over information resources, removing the need for a central point of control [78]. Information agents are situated in this information environment, and can independently specialise and adapt to the features of the information resources locally available to them.

The definition of an information agent was initially often limited to that of agents for information retrieval, that is, agents that can retrieve information for end-users. This stance is typified by [91] – "*information agents are computational software systems that have access to multiple, heterogeneous and geographically distributed information sources. Such agents may assist their users in finding useful,*

relevant information; in other words, managing and overcoming the difficulties associated with 'information overload'. Information agents not only have to provide transparent access to many different information sources in the Internet, but also to be able to retrieve, analyse, manipulate, and integrate heterogeneous data and information on demand, preferably in a just-in-time fashion". This definition later became extended to include the management and organistion of information and resources [137]. Therefore, information agents can be characterised as agents offering one or more of a variety of key information services:

- Acquisition – obtain and provide access to information.
- Management – maintain resources and ensure up-to-date information.
- Search – locate and retrieve information matching a request.
- Integration – aggregate and integrate information from heterogeneous sources.
- Presentation – layout information in a suitable format.
- Adaptation – tailor information to users, tasks, context, etc.

A multiagent system inhabited by a variety of trusted information agents that offer this range of services can then perform value-added information services for users. Examples of such services provided by multiagent information systems are given in [91], and include:

- access to multiple heterogeneous distributed sources to locate, retrieve, process, integrate, and store information,
- notification to users of publication of new information within a specified topic area,
- negotiation for and purchase of information, goods and services,
- explanation of the relevance and reliability of retrieved information,
- evolution and adaption to the dynamic information environment.

Therefore, multiagent information systems use their inherent dynamism and heterogeneity to manage the dynamic and heterogenetic nature of open information environments. The different tasks involved are performed by different agents within these systems, and these agents communicate and cooperate integrate the available information resources. The reasons for the suitability of multiagent systems to this task of providing homogeneous access to heterogeneous information can be summarised in four main points. Firstly, the information sources may be heterogeneous in a number of dimensions,

which requires local adaptation. Secondly, the information sources themselves are distributed, which requires system-level cooperation to integrate them. Thirdly, no centralised control is required, meaning that there is no single point of failure or bottleneck within the system. Finally, the inherent complexity calls for modular solutions and clear task divisions to manage it, with minimal user involvement.

There are a variety of multiagent information systems in existence that aim to integrate information from heterogeneous sources, and most of these approaches utilise agents in three main roles [93]:

- **Information consumers** – that query information managed within the system and receive the result of the information integration process. Such agents are often user interface agents or applications.

- **Information providers** – that link the information sources into the agent-based system. These often take the form of 'wrapper'[11] agents that provide an agent interface to an existing information resource. Such agents are often used to 'translate' the information from the data-schema used in the information source to that used in the multiagent system.

- **Middle agents** – that mediate between the provider and consumer agents by directing queries to those information sources that are best able to answer them [35]. This is achieved by matching a query to the provider's advertisements of their capabilities [167]. These middle agents can be further sub-divided into facilitator, broker and mediator agents that provide different 'middle-man' services on top of their basic matchmaking task.

Agents often play more than one role in these systems, such as agents that are both consumers and providers of information. In this case the agent receives information, processes or otherwise adds value to it and then provides this processed information to other agents [93]. Examples of information integration systems that utilise agents in this sort of architecture are iBROW [12], RETSINA / LARKS [146, 145], InfoSleuth [115] and KRAFT [120].

This three-layer system of provider, middle and consumer agents is reflected in the architecture of the Esperonto project – introduced in Chapter 1. As depicted in Figure 2.3, the provider agents are represented by the wrappers attached to each of the information sources, the demonstration applications (and the user query interface within SERSE) represent the consumer agents, and the semantic indexing and routing system (SERSE) represents the middle agents. Therefore, SERSE requires many of the functionalities of middle-agents: locating resources, matching queries to resources, connecting requestors

[11] So named because they can be seen to wrap around the existing resource.

and suitable providers, etc.

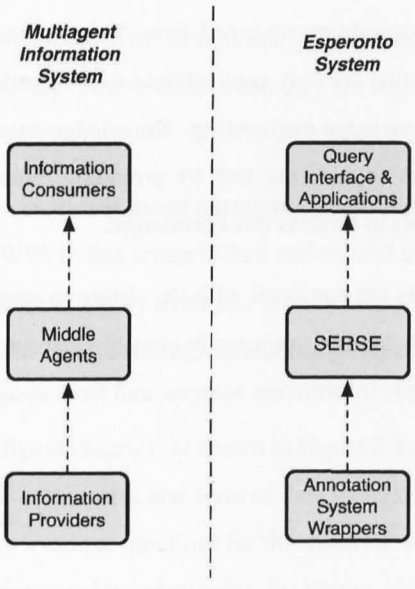


Figure 2.3: Multiagent Information Systems architecture compared to Esperonto architecture.

Many approaches to middle agents require the provider agents to broadcast their capability descriptions which raises a question over the scalability of such systems due the communication overheads inherent in such an approach [129]. In addition, many approaches utilise a central register of content and/or a hierarchical structure of middle agents [93]. Any large-scale open and dynamic environment, such as the Semantic Web, requires certain features from any information integration architecture, including:

- **Scalability** – meaning that the architecture should be able to function effectively over any number of information sources. The scalability is affected by the volume and efficiency of communications between the agents, by any hierarchical relations between the middle agents, etc.
- **Failure recovery** – the architecture should be able to cope with the failure of individual agents within the system, and should support self-maintainance to manage such failures.
- **Maintainability** – the architecture should maintain up-to-date references to the information sources to allow for the arrival and departure of information sources into and from the system.

Approaches to constructing information agents and multiagent information systems can be distinguished into three main types [92]:

- User Programming – agents are equipped, by the user, with the requisite rules and knowledge to

perform their specific information processing task. This approach is often made impractical by the degree of knowledge and understanding needed from the user.

- Knowledge engineering – agents are equipped, from design-time, with application-specific knowledge. This creates agents that are only applicable to their intended information environment, and needs significant prior knowledge engineering. Knowledge-based environments, such as the Semantic Web, assist in this engineering task by providing reusable ontologies, annotations and knowledge-bases, and tools to process this knowledge.

- Machine learning – agents are equipped with the ability to acquire the knowledge they need to process information. This learning process is usually performed by abstracting useful patterns from user-provided example information sources, and from system usage monitoring.

2.3.1 Digital Libraries

Digital libraries have been proposed as a solution to the problems of information management and integration in distributed, multimedia environments, such as that represented by the WWW [96]. As such they have a number of similarities with the role of SERSE within the Esperonto project. They also envision a network of distributed repositories, where all types of objects and resources can be located and retrieved, both within and across different indexed collections. A general definition of a digital library is 'a distributed technology environment which dramatically reduces barriers to the creation, dissemination, manipulation, storage, integration and reuse of information by individuals and groups' [99]. Therefore, a digital library is, as with traditional libraries, not simply a collection of information resources, but provides additional services relating to those resources. Reflecting this, digital libraries have also been defined as [102]:

> *"An electronic information access system that offers the user a coherent view of an organised, selected, and managed body of information"*

In order to achieve these aims within the intended environments Digital Libraries have had to address many of the same issues as encountered in the design and development as SERSE. Therefore, many of the approaches adopted by Digital Libraries may provide useful pointers to the solution of common problems within distributed and heterogeneous information environments.

The main functions of a digital library can be summarised as (from Esposito *et al.* [44]):

- Collection – refers to the detection of information resources that are useful to the clients of the library. It does not necessarily refer to the physical acquisition of such resources.

- Organisation and Representation – these require that the information resources are classified and indexed, according to criteria relevant to the potential users.
- Access and Retrieval – which involve the design and organisation of the materials within a physical space in order to effectively retrieve them when they are required.

A digital library system has to address some of the major issues of information retrieval, particularly in an environment such as the WWW. In this scenario data and control are distributed, there is no central control or viewpoint, and the system is highly dynamic. In addition, a digital library might reasonably be expected to provide personalised services to individuals and groups [54].

The general architectures of digital libraries, as shown in Figure 2.4, are based around the concept of Digital Objects [83], which represents a particular form of data structure containing the digital material (data) and metadata for it, such as a unique identifier for the material. In the system proposed by Khan and Wilensky the digital objects are stored in repositories, the unique identifier (handle) is obtained from a global naming authority and registered with a handle server. The repositories have mechanisms for depositing and accessing digital objects, and store a properties record containing the metadata for each of the digital objects within them. The system described provides methods for the naming, identification and retrieval or invocation of digital objects in an architecture of distributed repositories, and enables the location of objects independently from their physical location in the system.

A good example of a digital library system not based upon agent technologies is the Cheshire System [96]. This system was originally designed as an online library catalog that provided a combination of probabilistic and boolean retrieval methods. Subsequently, Cheshire has evolved into a flexible information retrieval tool for textual and structured metadata resources, and is utilised in a wide range of applications – including web-based library catalogs and specialised document search engines. The key features of the system are:

- SGML and XML metadata representation, manipulation and storage.
- Web-based client-server system, supporting the z39.50 v.3 Information Retrieval Protocol.
- Natural language queries, incorporating boolean logic.
- Probabilistic document ranking and document-topic clustering methods.
- Dynamic document linking, supporting exploratory browsing.

A more recent development of the Cheshire system, aims to provide a generic framework for Information Retrieval applications – based around use of modular components to provide the required functionality. This Cheshire 3 system [165] is built upon the integration of a number of technologies, including 'Data Grid' technologies for distributed resource storage and access, a content management model and model-based document parser, and text and data mining technologies. The intention is to develop a framework within which Information Retrieval and Digital Library systems can be developed, based upon existing, best-of-breed tools for the required functionality.

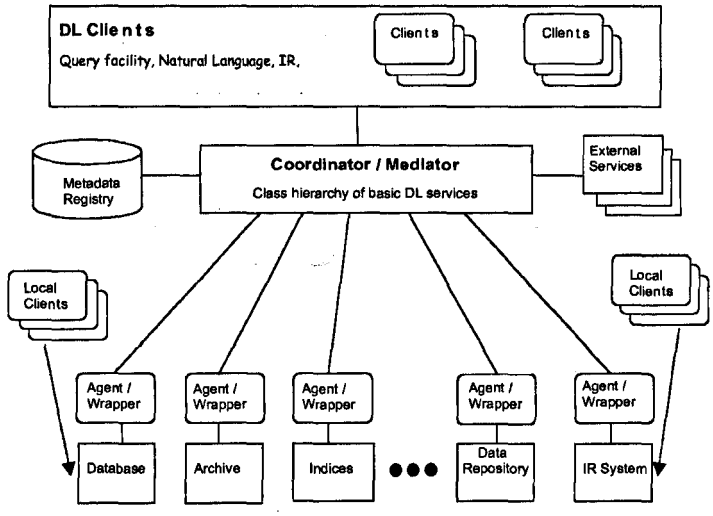


Figure 2.4: Generic agent – based digital library architecture.

A classic example of agent based digital library technologies is that of the University of Michigan Digital Library (UMDL) [39], which is based upon the use of an agent architecture in order to deliver the library services in a decentralised environment. The UMDL system supports an evolving, open network of capability providers and consumers that each act autonomously in the interest of their 'owners', but which collectively provide the whole digital library service. The library is characterised as an 'evolving agent economy' in which individual agents provide and consume different library services, and communicate and cooperate with each other in order to achieve their goals. The basic services, such as those for locating other agents, reaching agreements with them, etc., are standardised within the system and are realised using agents and teams of agents. The extensible basis of the system means that third-party agents, which might offer specialist services such as natural language processing, are free to join the system and offer their information and / or services to the library. The UMDL system is designed as a number of layers, which build upon each other to provide the library functionality. The lowest layer

is the system and network environment that provides the network transport (TCP/IP), agent communication transport layer (CORBA), and the operating environment for the agents (Unix). Above this lies the 'agentware' that defines agent communication interfaces, which are implemented using the KQML performatives and message structure. Built upon this basic structure are three levels of UMDL-specific functionalities:

- Protocols that enable system-wide agent interaction and interoperability, by defining a small number of patterns of message exchange that enable all the agents to tackle common tasks, such as location, selection, etc., in the same way. The semantics of the terms used in the KQML messages are defined in a UMDL ontology that defines the basic concepts used in the system, such as the types of things that can belong to a digital library (documents, articles, pictures, etc.) and the types of library services available. The ontology is not designed as a source of metadata to describe the content of library resources. This layer also defines a number of languages used by the system, such as that for registration and querying the registry, but leaves open the option to utilise more specialised languages for specific library tasks.

- Facilitators are those library agents that collectively provide the UMDL services for agent location, content-based message routing, security, negotiation and renumeration. These services underlie and underpin the task-specific services of the system that users see. The principal facilitator is the Registry agent, which maintains records of the agents within the system and their capabilities, and retrieves lists of agents that satisfy the capabilities required by a query.

- Services form the top-level of the UMDL architecture and provide the task-specific library functions to the users of the system. A task-specific agent operates in a narrow task domain, and the system implements, or envisions the need for, agents that offer services for a registry, content publishing and retrieval, a thesaurus, query planning, a content metadata vocabulary, task planning, a user interface and a collection interface.

The concept of agent-based digital libraries has been developed by a number of research groups, for example ZunoDL [54]. In this approach the digital library is implemented as a collection of autonomous agents operating within an 'information economy', in which producers and consumers of information interact in a market-like environment. The system has producer agents that link information 'owned' by third parties to the digital library, facilitator agents which map user's requirements onto the producers best able to satisfy them, and consumer agents which interface between the library and its users.

2.4 Peer-to-Peer Systems

Peer-to-peer systems can be characterised as distributed systems in which all nodes have minimal set of equivalent capabilities and equal responsibilities towards each other, and all communication is symmetric [129]. In P2P systems, each agent or node in the system is on an equal footing with all the other agents in the system, and communication between two peers is an exchange of information between two equals, though the overall capabilities and responsibilities of the nodes may differ widely. There are a number of definitions of what constitutes a peer-to-peer system, many of which overlap or are complementary, however:

> *"Reduced to a common denominator, P2P refers to a technology that enables two or more peers to collaborate spontaneously in a network of equals (peers) by using appropriate information and communication systems without the necessity for central coordination. " [134].*

Multiagent systems and peer-to-peer systems can be viewed as being compatible – in that agents can be designed to behave as peers in a network. Peers within a peer-to-peer system can be characterised as software objects that have a shared set of responsibilities towards each other [129], that is each peer will respond to a specific message from another peer in a known manner. Furthermore, this implies that the peers have a core set of shared (or equivalent) capabilities in order to provide the required services to other peers in the network. Such characteristics can also be seen in typical multiagent information systems (see Section 2.3) where the agents within the system are designed to offer a specific set of services to each other to enable the overall cooperative behaviour of the system.

The idea of peer-to-peer systems is not a new one, as the telephone system or Usenet groups can be characterised as P2P systems. More recently the use of peer-to-peer technologies has become more widespread, primarily through the very large numbers of users for file sharing systems, such as Napster[12] and Gnutella[13], and instant messaging services, such as Jabber [80]. However, there are methods used in many of these systems, in particular the use of a central index of content or a message broadcast protocol, that deviate from those of a 'pure' peer-to-peer system.

A comprehesive survey of peer-to-peer systems is presented in [2]. In this survey the authors present a classification of peer-to-peer systems based upon the application category of the system:

- Communication and Collaboration – Systems that provide the infrastructure for facilitating direct, real-time communication and collaboration between peer computers. These systems include chat

[12] http://www.napster.com/
[13] http://www.gnutella.com

and instant messaging applications, such as IRC, MSN and Jabber.

- Distributed Computation – Systems that aim to use available idle CPU processing cycles on peer computers. The main approach is to break down a processing-intensive task into many small work units, which are then distributed to peers to be processed. This approach usually requires central coordination to divide original task and re-combine the work-unit results. Prime examples of distributed computation P2P systems include seti@home [135] and genome@home [63].

- Internet Service Support – A variety of different systems exist to support internet-wide services. Such systems include a variety of peer-to-peer multicast systems, like those described in [162] and [23]; indirection infrastructures [140]; and web security applications [164, 87, 81].

- Database Systems – There are a number of published approaches to constructing distributed database systems based upon peer-to-peer infrastructures. These approaches include:

  - the Local Relational Model (LMA) [14] in which the totality of the data within a P2P network is treated a set of inconsistent local relational databases, which are interconnected by 'acquaintance' links that define mappings between the databases. This 'acquaintance' concept bears significant similarity with the 'neighbour' concept utilised within SERSE, though in LMA these connections are not predicated upon any specific semantic relationship between the linked databases.

  - the PIER system [76] is a scalable, distributed query engine built upon a P2P overlay network, enabling large scale (1000s of nodes) distributed processing of relational database queries.

  - Piazza [70] is, like SERSE, a Semantic Web [151] (see Chapter 3) peer-to-peer system in which the individual nodes act as sources of data (e.g. obtained from a database to which the node has access) and/or ontologies (see Section 3.3) in order to provide an information infrastructure for Semantic Web applications. The nodes in Piazza are all interconnected by chains of pair-wise mappings between nodes, again as in SERSE, but in Piazza individual nodes have heterogeneous data semantics and queries are replicated and distributed through the peer network to multiple source nodes, whereas SERSE sub-divides the queries and moves each one through the P2P network to a single source node.

  - Edutella [113] is also a SW P2P system, that uses standardised metadata representation to provide a metadata-based infrastructure and querying facility to applications layered upon it, such as Bibster [69]. Edutella and Bibster together bear some similarity to SERSE, such as the use of an onotology-based querying process, semantically annotated resources, semantics-based query routing, and indexed content being distributed over a number of

peers. However there are also a number of differences between the two systems. The two most significant differences are that SERSE uses a network of agents to host a distributed metadata index, in which each agent has a specified and unique area of expertise, selected from *any* of the ontologies used in the annotations. On the contrary, Bibster's peers have expertise that is based on their own resources, that are annotated according to two design-time specified ontologies. These differences reflect the differing goals of the two systems – Bibster aims to enable the the sharing of local resources regarding only bibliographic information over a network of peers; whereas SERSE aims to provide an open indexing system for resources annotated according to any ontology, and distributes the index over a network of agents in order to ensure the scalability and robustness of the system.

- Content Distribution – This forms the largest category of existing peer-to-peer systems, in which the intention is to enable end users, represented by peer nodes, to share digital media and data over the internet. Content distribution P2P applications range from relatively simple direct file-sharing systems, such as the aforementioned Napster and Gnutella, Kazaa [85] and FreeNet [26]; to more sophisticated systems that provide a secure and efficient distributed data storage, often built upon distributed hash tables, such as Oceanstore [95], PAST [129], Chord [141] and Groove [66].

The focus of the survey presented in [2] is upon P2P content distribution systems. Consequently the authors present a further two-level classification of such systems based upon the variety of available (up to 2004) technologies, which consists of:

- Peer-to-Peer Applications – The essential features of a P2P content distribution application are the publishing, searching and retrieval of files by peers in the P2P network. They can be further sub-divided into two types of system:
  - File Exchange Systems: Intended to provide direct transfer of individual files between pairs of peers. The system facilitates the setup of the peer network, and enables the search and file transfer functions. Most examples of such systems are lightweight implementations that largely ignore security, availability and persistence issue, and focus upon basic, best-effort file exchange. This category of system includes the most well-known examples of peer-to-peer systems, such as Napster, Guntella and Kaaza.
  - Content Publication and Storage Systems: Intended to provide a distributed storage medium for users to publish, store and distribute digital content, in a way that is secure and incorporates persistent content management (update, removal and version control). In many such systems peers require appropriate privileges to access specified content, and facilities are provided for anonymity, accountability, and censorship resistance.

- Peer-to-Peer Infrastructures – Service and application frameworks based upon P2P networks, that provide distributed storage and retrieval facilities to end-user applications. They can be further sub-divided into three service types:
  - Routing and Location: All P2P systems rely upon a mechanism for routing messages between peers within the peer network. Ideally mechanisms should be both efficient and fault-tolerant. Within SERSE this vital task is performed by the Semantic Relatedness Metric, described in Section 4.4.
  - Anonymity: Enabling peers, and thus end users, to remain anonymous when publishing and retrieving files.
  - Reputation Management: As in many mass-user systems, reputation management is utilised in some P2P content distribution systems to enable user to select optimal sources from their content requirements. In P2P systems, with no central authority to manage such facilities, complex solutions are needed maintain a distributed reputation management system that is secure, up-to-date and has high-availabilty.

Essentially a peer-to-peer system consists of a set of methods, services and protocols that allow nodes to pass messages amongst themselves so that messages are directed to the correct node. A node is any computer program, application, agent, etc. that implements the methods and protocols, and so can communicate and interoperate with other nodes in the system. Each node requires methods to locate and contact other nodes and to cope with nodes arriving in and departing from the system. The nodes also require a mechanism to determine which of their neighbour nodes to pass a message to. In addition to passing messages, nodes provide resources (information, data files, etc.) under their control to the peer group. The messages themselves can represent queries, responses, maintenance functions, etc., and are directed to nodes by means of location independent addresses, that point to nodes irrespective of the node's physical location in the network. The P2P system has to address a number of issues relating to efficiency, such as keeping the number of messages to a minimum to reduce network load and ensuring that messages traverse the nodes between their source and destination by the most direct route. Implementations of peer-to-peer systems have built the network of nodes on top of existing network communication and location infrastructures such as TCP/IP.

In order to direct messages to their intended destinations, a peer-to-peer system requires an index of the system content. This index can be a simple hash table, linking resource identifiers to resource locations, or may be more complex, providing metadata about the resources. The index needs to be stored within the P2P system, and this can be achieved in three main ways:

1. As a centralised index held in a single location. This acts as a hub through which all queries pass, and provides a global view of the content of all the peers. However, this central point is a source of significant vulnerability, and failure of the index effectively disables the whole system.

2. Distributed amongst a sub-set of the nodes in the system. These index nodes are referred to as super-peers[14] and can be seen as peers with additional capabilities that provide a specialised service to the P2P system.

3. Distributed amongst all the nodes in the system. In this case every node has responsibility for a specific part or parts of the global index. Ideally the index distribution will be even amongst the peers, ensuring that storage, computation and networking costs are approximately equal for all nodes.

In peer-to-peer architectures messages can be passed between peers using two different methods. Broadcast messages are sent from one peer to all the other peers, or a large subset of the peers, in the network. This is not a technique that will scale, due to the network traffic generated by these multiple messages. Point to point messages are sent between specific peers, by a process of dynamic routing whereby the message passes from neighbour to neighbour and can traverse a number of nodes in the system en route from its source to its destination. Neighbours are those nodes that have a logical link between them, determined by the P2P application, and this bears no relation to the underlying physical network connections. Figure 2.4 shows an example message route, where the message originates from a source node S, and is passed from node to node until the destination node D is reached. In such a system of dynamic routing, each node en route needs to select the best neighbour to send the message on to, which is usually achieved by numerical closeness of the neighbours ID-key to that of the messages intended destination. The messages marked as 3 and 4 on the diagram demonstrate an incorrect message routing. In this case the node that receives message 3 is not the intended destination, but does not have any other neighbours to which to send the message on to. Such errors can occur due to imprecision in the method used to select the best node to route a message to. Therefore, the node returns message 4 to the originator of message 3 – which then determines the next closest neighbour to route to[15]. This results in message 5 to node *D*, which is the message's destination.

Routing indexes [33] are intended to extend the information available at each peer that can be used to identify the best neighbour to which to route a query. This is achieved by including information from 'over the horizon', that is, from beyond the current node's immediate neighbours. In the system proposed by [33], documents and queries are characterised using manually assigned topic (or class)

[14]That is, peers that have additional functionalities / responsibilities within the peer-to-peer system.
[15]Temporary message routing histories can be kept by nodes in order to avoid circular message routing.

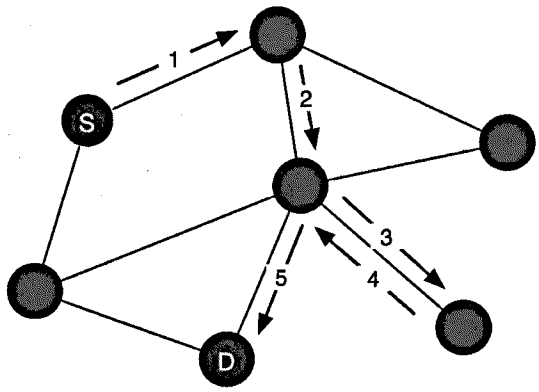


Figure 2.5: Message routing in peer-to-peer architectures.

labels, which are matched as strings. Each peer in the system has a routing index that records the number of documents available, subdivided by their characterising topics, that indicates which neighbouring peer represents the best source for information on any specified topic. Three types of these routing indexes are defined:

1. *Complete* – each peer's routing index enumerates by topic the total number of documents available from all peers.

2. *Hop-count* – the routing indexes are subdivided by topic and by the number of neighbour peers that must be traversed in order to reach the documents. Therefore, for a particular topic, there is a record of how many documents can be found one 'hop' away, two hops, etc. The size of the index can be controlled by imposing a maximum number of hops for which the indexes record document counts.

3. *Exponential* – the routing indexes contain, for each topic, a number that is algorithmically generated as the sum of the number of documents available multiplied by an exponential factor based on the number of hops away those documents are located.

Semantic Overlay Networks [34] are a method to group together peers based on the semantics of their content resources. As in [33], document semantics are represented by assigned topic labels. For each topic there exists a group of peers that contain documents characterised by that topic, and these peers form the overlay network for that topic. Nodes can have different neighbours in a number of different overlay networks. A query is processed by identifying which semantic overlay network (SON), or set of SONs, are best able to answer it. The query is then sent to a node in that SON and is only forwarded to other members of the SON. [34] proposes the use of a classification hierarchy as the basis for the formation of the SONs, where documents and queries are classified into one or more leaf concepts in the hierarchy. However, in practice, it may not be possible to determine which concept

they should fall within, in which case they can be classified into one of the non-leaf concepts that is the immediate ancestor of the possible concepts for classification. Since document classifications usually change infrequently, SONs seek to classify documents in advance of queries, placing them in 'buckets' that correspond to concepts in the hierarchy. The authors consider that a suitable classification hierarchy is one that:

- Produces 'buckets' with documents that belong to few nodes, because the smaller the number of nodes to be searched the better the query performance.
- Ensures nodes have documents in a small number of 'buckets', in order to reduce the number of SONs that a node belongs to and, therefore, to reduce the load at that node.
- Allows for easy-to-implement classification algorithms that make a low number of errors.

Semantic Overlay Networks utilise the fact that when a user is searching for a topic they do not necessarily want to find every available resource in that topic, but require a timely response with a significant proportion of the available resources. Therefore, nodes that only contain a few resources in a topic may not join the SON for that topic, thus reducing network traffic and the load upon that node without significantly reducing the utility of the system. Nodes that have a significant percentage of their documents (the authors use 15%) classified in a topic are assigned to the respective SON. Documents in topics that do not reach this threshold are grouped together under successive super-concepts until the number of documents within a concept passes the threshold (not including those documents already assigned to a more specific concept). The mechanism is referred to as Layered Semantic Overlay Networks, and produces an improvement in the efficiency of queries (in time, message overhead, etc.) with only a small reduction in the maximum recall level achievable.

Current research in the peer-to-peer domain is largely focused upon distributed indexing, dynamic message routing, reduction of messaging overheads, and fault tolerance through redundancy and replication [8]. In addition, there is significant ongoing work in the usage of peer-to-peer system approaches within the Semantic Web and (Semantic) Grid research communities [2].

2.5 Summary

This chapter has described a number of areas of research in autonomous agents and mutiagent systems, and the related topics of (agent-based) Digital Libraries and peer-to-peer systems. Each of the research areas described have contributed different technologies, approaches and techniques that can be used in

the design and construction of a distributed system for semantic content indexing, such as SERSE.

The core concepts of autonomous agents – proactivity, reactivity, social ability and autonomy – and the organisation of agents into multiagent systems all provide clear means by which to achieve many of the stated aims of SERSE. Autonomy is important because the indexes within SERSE are intended to 'know' which topics they index, and act independently to mange their own index. Social ability fulfills the need for the indexes to communicate and cooperate with each other to achieve system-level functions. Reactivity enables the indexes to act appropriately on receipt of different messages from other components of the SERSE system. Finally, both reactivity and proactivity provide the means by which the individual components of the index system can act in order to maintain the functioning of the overall system in case of disruptive events. When autonomous agents are designed and constructed in order to work together as a multiagent system, the combination of the different agent capabilities enables the system to exhibit system-level functionality that cannot be provided by any of the component agents individually. Furthermore, the sub-division of a multiagent system into independent functional elements, that communicate with other elements through messaging, means that a multiagent system lends itself naturally to distribution over multiple networked host systems.

The existing work on information agents and multiagent information systems described in this chapter demonstrate how a multiagent system can be organised in order to achieve specific information retrieval tasks. The use of multiple specialised agent types, with different capabilities, to perform different roles within the system provides an effective way to modularise the internal functionality of the system. Furthermore, the separation of information system components into provider, mediator and consumer, when applied to the function of SERSE within Esperonto, clarifies the intended role of SERSE within the overall information retrieval task – allowing design to focus on the mediation role, connecting queries to relevant content. Digital Libraries (DLs) offer an alternative approach to the construction of large-scale information storage and retrieval systems, with a focus on replicating the functions of an actual library in digital form. However, DLs also demonstrate the viability of large-scale information systems, and, with agent-based digital libraries such as UMDL, there is a clear confluence of ideas between DLs and multiagent information systems that can further inform the design of SERSE.

There are three primary contributions towards the design of SERSE from the peer-to-peer (P2P) technologies described in this chapter. The first is the way in which the peers in the system have only limited knowledge about the other peers in the system. That is, each peer is only aware of and able to communicate with a relatively small sub-set of the other peers, supporting system scalability by limiting the size of each peer's routing index. The second regards the logical organisation of peers using semantic

overlay networks to inter-connect peers, so that peers only know about other peers whose content has similar meaning. This leads to peers having routing indexes organised on the basis of specified 'topics', so that each peer can determine which other known peer is the best source for information on a topic. Finally, the concept of routing messages from one peer to another via multiple other peers in a series of hops provides an efficient communication mechanism for a large-scale distributed system, in which each node cannot know all of the the other nodes in the system.

Chapter 3

The Semantic Web

This chapter presents an overview of the Semantic Web – its background, vision, technologies and languages. It is not intended to cover all aspects of this multi-disiplinary endeavour, but will focus upon those aspects that have relevance for the SERSE *system presented in the next two chapters. The purpose of this chapter is to highlight the specific Semantic Web technologies and approaches that have contributed to the design and development of* SERSE *– primarily knowledge representation, manipulation and query languages, dynamic mapping between such representations, and resource annotation.*
Section 3.1 presents the general vision of the Semantic Web, as an extension of the current web. In section 3.2 we present the concept of metadata in this context, and its use on the Semantic Web. Section 3.3 goes on to describe ontologies, and their role as a fundamental building-block of the Semantic Web. Section 3.4 then describes how such ontology-based metadata can be queried and retrieved, using Semantic Web languages designed for this purpose. Finally, in section 3.5 we discuss semantic heterogeneity and matching, including the types of heterogeneity that may occur, semantic similarity between entities, ontology alignment techniques, and semantic indexing methods.

3.1 The Semantic Web Vision

The Semantic Web is intended as an evolution of the World Wide Web in which the information within it is also expressed in a machine understandable way, rather than only be human interpretable [151]. That is, on the current web the elements of web resources that are processable by software only regard the layout, navigation, etc., and express nothing about the *content* of the resource. The main purpose of the languages and technologies developed within Semantic Web research are to enable the addition of machine-processable metadata to web resources, giving software the means to 'understand' the meaning of the information contained within them. The semantics of the metadata are underpinned by use of ontologies to unambiguously define sharable terminologies – as introduced in the previous chapter in

the context of agent communication and multiagent information systems. Providing software with the means to meaningfully process self-describing information resources promises to significantly increase the power of the web – freeing users from time-consuming information processing tasks, and allowing user applications to provide sophisticated, 'knowledge-level' services that leverage the vast quantities of information available. However, in addition to stressing the *semantic* of the Semantic Web, we must also retain the *web* by retaining the decentralised, heterogeneous, fault-tolerant and scalable approaches that have made it such a success. Furthermore, The Semantic Web has the potential to make far more knowledge available, through the combination of different pieces of information from diverse sources, thus making the provision of scalable, decentralised control mechanisms even more crucial for its success [151].

The development of the Semantic Web has progressed on the basis of extending the current web with knowledge representation languages that enable the encoding of structured information and processing rules, sufficient to support automated reasoning over the information contained. Underpinning the Semantic Web with a set of 'standard' knowledge representation languages overcomes the key problem of representational heterogeneity that has hindered interoperation of traditional KR systems. Furthermore, the languages developed for the Semantic Web are intended to strike a balance between their expressivity and the degree to which they can be reliably processed – underlining the focus on the Semantic Web as a practical exercise. The Semantic Web is often depicted as a 'layer cake' of languages and standards, in which each layer builds upon the expressivity and functionality offered by the layers below it. This 'layer cake' is shown in figure 3.1[1].

The aim of the Semantic Web is to provide machine-processable descriptions for any kind of web resource or service, thus creating the kind of open information system discussed in the previous chapter. The standard languages and protocols of the Semantic Web provide the framework within which a wide range of knowledge-based and artificial intelligence techniques can be applied to create intelligent service that can solve complex problems for users. The interoperability enabled by the use of Semantic Web standards means that software components are able to reliably and consistently exchange, share and process the represented knowledge.

[1] www.w3.org/2005/ 12/31-daml-final.html

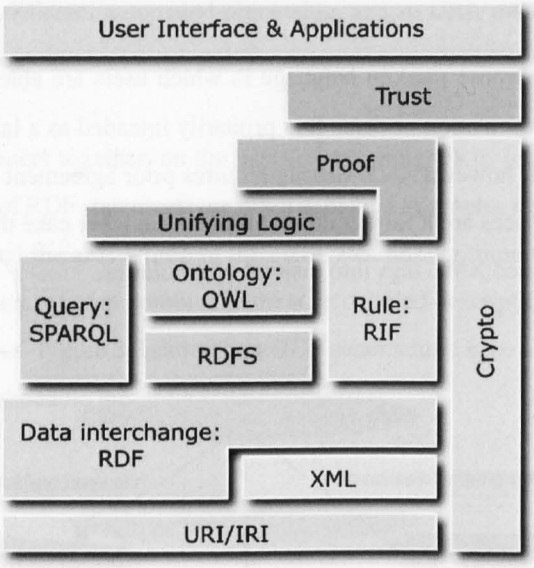


Figure 3.1: Semantic Web 'layer cake'.

3.2 Metadata and RDF

Metadata is 'data about data'. That is, metadata represents an additional layer of encoding that expresses supplementary information about the information item(s) under consideration. Metadata is used to facilitate the understanding, use and management of data. However, the metadata required for description will vary with the information being described, and so any metadata scheme for heterogeneous data must be sufficiently flexible to handle this heterogeneity and sufficiently extensible to adapt to any information domain.

On the Semantic Web, the representation of metadata is achieved by use of a number of underlying technologies that enable identification of entities and provide a syntax for metadata expression. This lowest level of the Semantic Web 'layer cake' – often referred to as the *syntactic* layer – refers to the fundamental technologies of Universal Resource Identifiers (URIs), Unicode, and the eXtensible Markup Language (XML)[2].

1. URIs mean that any entity on the Semantic Web can be assigned a unique identification that enables unambiguous reference to that entity from anywhere on the web. URIs are utilised in metadata and ontological representations to provide an identification scheme that ensures any information regarding an entity can be reliably associated with the correct entity.

2. Unicode is an industry standard scheme that allows the consistent representation and manipulation of text – that is, representation of the characters, digits, punctuation, etc. present in the variety of

[2]www.w3.org/XML/

human writing systems.

3. XML is a general purpose markup language in which users are able to define their own 'tags' to structure the representation of data. It is primarily intended as a language data sharing across information systems, however, such sharing requires prior agreement on the *meaning* of the tags used. XML Namespaces are a further contribution to the layer cake that enable the separation of different sets of defined XML tags into disjoint vocabularies.

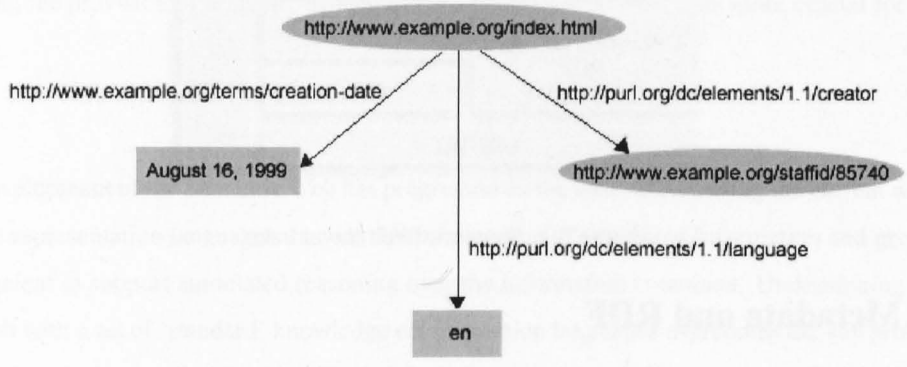


Figure 3.2: RDF graph example 1.

3.2.1 RDF

Built upon this syntactic layer are the *semantic* layers that enable the representation of knowledge. The lowest of these semantic layers consists of the Resource Description Language (RDF)[3]. RDF provides the means to make statements regarding entities, i.e., metadata statements, by reference to their identifying URIs. Statements in RDF are composed as triples that loosely correspond to the subject, verb (or predicate), and object structure in a basic sentence. These statements are encoded as a set of three XML tags, where the subject and predicate are represented by URIs, and the object is represented either by a URI or by an XML datatype value. An example of such a statement would be:

$$\langle thisthesis \rangle \langle writtenby \rangle \langle IanBlacoe \rangle$$

In this case, the subject would be identified by a URI that represents this document, the predicate is a URI that refers to a published definition of 'written by', and the object is identified by a URI representing a specific human being – myself. The use of URIs to refer to entities and predicates ensures that the statements are not just words – they are tied to unique, public definitions that define the implied semantics. In human language polysemy and synonomy add to the richness of the language, but such

[3] www.w3.org/RDF/

ambiguity significantly complicates automated processing, and so URIs enable statements to be based upon unambiguous meanings.

Statements in RDF connect together, on the basis of matching URIs, to form webs of information about entities. Collections of RDF statements are often depicited as graphs, in which the nodes represent the subjects and objects, and the arcs represent the predicates linking these entities. Figure 3.2 shows how a number of RDF statements can combine to create a detailed description of an entity, in terms of its properties and their values. Figure 3.3 shows how RDF statements can be chained together to connect such descriptions, and, thus, enable the 'information web'.

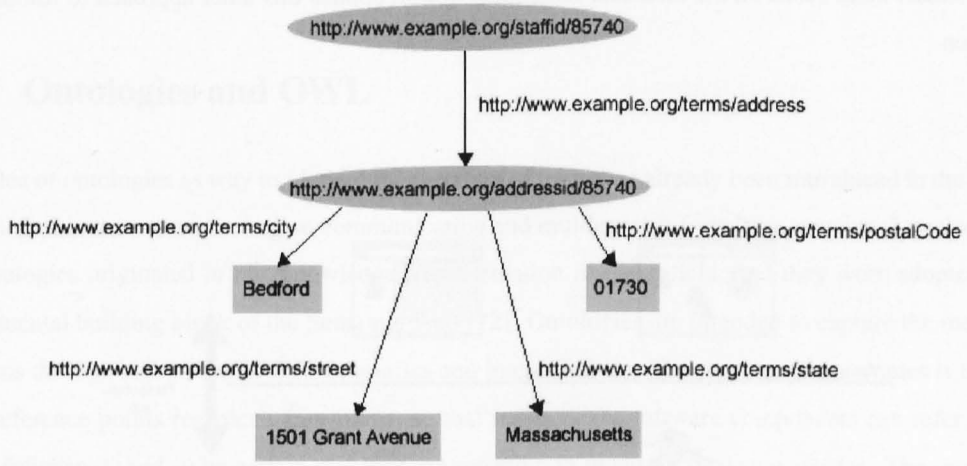


Figure 3.3: RDF graph example 2.

3.2.2 Annotation of Resources

In order to describe web resources with metadata expressed in RDF, it is necessary to link the metadata to the respective web resource. RDF statements can be linked to web resources by use of the resource's URL as the subject URI in statements about the resource. However, to enable software to discover metadata regarding a resource there is a requirement for the resource itself to contain, or indicate the location of, this metadata. This ability to effectively combine RDF descriptions within the syntax of web pages is an ongoing issue within the Semantic Web. Current proposals for the inclusion of metadata statements in web resources include RDFa and GRDDL. RDFa[4] is an approach to directly embed RDF statements into XHTML[5], by using existing HTML constructs and defining a number of HTML-compatable extensions to specify RDF content. A specific aim of RDFa is to reuse existing HTML content, so that

[4] www.w3.org/TR/xhtml-rdfa-primer/
[5] www.w3.org/TR/xhtml1/

the metadata does not repeat information already contained within the HTML tags. GRDDL (Gleaning Resource Descriptions from Dialects of Languages)[6] is a technique for obtaining RDF metadata from XML documents and XHTML pages, by declaration of an XSLT transformation for the source XML. This is intended to shift the workload from generating RDF directly to the generation of transformations for specific XML dialects.

Prior to these developments, a variety of means have been used to connect metadata descriptions to their resources. These include centralised knowledge-bases ensuring that any metadata about a resource can be discovered in one place, annotation ontologies defining specific links between resources and metadata statements, and separate indexing approaches that connect metadata and resources, and interconnect them based on the metadata semantics. SERSE applies this latter approach to resource annotation.

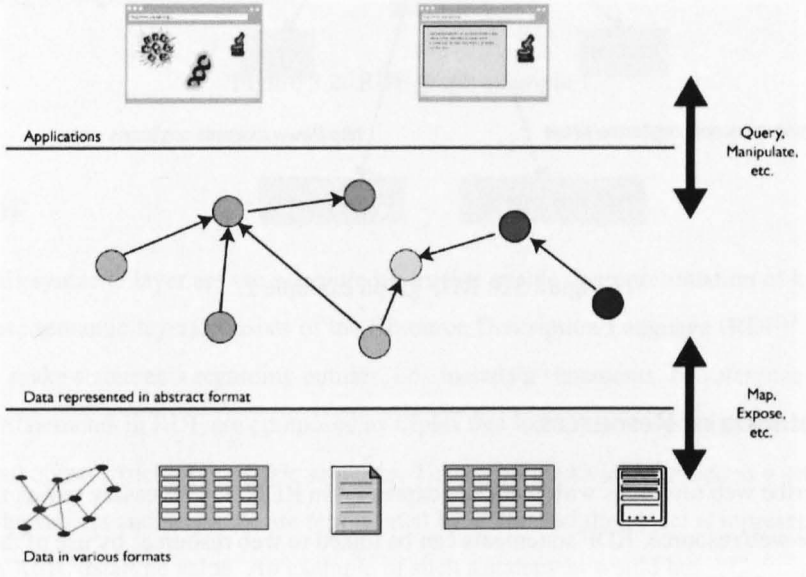


Figure 3.4: Semantic Web resource annotation.

As stated in the introduction to this chapter, the Semantic Web is intended as an extension of the current web. This implies that the Semantic Web will encompass the existing web content, upgrading this content to Semantic Web standards. This upgrade is achieved by a process of *annotation* in which

[6]www.w3.org/TR/grddl

resources are analysed and suitable metadata generated to describe them. This can be performed as an entirely manual process, where users hand-build the metadata based on their interpretation of the resource. However, the sheer volume of resources available on the web renders this an impractical approach. Recent work within the Semantic Web community has focussed upon semi-automatic annotation methods, whereby the metadata is generated automatically by software. This is achieved by using techniques from information retrieval, information extraction, etc. to analyse the resource content, and determine what this content means. The metadata is then generated based upon this analysis, by reference to the available ontological definitions to select the most appropriate descriptive terms. This general process of metadata annotation is shown in Figure 3.4.

3.3 Ontologies and OWL

The idea of ontologies as way to 'define' the meaning of terms has already been introduced in the previous chapter in the contexts of agent communication and multiagent information systems. Development of ontologies originated in the Knowledge Representation research field, and they were adopted as a fundamental building block of the Semantic Web [72]. Ontologies are intended to capture the meaning of terms through expression of their properties and inter-relations. The purpose of ontologies is to provide reference points for vocabulary terms, so that independent software components can refer to the same definitions, and so be certain that they are referring to precisely the same entities. The prototypical ontology for knowledge representation on the Semantic Web consists of a taxonomy of terms, and their properties, and an associated set of inference rules. The taxonomy enables the definition of classes of entities, and the expression of sub-class relations among them. This approach to categorising and inter-relating entities is a very powerful and natural means to encode basic knowledge for software to process [139].

The term ontology is taken from Philosophy, where it regards a systematic explanation of being [3]. More recently, the term has been co-opted by the knowledge representation engineering research community. Early work in this context provide this definition: "an ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary" [112]. This descriptive definition identifies basic terms and relations between terms, identifies rules to combine terms, and provides the definitions of such terms and relations. Note that, according to this definition, an ontology includes not only the terms that are explicitly defined in it, but also the knowledge that can be inferred from it. A later definition by Gruber [67] became the most quoted, utilised and subsequently extended by the community – in this

work an ontology was charaterised as "an explicit specification of a conceptualization". One significant modification was proposed by [17]: "Ontologies are defined as a formal specification of a shared conceptualization". These two overlapping definitions were then merged by Studer and colleagues [142] to provide the now widely accepted general definition:

"An ontology is a formal, explicit specification of a shared conceptualisation."

The elements of this definition bear closer examination:

- *Formal* – an ontology should be rendered machine-readable by encoding it in a language that has formally based semantics.
- *Explicit* – the types of concepts and the constraints on their use are explicitly defined within the ontology.
- *Shared* – an ontology captures consensual knowledge that is not private to an individual, but is shared by a group.
- *Conceptualisation* – an ontology represents an abstract model of some phenomenon that identifies the relevant concepts of that phenomenon.

Further definitions in the knowledge representation research community highlight the relationship between ontologies and knowledge bases, for example: "[An ontology] provides the means for describing explicitly the conceptualization behind the knowledge represented in a knowledge base" [13]. Thus, based on this definition, the same ontology can underlie any number of knowledge-bases. Furthermore, an ontology can be extended by adding more specific sub-concepts of a defined term, or by adding higher-level concepts that extend the domain over new areas. However, systems using the same ontologies share the same underlying knowledge structure, and so any subsequent sharing or merging of their knowledge-bases is supported by the ontology.

Due in part to the vagueness of such definitions, there is a wide range of representations that can be considered to be ontologies [142]. The research community distinguishes two main classes of ontologies:

- *Lightweight ontologies* – are mainly formed by taxonomic hierarchies, such as a web directory. Lightweight ontologies include concepts, concept taxonomies, relationships between concepts, and properties and their values to describe concepts.

- *Heavyweight ontologies* – model the knowledge domain in more detail and provide more restrictions on domain semantics. Heavyweight ontologies add axioms and constraints to lightweight ontologies.

Although there are many definitions of the word "ontology" in the research literature, there is general consensus among the ontology and SW community. The various definitions described here provide different and complementary points of view – some that are independent of the ontology construction process and of the ontology's application, and others that are influenced by the development process. However, given the wide-ranging purposes for ontologies, and their use in different research communities, an new definition was proposed in order to encompass this broad purpose [157]: "An ontology may take a variety of forms, but it will necessarily include a vocabulary of terms and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms." This broad definition of an ontology provides an 'umbrella' description of these knowledge representation constructs within a variety of research domains, including the Semantic Web.

The application of ontologies on the Semantic Web provides a means to represent what is meant by a concept or a relationship between concepts. Ontologies allow these definitions to have URIs, which are referred to from the RDF metadata statements discussed earlier. Thus, different metadata statements can refer to the same concepts and properties by use of the appropriate URI. Not only does this support interoperation between software using these ontologies, it also supports reuse of the knowledge captured in the ontological concept definitions. The expression of ontologies is not, in general, bound to any particular formalism. However, as has previously been indicated, one of the key actions of the Semantic Web endeavour is to provide a degree of standardisation, particularly with regard to languages – to enable widespread interoperation and reuse. Therefore, the W3C has proposed RDF Schema (RDFS)[7] and (most recently) the Web Ontology Language (OWL) [106] as the Semantic Web languages for the representation of ontologies.

3.3.1 Components and Types of Ontologies

Ontologies formalize the knowledge within a conceptualisation of a domain by representing different components – concepts, relations, functions, axioms and instances [67]:

- *Concepts*: A concept, also known as a class, represents the abstractions used to describe an object in the conceptualisation; where a concept can be abstract or concrete, simple or complex, real

[7] www.w3.org/TR/rdf-schema/

or fictitious. Formally, a concept is described by a term (generally a symbol), an extension, and an intension. The extension of a concept is the set of objects (i.e., instances) that the concept can be applied to: for instance, the extension of `Car` includes: "`the blue Porsche`". On the other hand, the intension of a concept is the set of properties, features, and attributes specifying the semantics of the concept – i.e., the set of features shared by these objects. For instance, the intension of the concept `Car` includes the features: `a road vehicle with engine, and usually with four wheels`. Moreover, in the description of a concept we can also use specific elements, such as: *Metaclasses* that are classes which have classes as their instances; *Slots / Attributes* which belong to a specific concept, for instance, the attribute `Age` may belong to the concept `Person`; *Facets* that allow the specification of an attribute; e.g., default value, type, cardinality, operational definition. Concepts in an ontology are usually organized in a hierarchical taxonomy. *Taxonomies* are used to organize ontological knowledge in the domain using generalization / specialization relationships through which simple and multiple inheritance can be applied. The semantics of these relationships may be based on definitions of *subclass of*, *partitions*, *disjoint composition*, etc.

- *Relations*: Relations represent different types of linkage among domain concepts. As with concepts, a relation is described by a term, an extension, and an intension. The extension of a relation is the set of possible tuples of the instantiated relation. For example, the extension of the relation `Parent` can include: "`Peter and Diane are the parents of Ian`". The intension specifies the types of the concepts linked by the relation, e.g., the intension of the relation `Parent` may be: "`the raising of children, and all the responsibilities and activities involved in it`". Examples of binary relations are `is-a` and `links to`.

- *Functions*: Functions are a particular type of relation that are defined on the set of available concepts and return a concept. For example, the function `Price-of-flat` is a function of the concepts `Size` and `Location`.

- *Axioms*: Axioms are assertions that are always true and specify the semantics of the concepts. They generally describe how the vocabulary (concepts and relations) can be used to reason over the domain. They are included in an ontology as information constraints, correctness checks, or knowledge inferences. In particular, they may express the type of relation between concepts, the signature and the cardinality of a relation, algebraic properties of a relation (e.g., symmetry or transitivity), and other conceptual properties, such as exclusivity, generality, or identity.

- *Instances*: The instances of concepts and relations are the elements of the domain – the actual objects of the conceptualised world. The instances of a concept are also known as *Individuals*.

In some cases, ontologies can be also complemented by rules and procedures:

- *Rules*: Rules follow an `if-then` structure and are used to express a set of actions or heuristics to represent decision making and business logic, and to facilitate enhanced representation and reasoning capabilities. Rules play an integral role within the Semantic Web, and the rule layer is intended to enable the deduction of knowledge and combination of information.

- *Procedures.* Procedures represent operational definitions to infer values of arguments, or to execute formulas and rules. Procedures have been particularly used for *problem solving methods* and *task ontologies* (see [110, 155] for details).

Depending on the representation language utilised and on the scope of an ontology, only a subset of the components presented above may be used.

A number of different types of ontologies are presented in the literature, that can be classified along different dimensions, which range from the level of generality of the concepts they describe, to the type of knowledge they model (regarding the domain or the task). According to [68], ontologies can be classified into four categories:

- *Upper-level / top-level ontologies.* Describe general-purpose concepts and their properties, such as space, time, etc., which are independent of a particular problem or domain.

- *Domain ontologies.* Are used to model specific domains, such as medicine, engineering or academia. The scope of these domains may vary widely, and is related to the intended application(s) of the ontology.

- *Task ontologies.* Describe general or domain-specific activities, such as diagnosis or sales.

- *Application ontologies.* Are instantiations of domain ontologies that relate to particular application requirements, and may be associated with application-related task ontologies.

An additional category of ontologies, which has not been covered by the classifications presented so far, are the so-called *meta-ontologies* or (knowledge) representation ontologies. They describe the primitives used to formalize knowledge in order to to conform with a specific representation paradigm.

Ontologies may also differ in the degree of formality by which the terms and their meanings are expressed in the ontology. In [156], for example, the authors classify ontologies as:

- *Highly informal*: are ontologies expressed in natural language. Term definitions may be ambiguous, due to the inherent ambiguity of natural language.

- *Semi-informal*: are ontologies expressed in a restricted and structured form of natural language. Restricting and structuring natural language achieves an improvement in clarity and a reduction in ambiguity.

- *Semi-formal*: are ontologies expressed in formally defined artificial languages.

- *Rigorously formal*: are ontologies whose terms are precisely defined with formal semantics, theorems, and proofs of desired properties, such as soundness and completeness.

A similar classification is given by McGuinness [105], who defines an "*ontological continuum*" that specifies a total ordering between common types of models . This essentially divides ontologies (or ontology-like structures) into informal and formal types, as follows:

- *Informal models*: are ordered in ascending order of their degree of formality – as controlled vocabularies, glossaries, thesauri and informal taxonomies.

- *Formal models*: are ordered in the same manner, starting with formal taxonomies, which precisely define the meaning of the specialization / generalization relationship. More formal models are derived by incrementally adding formal instances, properties / frames, value restrictions, general logical constraints, disjointness, etc.

In the first category we usually encounter thesauri such as WordNet [108], taxonomies such as the Open Directory[8] and the ACM classification[9], and various eCommerce standards [52]. Most of the ontologies available on the Semantic Web can be localized at the lower end of the formal continuum (i.e., as formal taxonomies), a category which coincides with the semi-formal level in the previous categorization.

3.3.2 Representing an Ontology

The definitions presented at the beginning of this section highlight the fact that ontologies are not bound to any particular formalism, but focus on the aspect of sharing a conceptual model. However, a fundamental requirement for this aspect is that ontologies are represented in some formal language, in order that *"detailed, accurate, consistent, sound and meaningful distinctions can be made"* [72]. Ontologies have typically been represented using frames (e.g., FLogic [88] and OKBC [25]), conceptual graphs, first-order logic or description logics. Currently, the most dominant are the representation schemas

[8] http://www.dmoz.org
[9] http://www.acm.org

based on description logic languages, such as OWL [106]. We will begin with a short introduction to Description Logics, and will use that as a foundation for the presentation of OWL, which is the current W3C standard ontology language.

Description Logics (DLs) [7] are a family of logic-based Knowledge Representation (KR) formalisms devised for the representation of and reasoning about the knowledge within an application domain in a structured and unambiguous way. For such a purpose, DLs are equipped with a well-defined semantics, which provides each of its constructs with a precise logical meaning. DLs structure the knowledge about a domain by defining the relevant atomic *concepts* and *roles* of the domain, and then using these to specify the properties of objects and individuals occurring in that domain. A DL provides a set of operators, called constructors, which allow the formation of complex concepts and roles from atomic ones. Concepts are sets of individuals and roles are binary relationships between individuals. For example, by applying the concept disjuncton constructor $\sqcup$ on the atomic concepts *UndergraduateStudent* and *GraduateStudent*, the set of all students can be represented by the following complex concept:

$$UndergraduateStudent \sqcup GraduateStudent$$

In addition to concept disjunction ($\sqcup$), the boolean concept constructors are concept conjunction ($\sqcap$), and concept negation ($\neg$). A Description Logic that provides, either implicitly or explicitly, all the boolean operators is called *propositionally closed.* Moreover, DLs typically provide concept constructors that use roles to form complex concepts. The basic constructors of this type are *existential* and *universal* restriction operators, which represent restricted forms of quantification. For example, the following concepts would describe all those researchers whose only affiliation is to a university, and all those persons who have obtained at least one bachelors degree:

$$Researcher \sqcap \forall affiliatedWith.University$$

$$Person \sqcap \exists hasDegreeBachelors$$

In a DL, a set of concept and role assertions constitute an ABox. Statements about how the concepts and roles are related to each other are defined by *terminological axioms.* A set of terminological axioms constitute a TBox. A TBox describes the structure of a conceptualisation in terms of classes (concepts) and properties (roles). This means that in a DL, concepts are defined intensionally using descriptions that specify what properties objects must have in order to belong to a certain class. In its simplest form, a TBox consists of concept definitions, i.e., a restricted form of concept inclusion axioms: sentences of the form $C_1 \sqsubseteq C_2$ or $C_1 \equiv C_2$, or C_2 atomic, which describe necessary or necessary and sufficient

conditions, respectively, for individual members of C_1 to be members of C_2. For example, the axiom:

$$Student \sqsubseteq UndergraduateStudent \sqcup GraduateStudent$$

introduces the atomic concept *Student*, and states that a *Student* is necessarily either an *UndergraduateStudent* or a *GraduateStudent*, or both. However, TBox axioms can also be used to describe more complex sentences. For example, the axiom:

$$\forall studiesAt.University \sqsubseteq UndergraduateStudent \sqcup GraduateStudent$$

states that everybody studying at a university must be either an undergraduate or a graduate student, or both.

An ABox consists of assertions about named individuals, using the concepts and roles defined in the TBox, i.e., terminological axioms. Thus, an ABox contains axioms of the form *C(a)*, called concept assertions, and *R(a, b)*, called role assertions – where *a, b* are object names, *R* is a role and *C* is a concept. For example, the axiom *University(University of Liverpool)* states that the University of Liverpool is a university. On the other hand, a role assertion is used to state that two objects are related by a role. For example, the axiom *locatedIn(University of Liverpool, UK)* states that the University of Liverpool is located in the UK.

3.3.3 OWL

The Resource Description Framework Schema language provides a relatively simple language for representing RDF vocabularies[10]. RDF enables the specification of inter-relationships between resources, in terms of named properties and values. However, RDF does not provide any means to declare such properties, nor does it provide any method for defining relationships between properties and other resources. RDFS fulfills the requirement for describing a taxonomy of concepts and their simple relationships and properties. RDFS specifies a basic type-system for RDF, by providing the mechanisms to define classes of resources, to define sub-class relations between them, and to restrict the application of properties to certain classes. However, RDFS falls short of the full requirements of an ontology language for the SW, by not providing sufficient expressivness to represent desired features, such as means to limit the properties with respect to number and type, means to infer that the presence of certain properties implies specific class membership, a well-defined model of property inheritance, etc.

The Web Ontology Language (OWL) [106] is the end result of the combined research efforts of a number of projects investigating knowledge representation for the SW, and is now the World Wide

[10]http://www.w3.org/TR/rdf-schema/

Web Consortium (W3C) standard for representing ontologies on the Semantic Web. The main ancestors of OWL were the Ontology Inference Language (OIL) and the DARPA Agent Modeling Language (DAML). Through the results of this prior research, OWL was based upon Description Logics (DLs), in order to provide the formal basis for representation. OWL is layered on top of RDF, conforming to the same underlying syntax, but extends RDF with an additional DL-based vocabulary for the specification of ontologies. This means that OWL ontologies can be encoded in RDF/XML documents, and parsed into normal RDF graphs. However, it also means that the logic underlying OWL is somewhat different from classical DL semantics.

OWL defines three sub-languages: OWL-lite, OWL DL, and OWL Full. These are intended to provide differing level of functionality to different communities of users, which differ in the level of expressivness offered by their underlying logic:

- **OWL Lite** – Represents the lowest level of OWL expressiveness. OWL Lite is intended to support those users that mainly need a classification hierarchy, along with a reduced set of possible constraints. The expressivness restrictions mean that tool support for OWL Lite should be simpler than with the other variants, given its reduced formal complexity.

- **OWL DL** – Represents the 'normal' level of OWL expressiveness. OWL DL is intended to strike a balance between expressivness and computability (in terms of completeness and decidability), providing users with all of the OWL language constructs, but placing some restrictions on their application to reduce them to DL complexity.

- **OWL Full** – Represents the highest level of expressiviness, and is beyond that of description logic. OWL Full semantics are based on those of RDF, providing greater expressivness by imposing fewer constraints than OWL DL, but at the cost of making OWL Full undecidable. OWL Full enables the extension of the RDF and OWL language primitives themselves, reducing the possiblity for tools to offer complete OWL Full reasoning support.

The semantics of OWL are defined using a standard model theory [106]. However, the theory is somewhat complex, given the (partial) integration with RDF semantics. OWL DL semantics are largely those of a 'normal' Description Logic – that is, an OWL interpretation has a domain that is a set of abstract objects and datatype values[11]. An ontology in OWL DL conforms to DL satisfiability (an ontology O is satisfied by an interpretation I), but of greater significance for inference is entailment, as in RDF. Inference in OWL is based on ontology entailment – an ontology $O1$ entails $O2$ iff all interpretations satisfying $O1$ also satisfy $O2$. Inference in OWL is aligned with classical DL by the fact that

[11] OWL datatypes are based upon a subset of the XML Schema datatypes.

entailment in OWL DL and OWL Lite can be reduced to checking for unsatisfiability in the respective DL model [119].

The main modelling primitives provided by OWL DL – concepts and properties – can be simply described as follows:

- **Classes (Concepts) in OWL DL** – A class defines a group of individuals that belong together because they share some properties. There are two built-in classes: *Thing* is the class of all individuals and is a superclass of all OWL classes, and *Nothing* is the class that has no instances and is a subclass of all OWL classes. New classes are introduced with either complete or partial descriptions. Complete descriptions are introduced by axioms stating equivalence of classes, while partial descriptions specify that a class is a *subclass* of another class. Furthermore, the *oneOf* class axiom gives a complete definition by enumerating all individuals belonging to this class. Additionally, classes can be specified to be disjoint with other classes. Classes can also be specified via restrictions. Cardinality restrictions specify a lower or upper bound, or an exact number, of properties that must be present. More precisely, this means that an individual belongs to such a class if and only if the number of individuals that it is related to (via some property, which is a binary relation) meets the specified bounds. The restrictions *allValuesFrom* and *someValuesFrom* are stated on a property with respect to a class. The former meaning that this property on this particular class has a local range restriction associated with it. The latter means that a particular class may have a restriction on a property where at least one value for that property is of a certain type.

- **Properties (Roles) in OWL DL** – Properties are used to state relationships between individuals (*object properties*), or from individuals to data values (*datatype properties*). Property hierarchies may be created by making one or more statements that a property is a *subproperty* of one or more other properties. Both types of property can be restricted to a certain domain and range. A domain of a property limits the individuals to which the property can be applied. The range of a property limits the individuals that a property may have as its value. OWL DL also includes primitives related to equality or inequality of properties. Equivalent properties relate one individual to the same set of other individuals. There are also special identifiers used to provide information concerning properties and their values. A property may be stated to be the inverse of another property. Properties may be stated to be transitive and/or symmetric. Moreover, properties may be stated to be functional (i.e., have a unique value – shorthand for stating that the propertys minimum cardinality is zero and its maximum cardinality is 1), or inverse functional (also referred to as an unambiguous property).

'Clearly, OWL is not the final word on ontology languages for the Semantic Web.' [73]. A number of useful features for a web ontology language had already been identified in the OWL Requirements [72] and were never incorporated into the final language. These non-implemented features include enhanced modularisation and import facilities, default inherited property values, flexible application of closed world assumption reasoning, and an integrated rule-expression language. Furthermore, an number of additional limitations have been observed in OWL:

- **Ill-defined Layering** – The main limitations of OWL are because of the problems of layering the language on top of RDF, and the (politically defined) layering within OWL itself. Specifically, OWL Lite does not meet its requirement to be an 'easy' language for knowledge representation, that would aid wide adoption.

- **Monolithic Ontologies** – OWL ontologies can be viewed as being monolithic in the sense that they are divided into a set of individual ontologies, but which have to be interpreted as one unified ontology. This is due to the poorly defined import semantics of OWL, that only provides means to include entire ontologies, rather than importing particular subsets of other ontologies.

- **Datatyping** – Datatyping in OWL remains weakly defined, and continues to be poorly aligned with XML datatypes.

- **Tractability** – As has been widely reported [73], the language has unimpressive complexity results for reasoning tasks. While several optimizations for the standard TBox problems are known, which result in tractability of realistic TBox problems, the same degree of tractability has not yet been achieved for ABox problems.

To address a number of these short-comings an update to the OWL language – OWL 1.1 [118] – was proposed in 2006. This update adds features to the language in four main categories: syntactic sugar, new Description Logic constructs, expanded data-type expressiveness, and meta-modeling constructs. The syntactic sugar is intended to make commonly stated constructs easier to express, e.g. disjoint unions, and negative property membership. The additional DL constructs include qualified cardinality restrictions, local reflexivity restrictions, disjoint properties, and property chain inclusion axioms. The expanded expressiveness in data-type handling relates to the ability to include user-defined data-types, built upon XML Schema data-types. Finally, the meta-modeling introduced is known as 'punning', and allows a specified name to refer to any or all of an individual, class or property with that label.

3.4 Semantic Query Languages

In order to take advantage of large collections of RDF metadata, a method is required to effectively query them, in order to extract specific values, check for the existence of statements, etc. In the RDF Model and Syntax Specification[12], that formally specifies a model-theoretic semantics for RDF, the primary syntax is as a graph. This graph is (partially) labeled and directed, in which the node labels are either URIs or data-type literals, and the arc labels are URIs. Constraints upon this graph syntax include: no two nodes can have the same label, no two arcs between two nodes can have the same label. An additional point is that nodes do not always need to be labeled with a URI – producing anonymous nodes (also known as b-Nodes).

The proposed query model for RDF [94] is that the RDF query itself be expressed as an RDF model, in which any of the nodes and arcs can be replaced by a variable. The result of such a query would be formed by a sub-graph of the target knowledge-base that matches the query, and sets of valid values for the specified variables. Further expressed intentions of a query model for RDF are that it leverages the inferencing available from use of class and properties hierarchies, and that a query might specify if it is to be executed against the original knowledge-base RDF graph or against a deductive closure of that graph. This query model provides a basic level of Semantic Web querying, by using the RDF model itself rather than applying any inference or interpretation over the higher-level language constructs.

A variable within an RDF query is from an anonymous node (an unlabeled 'blank' node) in an RDF graph. Despite bNodes being considered as existentially bound variables in the model theory, they differ from variables in query patterns in a number of ways:

1. The set of variables specified in a query is distinct from the set of bNodes in the queried graph.
2. Query variables can match with graph nodes, and so can be resource URIs, literals, or bNodes.
3. Query variables can also match with graph arcs, and so can be property URIs.

3.4.1 SQL-Type languages

There has been ongoing development of various semantic query languages, based on RDF graph matching and SQL-type syntax. The following descriptions of current semantic web query languages is not intended to be an exhaustive listing, but rather an overview of the development and current state-of-the-art in this field.

[12] http://www.w3.org/TR/rdf-mt/

TRIPLE – The term Triple denotes both a query and rules language as well as the actual runtime system [138]. The language is derived from F-Logic [88]. RDF triples (S,P,O) are represented as F-Logic expressions S[P→O], which can be nested. Triple does not distinguish between rules and queries, which are simply headless rules, where the results are bindings of free variables in the query. Since the output is a table of variables and possible bindings, Triple does not fulfill the closure property. Similarly, Triple is not safe in the sense that it allows unsafe rules such as FORALL X (X[rdfs:label→"foo"] ←(a[rdfs:label→"foo"])@default:ln.. While Triple is adequate and closed for its own data model, the mapping from RDF to Triple is not lossless. For example, anonymous RDF nodes are made explicit. Triple is able to deal with several RDF models simultaneously, which are identified via a suffix model. Triple does not encode a fixed RDF semantics. The desired semantics have to be specified as a set of rules along with the query. Triple does not support datatypes, and updates to the fact base are also not possible.

SquishQL – SquishQL [109] adds to the baseline query model, introducing filter functions over the variables, which restrict the values that the variables can take. These filter functions do not change the expressive power of the graph pattern. The pattern language is formed from:

- Triple patterns, which describe one edge of the graph, allowing either a variable or an explicit value for each of subject, predicate and object. In the syntax, variables are indicated by '?': the most general pattern is (?x, ?y, ?z) which matches any triple.
- Graph patterns, which describe the graph shape, expressed as a collection of triple patterns. In the syntax below, there is a list of triple patterns which are interpreted as the conjunction of the triple patterns. This list is an edge-list of the graph pattern.

RDQL – RDQL was a previous W3C submission[13], now superceded by SparQL. The syntax of RDQL follows a SQL-like select pattern, where a from clause is omitted. For example, Select ?p Where (?p, $\langle rdfs : label \rangle$, "foo") collects all resources with label "foo" in the free variable p. The Select clause at the beginning of the query allows projecting the variables. Namespace abbreviations can be defined in a query via a separate "using" clause. RDF Schema information is not interpreted. Since the output is a table of variables and possible bindings, RDQL does not fulfill the closure and orthogonality property. In addition, RDQL is type-safe and offers preliminary support for datatypes.

SeRQL – The Sesame RDF Query Language (SeRQL) [20] is a querying and tranformation language loosely based on several existing languages, most notably RQL, RDQL and N3. Its primary design goals are unification of best practices from query language and delivering a light-weight yet expressive

[13]http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

query language for RDF that addresses practical concerns. SeRQL syntax is similar to that of RQL though modifications have been made to make the language easier to parse. Like RQL, SeRQL is based on a formal interpretation of the RDF graph, but SeRQL's formal interpretation is based directly on the RDF Model Theory. SeRQL supports generalized path expressions, boolean constraints and optional matching, as well two basic filters: select-from-where and construct-from-where. The first returns the familiar variable-binding/table result, the second returns a matching (optionally transformed) subgraph. As such, SeRQL construct-from-where-queries fulfill the closure and orthogonality property and thus allow composition of queries. SeRQL is not safe as it provides various recursive built-in functions. SeRQL is implemented and available in the Sesame system. A number of querying features are still missing from the current implementation. Most notable of these are functions for aggregation (minimum, maximum, average, count) and query nesting.

SparQL – SparQL now has the status of a W3C Candidate Recommendation[14]. SparQL was developed on the basis of many of the previous query languages, and attempted to combine the best-practices identified in the earlier languages. SparQL queries consist primarily of RDF triple patterns, along with conjunctions and disjunctions, and additional optional patterns. Variables within a SparQL query are denoted by a **?** preceding the variable identifier. Execution of a SparQL query over a knowledge-base will cause the query processor to seek to match the triple patterns expressed in the query with those in the knowledge-base – binding the variables to the corresponding parts of each triple. As an example the following SparQL query will select the names of all of the capital cites within Europe (given a sufficiently detailed geographical ontology and knowledge-base).

```
PREFIX  geo:  <http://example.com/geographyOntology>
SELECT   ?capital
WHERE  {
    ?x   geo:cityName   ?capital
    ?x   geo:capitalOf   ?y
    ?y   geo:inContinent   geo:europe
}
```

The ability to express and process semantic queries is intended to enable a SparQL compliant query processor to act as the hub of an ontology-based data-integration system (as shown in Figure 3.5, that can draw information from a number of heterogeneous sources and then pass the result to a user-facing application.

[14] http://www.w3.org/TR/rdf-sparql-query/

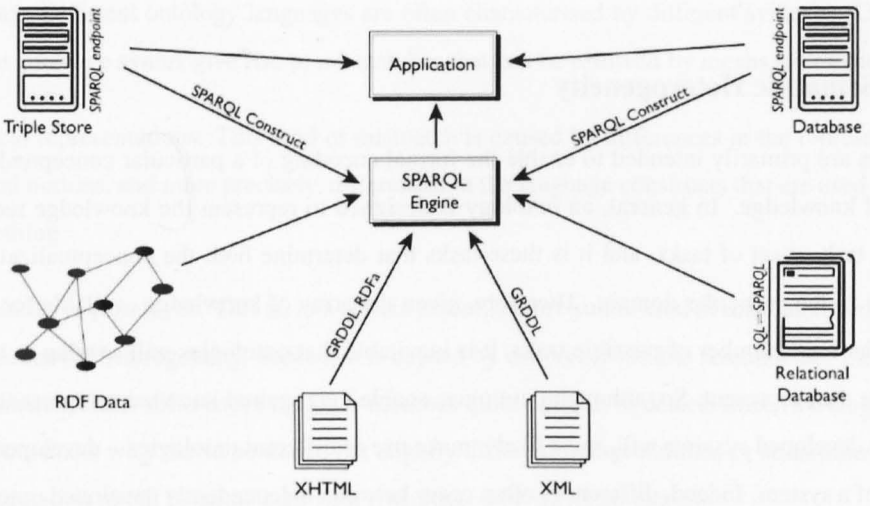


Figure 3.5: SparQL underlying a data integration system.

3.5 Semantic Heterogeneity and Matching

Ontologies provide the ability to define a vocabulary for a specific domain of knowledge, and this vocabulary can be shared among those entities participating in a task to provide an unambiguous definition of the terms used in communication, etc. However, the Semantic Web explicitly anticipates an environment in which there are many different ontologies, developed independently, whose domains of knowledge overlap. In this situation it is intended that the entities using the different ontologies engage in a process of mapping between the concepts and properties they define in order to establish those terms that refer to the same real-world entities. This process should determine which concepts in the ontologies are sufficiently *similar* to each other to be considered as being a reference to the same abstraction. The generation of such ontology entity mappings forms the basis of *ontology alignment*, where a set of mappings are generated to represent the similarities between all entities in the two ontologies involved.

In this section we first examine the underlying causes of heterogeneity between ontologies, and then describe the basis for determining similarity between them. We then go on to examine the various means for assessing ontological similarity, and examine the state-of-the-art approaches to ontology alignment.

Finally, we examine the techniques of semantic indexing, which often rely upon an evaluation of semantic similarity between index terms.

3.5.1 Semantic Heterogeneity

Ontologies are primarily intended to enable the formal encoding of a particular conceptualization of a domain of knowledge. In general, an ontology is designed to represent the knowledge required for a particular task or set of tasks, and it is these tasks that determine both the conceptualization and the delimiting of the particular domain. Therefore, given the array of knowledge available for representation and the huge number of possible tasks, it is inevitable that ontologies will overlap in terms of the knowledge they represent. So, although ontologies enable the required knowledge representation, independently developed systems will, most likely, make use of different ontologies – developed to support the tasks of a system. Indeed, differences often occur between independently developed ontologies even when they regard the same (or similar) domains of knowledge [112, 163]. When ontologies are developed independently they can differ in a number of ways with regard to the semantics they represent. The forms that this semantic heterogeneity can take and the knowledge mismatches they can cause are briefly discussed below.

The large volume of literature on the integration of heterogeneous information sources is sometimes confusing regarding the kinds of heterogeneity and the mismatches that can arise, especially where the knowledge engineering and data modelling fields meet. This makes it less easy to compare the different approaches [149, 161]. An attempt to reconcile and compare the different definitions presented in the literature and to find commonalities is given by Klein [90] and Chalupsky [24]. These works and that of Visser and colleagues [163] form a starting point for reviewing the different types of heterogeneity that might affect resources.

The importance of dealing with heterogeneity is that it causes mismatches that need at least to be taken into account, if not reconciled, when manipulating this knowledge. We can broadly distinguish between mismatches caused by non-semantic and semantic heterogeneity [89]. The former type of heterogeneity is also known as syntactic or language heterogeneity in [90], while the latter is also called ontology heterogeneity by Visser and colleagues [163]. Syntactic heterogeneity denotes the differences in the language primitives that are used to specify ontologies, while semantic heterogeneity denotes differences in the way the domain is conceptualised and modelled.

Syntactic heterogeneity occurs when resources and their underlying ontologies, that are written in

different ontology languages, are combined. In [90] and after [24] four types of mismatch due to language heterogeneity are recognised:

- Syntax: Different ontology languages are often characterised by different syntaxes. Differences in the language syntax give rise to mismatches that can be resolved by means of rewrite rules.

- Logical representations: This kind of mismatch is caused by differences in the representation of logical notions, and more precisely, differences in the language constructs that are used to express something.

- Semantics of primitives: This is, to a certain extent, a more subtle kind of mismatch deriving from non-semantic heterogeneity. Indeed, it is caused by differences in the semantics of the language statements. These differences can be sometimes quite difficult to detect, since two languages can use constructs with the same name, but slightly different interpretations, or sometimes the same interpretation might be associated with constructs with different names.

- Language expressivity: Mismatches due to differences in the expressivity between two languages are those which have the most impact on the problem of integrating/merging ontologies. Differences in the expressive power of the languages imply that one language can express something that the other language cannot express. For example, some languages support negation while others do not. A complete comparison of different ontology languages can be found in [30].

We have listed here the four types of syntactic heterogeneity, however, we should point out that mismatches due to syntactic heterogeneity can be overcome by providing the means to translate ontologies into different ontology languages in an automatic fashion. Facilities of this kind are offered by various ontology editors such as WebOde [4], which permits the editing of ontologies in a language independent representation and their automatic translation at a later stage.

Mismatches caused by semantic heterogeneity occur when different ontological assumptions are made about the same domain. This kind of mismatch also becomes evident when combining ontologies which describe domains that partially overlap. In particular, mismatches due to ontology heterogeneity can occur while conceptualising and/or explicating [163] the domain. Visser and colleagues use these terms to refer to the definition of ontology given by Gruber [67] stating that 'an ontology is the explicit specification of a conceptualisation'. That is, the process of designing an ontology is comprised of two main stages, the conceptualisation of the domain and the subsequent explication of this conceptualisation, and the idea is that ontology heterogeneity can be introduced in both stages of the design [163].

Mismatches due to ontology heterogeneity can, therefore, be subdivided into conceptualisation and explication mismatches. Conceptualisation mismatches are semantic differences arising from different conceptualisations of the concepts and the relations between them in the ontology domain. Conceptualisation mismatches can be caused by the following types of heterogeneity:

- Model coverage and granularity: This type of ontology heterogeneity occurs when different conceptualisations, and thus different ontologies, model the same part of domain differently both with respect to model coverage and granularity.

- Scope: This mismatch occurs when two concepts or relations in the ontologies seem to be the same but their extensions (that is the set of their instances) are not the same although they are not disjoint. Relations mismatches also include mismatches concerning the assignment of attributes to concepts, since those represent relations between conceptual entities [169].

Explication mismatches arise because of differences in the specification of the domain conceptualisation. During the conceptualisation phase the concepts describing the domain are selected. In the explication phase these concepts are made explicit, usually by labelling each of them with a term (which is one or more words in natural language) and associating a definition with each term, which could be expressed in natural language or in a formal ontology language. We distinguish six types of mismatches, in which the first three concern the modelling choices, the following two concern the choice of terms that are used to label a concept in the ontology, whilst the last type of mismatch concerns the way in which concepts are encoded:

- Representation paradigm: This type of mismatch depends on different representation paradigms used to model the same domain. It can become apparent with concepts such as time, actions, plans, causality, etc.

- Top-level concepts: Top-level concept mismatches arise because ontologies differ in the top-level ontologies they refer to.

- Modelling conventions (Also known as concept description in [90]): Modelling convention mismatches depend on modelling decisions made while designing the ontology. For instance, it is often the case that an ontology designer has to decide whether to model a certain distinction by introducing a separate class or by introducing a qualifying attribute relation [24, 58].

- Synonym terms: This type of mismatch is discussed in length in [163], where it is called term mismatch. It occurs when the same concept, attribute, or relation is referred to by different terms and/or described by different definitions, which are semantically equivalent.

- Homonym terms: This type of mismatch occurs when a term can refer to different concepts depending on the context. It is mainly due to the existence of homonyms in natural language, such as the English word wood, which can mean a collection of trees or the material that forms the main substance of the trunk and branches of a tree. Homonym terms can appear in different ontologies concerning the same domain if these operationalise the term in different ways. For example the concept Year might be described as a period of time divided into 12 months in two different ontologies, O1 and O2. If the first ontology considers a month as a period of time of 30 days, whereas the second ontology considers a month as a period of time that can have a number of days between 28 and 31, then the term Year in O1 is a homonym of the analogous term in O2.

- Encoding: This is maybe the easiest mismatch to resolve. It occurs when different ontologies encode values in different ways.

Heterogeneity, and especially ontology heterogeneity, can seriously hinder attempts to share and reuse knowledge automatically. In fact, in order to recognise whether two concepts from heterogeneous knowledge source are similar, we cannot only rely on the terms denoting them and on their descriptions, and we need to have a full understanding of the concepts in order to decide whether they are semantically related or not. However, the vision of the Semantic Web is that of open communication between 'knowledge-based' systems, using the ontologies to unambiguously identify the vocabulary terms used. Therefore, some method is required to reconcile terms within heterogeneous ontologies that describe the same real-world entities. Such reconciliation is based on a determination of semantic similarity between ontology terms, based upon the issues described in the following sub-section.

3.5.2 Semantic Similarity

There is extensive literature on measuring similarity in general and on word similarity in particular. The classic work by Tversky is based upon a psychological view of similarity, where similarity is treated as a property characterized by human perception and intuition [152]. Several similarity measures or semantic distance functions have been developed in Artificial Intelligence. Many of these have been provided to evaluate similarity between simple objects, in which the objects are represented as vectors of attribute values and similarity measures are defined in terms of those vectors [46]. More recently, there has been some work, such as in the SODAS project [38], that deals with similarity between complex objects [45]. However, these measures can only account for structural similarity, but can say very little on the similarity in meaning, that is, similarity between concepts rather than objects.

Semantic similarity is a form of semantic relatedness using network representation, a problem that has received much attention in the artificial intelligence field [122, 29]. Rada et al. [123] suggest that similarity in semantic networks can be assessed solely on the basis of the IS-A taxonomy, without considering other types of links. One of the easiest way to evaluate semantic similarity in taxonomies is to measure the distance between the nodes corresponding to the items being compared, that is the shorter the path between the nodes, the more similar they are. This idea is the basis of some definitions of dissimilarities defined for cluster analysis, namely ultrametrics, tree distances and strong Robinsonian dissimilarities [46]. A hierarchical / tree distance characterises an additive tree, i.e., a tree $\mathcal{T}$ with n vertices and n-1 weighted edges. The dissimilarity d(a,b) is then reproduced as the sum of the weights of all edges of the (unique) path connecting two given vertices α and β in $\mathcal{T}$.

There are two principal problems with similarity measurements based upon evaluation of an IS-A taxonomy:

1. It assumes that taxonomic links represent uniform distances, whereas in real taxonomies there is typically a wide variation in the 'distance' covered by a single link.
2. It is based upon the assumption that the two concepts being compared have a common ancestor within the taxonomy [150].

The first problem has been addressed in a number of ways, in particular by the use of weighted path measures. The weighting calculation for each link can be based on many different factors, such as:

- the types of links present [143],
- the depth of a link in the taxonomy [143, 57],
- the density of concepts in the immediate neighbourhood of the link [57, 127].

The second problem limits the applicability of this sort of measure to those concepts that have an ancestor that is common to both of them.

3.5.3 Ontology Alignment

Determining similarity between knowledge entities is usually referred to as a *matching* process, and the majority of work in this area has been done within the fields of information integration using database schemas, within XML schemas and web catalogs, and with knowledge representation using ontologies [136]. Many traditional applications based on structured data models utilise matching processes,

for example information integration, data warehousing, distributed query processing, etc. However, these processes are usually based on an initial design-time matching. More recent application scenarios, such as web service integration, agent communication and peer-to-peer systems, are more dynamic and require the ability to perform run-time matching – and this is usually achieved by use of a 'richer' knowledge model [136].

In the context of the Semantic Web (SW), this knowledge model is provided by the ontologies, and it is this model that is utilised when attempting to achieve accurate dynamic matching. The open nature of the SW environment implies that there will be multiple ontologies available for any particular domain, task, etc. among which interoperability is required, and so matching of ontology entities is a core question for the Semantic Web [40]. In addition, due to both the dynamism of the SW, and the number, size and complexity of available ontologies, it is essential that such matching systems operate in an automatic (or at least semi-automatic) manner. Furthermore, in an open environment, no assumptions can be made about the ontologies and their content, thus establishing entity matches cannot assume the existence of common entities employed as points of reference between the ontologies. The matching process that we are considering here is then based solely on the ontological information, that is, on the entity features, and on the ontology structure. When discussing ontologies in a SW environment, the product of a matching process is usually termed an *ontology alignment*. Such an alignment consists of a set of *mappings* between the entities (concepts, properties, etc.) in the aligned ontologies. Formally, following the definition in [136], an ontology alignment is defined as follows:

Definition Given two ontologies O and O', an alignment between O and O' is a set of correspondences (or mappings). A correspondence is described as a 4-tuple: $m = \langle e, e', n, r \rangle$, where e and e' are the entities (concepts, relations or individuals) between which a relation is asserted by the correspondence; n is a numeric value expressing the degree of confidence in that correspondence; and r is the relation (e.g., equivalence, more general, etc.) holding between e and e' asserted by the correspondence.

Therefore, ontology alignment systems utilise semantic similarity techniques to determine mappings between ontology entitlies, primarily by exploiting knowledge encoded within the ontologies [136]. Due to the consistent underlying semantics of ontologies, the modelled coherences are rendered interpretable, and so further knowledge can be derived, such as the similarity of entities in different ontologies [40]. This knowledge about entity similarity is derived from the features and relationships of the entities under consideration, as described in the ontologies. A simple example of how this is achieved can be seen when considering the labels assigned to ontological entities. These labels are a natural-language description of the entities, and so it can be assumed that entities having the same label are highly likely to be similar. This rule does not always hold true, but can be a strong indicator of similarity, and other

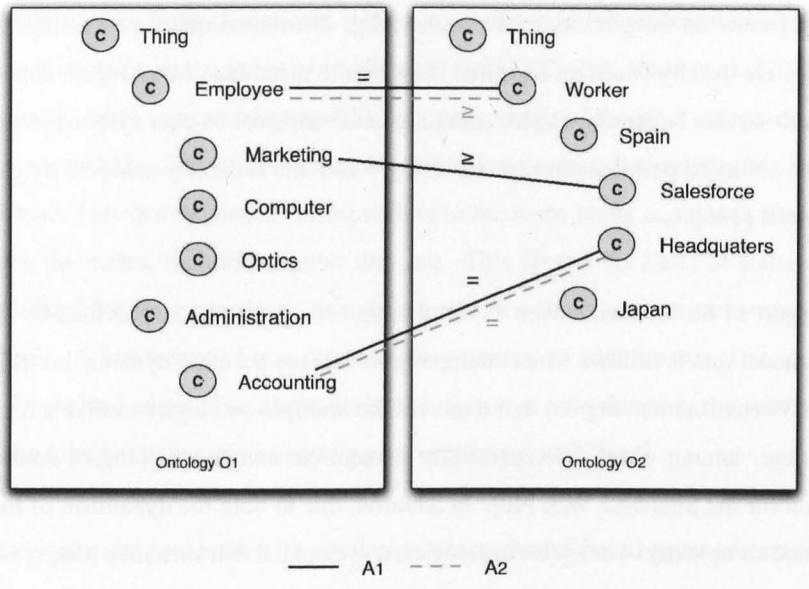


Figure 3.6: Ontology alignment example.

constructs, such as subclass relations and type definitions can be interpreted similarly [41].

Following on from the descriptions and definitions of ontologies and similarity provided previously, below we provide a formal definition of similarity for ontologies (adapted from [41]):

- $\mathcal{O}_i$: ontology $\mathcal{O}$, with ontology index $i \in \mathcal{N}$ – where $\mathcal{N}$ represents the set of all integers..
- *sim(x, y)* : similarity function.
- e_{ij} : entities of $\mathcal{O}_i$, with $e_{ij} \in \{\mathcal{C}_i\ ,\ \mathcal{R}_i\ ,\ \mathcal{I}_i\}$, and entity index $j \in \mathcal{N}$. Where $\mathcal{C}$, $\mathcal{R}$, and $\mathcal{I}$ respectively represent the concepts, relations and instances present in the ontology.
- *sim*(e_{i1j1}, e_{i2j2}) : similarity function between two entities e_{i1j1} and e_{i2j2} (where $i_1 \neq i_1$).

The key question then remains how the function *sim*(e_{i1j1}, e_{i2j2}, $\mathcal{O}_{i1}$, $\mathcal{O}_{i2}$) is computed.

In essence, aligning ontologies amounts to defining a pair-wise distance between entities (which cannot be reduced to an equality predicate), and often relies on computation of the 'best match' between them, i.e., the one that maximizes a similarity measure (or minimises a distance) between them. There are many different ways to compute such a semantic similarity measure, as shown in the following classification [136]:

- *Terminological methods.* These methods compare lexical information, and can be applied to labels, comments, etc. The comparison may be based on consideration of character strings alone,

by use of common substrings, etc., or may use some additional linguistic knowledge, such as Wordnet [108].

- *Internal structure methods.* These methods calculate the similarity between entities by comparing their internal structure, such as the value range of their properties (attributes and relations), their cardinality, and their transitivity and/or symmetry. These internal structure based methods are sometimes referred to as constraint based approaches in the literature [124].

- *External structure methods.* These methods compare the relationships between the entities under consideration and other entities. This comparison is primarily based upon the relative position of the entities within a hierarchy – if two entities (from different ontologies) are similar, their ontological neighbours may also be similar in some respects.

- *Extensional methods.* These methods compare the known extensions of entities, that is the set of other entities that may be attached to them (i.e., instances of classes). In some applications (e.g., agent interactions or web service integration) no concept instances are given before alignment, and so it is necessary to perform matching based only on the schema-level information [136].

- *Semantic methods.* These methods compare the interpretations, or more exactly the models, of the entities. Examples of such comparisions may include use of subsumption or satisfiability tests, which are well-studied reasoning tasks in description logics [6]. The main limitation of these methods is that they can only be applied after a pre-processing phase in which a number of concept mappings are declared. Therefore, when no relations are known to exist between ontologies, these techniques cannot be used to derive ontology mappings.

This classification demonstrates that ontology alignments can be derived using a variety of different methods that based on different underlying research. Ontology alignment systems can operate using different matching techniques, and differing combinations of these techniques. A number of representative approaches for ontology alignment have been proposed in recent years.

Ontology Alignment Systems

Many existing systems utilise lexical matching and synonym sets using terminological taxonomies, such as WordNet, and semantic neighbourhoods to compute semantic similarity, for example, SymOntos [57] (developed during the IST project Harmonise) and OBSERVER [107]. Some approaches also use additional information encoded into the ontology concepts, such as the mereology (part-whole relations) [57] or typical and distinguishing features of concepts [139]. Several ontology alignment approaches have

been developed within the Semantic Web research area, some of which present features such as full automation and efficiency that are particularly suitable for such an open environment. The following list of ontology alignment systems and approaches is not intended to be an exhaustive review, but rather is intended to briefly introduce some of the key existing systems, and to indicate the variety of techniques employed in different situations:

NOM – NOM (Naive Ontology Mapping) [41] is the forerunner of the QOM system [40] described in Section 3.5.4. These systems are primarily intended to allow for the ad-hoc mapping of large, light-weight ontologies. NOM is based on heuristically calculated similarity of the individual ontology entities, using a set of specialised matching rules.

FOAM – Foam [42] is a successor to the QOM system [40], and uses the same approach to computing similarity between entities.

Crosi – Crosi [84] is constructed a set of matching algorthims, that may be applied individually or in combination to achieve ontology alignment. The selection of matchers used and the relative weighting factor applied to the result of each matcher when combining results are set by the user on invocation of the system. The matchers developed within the Crosi system include: string-based matching algorithms that operate on the entity labels, structural matchers that compare entities positions in taxonomic hierarchies, and terminological matchers that make use of Wordnet to leverage synonyms, etc.

OLA – OLA (OWL Lite Aligner) [51] is intended for the alignment of ontologies expressed in OWL-lite. The matching process is designed to balance the contribution towards (dis-)similarity made by each type of component that comprises an ontology. OLA determines semantic distances between entities using matchers examining different ontology components. OLA uses all the available information (i.e., lexical, internal and external structural, extensional, and data-types) extracted from the two given ontologies. OLA then converts these distance measurements into a set of equations, and attempts to find an ontology alignment that minimises the total distance between matched entities.

FalconAO – FalconAO [82] is an automatic tool for aligning ontologies that employs a combination of two distinct matching approaches. The first is a linguistic matcher, called LMO, that utilises entity labels to generate intitial alignments. The output from LMO then forms the input to the second, graph-based matcher, known as GMO, that then generates additional mappings based on the ontology structure. Alignments are generated by the LMO and GMO according to the concept of reliability, which is obtained by observing the linguistic and structural comparability of the two ontologies being aligned.

Anchor-PROMPT – Anchor-PROMPT [59] assesses both lexical and semantic matches exploiting the content and structure of the source ontologies. Anchor-PROMPT is able to produce new concept mappings by analyzing similar paths between a set of *anchor* matches, which have previously been identified, either manually or automatically.

HICAL – HICAL (HIerarchical Concept ALignment system) [79] provides concept hierarchy management for ontology alignment, enabling one concept in a concept hierarchy to align with a concept in another concept hierarchy. HICAL uses a machine-learning approach for aligning multiple concept hierarchies, and exploits the overlap in concept instances from the two taxonomies to infer mappings. In addition, HICAL uses hierarchical categorization and syntactic information so that it can categorize different terms as identifying the same concept.

COMA++ – COMA++ [43] is a customizable and generic tool for matching schemas and ontologies, expressed in SQL, XML Schema and OWL. COMA++ provides a GUI to visualize ontology models, and to manage the matching process and the mappings produced. It supports the combined use of several matching algorithms, and different matching strategies can be applied. Of particular note is the 'fragment-based' matching approach, which decomposes a large matching problem into smaller problems, by reusing the results of previously applied matchers in subsequent matching algorithms. In addition to addressing schema-based matching problems, COMA++ also enables the comparative evaluation of the relative effectiveness of different matching algorithms and strategies.

This concludes the overview of the principle current approaches to achieving ontology alignment. Interested readers are referred to [48] for a full coverage of the state of the art in this area. We now turn to a detailed description of the previously mentioned successor to the NOM system – the Quick Ontology Mapping (QOM) system – as many of the approaches utilised in QOM are re-used and / or adapted for use within SERSE to perform determinations of semantic relatedness between concepts.

3.5.4 Quick Ontology Mapping

The Quick Ontology Mapping system (QOM) [40] was designed in order to find a balance between the effectiveness (i.e., quality) and the efficiency of ontology mapping generation algorithms. In order to present an efficient ontology mapping algorithm QOM optimizes its predecessor – the effective, but inefficient Naive Ontology Mapping (NOM) approach – by adapting (or removing) the computationally

expensive feature comparisons. Like many other state-of-the-art ontology mapping systems, QOM employs a number of different comparisons over different features of the concept descriptions, and then seeks to aggregate the results of these comparisons into a single similarity measure. Therefore, QOM represents a system that provides a widely adopted general mapping approach that produces high quality mappings, but with specific modifications for efficiency. These are precisely the features required for the semantic similarity calculation within SERSE, making QOM a suitable basis for the design of this function.

The basic ontology mapping procedure of QOM is that, given two ontologies O_1 and O_2, mapping one ontology onto another means that for each entity (concept C, relation R, or instance I) in ontology O_1, the system tries to find a corresponding entity, which has the same intended meaning, in ontology O_2. To compare two entities from two different ontologies, and to determine the degree to which their intended meanings coincide, one considers their characteristics, i.e., their features. These features of ontological entities (of concepts, relations, instances) need to be extracted from the extensional and intensional ontology definitions. Possible concept characteristics considered in QOM can include: *identifiers* – strings with dedicated formats, such as URIs or RDF labels; *RDF/S primitives* – such as properties or subclass relations; *derived features* – which constrain or extend simple RDFS primitives; *aggregated features* – aggregating more than one simple RDFS primitive; *OWL primitives* – such as an entity being the sameAs another entity; and *domain specific features* – which only apply to a certain domain with a pre-defined shared ontology. See also [41] and [50] for an overview and classification of possible ontological mapping features.

In order to compare the different features that can be obtained from concept descriptions, QOM employs a number of different means of measurement. The following four similarity measures are used to compare the features of ontological entities, at any particular iteration of the matching algorithm:

- Object Equality – is based on existing logical assertions – especially assertions from previous iterations.
- Explicit Equality – checks whether a logical assertion already forces two entities to be equal.
- String Similarity – measures the similarity of two strings on a scale from 0 to 1 (cf. [103]), based on Levenshteins edit distance [100].
- SimSet – to determine to what extent two sets of entities are similar. Multidimensional scaling [32] measures how far two entities are from all other entities and assumes that if they have very similar distances to all other entities, they must be very similar.

These results obtained from the use of these measures are all input to a similarity aggregation method, in order to produce a single similarity measure. Similarities are aggregated by a function that weights and sums the individual measures, and then normalises the result. The similarity aggregation function is:

$$sim_{agg}(e,f) = \frac{\sum_{k=1}^{n} W_k \cdot adj(sim_k(e,f))}{\sum_{k=1}^{n} W_k}$$

– where W_k is the weight for each individual similarity measure, which are assigned manually following evaluation to maximise the f-measure results on a number of training-data ontologies. *adj* is a function that transforms the original similarity values into a [0 - 1] range, providing comparable results between measures.

From the individual similarity values between the ontology entities, the actual mappings are derived. The basic idea is that each entity may only participate in one mapping and that mappings are assigned based on a similarity threshold value *t*, and a greedy strategy that starts with the largest similarity values first. Ties are broken arbitrarily by *argmax(g,h)*, but with a deterministic strategy.

The ontology mapping process performed in QOM is based upon a canonical process model that subsumes the vast majority of existing mapping approaches [40]. The steps of this process, and the specific feature comparisons performed within QOM, are as follows:

1. **Feature Engineering**: Feature engineering transforms the initial representation of ontologies into a format digestible for the similarity calculations. QOM exploits RDF triples, extracted from RDFS definitions, that represent the ontological features, such as identifiers and language primitives, which are detailed above.

2. **Search Step Selection**: The derivation of ontology mappings takes place in a search space of candidate mappings. A major ingredient of run-time complexity is the number of candidate mapping pairs which have to be compared to actually find the best mappings – so QOM uses heuristics to lower the number of candidate mappings. The system makes use of ontological structures to classify the candidate mappings into promising and less promising pairs. In particular, QOM employs a dynamic programming approach [15], in which there are two main data structures. First, there are candidate mappings which ought to be investigated, and, second, an agenda orders the candidate mappings, discarding some of them entirely to gain efficiency.

3. **Similarity Computation**: The similarity between an entity of O_1 and an entity of O_2 is determined using a range of similarity functions. Each function is based on one of the available ontological features, and employs a particular similarity measure. However, in order to optimize QOM, the range of costly features have been restricted. In particular, QOM avoids the complete pair-wise comparison of trees in favor of a (incomplete) top-down strategy. Some feature comparisons were changed from features which point to complete inferred sets to features only retrieving limited size direct sets.

4. **Similarity Aggregation**: In general, there may be several similarity values for a candidate pair of entities e, f from two ontologies O_1 and O_2, e.g., one for the similarity of their labels and one for the similarity of their relationship to other terms. These different similarity values for one candidate pair must be aggregated into a single aggregated similarity value. QOM seeks to emphasize high similarities and de-emphasize low similarities by weighting individual similarity values with a sigmoid function, prior to summing of these modified values. The aggregation of single methods is only performed once per candidate mapping and is therefore not critical for the overall efficiency.

5. **Interpretation**: Similarity results are interpreted by two means. Firstly, a threshold value is used to eliminate spurious similarity determinations, and, secondly, the mapping bijectivity is enforced by favouring those mappings with the highest aggregate similarity scores.

6. **Iteration**: Several algorithms perform an iteration over the whole process in order to bootstrap the amount of structural knowledge. Iteration may stop when no new mappings are proposed. QOM iterates to first find mappings based only on lexical knowledge, and then subsequently based on knowledge structures. By performing the computation in several rounds, the algorithm can access the previously computed pairs, and then use more sophisticated structural similarity measures.

The output returned from this process is a mapping table representing the relation $map(O_1, O_2)$ – this defines the ontology mappings comprising the calculated ontology alignment. Due to the aim of increasing mapping efficiency, achieved by limiting and adapting the most computationally expensive feature comparisons, QOM has lower run-time complexity than existing prominent approaches. The complexity of QOM is of $O(n \cdot log(n))$ - (where n is the number of the entities in the ontologies) against $O(n^2)$ for approaches that have similar effective outcomes. This focus upon efficiency, whilst retaining the effective mapping functionality, made the QOM approach a promising starting point for the development of the semantic relatedness determination required within the SERSE architecture.

3.5.5 Semantic Indexing

The purpose of an information retrieval (IR) system is to process files of records and requests for information, and identify and retrieve from the files certain records in response to requests. The retrieval of particular records depends on the similarity between records and the queries, which in turn is measured by comparing the values of certain attributes to records and information requests [132]. In order to be able to compare the similarity between records (resources) and queries, both need to be represented in a compatible way. Compatible representation makes it possible to automate the process of calculating the relevance between queries and resources. The term indexing has been widely used to refer to the process of building such representations. Indexing techniques have been developed in order to make possible the identification of the information content of documents (be they text documents, hypermedia or multimedia ones).

In general, indexes permit the representation of knowledge about a domain in order to facilitate access to information. It simply means pointing to or indicating the content, meaning, purpose and features of messages, texts and documents [1]. Traditional indexing is based on the assignment of semantic labels or more formal typing to authored links [111]. Typically the indexing of a textual document is obtained through the identification of a set of terms or keywords which characterise the document content, that is, terms which describe the topics dealt with in the document. The terms included in this set have not only to be representative of the topics covered in the documents, but they also need to be distinguishing, in that they should make it possible to discriminate one document against the other documents in the collection covering the same or similar topics.

Indexing systems can be categorised along three dimensions [160]:

- index terms are automatically derived or manually assigned;
- index terms belong to a controlled vocabulary or are uncontrolled;
- terms can be combined as ordered strings representing a single concept when indexing (pre-coordinated terms), e.g., Association of Computing Machinery (ACM)[15], or must be post-coordinated on retrieval.

Information retrieval applications concerning textual documents use automatically generated free text index terms (post-coordinated), which are weighted by the statistical frequency of terms in documents and collections. On the other hand, distinguishing features of a semantic index are that semantic relationships exist between controlled index terms, usually (but not necessarily) the result of manual

[15] http://www.acm.org/

cataloguing. Semantically indexed hypermedia links are, by definition, computed corresponding to Intensional-Retrieval links [37]. This allows the possibility of flexible query-based navigation tools.

We will now concentrate on the first type of indexing in the list above, since it is the most commonly used and can allow for all types of resources. Indexing can be either manual or automatic. The former is based on human analysis whereas the latter depends on the use of some type of algorithm, typically machine learning. A study by Anderson and Perez-Carballo [1] has compared the two approaches, taking into consideration the different aspects of indexing and concluded that there is no real motivation to prefer one approach over the other. However, there are some considerations concerning the domain, the type of documents to index and on the number of documents available. Manual indexing can be rather expensive to perform and it might become difficult to perform with very large collections of documents such as those stored in digital libraries. Furthermore, it is not sufficiently flexible to support different indexing strategies. On the other hand, automatic indexing is less expensive to perform and can easily support different indexing strategies, but it might result in less precise indexing, since it is based on some mathematical or statistical formulations and not on a real understanding of the semantics of the terms used for the indexing.

3.6 Summary

This chapter has described a number of areas of research under the umbrella of Semantic Web technologies, primarily knowledge representation and query languages, and semantic similarity and matching approaches. Each of the research areas described have contributed different technologies, approaches and techniques that can be used in the design and construction of a distributed system for semantic content indexing, such as SERSE.

The basis of the entire Semantic Web effort is the development and use of 'standardised' knowledge representation languages, so that encoded knowledge can be easily combined, re-used, etc. The aim is to overcome the limitations of prior knowledge representation systems, that encoded knowledge in differing ways that discouraged inter-operation and re-use of the knowledge. Therefore, a 'layer-cake' of Semantic Web languages have been developed to provide suitable representations of a wide variety of knowledge types. The basic language is RDF, allowing the specification of relationships between entitites. The semantics of these relationships and entities are described in the next layer using OWL to define ontologies of terms and properties, that are then instanciated in knowledge-bases. Further Semantic Web languages provide means to query collections of RDF statements, attach RDF statements

to multimedia resources, express rules that operate over ontology definitions, and handle a wide variety of knowledge representation and manipulation tasks. Widespread adoption of this set of languages and their application to the vast store of implicit knowledge on the WWW has the potential to make available exponentially more semantically annotated resources than any other knowledge representation and storage system, thus making a clear case for the adoption of these languages within SERSE. Furthermore, the rich expressivity and easy knowledge manipulation provided by the languages should aid the complex use of knowledge required within the intended distributed semantic index.

A significant requirement of any knowledge indexing system that aims to operate using multiple, heterogeneous ontologies is the ability to make connections between the terms and properties described in the ontologies on the basis of the similarity (or other mode of connection) of their defined semantics. In the Semantic Web context, this requirement for determining semantic similarity between defined terms has been approached through the process of ontology alignment. The approach involves the determination of pair-wise mappings between ontologically defined entities, by use of the respective properties and relationships defined for each entity. Such mappings can be pre-computed and then stored in libraries of mappings, or determined dynamically at point at which the degree of similarity is required. Therefore, this Semantic Web approach to knowledge representation through multiple, heterogeneous ontologies, and the means to semantically relate this separately defined knowledge, provide the means for a system like SERSE to inter-connect distributed, topic-based indexes using the semantics of their contents. More specifically, efficient ontology alignment systems like QOM provide a clear approach to on-demand determination of semantic similarity between ontological entities, using only the ontology definitions. Finally, the existing work on application of relevant semantic information to the indexing of resources can be adapted to the large-scale Semantic Web domain, with the ontology entities acting as the index terms.

Part II

Scalable Search on the Semantic Web

Chapter 4

SERSE Design

In this chapter we present the conceptual and detailed design of SERSE, *beginning with functional and non-functional requirements for the system – engendered by the intended functions and the system's context within its encompassing Esperonto project. The chapter then focusses upon the system architecture and algorithms that enable it to perform its primary intended distributed semantic indexing task. Section 4.1 presents the intended tasks of* SERSE, *with the overall context of the Esperonto project, within which it was conceived. In section 4.2 we describe the design objectives of the system, and the principles underlying the design approach. We then go on, in Section 4.3, to present the architecture of* SERSE*'s multiagent system, and detail the specialised roles and tasks of the agents within it. Finally, Section 4.4 presents in detail the concept of semantic relatedness, and describes how this was applied as a fundamental component within* SERSE.

4.1 Purpose of SERSE

The intended purpose of SERSE was as a component of the Esperonto (IST-2001-34373) project architecture. The overall project goal was to develop a suite of tools and techniques to 'provide a bridge between the current web and the Semantic Web' [11]. The global architecture of the Esperonto project deliverables is described in the following sub-section. The role of SERSE within this architecture was to provide semantic indexing and retrieval semantically annotated resources. A fundamental intention was that this semantic indexing system should be robust and scalable, in order to handle both the potential volume of SW resources, and deal with the dynamic nature of the web in a fault-tolerant manner. In this context, *scalable* refers to the ability of a system to adapt to increased demands[1]. For example, a scalable network system would be one that can start with just a few nodes but can easily expand to thousands of nodes – the distributed nature of the Domain Name System (DNS) allows it to work efficiently even

[1] http://www.webopedia.com

when all hosts on the Internet are served, and so it is said to "scale well". *Robust* refers to a system that does not break down easily or is not significantly adversely affected by a single application or component failure. Robustness can also refer to a system that holds up well under exceptional circumstances, and is able to recover quickly from the effect of such circumstances. For example, an algorithm is robust if it can continue to operate despite abnormalities in its input, calculations, etc. A further fundamental intention for SERSE was that that this indexing system should leverage the semantics of the resource metadata in order to aggregate resources over specific topic areas. The intended role and functions of the 'semantic indexing and routing system' are fully described in Section 4.1.3.

4.1.1 Esperonto Architecture

The objective of the Esperonto project was to provide a bridge between the current web and the Semantic Web (SW). In spite of the big advantages that the SW promises, its success or failure will – as with the WWW – be determined to a large extent by the availability of content. Currently, there is very little SW content available, since the infrastructure is still being developed (RDFS, OIL, DAML+OIL, OWL, etc.). Apart from the infrastructure, researchers are currently building tools to support semantic annotation of web content. Such tools are important and critical to the success of the SW, but, in general, they have two main limiting characteristics. Firstly, most of them annotate only static pages – they do not consider dynamic content (content generated from databases), known as the 'Deep Web', whose size was estimated in March 2000 to be 400 to 550 times larger than the commonly defined World Wide Web (more than one billion static web pages) [97]. Secondly, many of these tools only focus on the annotation of new web content, i.e., annotating web resources as they are constructed, rather than annotating existing resources.

In light of these problems, the first objective of Esperonto was to construct a service that provides content providers with tools and techniques to publish their (existing and new) content on the SW. Once available on the SW, the content becomes accessible for software agents and other applications like intelligent navigation, visualization, filtering, aggregation, etc. The service – called SEMantic Annotation Service Provider (SemASP) – operates as an ASP (application service provider) with a variety of services aimed at different users. These services include, but are not limited to:

- Manual annotation tools that users can employ to annotate static web documents.
- (Semi)-automatic annotation tools to annotate static, dynamic and multimedia content, as well as web services.
- Ontology construction, selection, browsing, consistency checking and maintenance tools.
- Ontology import and mapping tools.

- Multiple European language support.

The main innovation of the SEMantic Annotation Service Provider with respect to other ongoing SW annotation efforts is that it not only considers static web pages, but also dynamically generated pages. The service also includes multimedia content and multilingual capabilities, providing ontology-based translation services. Besides constructing SemASP, other objectives of the Esperonto project included the exploration and exploitation of innovative knowledge-based services built on top of the basic technology of the SW. This included the semantic indexing and routing system, and innovative visualisation techniques and user interfaces. The rationale for this overall architecture was to provide the SEMantic Annotation Service Provider as a core set of services that covered the essential Semantic Web tasks of ontology construction and maintenance, and resource annotation based upon ontological knowledge models. The implementation of SemASP was further sub-divided into the two self-contained modules: an ontology engineering platform; and a multimedia resource annotation service. The semantic indexing and routing system and the knowledge model visualisation tools where conceptualised as a second layer of functionality that layers on top of the core functionality of SemASP, utilising the products of the ontology engineering and resource annotation modules to perform their own functions.

The technology was demonstrated in three case study pilot applications that use the output of the semantic annotation process to provide innovative services and innovative uses of the SW content in order to illustrate the added value of SW technology. The case study prototypes can be grouped into two application areas:

- **Semantic-based navigation** – Innovative visualization techniques and user interfaces are employed for SW navigation. These techniques for semantic-based navigation were applied to existing content, annotated using SemASP. The resulting application offers a cultural tour through Spanish art and literature.

- **Semantic Web search** – The semantic indexing and routing infrastructure (i.e. SERSE) enables the automatic aggregation of information contained (and annotated) in distributed web resources, and this enables new approaches for intelligent access, navigation, etc. The following pilot applications have been developed on top of the semantic indexing and routing infrastructure:

  - European Fund Finder for R&D. Aggregates information from varied web resources related to funding for R&D in the European Union and specifically in Catalunya.

  - Scientific Discovery. Computer assisted literature search for scientific knowledge discovery. Identifies interesting but previously unknown links between two concepts (for example

a chemical compound and a disease) by providing suggestive juxtapositions of scientific articles. It uses scientific papers in the bio-technology area as the source.

The five main components of the Esperonto system architecture can be seen in Figure 4.1:

- Ontology Server
- Annotation System
- Semantic Indexing
- Visualization
- Multilinguality

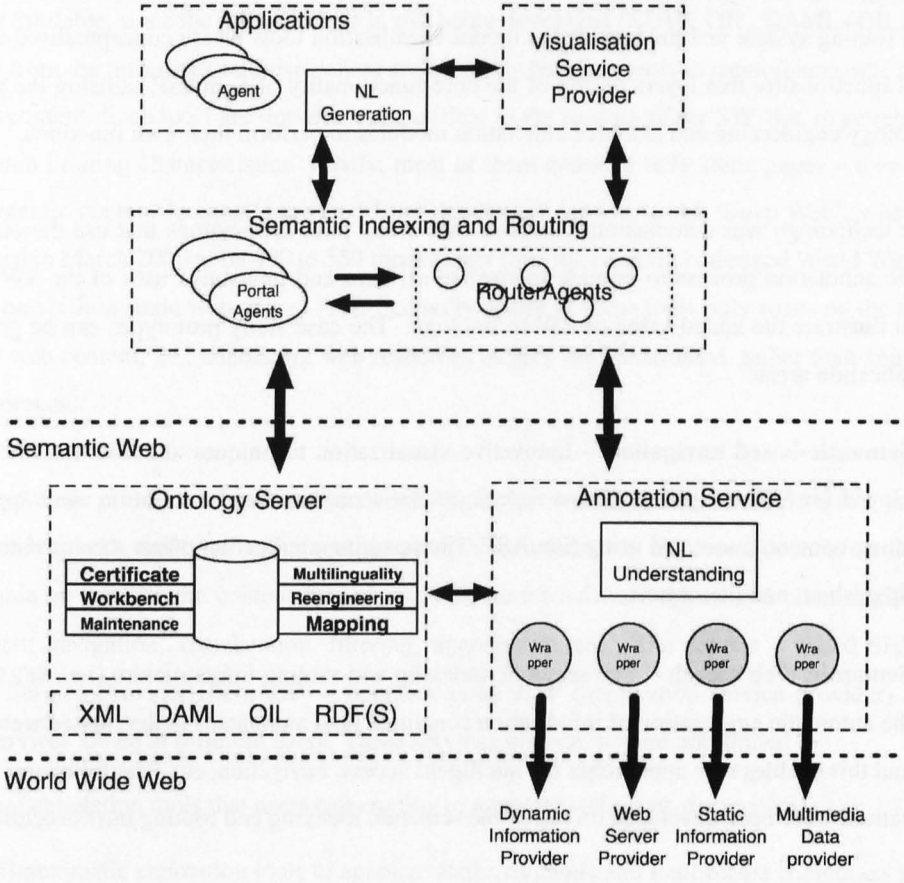


Figure 4.1: Esperonto architecture.

In more detail these five components of the Esperonto architecture can be described as follows:

Ontology Server – Ontologies provide the vocabulary and semantics for annotating Web content; this is the basis for having machine-understandable content. In Esperonto there were two goals concerning ontologies. Firstly, the construction of kernel ontologies to be used as upper level ontologies, as well as domain ontologies for culture (art), funding organizations and biotechnology within the case studies. Secondly, to provide a workbench along with methodological support for the activities of the ontology development process, including learning, construction, evaluation, version control and evolution, alignment, import and export, etc.

In the context of these requirements, an ontology server, named WebODE [4], was developed as a scalable ontological engineering workbench that gives support to most of the ontology development and management activities. WebODE also includes middleware services to aid in the integration of ontologies into real-world applications, as well as rapid development tools for building ontology-based Web portals (ODESeW [31]) and ontology-based knowledge management applications (ODEKM [31]). In addition, WebODE has been created to provide technological support to Methontology [55], a methodology for ontology construction. However, this does not prevent it from being used with other ontology development methodologies, or using no methodological approach at all. In addition, the ontology server was intended to provide the basis for ontology middleware services that allow the easy use and integration of ontology-based technology into existing and future information systems, such as services for ontology library administration and access, ontology upgrading, querying, metrics, etc.

In Esperonto the Ontology Server provides all the ontology management functions within the architecture. Specifically, it provides functions for ontology building (manually or semi-automatically by processes of ontology learning), ontology selection, browsing, consistency checking, maintenance, evolution, importation and mapping.

Annotation System – As described in the previous chapter, the resource annotation on the Semantic Web takes existing content, either structured (e.g., databases), semi-structured (e.g., spreadsheets, dynamically generated HTML pages) or unstructured (e.g., free text, static HTML pages), and provides as output a semantic annotation of that content. The semantics are defined in ontologies, and the annotations provide pointers to these ontologies. Annotation can be performed in several ways, ranging from completely manual, through tool-assisted to fully automatic. The type of annotation approach to be chosen depends on the degree of structure the content exhibits. More structure allows for more automation, while maintaining the quality of the annotations. This diversity in annotation approaches leads to use of different 'wrappers' to provide access to information sources of different types.

A wrapper is a software 'shell' around a source of information that makes its content accessible regardless of its implementation details. That is, the wrapper is able to extract the information content, manipulate it, and then make it available in a standardised form for use by another system. With the growth of the web, wrappers are becoming increasingly important for accessing heterogeneous content. In order to deal with the weak structure web content usually has, wrappers can rely on several techniques, such as heuristics to guess where the necessary information resides, NLP for reducing the ambiguities in the content, Information Retrieval techniques, etc. Wrappers can be constructed in various ways ranging from manual programming to semi-automatic generation. The main problems of wrappers reside in the fact that they are failure prone (simple layout changes of the sources cause problems), and they are hard to validate and to maintain. Wrapper construction is usually considered the main bottleneck. However, experience shows that source identification and wrapper maintenance are equally time-consuming tasks [10].

Esperonto has developed wrapping generation technology for different types of sources and with different degrees of automation. ODEMapster transforms database content into Semantic Web content (i.e., RDF annotated) according to the specified mappings. Knowledge Parser transforms less structured content into Semantic Web content according to both the source description – provided by means of a wrapping ontology – and to the evaluation of a set of ontology population hypothesis with the extracted content. SeMMannot annotates multimedia files by means of the contextual information that can be found close (and possibly attached) to them. ODESWS allows creating Semantic Web Services by means of a user interface that relies on problem solving method modelling. Finally, Esperonto has also created a supervised annotation tool for documents, which provides suggestions about the contents to be annotated and allows users to validate those suggestions or create new annotations.

Semantic Indexing and Routing – This is the Esperonto component that has been implemented as the SERSE system presented in this thesis. Semantic indexes are intended to group Semantic Web content based on particular topics. This is a necessary step to enable applications to aggregate resources in order to provide added value services. Semantic indexes are generated dynamically using ontological information and annotated resources. Within Esperonto the semantic indexing and routing system is in charge of indexing the static and dynamic Semantic Web content generated by the Annotation System, or otherwise available on the Semantic Web.

For the coordination between semantic indexes, a peer-to-peer (P2P) architecture has been explored. This architecture is based on the notion of peers of equal capabilities, each of them knowledgeable about

a 'fragment' of the Semantic Web, and each having an index to access the resources within that fragment. Naturally, the Semantic Web is composed of many of these fragments, and the idea of maintaining a large index of the whole SW is clearly impractical. Therefore, there is a need to decentralise not only the indexing mechanism, but also decentralise the search for indexed resources. Such decentralised organisation is in the spirit of the original WWW.

Visualisation – Due to the massive amount of information and knowledge available to visualize, scalable technology is required for visualising any semantic web content – without requiring manual design and implementation of a specific interface. Three-dimensional and other visualisation techniques have been developed to visualise the content of the semantic web, making a major step forward with respect to traditional site maps that represent link structures. Furthermore, the same visualisation techniques have been applied to the monitoring of the semantic indexing and routing system, showing the activation of nodes and the exchanges of messages between them.

Multilinguality – Multilinguality plays a role at various levels in the project: ontologies, annotations and interfaces. At the ontology level, ontology builders may want to use their native language for ontology construction. Existing multilingual and linguistic resources, such as EuroWordnet[2], have been integrated into the ontology engineering platform (WebODE). The ontology editor gives support for the translation of ontology terms to multiple languages (Spanish, English, and German) and its interface is also available in those languages. Besides this, the platform also allows the storing and retrieving of ontology terms in multiple languages. Manual annotation of content can be performed in various languages, and users from different countries may want to access content in their native languages. The project explored the possibility of (semi)-automatic multilingual annotation of content, basing the work on experiences done with multilingual Information Extraction (IE). IE allows as a side effect of the extraction procedure to provide for domain specific annotations of text corpora. We have generalized this approach to extraction and annotation of content as such. At the user interface level, Esperonto has explored the possibility of generating natural language texts in multiple languages from multilingual ontologies.

Finally, the various primary components of the Esperonto system, Ontology Server, Annotation System and Semantic Index, communicate with each other in a decoupled fashion through use of the Notification Server. The Notification Server uses the Java Messaging Service[3] to maintain a number of publish and subscribe 'topics' that each relate to certain system events, and to which the system components subscribe. System components announce events, such as ontology publication or content

[2]http://www.illc.uva.nl/EuroWordNet/
[3]http://java.sun.com/products/jms/

annotation, to the appropriate topic on the Notification Server. Components subscribing to a specific topic will then receive all topic publications from the Notification Server. In this way, components can publish their events without a priori knowledge of the interested recipients, and can receive publications on topics of interest without any knowledge of how and where these publications are generated.

4.1.2 Non-functional Requirements for SERSE

Before moving on to describe in detail the intended task of SERSE within the context of the Esperonto project, there are a number of non-functional requirements resulting from this context. Firstly, SERSE should integrate with the other software products of the project. That is, SERSE should make use of the other Esperonto components to provide elements of the required functionality. The two primary elements of required functionality are the provision of ontologies and the provision of semantically annotated information resources. Therefore, SERSE is required to utilise the Ontology Server for the provision of domain ontologies, and be capable of reacting appropriately to nofifications from the Ontology Server regarding the publication of new ontologies and regarding the modification of existing ontologies. SERSE is also required to receive notifications from the annotation system regarding newly available semantically annotated web resources, and to use this information to generate and populate the distributed semantic index.

Secondly, SERSE is required to function as a Semantic Web application – using the standard Semantic Web knowledge representation languages of RDF, RDFS and OWL. Specifically SERSE is intended to function as an RDF metadata index, that utilises the semantic knowledge encoded in the OWL entities referenced in the RDF statements. Thirdly, SERSE is required by the aims of the Esperonto project to pursue a distributed indexing approach. That is, the RDF metadata index maintained by SERSE is to be sub-divided amongst a number of nodes within a network. The nodes that manage the index elements should act in manner consistent with them 'knowing' which semantic sub-domains they are responsible for. Furthermore, the organisation of the network of index nodes should make use of the underlying semantics of the contents of the indexes.

Finally, SERSE is required to provide an end-user query interface that enables the submission of queries to SERSE, and consequent display of the (semantically-annotated) resources that SERSE determines to be a match for the query semantics. In addition, this user interface should seek to aid the end-user in the construction of semantically-specified queries, whose underlying representation should utilise a suitable Semantic Web query language. The external interface exposed by SERSE should be able to accept queries from any external application, provided the query is submitted in a specified SW

query language. This end-user interface is one such external application, and the FundFinder business funding search application and the Scientific Discovery biotech literature aggregation application case studies within Esperonto are two others.

4.1.3 Intended Task of SERSE

In order to provide access to annotated web resources, this part of the Esperonto project was concerned with exploring the creation of an infrastructure of semantic indexes and routers for the SW, following a decentralised peer-to-peer (P2P) approach. This infrastructure was intended to enable the automatic aggregation of information contained in semantically annotated web resources. The intended general architecture of the semantic indexes and routers was:

1. The semantic indexes group SW content on particular topics. They are dynamically constructed using the annotations contained in the web resources and the definitions in ontologies of the annotation terms used. The indexes were envisioned as active agents that 'know' what topics they can handle (i.e., find content for) based upon the known ontologies.

2. Queries received by an index for a topic that is not handled by that index are then semantically routed to a 'neighbour' index. In this context the closeness of a neighbour is a function of the semantic relationships between the topics covered by a pair of indexes. Determination of the semantic distance between indexes is based on comparison of the terms defined in their ontologies, and so addresses the issue of semantic heterogeneity between different ontologies. The comparison of these ontology terms utilises techniques for evaluating semantic similarity to identify the neighbouring index that is best able to answer the query.

3. In order to operate within an open and dynamic environment such as the SW the indexes needed to exist in a distributed, decentralised and scalable architecture. Peer-to-peer systems exhibit these qualities [64] and were, therefore, seen as a useful paradigm on which to base the index infrastructure.

In the above description it is indicated that the structure of the distributed semantic index should be influenced by the peer-to-peer paradigm, as such systems provide good prospects for both distribution and scalability. However, this does not conflict with the intention to design SERSE as a multiagent system – a choice determined by the requirements of proactivity, knowledge-level behaviour and fault-tolerance for the system. As described in Section 2.4 previously, multiagent systems and peer-to-peer systems can be viewed as being compatible technologies. The design principles drawn from peer-to-peer approaches and their integration into a multiagent system architecture are fully described in Section 4.2.1. Drawing

on these twin threads of multiagent systems and peer-to-peer systems, the basis of the intended semantic indexing and routing system is of a network of agents, that are each responsible for indexing a subset of the global knowledge of the system. Furthermore, the agents communicate with each other in order to 'route' messages, either notifications of content or queries for content, around the network to those agents best able to handle the message – based upon the semantics of the message content and the knowledge handled by individual agents.

Therefore, within the tasks assigned to it as a part of the Esperonto system architecture, SERSE presents four main use cases. The first, Ontology Publication use case, covers the receipt of notifications of the publication of new ontologies and the incorporation of this new ontological knowledge into the indexing system. The second, Ontology Update use case, covers receiving notifications of ontology modifications, routing these to the appropriate agents and modifying both their content indexes and their inter-connections with other system agents to reflect the change in ontological knowledge. The third, Content Acquisition use case, covers receiving notifications of newly acquired semantic web content, routing these notifications to the appropriate indexes and storing both the metadata annotations and the pointers to these semantically annotated resources. The fourth, Answer Query use case, covers the process of receiving a semantically specified query for semantic web content, semantically routing this query to the appropriate agent(s) and re-aggregating the generated answer sets into a set of pointers to web resources matching the original query.

1) Ontology Publication Use Case

Purpose: Receive an ontology publication notification from the Notification Server, and intialise the ontological knowledge it contains within the indexing system.

Overview: SERSE receives an ontology publication notification message from the Notification Server component of Esperonto. This ontology publication message then needs to be distributed throughout the SERSE network for two key purposes. Firstly, the new ontological knowledge needs to be integrated into the index network, by making the existing index agents in the system aware of the concepts available in the new ontology. Secondly the new ontology needs to be made available for use in the user interface, so that it can be used within the query generation process.

Course of events: Messages are generated by the Notification Server and sent to SERSE, via some suitable asynchronous messaging system, with the purpose of notifying SERSE about the publication of a new ontology. The notification must first be distributed to each of the host systems that is hosting a

part of the distributed SERSE system. This may be achieved by a specialised component that is aware of all host systems within SERSE, that receives all notifications and sends a copy of the notification to a known contact point within each separately hosted fragment of the distributed index. Then upon each of these separately hosted fragments of the index system, the notification needs to be used in two ways. Each of the indexing agents on the host must be made aware of the availability of the entities within the new ontology. This may be achieved by adding the ontology to a list of available ontologies, held in an accessible point on each host, that the individual indexes then query to obtain the list. In addition, each of the components within the SERSE system that provide a user interface need to be made aware of the newly published ontology, thus allowing users to make use of the new ontology definitions within queries for semantically annotated resources.

Exceptional flows: Exceptional flows would occur if the notification message is not received by one of the host systems, or by the any of the intended recipients on any of those systems. Such exceptional flows can be straightforwardly handled using a variety of available message management techniques.

2) Ontology Update Use Case

Purpose: Receive an ontology update notification from the Notification Server, and apply the updates indicated within the message to the appropriate indexes within SERSE.

Overview: SERSE receives an ontology modification notification message from the Notification Server component of Esperonto. The modifications to the ontology entities then need to be notified to the appropriate indexes with the SERSE distributed index. The appropriate indexes are those that index content whose semantic annotations are directly affected by one or more of the notified ontology notifications.

Course of events: Messages are generated by the Notification Server and sent to SERSE, via some suitable asynchronous messaging system, with the purpose of notifying SERSE about one or more modifications to a previously published ontology. The notification must first be distributed to each of the host systems that is hosting a part of the distributed SERSE system – which should be achieved by the same means as in the previous use-case. The multiple ontology modifications within a notification message then need to be made available to each of the indexes on the host system that index content whose semantics are affected by the modifications. The modifications to the ontology entities should then be reflected within the SERSE index – changing, adding and deleting entity definitions – but without discarding information indexed under the previous ontology definition. Such changes should then be reflected in the topographical structure of the interconnections between the distributed indexes, driven

by the notified changes in the defined semantics of the ontological entities. Finally, each of the components within the SERSE system that provide a user interface need to be made aware of the ontology modifications, so that the user interface is utilising up-to-date ontological information.

Exceptional flows: Exceptional flows would occur if the notification message is not received by one of the host systems, or by the any of the intended recipients on any of those systems. Such exceptional flows can be straightforwardly handled using a variety of available message management techniques.

3) Content Acquisition Use Case

Purpose: Receive a content acquisition notification from the Notification Server and incorporate this information into the SERSE index.

Overview: SERSE receives a content acquisition notification message from the Notification Server component of Esperonto. The information about newly acquired semantically annotated resources then needs to be notified to the appropriate indexes with the SERSE distributed index in order to incorporate this new content into the global content index. The appropriate indexes are those that are specialised to handle each of the ontology entities referred to in the notified resources semantic annotations.

Course of events: Messages are generated by the Notification Server and sent to SERSE, via some suitable asynchronous messaging system, with the purpose of notifying SERSE about one or more online resources that have been newly acquired (i.e. appropriately semantically annotated) by the Annotation System component of Esperonto. The notification must first be distributed to each of the host systems that is hosting a part of the distributed SERSE system – which should be achieved by the same means as in the previous use-cases. The multiple annotated resources within a notification message then need to be made available to each of the indexes (on each host system) that are specialised to handle the ontology entities referred to in the semantic annotations attached to those resources. Once the message is received by the indexes that handle the ontology entities referred to in the notification, they should then update their content indexes by adding, deleting and modifying entries in the index, according to the information contained to in the notification message. If the content acquisition notification relates to an ontology entity that is not handled by any existing and operational index in the SERSE network, the system should act in order to assign responsibility for the new entity to a (newly created or existing) index within SERSE.

Exceptional flows: Exceptional flows would occur if the notification message is not received by one of the host systems, or by the any of the intended recipients on any of those systems. Such exceptional flows can be straightforwardly handled using a variety of available message management techniques.

4) Answer Query Use Case

Purpose: To pose a semantically specified query to SERSE which then generates a reply consisting of a list of semantically annotated resources whose annotations match the constraints imposed by the terms of the query.

Overview: The system receives a message from an external application that contains semantically specified query for annotated resources. SERSE should respond to the query with a single reply, containing pointers to all those resources indexed within the distributed system that are a match for the query. In order to do this the query needs to be directed to the most suitable of the index fragments within SERSE in order to answer the query. The most suitable indexes are those that contain resources whose metadata annotations may be a match for the constraints expressed in the query. All resources returned from the distributed indexes should be formed into single reply message, which is then sent to the external application that posed the query.

Course of events: An external application generates a query for semantically annotated resources, and submits this query in a suitable message format to an externally visible contact point within the SERSE system. The query itself should be expressed expressed in a pre-defined semantic query language intended to operate over collections of semantic metadata, such as that indexed by SERSE. Once received by SERSE the system should reply as if it were a single content index. That is, the distributed nature of the resource index implementation should not affect which resources are contained in the generated reply. Therefore, the query must be directed to each of the indexes (on each of the host systems in SERSE) that are specialised to handle the ontology entities, that are referred to in the semantic constraints expressed within the query. Once the message is received by the indexes that handle the ontology entities referred to in the query, they each should then retrieve the resources from their fragment of the global content index that are a match for the query. All the resources returned from all of the queried indexes should then be aggregated into a single reply message, and this aggregate reply is then returned to the application that posed the query to SERSE.

Exceptional flows: Exceptional flows would occur if the notification message is not received by one of the host systems, or by the any of the intended recipients on any of those systems. Such exceptional

flows can be straightforwardly handled using a variety of available message management techniques.

4.1.4 Functional Requirements for SERSE

On the basis of the non-functional requirements identified in Section 4.1.2 and the four primary use-cases detailed above, a number of key functional requirements can be identified, that are necessary for SERSE to be able to fulfill its intended purpose:

1. Scalable network of limited-knowledge domain indexes – the basis of the indexing system is to sub-divide the global resource index into a number of sub-indexes that are each responsible for indexing resources falling within a limited domain of knowledge. The stated intention is to implement each of the managing entities of the indexes as autonomous agents. Furthermore, each of the agents will be linked to other agents, forming a network of index nodes. However, each agent will only have knowledge of a sub-set of the other index agents, so that the number of interconnections each agent has to maintain remains manageable – underlying the scalability of the system. Finally, the inter-connections between the index agents should be on the basis of the semantic 'closeness' of the respective limited domains of knowledge of the agents involved.

2. Inter-index communication – all of the agents in the index network are potentially able to communicate with all other agents, but agents can only send messages directly to the sub-set of the other index agents they 'know' about. This then leads to a requirement for messages to be 'routed' via a number of index agents from their source to their target, potentially traversing the entire index agent network.

3. Semantic similarity function – a key requirement of the system is to have a means to determine the degree of semantic similarity between ontologically defined concepts. The degree of similarity between concepts is need for both the formation of index inter-connections on the basis of their semantic proximity, and for the index agents to determine which of other index agent they know of is the most appropriate recipient of the semantics expressed in a message that is being routed through the network.

4. Robust system-level functioning – the system as whole should be robust and fault-tolerant, that is, able to continue system-level functioning despite the failure of any of the index agents, and attempt to recover from any such failure. The sub-division of the global resource index between a large number of index agents supports this aim, as temporary loss of a small fraction of the global index will not significantly hinder overall system performance – only queries directly addressing the knowledge domain managed by the absent index agent will not be fully answered. The index

agents themselves can support the fault-tolerance of the system, through the ability to re-structure their inter-connections when other index agents are absent.

Thus, drawing together these functional requirements, SERSE is designed as a multiagent system composed of specialised agents capable of functioning in a scalable, self-managing, open, and dynamic fashion. The core of the system is represented by a network of specialised agents providing *indexing* and *routing* functionalites, that permit them to efficiently retrieve resources based on the semantics of their content. Each agent is *specialised* with respect to a single concept, meaning that it can access the resources whose annotations contain instances of that concept, and it is only aware of those agents specialised with concepts that are semantically *related* to its own. The decision to base the system architecture upon one index agent being responsible for indexing a single concept was taken for a number of reasons – the primary ones being:

- The sub-division of the global resource index has to be performed at some level of granularity. An initial consideration was for individual indexes to aggregate over a number of very similar concepts. However, the concepts handled by a single index would then have to vary as new concepts became known to the system – meaning that the aggregate semantics of an index would not be fixed. The end result of this would be that index agents would have to continually re-calculate their relative semantics and re-adjust their inter-connections appropriately. Selecting the lowest possible sub-division granularity of one concept per index retains the possibility of concept aggregation in the query process, whilst keeping the semantics of each agent, and thus their inter-connections, fixed (and not requiring computationally expensive continual re-adjustment).

- Reducing the granularity of the index sub-division to the single concept level has significant benefits to the semantic similarity function that the system requires. That is, the function can be built upon the basis of a comparison of two individual concepts – whilst accounting for the semantics provided by the respective ontological contexts. If a coarser granularity were selected, with an individual agent managing more than one concept, the semantic similarity function would need to be able to determine similarity between two sets of composite semantics. This would be far more computationally expensive, and would present significant semantic aggregation problems, when compared to the single concept granularity function.

- Sub-dividing the index amongst the maximum possible number of index agents, without sub-dividing the metadata referencing a single concept, helps to support the system robustness. As explained above, the more index agents hosting the system's knowledge, the smaller part of that knowledge is lost by the failure of a single index agent. Furthermore, keeping individual indexes as small as possible (whilst keeping index inter-communication under control) supports the

scalability of the system, by enabling the indexing system to be distributed over as many host computers as there are concepts contained in the global index.

The inter-connections between semantically similar index concepts means that index agent network is organised into *semantic neighbourhoods* that mirror the structures of the known ontologies (in terms of the hierarchical and other specific relationships defined in these ontologies). That is, more formally, if we consider an ontology as a directed, labelled graph *G=(N,E)*, where *N* is a finite set of labelled nodes, each corresponding to a concept in the ontology, and *E* is a finite set of labelled edges, then the topology of the network of routers is determined by the structure of *G* (that is by the semantic relations between the nodes), and there is a one to one correspondence between the nodes *N* and the set of agents composing the nework *A*. For each agent in the network a_i, following [69], we can define a function $knows \subseteq A \times A$, such that $knows(a_i, a_j)$, if there is an edge e_k between the concept n_x of the agent a_i, and the concept n_y of the agent a_j, or, more informally, if there is a relationship linking n_x to n_y. The function *knows* is symmetric, thus $knows(a_i, a_j) = knows(a_j, a_i)$. The neighbourhood of an agent a_i is then given by the set $Neighbour_{a_i} = \{a_j \in A | knows(a_i, a_j)\}$.

Neighbourhoods are partially overlapping, and this permits the routing mechanism to find the answer to a query in a limited number of hops, without having to browse through all of the known ontologies and without having to flood the network with a large number of messages (see explanation in Section 5.2). Semantic neighbourhoods are automatically determined when the system receives a notification of new ontological content – received as new concepts are used to annotate resources. The neighbourhoods are not static but they dynamically change as the system is required to handle further notification of new ontological content, or if an ontology is modified (and a new version of the ontology is used in the annotation). In this way, we have multiple overlapping neighbourhoods, each centred on one concept, and agents have knowledge only of the agents composing their neighbourhood.

Indexing ontological content consists of creating structures that link resources, identified through their URLs, to RDF statements describing instances of the concepts in the ontologies. The routing functionality permits SERSE to route queries to the index agents that are specialised to retrieve resources annotated with metadata that references particular concepts. SERSE handles queries expressed in RDQL (see Section 3.4) on any combination of concepts and concepts' properties. Complex queries are decomposed into simple ones, that each query over a single concept. Each simple query is routed to one of the agents in the network of routers, and the agent consults its index to determine whether it can answer the query. If the agent cannot answer the query, then it routes the request to that agent within its neighbourhood that handles the concept *closest* to the one in the query. This next agent then performs

the same task – answering the query if it is able, and if it cannot, routing it onto its neighbour (not including the previous agent) handling the concept most closely related to the queried concept. This process thus iteratively moves the message from neighbour to neighbour, with each step moving the message semantically closer to its target.

This simple approach to the semantic routing process was later amended and elaborated, following early evaluation of its functioning. When the network of index agents became sufficiently large (the specific number being entirely dependent on the semantic granularity of the ontologies used) it became possible for messages to become 'lost' in a section of the network that was significantly semantically dissimilar from the message concept. The problem was that the agents became unable to determine suitable neighbours to whom to route messages, as all their neighbours handled concepts that were not at all semantically similar to the target concept. Clearly, the semantic similarity function could only function when there was some degree of similarity to evaluate. When all available neighbouring concepts within five or more hops had no similarity with the message concept the message could not be routed along a definite semantic direction across the agent network, and often began to 'wander aimlessly'. Therefore, a decision was taken to adopt a two-tier semantic routing system whereby messages would be initially routed, by specialised system elements, to the correct general semantic 'area' of the index agent network. This approach draws upon P2P super-peer hybrid architectures – as described in Section 2.4.

These specialised elements would, like the index agents, only have knowledge of a sub-set of the index agent network (to support system scalability). The approach calls upon these elements, that offer a 'portal' into different areas of the semantic network, to be linked to a set of 'significant points' within the index agent network. Such 'significant points' could be selected on the basis of many index agent features, such as number of neighbours, local network density, total message traffic, etc., but would need to be fairly evenly distributed throughout the network to fulfill their 'portal' function. The selection of which index agent (that is listed as a 'significant point') to route a message to is made, as with the other index agents, by use of the semantic similarity function within SERSE. The detailed functioning of these portal elements and the implemented 'significant point' selection criteria are described in Section 5.2.

SERSE evaluates similarity between concepts according to the method proposed in Ehrig and Staab's Quick Ontology Mapping (QOM) approach [40]. However, the algorithm has been modified so that it exhibits a task-specialised behaviour, implemented through heuristics, that provides a higher number of potential matches for a given concept. Ehrig and Staab's approach is aimed at ontology mapping, a process that can be taken off line and requires high precision in order to establish the correct mappings. Semantic routing is different in nature: the evaluation of similarity should be sufficiently precise

to determine a new agent to whom the query can be routed, not necessarily the very *best* agent available. In addition, semantic routing is a dynamic process executed on-line, and therefore it requires fast computation in order to minimise the time spent by the user waiting for an answer to a query. The semantic routing process and the algorithms underlying the semantic relatedness metric are described in Section 4.4.

In addition to the main indexing and routing facilities, the system is also intended to be robust and self-maintaining. SERSE uses autonomic computing techniques to preserve index knowledge and to adjust the index connections when one or more indices within the system are unavailable. Autonomic behaviour is also used to ensure the system remains operative in case of failure of one or more agent or one platform. Section 4.2.3 describes the mechanisms used to implement autonomic behaviour in SERSE.

4.2 Design Objectives

Utilising the desiderata for searching the Semantic Web, presented in Section 1.2, a number of required features can be identified that SERSE should exhibit:

1. *Decentralisation*: Efficient navigation toward a specific topic depends on the ability to identify the 'right direction' in which to travel. A centralised approach would imply maintaining a directory of all topics, acting as a centralised server for queries. However, this can prove too inefficient and cumbersome for the SW [22]. An alternative is to have system components each with equal roles and the capability to exchange knowledge and services directly with each other. Peer-to-peer technology such as Edutella [113] or Morpheus[4] is a possible answer to this quest for decentralisation. P2P systems are networks of peers with equal roles and capabilities, and recently peer-based management systems have been proposed, which exploit P2P technology for sharing and retrieving huge amounts of data [70].

2. *Openness*: Openness is inherent in both the syntactic and the Semantic Web. In the case of the SW, openness is enhanced, because ubiquitous software components – from computers to mobile phones and TVs – are involved in the process of using it [151].

3. *Autonomy and social ability*: P2P systems are usually comprised of simple data-storing systems, and the sharing and retrieval of data is performed on the basis of a central directory, while communication among peers simply relies on raw TCP/IP protocols. However, we require that our system components

[4] http://www.morpheus.com/

be able to interact with users and other peers, and decide which peer supervises the 'fragment of knowledge' closest to their own one. This implies that our components must exhibit autonomy and social ability: characteristic features of multi-agent systems [170].

4. *Scalability*: As with the Internet itself, the SW is expected to grow exponentially [151]. Scalability is thus critical: the system must adapt flexibly and dynamically to the highly dynamic nature of the SW. This implies that the system must be able to incorporate and integrate new knowledge as it becomes available.

5. *Semantic based content retrieval*: Content retrieval can exploit the ontology-based annotation of the SW. System components must exhibit the ability to understand and process ontological descriptions. When knowledge is factored into fragments, navigation is determined by means of the ontological descriptions of the concepts to search. Semantics is used to establish the right path to a neighbourhood, by evaluating the relatedness between concepts (as described in Section 4.4).

A key issue when searching annotated content in such an architecture is to determine the approximate size of a neighbourhood. This has implications for the topology of the network of peers – in particular, on the level of centralisation – and on the overlay networks, on the performance of the system, and on scalability. There is clearly a spectrum, where at one extreme a small neighbourhood means a search component has only knowledge local to few concepts of the ontology(ies), and at the other extreme, there is global knowledge (every search component has knowledge of the whole ontology or ontologies used to annotate the instances to search). Local knowledge favours a purely decentralised approach, but gives little help in determining the right message routing direction, while global knowledge creates a single point of failure that affects the performance of the system [133].

In the following three sub-sections we will examine in further detail the way in which the design of SERSE seeks to address the objectives of scalability, semantic-based information retrieval, and system robustness.

4.2.1 Scalability

Scalability is a fundamental feature of the design of SERSE, intended to enable the system to handle both the size and growth of the Semantic Web, and the dynamic nature of the information contained within it. As indicated earlier, a potential system architecture that would support such scalability is that of peer-to-peer (P2P). Within the SERSE design the P2P architecture adopted represents a hybrid between super-peer indexing and individual peer indexing – the index agents represent the individual

nodes within the P2P network, and these agents handle and maintain the index of content – see the description of `RouterAgents` in Section 4.3 below. The super-peers are utilised to enhance the efficiency and effectiveness of the semantic message routing process between peers by providing direct access into different semantic neighbourhoods (clusters of related terms) within the P2P network – see the description of `PortalAgents` in Section 4.3 below.

In the design of SERSE the concept of routing indexes has been adapted to enable the index agents to manage their 'semantic neighbourhoods'. Each of the index agents has a routing index that contains information about those other index agents that manage concepts that are 'closely related' to its own concept. However, this routing index describes only immediate neighbours and does not include information from 'over the horizon'. This is because of the different organisation of the information by the peers within the system. Within the system proposed in [33], different peers can index documents belonging to the same topic, and so the routing index enables a peer to determine which of its neighbours leads towards other peers that index higher number of documents within a topic. In SERSE each of the index agents provides indexing information for a specified topic, and does not index resources that do not pertain to that topic. Furthermore, each index agent's assigned topic is unique within the system, so a search for a specific topic within the system is a search for a specific index agent[5]. Finally, the semantic message routing process within SERSE enables the index agents to determine the 'best' routing direction by indicating to what extent neighbours' topics are related to that in the message by use of the 'relatedness metric'. This allows the index agents to determine a direction based upon which of its neighbours' topics are 'closest' to the topic sought. The semantic relatedness metric is based upon well established ontology alignment techniques, and is described in detail in Section 4.4.

The application of Semantic Overlay Networks to the intended multiagent indexing and routing system again required some adaptation. In the system described in [34], individual peers can have heterogeneous content, and content for a single topic can be spread over many nodes. Conversely, in SERSE individual agents (nodes) within the system contain homogeneous content – referring to a single topic only. Therefore, SONs of the type described are not required, as each 'bucket' would contain only a single node, and, furthermore, in our case the classification hierarchy is already determined by the ontological relationships and cannot be adapted to 'fit' the criteria proposed. However, the idea of SONs remains applicable to our design, as the interconnections between the indexing agents are determined on the basis of semantically defined relationships in the underlying ontologies. So, these ontologically-based interconnections themselves represent a type of Semantic Overlay Network – connecting nodes with 'closely related' topics, and enabling messages to move between nodes on the basis of the semantic

[5]Can be seen as conflicting with the need for fault tolerance, but this is addressed by the autonomic system features described in Section 4.2.3

interconnections.

The application of these peer-to-peer approaches to our system, and their adaptation to the particular requirements of it, form the scalable basis of SERSE. This distributed organisation of the index knowledge, with individual agents being responsible for a specified sub-set of the system knowledge, provides a means to extend the system simply by the addition of agents to handle new areas of knowledge. Furthermore, the sub-division of the agent network into semantic neighbourhoods means that any additions to the system can be incorporated into the semantic overlay network on a 'local' basis, only requiring action from those agents identified as semantic neighbours of the new agent. Thus, the overall indexing system is distributed over a number of nodes, that themselves may be distributed over multiple host machines, and communication in the system only occurs on a point-to-point basis – underlying the scalability of the system as a whole, as demonstrated by Experiments 2 and 3 in Chapter 6.

4.2.2 Semantics-Based Resource Retrieval

As a Semantic Web application, SERSE is intended to utilise and leverage the knowledge encoded into the ontologies, annotations and queries to retrieve web resources. This is achieved in two main ways:

1. Firstly, the ontological definitions are utilised to generate the interconnections between the indexing agents, and to determine how messages are moved around the network in order to locate the index agent most suitable to answer a query.

2. Secondly, the queries within the system are 'semantically specified'. That is, the queries are encoded in RDQL and specify which resources are to be matched on the basis of the metadata annotations attached to them. Thus, once a query has been routed to the appropriate index agent, it is semantically matched against the the RDF knowledge base maintained by that index agent to identify which metadata annotations match the metadata template defined in the query. Once the metadata annotations have been identified the index agent then consults its content index to determine the URLs of those resources annotated which the identified metadata. The reply to the query is formed the by the URLs of this set of resources.

These twin approaches of semantic message routing within the agent network, coupled with semantic query answering by the individual agents, enables SERSE to retrieve resources purely on the basis of the semantics of their annotations.

4.2.3 Robustness

The multiagent system approach to indexing Semantic Web content is intended to be more robust than a centralised system. This implies that the system will continue to operate when existing index agents become unavailable – either through an individual agent developing a fault, a network communication error, or through one of the platforms hosting a part of the index system being shut-down. This will require the agents to have a number of behaviours that prevent the agent network as a whole being adversely affected by the state of any of its individual components. SERSE has been designed to autonomously react to both normal and exceptional events, such as the notification of a new ontology, or the failure of an indexing agent. This behaviour is motivated by the need to have a system that operates in an open and dynamic environment, such as the Semantic Web, and so must be scalable, robust, and require limited human intervention for its functioning. For this reason, SERSE has been designed as a multi-agent system in which agents can join and leave the network without any need to take (any part of) the system off-line, and without degrading the performance of the system as a whole.

Autonomic computing is an emerging branch of software engineering that advocates the design and implementation of self-managing systems, consisting of several interacting, autonomous components that in turn comprise large numbers of interacting, autonomous, self-governing components at the next level down [86]. Agents provide a ready vehicle for such techniques, in that they are already characterised as autonomous and self-governing, and are able to exhibit proactive behaviour to achieve the type of individual and cooperative self-management envisioned in autonomic systems. Autonomic behaviour in SERSE supervises two main functionalities: dynamic management of the network of index agents, and management of failures within the network. Index agent management primarily consists of on-demand creation of index agents, and pro-active maintenance of the neighbourhood connections between them. This management is designed to handle the construction of the index agent network from the initial notification of the first available ontology – building the system from scratch. Failure management consists of enabling the system to continue to operate despite the temporary or permanent loss of index agents, or even whole index agent platforms, from an existing network. Such failure management includes equipping the agents with the abilities to recover from failure to a previously saved state, thus prevent loss of indexing data held at each agent. Several autonomic systems mechanisms provide this dynamic management within SERSE:

Agent network creation: The creation of index agents and population of the agent network is triggered by the notification of new ontologies (in the case of ontology root concepts), and of annotated resources (when these regard a concept not previously known to the system). On receipt of a notification regarding publication of a new annotation ontology, the root concept(s) of the ontology are determined and a

creation request message is generated for each of these concepts, and is then sent to the platform management agent, that in turn, creates a index agent for each root concept. On receipt of a notification regarding a concept for which an index agent has not yet been created, the message will be semantically routed though the network until it reaches the 'closest' existing index agent for the message concept. This index agent will determine from its routing index that the index agent for the sought concept does not exist, and will then send a creation request message to the platform management agent for that concept. The platform management agent creates a new index agent for the concept, populating its content index with the content of the original notification message. Each of the index agents that handle concepts neighbouring this new one will then be contacted by the new agent, that has autonomously determined its appropriate semantic neighbourhood, and they will then update their routing indexes with this new neighbourhood knowledge.

Heartbeat monitor: A heartbeat monitor in an autonomic system is intended to enable components to know when associated components are available for communication. This is achieved by components periodically sending 'ping' messages to each other in order to determine their state. However, rather than having a dedicated heartbeat message system in place, which would impose a high message transport cost on the agent system, the existing system messages are used by the index agents to monitor the state of neighbouring agents. The monitoring is performed by maintaining a field in the routing index table that records the recent message transmission success history for each of the neighbours. In the situation where messages have not been attempted between two neighbours for a specified time period, a specific heartbeat message is employed to determine the neighbour's state – in this way the heartbeat monitoring is achieved by the normal system messages where possible, and falls back on a periodic heartbeat message when required. When messages are not received successfully the neighbour's routing index entry is first set to a warning level, and if failure continues for a short time the entry is marked as unavailable. The neighbour will become available again if a message is received from it within a specified time period. Once a neighbour has been un-contactable for this specified period, it is considered to be permanently unavailable, and the index agent(s) neighbouring this agent then update their routing indexes accordingly.

Shutdown procedure: Individual index agents within the network may be shut-down by the Agent Management System (AMS) of the platform hosting the agent, due to an internal error with that agent. If agent shutdown is instructed by the AMS, the index agent then performs a controlled shutdown procedure. In this procedure the index agent first backs up its index knowledge to permanent storage – saving all knowledge in both the content and routing indexes. The agent then sends a set of messages, one to each of its neighbours, informing them of its shutdown and subsequent unavailability. This enables

the neighbours to adapt their neighbourhood connections to reflect the loss of this agent[6]. A further shutdown message is sent to the platform management agent within SERSE, notifying the shutdown and providing a reference to the saved state of the index agent concerned. The platform management agent will the create a new index agent to handle the knowledge fragment managed by the expired agent – passing in the saved state knowledge to intialise the new index agent (effectively restoring the original agent). It may also be the case that an entire agent platform hosting a part of the SERSE system is subject to a controlled shut-down, due, for example, to system maintenance or the host machine failing. In this case all the index agents on the platform perform the controlled shut-down procedure, saving their knowledge and informing their neighbours (though with platform shut-down, the agents only send these messages to those neighbours on different platforms). Recovery from platform shut-down is initially a manual process to re-boot the platform itself and restart the SERSE platform management agent. However, once started the platform management agent will detect the saved-states of the index agents and restore these agents on the platform.

Index backup and recovery: Index agents periodically save their content and routing index knowledge into backup storage, which enables the recovery of knowledge following failure of the agent or of its platform. The location of this back-up file is shared by all agents on any one agent platform, and is pre-defined for each computer system hosting an agent platform forming part of SERSE. The actual location is specified at the point of manual initiation of each platform, so that a suitable location can be chosen dependent on the configuration of each hardware and software environment. The period of the back-up can be adjusted to suit the system usage, size of agents' indexes, available backup storage, etc.. Each successive back-up overwrites the previously saved state, though it would also be possible to enable the storage of multiple previous states. This behaviour operates in the same way as with the controlled shutdown procedure, and recovery from either individual agent or platform failure is also managed by the platform management agent. The behaviours differ in that the controlled shut-down triggers an immediate backup, ensuring no index knowledge is lost, whereas in this case the re-created index agent(s) will have a older saved state restored. Therefore, in case of uncontrolled failure and shutdown of part of the system, some index knowledge may be lost due to unrecorded updates.

4.3 SERSE Architecture

The requirements and goals identified previously have motivated the design of the semantic indexing and routing system. SERSE is implemented as a multi-agent system, whose index / router agents share a core of equal capabilities and responsibilities, and which are capable of retrieving web resources based

[6]Non-reciept of such messages simply means agents are not warned of the unavailability, but are still able to handle it through the heartbeat message system previously described

on the semantics of their annotations. The system is internally organised in a peer-to-peer fashion: each agent can communicate with its immediate neighbours, and the neighbourhood population and size is determined by the semantic proximity between the concepts known to the agents. No agent can broadcast messages to the whole system, and no agent has global knowledge of the network: this ensures decentralisation.

SERSE was implemented using JADE [9], a FIPA-compliant middleware platform. JADE is used to handle the transport aspects of agent communication: our implementation builds on JADE to provide a semantic overlay network, i.e., the *logical* organisation of the agents in a network of peers, which is based on the notion of semantic neighbourhood. Agents in SERSE have knowledge of a number of concepts forming the ontologies, that are expressed in OWL and stored in some ontology server. Agents have the ability to send FIPA messages to the agents belonging to their immediate semantic neighbourhood. Although limited, these 'social abilities' permit the agents to autonomously and dynamically determine the most appropriate index agent, i.e., the agent that can retrieve instances of a concept that is identical or semantically closest to the queried concept, and to route them an unanswered query.

4.3.1 Multi-platform Approach

SERSE's ability to retrieve annotated content is based on *distributed index structures* and *semantic routing mechanisms*. These functionalites are provided by a network of specialised indexing agents that allow the system to efficiently retrieve resources based on the semantics of their content. Semantic routing of messages between the agents permits SERSE to route queries through the network to the most appropriate agents, i.e., those that are capable of retrieving the resources annotated with instances of the concepts used in the query. Figure 4.2 illustrates the different roles played by the agents in SERSE and the primary message flows within the system.

Scalability of the indexing system is achieved through distribution: SERSE is designed to be distributed over a number of JADE agent platforms, on different host machines, with each platform containing part of the network of index agents and its own set of interface agents. This enables the system to operate even when reduced to one platform, as each platform in the system provides the necessary points of entry to the system. It also dynamically reconfigures the network of agents in response to temporary or permanent unavailability of individual agents, or whole platforms within the system. Figure 4.3 shows the interactions between the different types of agents on multiple platforms.

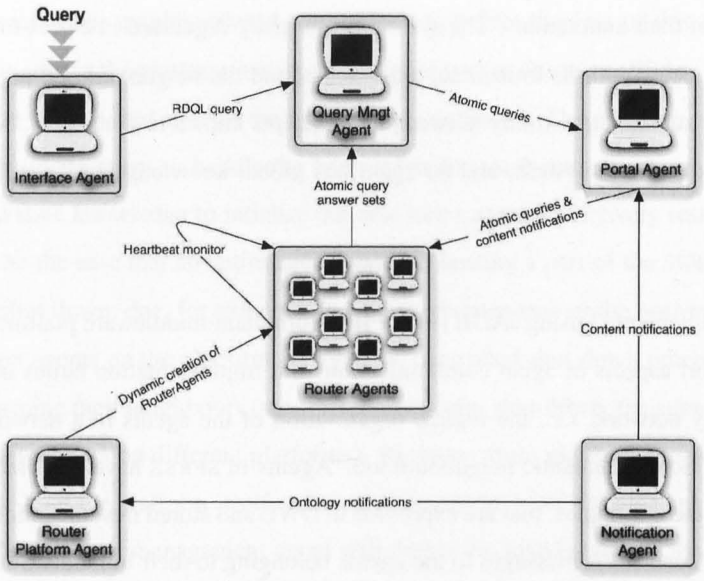


Figure 4.2: SERSE conceptual architecture.

The basis for new platforms being created and joining an existing SERSE system is that an administrator for the host system sets up the platform (giving web server and notification mediator addresses). The new platform then connects itself to the notification mediator and web server. It will now be available for the mediator to send notifications to, which will lead to index agents being created on the platform – so the platform now hosts part of the indexing network. The policy of multi-platform distribution is applied by the notification mediator – as it is here that the selection is made of which available platform to send a notification. The current policy is to apply a simple round-robin approach, with each of the n available platforms receiving $1/n$ of the total notifications. This approach means that the system is self-distributing, making use of all available platforms.

4.3.2 Agent Types

SERSE is implemented as a multi-agent system composed of six main types of specialised agents, each playing different roles in the provision of management, semantic indexing and routing services. The network of agents that perform the semantic indexing and routing constitute the core of the architecture, by each handling one concept out of all those known to the system. In addition, five other types of specialised agents provide various ancillary services, including interfacing with external systems, and system management functions. The different roles that these agents play in SERSE are described below.

Router Agent (RA): `RouterAgents` form the core of SERSE, by creating a highly interconnected network of distributed indexes – previously referred to as 'indexing agents'. These agents provide the

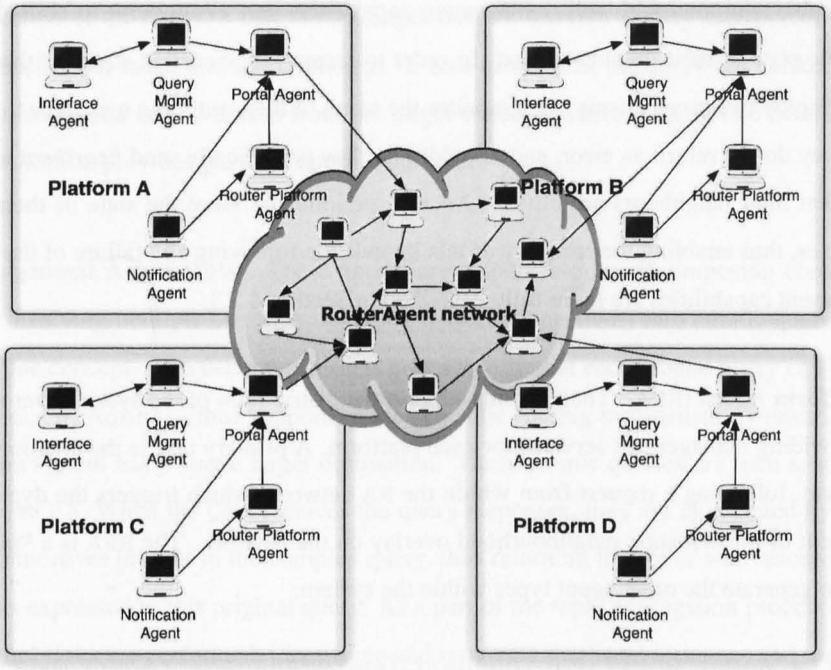


Figure 4.3: SERSE architecture distributed over multiple platforms.

semantic indexing and message routing, and some of the self-management functionalities of the system. RAs can be distributed over multiple agent platforms, while the other agent types described in the remainder of this section are replicated for each of the platforms within the system. The RAs maintain two types of index: a *content index* and a *routing index*. The content index stores the identifiers for the RDF metadata of an annotation, together with the URLs of the resources that have been annotated with it. The routing index stores the communications address for each of the RAs that are in the agent's semantic neighbourhood, along with the concept handled by each of these RAs and the semantic distance metric score between the concepts. In order to ensure that the routing process will perform correctly, despite the absence of RAs from the network (due to neighbouring concepts in the ontology(s) not yet having reported index content, and so the corresponding RA has not yet been created), routing indexes store neighbour entries of three types: *actual*: neighbouring concepts that are handled by existing RAs that this RA is aware of; *ontology*: neighbouring concepts (according to the ontology(s) known to the RA) for which no RA currently exists; and *implied*: concepts from outside of the normal neighbourhood for this concept, that should be indirectly linked except for the absence of one or more intervening RAs - providing a 'bridge' over gaps in the network. Actual neighbours represent the normal semantic links in the network, whilst implied and ontology neighbours enable messages to be routed even if the network of RAs is not fully populated. Through this sub-division of the network into semantic neighbourhoods, each RA is responsible for a sub-set of the total system knowledge, having only part of the global content index and only localised knowledge about its own semantic neighbours.

RAs are also equipped with self-management capabilities that allow them to actively respond to changes in the state of their neighbourhood. In order to ascertain the current status of their neighbourhood, RAs employ two mechanisms: they monitor the result of their outgoing messages to other RAs to verify that they do not return an error, and, in addition, they periodically send *heartbeat messages* [86] that verify that their neighbours are alive. RAs also periodically store the state of their content and routing indexes, thus enabling the recovery of this knowledge following any failure of the agent. These self-management capabilities are more fully described in Section 4.2.3.

Router Platform Agent (RPA): These agents support the distribution of the system over multiple platforms by providing management services for each platform. A primary task is the creation of new RAs on its platform, following a request from within the RA network, which triggers the dynamic creation and adjustment of the semantic neighbourhood overlay on the network. The RPA is a "factory" agent, that is able to generate the other agent types within the system:

- A `PortalAgent`, `NotificationAgent`, `QueryManagementAgent` and `InterfaceAgent` for the platform on which the `RouterPlatformAgent` is initialised.
- `RouterAgents` – in response to requests from `RouterAgents` as part of the content notification mechanism (see Section 5.4.2).

The RPA is also intended to provide the initial conduit for cross-platform messages, allowing RAs and PAs to discover neighbours with whom they can subsequently communicate directly. This is the only agent type within the system that has access to the central registry of router system platforms, therefore enabling messages to be sent to any of these platforms. This agent receives messages from the PA in its own platform, re-sends the message to the RPA on another platform, and this in turn passes the message on to its own PA.

Portal Agents (PA): These agents act as the gateway into the RA network – all atomic notification and query messages are passed through them, in order to get the messages into the correct approximate semantic 'area' within the RA network. Each PA maintains a list of *significant points* within the RA network, and send messages into the network by initially routing them to the most appropriate of these points using the semantic routing mechanism. The selection of such significant points could be based upon many different criteria – such as density of neighbour connections, volume of content, and message throughput. In SERSE these points are currently implemented as the set of root nodes (i.e., all concepts within the ontology that have no ancestor other than OWL:Thing) of all the ontologies currently known to the system. The PA forms an important element of the semantic routing system, by enabling messages to be sent to RAs that are in the same general semantic 'area' as the message concept. This reduces the

size of the section of the network that the messages have to traverse as they are routed, thus making the routing mechanism faster and more efficient. It also means that messages are unlikely to become 'lost' in a semantic area very different from the target concept, where it cannot be determined which neighbour connection provides the most appropriate route.

Query Management Agent (QMA): These agents are responsible for decomposing complex queries (i.e., queries involving multiple concepts linked by logical connectives) into atomic queries, that each refer to only one concept. This decomposition is performed so that each atomic query can be answered by a single RouterAgent – thus supporting the semantic routing mechanism by ensuring that each message in the system has a single target destination. These atomic queries are then sent into the RA network, via the PA. When the QMA receives the query responses, they are aggregated by re-applying the logical connectives present in the complex query, thus returning the set of web resources matching the constraints expressed in this original query. As a part of the reply aggregation process a number of result 'cleaning' tasks are performed, identifying and removing duplicate instances and resources from the aggregated result set. The query decomposition and the reply aggregation and optimisation techniques implemented in this agent are adaptations of existing techniques and mechanisms. A detailed description of the specific processes employed in query decomposition and reply re-aggregation, and the algorithms developed to implement them, are presented in Section 5.5.2 and Section 5.5.4 respectively.

In addition to its primary query decomposition and reply re-aggregation tasks, the QMA also provides some management of the query process. That is, the QMA seeks to cope with non-reciept of expected replies to atomic queries. This management is achieved by two different behaviours. Firstly, the QMA tries again to obtain a reply by re-sending 'failed' queries, where no reply has been received within a specified time limit – in order to handle situations where the RA handling the concept in the atomic query is temporarily unavailable. Secondly, where the QMA is unable to obtain any reply to one or more of the atomic messages, it then seeks to compile a partial reply from the available results within the replies that have been received. This is achieved by identifying the position of the missing replies within the tree-reconstruction that drives the re-aggregation process, and then simply forming the union of the aggregated result sets on either side of the missing element. Clearly there are cases where even a partial reply cannot be sensibly reconstructed, for example where the majority of the expected replies are missing or the missing reply represents the only element in the *SELECT* construct of the query. In these cases, identified by use of a set of simple rules, the QMA replies to the IA with an 'query failed' message.

Notification Agent (NA): These agents act as the interface between SERSE and the Notification Server

component of the Esperonto architecture. These notifications are received by the agent via the Java Messaging Service, from the Notification Server – to which the agent must subscribe in order to get the messages. NAs can receive three types of notifications:

- Ontology publications – notifies that a new ontology has been published on the Ontology Server. The NA then sends this notification to the local RPA which then generates the RAs for each of the root concepts within the ontology, in order to introduce the ontological knowledge into the RA network. The NA also passes the notification to the The IA, so that the query interface is updated, and can include all the available ontologies in the query generation process.

- Ontology modification – notifies that an existing ontology on the Ontology server has been modified in some way. The The NA extracts the, potentially multiple, concept mappings from the notification message, and then re-sends these mappings as a set of ACL messages – with each message relating to one of the original concepts that is to be modified. These messages are then routed to the appropriate RAs so that they can update their ontological knowledge.

- Content acquisition – notifies that new web resources have been semantically annotated by the Annotation System. NAs decompose these XML notifications, that may regard multiple concepts and resources, into 'atomic' notifications that each refer to only one concept. These atomic notifications are then sent into the RA network, via the PA, as Agent Communication Language (ACL) messages. This notification decomposition process is performed in the same manner as the query decomposition process performed by the QMA.

The role of the `NotificationAgent` within SERSE and its handling of these messages is further elaborated in Section 5.4.1.

Interface Agent (IA): These agents provide a connection between each agent platform within SERSE and an external user-query application that operates outside the platform, by creating a socket interface and passing query and response objects across it. An external application queries the routing system by using a published API, which consists of a set of objects that both parties use to communicate about queries and replies. The system has an implemented web-based query interface to enable users to generate semantic queries (as described in Section 5.4.1), that utilises the IA to pass queries into SERSE and receive the corresponding replies.

4.4 Semantic Relatedness Metric

The RouterAgents (RAs) within SERSE are each *specialised* with respect to a defined concept, meaning that they index and can access those resources whose annotations contain instances of that concept. Furthermore, each RA is only aware of those other RAs whose concepts are *similar* or *related* to its own. In this way, the network of RAs is organised into a set of overlapping *semantic neighbourhoods* that link together those agents whose concepts are determined to be sufficiently similar or related. Messages within the RA network each regard a single concept, and are semantically routed through the RA network by determining the degree of similarity or relatedness between the message concept and the concepts handled by the neighbouring RAs. Therefore, the RouterAgents require a means to determine the degree to which their concept is related to any other concept, that may appear in a routed message or as a potential neighbour. This evaluation of similarity and relatedness between concepts is performed by the Semantic Relatedness Metric (SRMetric).

We refer to both similarity and relatedness as ways in which two concepts can be 'related' because these two types of relation are used in different ways within SERSE, though both are determined on the basis of the ontological descriptions by the SRMetric. Two concepts are *similar* based on whether they refer to the same conceptualised entity, and this can be evaluated by comparison of the elements of the two ontological descriptions of the concepts. Similarity is the basis upon which the semantic message routing operates, in that messages are only routed to neighbouring RAs with the highest similarity determination. Two concepts are *related* if there is an non-hierarchical, ontological relation defined between them, that is, if they are related through an object property - for example, the concepts *car* and *person* would be related by the properties *drives* and *hasDriver*. Relatedness is only used within the formation of semantic neighbours, in conjunction with the similarity determination. The purpose of including relatedness links within the semantic neighbourhoods is to provide additional links across the RA network. This is primarily to support the semantic browsing mode of resource discovery, but also to aid the semantic routing of messages by enabling 'shortcuts' across the network – allowing a message to be moved to an RA handling a more closely related concept without traversing all of the RAs in between.

Thus, the semantic relatedness between two concepts, determined on the basis of their ontological definitions by the SRMetric, is utilised within SERSE in two main ways:

- To underlie the formation of semantic neighbourhoods, by making connections between RouterAgents based on the semantic relatedness of the concepts they are responsible for. Such relations include similarity and other relations, such as meronomy, ownership, usage, etc., between concepts, as specified within the defining ontologies. This usage of semantic relatedness to interconnect RAs results in a semantic overlay that effectively clusters and links the RAs based upon

the meaning of the terms they index.

- To enable the semantic routing of messages between `RouterAgents`, by enabling agents to determine the degree of similarity between their, and their neighbours, concepts and those concepts that form the subjects of the messages. By use of the SRMetric individual `RAs` are able to select which is the most appropriate neighbour to route a message to based purely upon the meaning of the message concept and those handled by each of the neighbours.

The SRMetric has been specifically designed and implemented to perform the determination of concept similarity and relatedness within the SERSE environment – with a strong focus on efficient and rapid calculation. The manner in which the determination is made owes a great deal to current approaches in ontology alignment (see Section 3.5.3), and in particular on the Quick Ontology Mapping (QOM) system described in Section 3.5.4.

4.4.1 SRMetric Implementation

The Semantic Relatedness Metric (SRMetric) has been implemented as a set of heuristic rules (based on existing semantic similarity research) that exploit the characteristics of a concept described in the ontology – lexical information, ancestors, descendants, properties, property values and constraints, etc. This element-based approach, using the features of the concept description for separate sets of comparisons that are subsequently aggregated, is common among state-of-the-art ontology alignment systems (see Section 3.5.3), and QOM is a leading example of such approaches. However, we have modified the adopted approaches, because they are intended to perform a process that requires high precision in order to establish the correct mappings. Semantic routing is a problem that presents different features: the evaluation of similarity should be sufficiently precise to determine a new target agent for the routing process, but not necessarily the *best* one available if the global knowledge of the system is considered. In addition, semantic routing is executed as a dynamic process, and requires efficient computation in order to minimise the round-trip query time, and so make the system usable for on-demand search.

A number of different modifications and adaptations were performed upon existing state-of-the-art approaches to fit these requirements. Some of the modifications follow a similar pattern to those modifications introduced by QOM itself – adapting and removing the most computationally expensive comparisions. In addition, the iterative behaviour of many alignment systems was also excluded from the SRMetric, because we are only comparing single concepts rather than whole ontologies, and so there

are no previously determined mappings available to be considered in a later iteration. Other modifications include the application of specialised element-weighting heuristics, so that the SRMetric generates a higher number of potential matches for any particular concept when compared to ontology alignment systems, and thus exhibits a less precise behaviour than is required for ontology alignment. Finally, the SRMetric does not make use of ontology extensions (i.e., instances of concepts) when determining similarity, because they are often not available, if they are available they may be a large set that take time to process, and, unlike with ontology alignment, there is no assumption that the extensions of two similar concepts should overlap (the two concepts being compared may be entirely unrelated).

Component	Feature	Comparator
Lexical	Concept label	String similarity
Lexical	URI	String equality
Structural	Super-concepts	Concept-set similarity
Structural	Sub-concepts	Concept-set similarity
Structural	Sibling-concepts	Concept-set similarity
Property	Data-type properties	Property-set similarity
Property	Object properties	Property-set similarity
Relatedness	Object properties	Co-reference property identifer

Table 4.1: SRMetric components.

The differences between similarity and relatedness in this context were discussed in the introduction to this section. Essentially, similarity assesses to what degree two concepts represent the same conceptualisation of the same abstract or concrete entity or object. Relatedness assesses to what degree two concepts may be related through means other than similarity. The Semantic Relatedness Metric outputs either a similarity score between 0 and 1, or a relatedness score between 0 and 1, and indicates the type using an additional output parameter. When comparing two concepts the metric will attempt to determine a score for both similarity and relatedness. The usual case is that one of the two scores will not be significant – due to the fact that two concepts that are significantly similar are usually not also strongly related – and the higher score is the one that is returned. In those cases where two concepts are both somewhat similar and somewhat related, the metric will still return the higher of the two scores, unless the similarity score is above a defined threshold value in which case the similarity score is always returned. This is because similarity is utilised as the primary semantic relation within SERSE, and the purpose of determining a relatedness score between two concepts is to provide additional interconnections within the `RouterAgent` network only when a significant similarity cannot be determined.

The semantic relatedness metric determination is built upon four sets of feature comparisons (summarized in Table 4.1), utilising different means of determining similarity and relatedness, that are combined to produce an overall metric. The components of the ontological descriptions utilised in these

comparisons are:

1. **Lexical component**: This utilises the labels and URIs of the concepts, but in two different ways. The URIs are checked for exact equality, and finding this would immediately result in the two concepts being determined to be the same (i.e., be totally similar). The strings representing the labels used to denote the concepts are compared to each other, using an edit-distance calculation [100], plus checks for term containment, sub-string match between terms, etc. The more similar these strings, the greater the contribution to the overall metric score.

2. **Structural component**: This compares the relative position of the two concepts in their respective ontology hierarchies. This is achieved by comparison of the sets of super, sub, and sibling concepts of the two concepts being compared. Such comparisons are achieved by recursive use of the SRMetric, to make concept comparisons that themselves form a part of the original comparison. Such recursive use of the metric requires addition constraints, as described below. Each item of similarity evidence obtained from these comparisons, indicated by significant similarity between any of the members of the concept sets, then contributes to the overall metric score.

3. **Property component**: This comparison utilises the concepts properties and their values, and makes reference to the property inheritance hierarchy in the two source ontologies. The property labels and URIs can be compared lexically, which is achieved by re-use of the same lexical comparison component used for concept comparisons, as described above. Additionally, the values of the properties can be compared, but in different ways for data-type and object properties. The values of data-type properties can be compared in terms of the type and allowable range of the specified data-types. The values of object properties are concepts, and their extensions, and so they are compared by recursive use of the entire metric (with additional constraints, described below). As with the structural component, each item of similarity evidence, obtained through these comparisons, contributes to the overall metric score.

4. **Relational component**: The object properties defined for each of the concepts are examined, and the concepts representing their ranges are identified. If any of these range concepts are the same as the other concept in the original comparison, then the original concepts are determined to be 'fully' related. Where one or more of the range concepts is determined to be similar to the other concept in the original comparison, then the original concepts are determined to be somewhat related. The degree to which they are determined to be related is based upon the highest similarity score found during this comparison. As with hierarchically related concepts and with the ranges of object properties, these concept comparisons are achieved by recursive use of the entire metric.

As described in the metric component descriptions above, the metric is used recursively when the

Algorithm 1 Semantic Relatedness Metric

Require: Concept $\mathcal{A}'$ and Concept $\mathcal{B}'$
Ensure: Metric value - $0 \leq \texttt{SRM} \leq 1$
1: Hierarchical depth index $\delta = 0$
2: Recursion flag $\mu = true$
3: SRM $(\mathcal{A}', \mathcal{B}') = CompareConcepts\ (\mathcal{A}', \mathcal{B}')$

degree of relatedness between two concepts is used as a part of the evidence in the comparison of another two concepts. This is the case when considering the ancestor, descendent and sibling concepts of two concepts, and with those concepts that are the subjects of the original concepts' properties. However, such recursive use of the metric requires careful implementation. In particular, there is the need to place a limit on how 'deep' into a concept hierarchy such recursion can go – otherwise the metric will ultimately compare the two entire ontologies as a part of the comparison of any two concepts from these ontologies, by traversing the whole inheritance hierarchy for each concept. To avoid this situation, and to ensure the rapid calculation of the metric, the recursion applies a 'depth-limit' control that specifies at which cycle of recursion the comparison should cease. Furthermore, the recursion depth is also applied as an additional weighting factor on the recursive applications of the metric, to reduce the significance of similarity scores contributing to an overall score, dependent on the number of hierarchy steps the found similarity is from the original comparison concepts. This depth-limit could be further refined by taking into account the degree of relatedness already determined at the point of recursion. This would allow the metric to terminate calculation earlier when the concepts being compared have not already demonstrated any significant relatedness, and, conversely, would also allow the metric to take into account further ontological information when the two concepts have already been determined to have significant relatedness.

The final features of the SRMetric calculation are the use of heuristic weighting factors and threshold values. The metric uses a number of weighting factors, with different weights applied to different components of the metric. The primary weighting factors are those applied to the four components of the metric calculation. However, additional weighting factors are applied at a number of points within the metric calculation, such as the recursive-depth weighting mentioned above, and the similar weighting applied due to traversal of the property hierarchy. There are three main threshold values utilised within the SRMetric calculation. The first threshold value has the most significant function, that is to determine if a potential neighbour concept is sufficiently similar to an RA's own concept for that concept to be included in the RA's semantic neighbourhood. In the current implementation of the SRMetric this threshold value has a fixed value, however, consideration has been given to enabling the RAs to have autonomy over this value, and determine individual values based on available evidence. This feature has been trialled within SERSE with some success, and highlighted the fact that the degree of autonomy over

Algorithm 2 Compare Concepts

Require: Concept $\mathcal{A}$ and Concept $\mathcal{B}$
Ensure: $0 \leq \texttt{SRM} \leq 1$
Increment δ
Find lexical matches - λ - between concept labels and URIs
Aggregate and weight lexical matches - $\mathcal{L} = \sum_{a=1}^{n} \omega_l \, \lambda_a$
if $\mu = true$ (i.e., comparing $\mathcal{A}'$ and Concept $\mathcal{B}'$) **then**
  for all sibling classes of $\mathcal{A}$ ($\mathcal{A}\gamma$) and $\mathcal{B}$ ($\mathcal{B}\gamma$) **do**
    $\sigma_b = CompareConcepts\ (\mathcal{A}\gamma_b, \mathcal{B}\gamma_b)$
  end for
  $\mu = false$
end if
if $\delta \leq \Delta$ (hierarchical depth limit) **then**
  for all super-classes of $\mathcal{A}$ ($\mathcal{A}\alpha$) and $\mathcal{B}$ ($\mathcal{B}\alpha$) **do**
    $\sigma_c = CompareConcepts\ (\mathcal{A}\alpha_c, \mathcal{B}\alpha_c)$
  end for
  for all sub-classes of $\mathcal{A}$ ($\mathcal{A}\beta$) and $\mathcal{B}$ ($\mathcal{B}\beta$) **do**
    $\sigma_d = CompareConcepts\ (\mathcal{A}\beta_d, \mathcal{B}\beta_d)$
  end for
end if
Aggregate and weight structural matches - $\mathcal{S} = \sum_{e=1}^{n} \omega_s \, \sigma_e$
for all properties of concepts $\mathcal{A}$ ($\mathcal{A}\phi$) and $\mathcal{B}$ ($\mathcal{B}\phi$) **do**
  Hierarchical depth index $\eta = 0$
  Recursion flag $\nu = true$
  $\phi_f = CompareProperties\ (\mathcal{A}\phi_f, \mathcal{B}\phi_f)$
end for
Aggregate and weight property matches - $\mathcal{P} = \sum_{g=1}^{n} \omega_p \, \phi_g$
for all object properties of $\mathcal{A}$ ($\mathcal{A}\rho$) **do**
  $\rho_h = CompareConcepts\ (\mathcal{A}\rho_h, \mathcal{B})$
end for
for all object properties of $\mathcal{B}$ ($\mathcal{B}\rho$) **do**
  $\rho_i = CompareConcepts\ (\mathcal{A}, \mathcal{B}\rho_i)$
end for
Aggregate and weight relation matches - $\mathcal{R} = \sum_{j=1}^{n} \omega_r \, \rho_j$
Aggregate all element-match scores - $\texttt{SRM} = \frac{\mathcal{L}+\mathcal{S}+\mathcal{P}+\mathcal{R}}{\omega_l+\omega_s+\omega_p+\omega_r}$

the threshold value must be limited in a number of different ways in order to ensure that routing paths remain intact, and RAs do not become unconnected from the rest of the network – which is ensured by the currently selected threshold value. That is, RAs cannot have total autonomy over the threshold value, due to the fact that this would enable RAs to select too high a value, resulting in them having no neighbours and so becoming isolated from the rest of the network, or too low a threshold, resulting in over-large neighbourhoods that entirely engulf the neighbourhoods of neighbouring agents. The most promising approach is to define a range of possible threshold values and enable RAs to exert autonomy over the value within this range, but determination of such a range is in itself problematic. The second threshold value used is that mentioned above in relation to determining whether to return the similarity or relatedness score from the metric. The role of this threshold value is to set a lower bound on the value

of the similarity score, below which the similarity can be discarded in favour of a greater relatedness score. The third, and final, threshold value is similar to the first, in that it determines the cut-off value for inclusion in a semantic neighbourhood, but applies to the relatedness score, rather than the similarity score. As in the development of the QOM system, the values of the weights and the threshold value are all manually determined on the basis of performance evaluations – such as that presented in Experiment 4 in Section 6.3.4.

Based on the descriptions given above, a formal definition of the overall Semantic Relatedness Metric is as follows:

$$SRMetric(\alpha, \beta) =$$
$$max(\mu_s(\sum_{i=1}^{n} W_l \cdot L_i(\alpha, \beta) + \sum_{i=1}^{n} W_s \cdot S_i(\alpha, \beta) + \sum_{i=1}^{n} W_p \cdot P_i(\alpha, \beta)), \mu_r(\sum_{i=1}^{n} W_r \cdot R_i(\alpha, \beta)))$$

where: W_x are individual weighting factors for each of the components, and μ_x are separate normalisation factors for similarity and relatedness, used to express the result of the metric in a [0–1] range. The implementation of this formal definition is divided into three functions. The overall SRMetric algorthim can be seen in Algorthim 1 – which itself uses the separate concept and property comparison functions described in Algorithm 2 and Algorithm 3 respectively.

Algorithm 3 Compare Properties

Require: Property $\mathcal{A}$ and Property $\mathcal{B}$
Ensure: $0 \leq \phi_i \leq 1$
1: Increment η
2: Find lexical matches - λ - between property labels and URIs
3: Aggregate and weight lexical matches - $\mathcal{L} = \sum_{a=1}^{n} \omega_l \lambda_a$
4: **if** $\nu = true$ **then**
5: **for all** sibling properties of $\mathcal{A}$ ($\mathcal{A}\gamma$) and $\mathcal{B}$ ($\mathcal{B}\gamma$) **do**
6: $\sigma_i = CompareProperties$ ($\mathcal{A}\gamma_i$, $\mathcal{B}\gamma_i$)
7: **end for**
8: $\nu = false$
9: **end if**
10: **if** $\eta \leq \Delta$ (hierarchical depth limit) **then**
11: **for all** super-properties of $\mathcal{A}$ ($\mathcal{A}\alpha$) and $\mathcal{B}$ ($\mathcal{B}\alpha$) **do**
12: $\sigma_j = CompareProperties$ ($\mathcal{A}\alpha_j$, $\mathcal{B}\alpha_j$)
13: **end for**
14: **for all** sub-properties of $\mathcal{A}$ ($\mathcal{A}\beta$) and $\mathcal{B}$ ($\mathcal{B}\beta$) **do**
15: $\sigma_k = CompareProperties$ ($\mathcal{A}\beta_k$, $\mathcal{B}\beta_k$)
16: **end for**
17: **end if**
18: Aggregate and weight structural matches - $\mathcal{S} = \sum_{b=1}^{n} \omega_s \sigma_b$
19: Aggregate all element-match scores - SRM = $\frac{\mathcal{L}+\mathcal{S}}{\omega_l+\omega_s}$

Chapter 5

SERSE Implementation

In the previous chapter we detailed the context and intended task of SERSE*, the principles underlying the system design, and the multiagent architecture of the system and the agent roles within it. In this chapter we present details on the implementation of this design, and describe how the intended functions of* SERSE *are achieved in practice.*
Section 5.1 discusses the means by which the capabilities of the agents are defined and constructed within the development environment, and describes the structure of the following descriptions. Section 5.2 describes the way in which the underlying semantic message routing process operates, abstracting this fundamental system-support function from the primary use-case function sections that follow. The first of these, Section 5.3, describes how SERSE *handles the notification of ontology publications and potential, subsequent ontology modification notifications. Then, Section 5.4 and Section 5.5 describe the operation of* SERSE *when performing its two primary tasks of, respectively, indexing annotated resources and answering semantic queries over these resources.*

5.1 Overview

This chapter is intended to provide a description of how SERSE has been constructed to fulfill the requirements and implement the system design presented in the previous chapter. The multiagent system design for SERSE has been implemented using the Java Agent DEvelopment Framework (JADE) [9], that was presented in Section 2.2.4. The required capabilities of the SERSE agents were developed following JADE's model of task-specific behaviours, and composition of behaviours to achieve complex tasks. All the agents within SERSE have a number of possible behaviours, that are each triggered by a particular set of environment circumstances – usually related to the receipt of a particular type of system message. Therefore, each agent makes use of a cyclic 'message receipt and behaviour dispatch' behaviour. This behaviour operates continuously within each of the agents, performing the following

general sequence of actions:

1. Check incoming message buffer for new messages received since the last behaviour cycle. Different system messages are assigned different priorities, and this determines the order in which received messages are removed from the buffer and processed.

2. Remove the oldest, highest priority message from the buffer, and parse the message contents according to the message template definition.

3. Select and execute the appropriate behaviour for this message type and content, passing in the message contents for subsequent action by the new behaviour.

The sub-behaviours spawned off by this cycle message handling behaviour fall into four main categories: semantic message routing, notification behaviours, query answering behaviours, system management behaviours. These task-specific behaviours are described in detail, as a part of the overall functional description of SERSE in the following sections.

The following sections describe the implementation of the primary SERSE functions presented in the use-cases descriptions in Section 4.1.3. Therefore, the sections present how SERSE handles ontology publication notifications, ontology modification notifications, content acquisition notifications, and query answering. However, the first section abstracts a key component of all of these primary use-cases – the semantic message routing function – as identified in Section 4.1.4. The behaviour implementing this semantic message routing, that underlies all communication between `RouterAgents`, is presented in Section 5.2. The next section 5.3 presents both the ontology publication notification and ontology modification notification functions, thus describing how SERSE learns about newly available concepts, and where necessary, their subsequent modifications. The content acquisition notification function, through which SERSE receives new knowledge about resources to index, is described in Section 5.4. The query answering function, through which the knowledge regarding semantically annotated resources is selectively retrieved from SERSE by use of semantically specified queries, are described in Section 5.5. Finally, the system management behaviours introduced above are not presented in a separate section, but are described within the other sections along with the more directly functional elements of the system that they support.

5.2 Semantic Message Routing

The semantic routing behaviour is the means by which messages are moved around the RouterAgent (RA) network on the basis of the semantics of the message contents, and those of the concepts handled by the RAs. The semantic routing mechanism is designed to move messages, addressed by concept, in a series of RA to RA hops across the network to the most appropriate RA for the message concept. Semantic routing is achieved by each RA in the system: firstly determining if it is the most appropriate for the message concept, and if it is, the agent handles the message itself; if it is not the most appropriate, the RA will route the message to the most appropriate other RA that it knows about. Both the determination of whether it handles the closest available concepts and of which of its neighbours handle the closest known concept is performed by the RA using the Semantic Relatedness Metric (as described in Section 4.4 of the previous chapter). This semantic routing mechanism builds upon three main threads of research: agent communication, peer-to-peer routing, and semantic overlay networks. The adaptation and combination of these approaches is a unique feature of SERSE, enabling dynamic inter-linking of the whole multiagent system by use of only 'local' semantic knowledge within each of the agents involved.

The intention of the Semantic Routing process is to incrementally, at the point of re-transmission of the message by an RA to its most appropriate neighbour, move a message semantically 'closer' to the correct RouterAgent (for the message concept), until that agent is reached. The Semantic Routing process aims to move a message along the most efficient route between its starting RA and its target RA, however this cannot be guaranteed given the dynamism of neighbourhoods of the RAs on this route. The routing process can guarantee a message reaches its target within a limited number of re-transmission hops, given that the RAs act in order to prevent cycling of messages (as described later in this section), thus ensuring that any one message will not traverse more hops than there are RAs within the entire SERSE network.

Each RA maintains a routing index to manage its semantic neighbourhood – recording the agent ID, concept handled, and recent message transmission success for each neighbouring RA. This routing index is initialised when the RA is created, and subsequently modified on the basis of changes in the RA network, by addition of new RAs to, or their removal from, the neighbourhood. The processes by which the routing index is created and maintained are triggered by the notification of new resources to the system, and these are fully described in Section 5.4. In order to dynamically manage their neighbourhoods, RAs utilise a simple type-system for the neighbours in their routing indexes. This system enables the agents to determine which concepts are present in their semantic neighbourhood, and which concepts should be in the neighbourhood (on the basis of the ontological knowledge) but do not yet have notified content – and therefore no existing RA. Furthermore, the type system enables the agents to 'bridge' over

these gaps in the RA network that are due to the absence of RAs that do not yet have content to index. The three types of neighbour used in the routing index entries are:

- **actual** – an actual neighbour is an already existing RA that handles a concept determined to be a semantic neighbour. These represent the 'normal' agent inter-connections in the RA network.
- **ontology** – an ontology neighbour is that of a semantic neighbour concept for which there is not yet an existing RA. The presence of these neighbours enables RAs to know when a concept is not indexed within the system – in that, if it was, its ontology neighbours would be aware of it.
- **implied** – an implied neighbour is an already existing RA that handles a concept determined to be just outside the semantic neighbourhood – that is a concept that should be a neighbour of this concept's neighbour. Implied neighbours are used to maintain this semantic connection when the immediate ontology neighbour is not present.

The operation of the semantic message routing process is based upon the fact that messages within the RA network always regard only a single concept. This is achieved by the decomposition of 'complex' messages that refer to multiple concepts into a set of 'atomic' messages, that each refer to only one of the concepts in the original message, and collectively maintain the semantics of the original message. This message decomposition process is applied to notification and query messages originating from outside SERSE by the `NotificationAgent` (NA) and `QueryManagementAgent` (QMA) respectively, and are fully described in Section 5.4 and Section 5.5. Following this decomposition the 'atomic' messages are passed from the NA or QMA to the `PortalAgent` (PA) for transmission into the RA network. Messages directed between RAs are generated as 'atomic' messages and do not require decomposition. The different types of messages utilised within the SERSE system, including those that are semantically routed in the RA network are presented in detail in Appendix A.

An RA receiving an 'atomic' message, from either the PA or another RA, follows the semantic routing algorithm shown in Algorithm 4. The execution of this algorithm produces the following process:

- The message concept is first compared to the concept indexed by this RA, using the semantic relatedness metric and the OWL concept definitions. If the message concept is sufficiently 'close' to the agents concept, i.e., the semantic relatedness score is above the matching threshold, then the message is dealt with by this agent. Depending on the message type, the RA handles the message as follows:
  - *ACLNotificationMessage* – the RA updates its content index, by reading the RDF file referred to in the message, and using this RDF data to add new annotation instances to the

Algorithm 4 Semantic Routing

Require: $\texttt{RA}_i$ receives message $\mathcal{M}$
Ensure: $\mathcal{M}$ handled by $\texttt{RA}_i$ or forwarded to most appropriate neighbour RA
1: **if** concept $\mathcal{C}_m$ in $\mathcal{M}$ = concept $\mathcal{C}_i$ handled by $\texttt{RA}_i$ **then**
2: $\texttt{RA}_i$ handles message $\mathcal{M}$
3: **else**
4: **for all** neighbour RAs – $\mathcal{N}_j \in \{\mathcal{N}_1, \cdots, \mathcal{N}_n\}$ – in routing index of $\texttt{RA}_i$ **do**
5: **if** type of $\mathcal{N}_j$ = *actual* **then**
6: Retrieve concept $\mathcal{C}_n$ of $\mathcal{N}_j$
7: Calculate $\texttt{SRM}(\mathcal{C}_m, \mathcal{C}_n)$
8: **end if**
9: **end for**
10: Select $\mathcal{N}_k$ where - $\texttt{SRM}(\mathcal{C}_n, \mathcal{C}_m) = \min\{\texttt{SRM}(\mathcal{C}_s, \mathcal{C}_m)\}$
11: Forward message $\mathcal{M}$ to $\mathcal{N}_k$
12: **end if**

index and to remove old instances (as identified by the annotation wrapper). See Section 5.4 for a full description of this process.

- *ACLOntologyModificationMessage* – the RA modifies its routing index. The ontology modification mappings in the message are used to map existing content index and routing index entries using the previous ontology definition into ones using the new definition.

- *ACLSimpleQueryMessage* – the RA extracts the RDQL query contained in the message, and executes this query against its stored RDF content index. The RA then responds to the originating QMA with a reply to this query. See Section 5.5 for a full description of this process.

- If the message concept does not match the agents concept, the RA then compares the concept to the concepts handled by each of the neighbours in its routing index. The concept that is semantically closest to the message concept is determined using the semantic relatedness metric (as described in Section 4.4) and, thus, which of the RA's neighbours is the most suitable recipient of the message. The following action is then dependent on the type of the neighbour:

  - if the neighbour is an actual neighbour or an implied neighbour then the message is sent to that agent.

  - if the neighbour is an ontology neighbour, the current agent can determine that no RA exists for the message concept (due to the creation of implied neighbours when a RA is created), the the RA takes one of two actions. If the message is a content notification then the RA sends a message (*ACLRouterCreationMessage*) to the `RouterPlatformAgent`, thus creating the new RA for the notified content. If the message is a query, the current RA can be certain that there are no indexed instances for the message concept and can reply accordingly to the

originating QMA.

Within this overall semantic message routing process there are a number of issues that require special handling:

- A RA must be able to determine when the same message has been received multiple times, to avoid the formation of 'cycles in the routing process. This is achieved by each RA keeping a history of its most recently handled messages, and checking incoming messages against this. The message is detected as cycling if a RA receiving a message determines that it has seen this message before (through use of the unique message ID) and that it has already sent it to all of its neighbours. When a cycle is detected, the message is sent back to the PA, which will then re-send the message into the RA network, but using a different RA as the starting point for the routing.

- When the neighbouring RA selected as the receiver of a forwarded message is unavailable, a RAs have a handling procedure that allows them to re-send a message to an unavailable neighbour, for a fixed number of attempts, and then to consider the neighbour RA dead and permanently unavailable for routing[1]. In this situation the message would then be handled as for a non-existant RA – contacting the local `RouterPlatformAgent` to create a new RA for content notifications, and returning an empty reply for queries.

The semantic routing process described here is designed to move messages through the RA network, from agent to agent in a series of iterative 'hops', on the basis of the semantics of the concept that is the subject of the message and of the concepts handled by each of the RAs. However, the potential size of the RA network, when indexing a large collection of knowledge, raises an additional problem. In the case where the ontologies available to SERSE cover a diverse domain of knowledge, there is the potential for an RA receiving a message being unable to determine which of its neighbours is best suited to handle the message, because the semantics of the message concept are sufficiently dissimilar to all the neighbours concepts that no clear routing direction can be determined - due to the fact that the SRMetric cannot effectively discriminate between neighbour concepts when they are all very dissimilar from the target concept. Initial evaluation of the semantic routing process demonstrated that if messages where initially sent to a RA in a neighbourhood that was semantically very dissimilar, there was a significant probability that such messages would become 'lost' in the RA network – with the query timing-out before the message reached the appropriate RA. In order to handle such situations, and to increase the efficiency of the semantic routing process in general, the concept of super-peers, within a hybrid P2P

[1] However, the autonomic system processes in place would seek to recover the failed agent, and would re-instate the neighbour connections – see Section 4.2.3.

architecture, was adopted. As described in Section 2.4, super-peers are intended to operate within the P2P network, but also have additional capabilities – usually associated with some management or control processes within the P2P system. This approach was adapted to suit the existing architecture of SERSE by devolving the additional capabilities into a separate agent (the `PortalAgent`) that was not responsible for indexing any resources, but was solely responsible for the additional capabilities relating to the semantic routing process. These capabilities were designed to enable the `PA` to direct a message into the correct general semantic 'area' of the `RA` network – enabling the receiving `RA` to determine the correct routing direction amongst its semantic neighbours. In order to achieve this initial semantic routing step, the `PAs` hold a routing index that records a set of `RAs` that have been determined to be 'significant points' within the `RA` network.

In the current implementation of SERSE, these `RAs` are those that handle the root concepts of each ontology that is known to the system. Although this initial routing step to the root-concept `RAs` can be seen as the potential cause of a bottleneck in the message routing, these `RAs` are those that are likely to have the smallest workload from responding to queries. In fact, in most domain ontologies the majority of the concept instances are instanciations of more specific descendant concepts (leaf nodes), whilst the root-node concepts have few (if any) instances. Therefore, the additional routing effort performed by these `RAs` is compensated for by responding to fewer direct queries. In addition, any such set of significant entry points in the `RA` network could act as a bottleneck, and the use of possible alternatives points is constrained by the computation required to dynamically identify these points, and by message overhead involved in continually aligning these points between the existing `PAs`.

5.3 Ontology Notifications

The Notification Server provides SERSE with notifications of ontologies that have been newly published on the Ontology Server. The knowledge regarding this availability must be incorporated into the indexing system, so that the system can store indexing information for the concept, and so that the concept can be utilised within queries for resources. This incorporation must support the subsequent process of indexing resources annotated with instances of concepts from the new ontologies – that is, provide an initial placeholder within the index network for the concepts within the ontology.

Once an ontology is published on the Ontology Server, the server generates a message that is sent via the Notification Server component of the Esperonto system and then Notification Mediator component of SERSE to all of the available `NotificationAgents` within the SERSE system. The Notification

Mediator forwards the details of the new ontology to all of the available platforms, so that the ontology is added to the master list of available ontologies maintained by all RPAs, forming the register of ontological knowledge maintained within the system. This then means that the ontology becomes available to all RAs within the network, which enable them to use the concepts in the new ontology when considering potential semantic neighbours. This means that the new concepts are rapidly integrated into the index network. The process of creating new RAs in this situation is described in detail in Section 5.4.2. The ontology publication notification is also forwarded by the RPAs to the InterfaceAgent on each platform. This is so that the query interfaces linked to each IA can be aware which ontologies are available for use within user's queries, and so make the concepts within the ontology available for selection in the interface.

However, despite the fact that the notification message is sent to all RPAs only one of them, on one of the available platforms within the SERSE system, is requested to act on the notification in terms of creation of RAs. That is, the initial action within SERSE following notification of a new ontology is to create RouterAgents for each of the 'root' node concepts described in the ontology. The purpose of this initial creation is to make the concepts available for portal routing (see previous section), and to ensure at least one concept from the ontology has an existing RA within the network – to perform the role of 'nearest available neighbour' in the subsequent creation of RAs for other concepts in the ontology (see Section 5.4.2). Therefore, only one of the RPAs receiving the notification should act on it with regard to root node RA creation, to prevent duplication of index agents.

The selection of which platform in the system to use for this initial RA creation is made by the Notification Mediator, whose function with SERSE is to distribute notified content over all the platforms available within the SERSE system, on the basis of some distribution policy. In the current implementation, this policy operates as a simple 'round-robin' selection, ensuring that each available platform receives approximately the same number of notifications. Clearly, other more policies could be adopted to achieve the same aim of appropriate distribution of indexing effort over available index platforms. Such other policies could include additional considerations, such as the processing and storage capacity of platform host systems, the network connectivity of hosts, existing query and notification traffic per platform, etc.

The Notification Server also provides SERSE with notifications of modifications to ontologies on the Ontology Server, consisting of one or more specific modifications to one or more individual concept definitions. This knowledge must be incorporated into the indexing system, so that up-to-date concepts definitions are being used, but without discarding resources indexed under prior definitions. As with all

other messages that potentially regard multiple concepts, ontology modification notifications are subdivided into atomic notifications that each contain the notified OWL mapping for one of the modified concepts. This mapping represents a declarative representation of each of the specific modifications to a concept description, specified in terms of OWL statements showing the existing descriptive elements and their respective replacement descriptions. These atomic 'concept modification' notifications are then sent as ACL messages into the `RouterAgent` network. There are three ontology modification cases, which cover all potential modifications within an ontology, that SERSE is equipped to handle:

1. New concepts in the modified ontology are processed in the same way as any other newly available concept – the availability of the concept is made known to all `RPAs`, and via them to all `IAs`. In addition, if the concept is a new root node of an ontology, its availability is also made known to all `PAs` for use in portal routing (see previous section).

2. Concepts that have had their existing descriptions modified are treated as being a new 'version' of the existing concept, and so are handled by the same `RouterAgent` (whether or not it already exists). The basis on which the distinction is made between a 'new' concept and a 'modified' concept, given that the modifications may entirely change the concept description, is whether the concept label has changed. The rationale for this is that if the label of the concept has not changed, then the intended semantics have not significantly changed. If the semantics of a concept had significantly changed within the context of an existing ontology, this should be reflected by some change in the label that is intended to provide some indication of those semantics.

3. Finally, one or more concepts in the original ontology may have been removed in the modified ontology, but it would be unreasonable for SERSE to mirror this action within the RA network. That is, SERSE should retain any information about resources indexed under these concepts, and should continue to make the concept available in the user interface for use in queries. Therefore, SERSE takes no action with regard to any concepts that were previously known to it, but have subsequently been removed from a later version of the ontologies in which they were described.

5.4 Indexing New Resources

A core function of SERSE is to index the annotated resources that are notified to it from the Annotation System component, via the Notification Server. The Annotation System, through a set of resource 'wrappers', creates a metadata description of the web resources it examines. This is achieved by the use of instances of the concepts defined in the ontologies available on the Ontology Server – that are dynamically created to provide a semantic representation of the resource content. Wrappers within

the Annotation System access and analyse specified web resources using techniques from Information Retrieval, such as document layout analysis, Term Frequency / Inverse Document Frequency (TF/IDF) analysis, etc., and then generate metadata descriptions on the basis of this analysis. Wrappers then report batches of newly annotated resources to the Notification Server, providing references to the resources and the generated metadata. The Notification Server then forwards these notifications to the Notification Mediator component of SERSE, and the task of the multiagent system is then to incorporate knowledge about these new resources into the distributed indexing system. This process requires the identification, and possible creation, of the appropriate `RouterAgents` to handle this indexing knowledge.

This process involves the creation of new RAs, and the creation of their interconnections with their semantic neighbours. Therefore, this RA creation process manages the dynamic construction of the semantic overlay network superimposed on the multiagent system. In the following subsections we firstly describe in detail the manner in which SERSE handles notifications of new resources. Secondly, we explain the process of RA creation following such notifications, and thirdly, we describe how these new RAs then go about the dynamic generation of interconnections with their semantic neighbour RAs.

5.4.1 Content Notification

Content notification is the process by which newly annotated resources are reported to and subsequently indexed by SERSE. These notifications originate from the Annotation system component of Esperonto, and are forwarded to SERSE via the Notification Server. If a notification regards newly annotated content, the system will seek to direct the notification to the appropriate RAs for the concepts used within the annotations. The process of indexing newly annotated resources begins with a message sent from one of the wrappers within the Annotation system, via the Notification Server and Notification Mediator, to one of the NAs in the SERSE system. As with ontology notifications, the Notification Mediator determines which NA to use for any particular notification using the 'round-robin' selection policy. The content acquisition message from the Annotation system describes one or more annotated resources by recording the URL of the annotated resource and the URL of the RDF file, generated by the annotating wrapper, that details the annotation metadata. The receiving NA first decomposes the message content into separate notifications for each concept referred to in the RDF metadata file - in a similar manner to the query decomposition (see Algorithm 5). These 'atomic' notification messages are then sent into the RA network, via the local PA, to be semantically routed to the appropriate RA.

On receipt of an *ACLContentNotificationMessage* an RA will extract the concept referred to in the notification, and will compare this to its own concept and, subsequently, the concepts handled by its

neighbour RAs. The receiving RA will then determine that it is the 'most appropriate' to handle the notification if either of the two following conditions hold:

1. The concept referred to in the notification message matches the concept handled by the RA. In this case the RA will index the notified content – as described below.

2. The concept referred to in the notification message matches the concept handled by a neighbour RA that is recorded as an *ontology neighbour*, or if it is 'bridged over' by an *implied* neighbour. In this case the RA then knows that the relevant agent has not yet been created, and will initiate a process to create a new RA to take responsibility for indexing content referring to this concept and, thus, to handle the notified content – as described in the following sub-section.

A RouterAgent, having determined that it is the most appropriate location at which a content notification should be handled, needs to update its content index using the new annotation metadata. This is achieved by extracting the message parameters that specify the URLs of the annotated resources and the URL of the RDF metadata file generated by the wrapper performing the annotation. The RA then accesses the RDF file and extracts all those triples relating to instances of the concept handled by this RA. Wrapper metadata files are continually updated by a wrapper, with new annotations being added to the file and annotations that are no longer valid being removed from the file. Update of the content index is achieved by forming ⟨Resource URL⟩⟨InstanceURI⟩⟨WrapperID⟩ tuples for each unique combination of resource in the notification and instance in the wrapper file, and then comparing these with the existing set of tuples in the content index. If a tuple is not found in the current content index, the RA adds the tuple to the index – linking the resource to the metadata. All unique combinations of instance and resource are stored – i.e., the same resource, described by a different instance, or a different resource described by the same instance will have separate entries in the index. The RA then adds all the RDF triples extracted from the wrapper file, that relevant to the instance, to the RDF model that forms part of the RA's content index. If a tuple is found in the RAs content index that originates from the currently notifying wrapper and refers to an instance that is no longer recorded in the wrapper's RDF file, the entry is determined to be invalid. In this case the RA will remove the relevant tuple from the content index, and, if they are not referred to by any other entry, remove the RDF triples relating to the instance from the stored RDF model. The structure of the tuples within the content index means that the same resource annotated with the same instance but reported by two different wrappers is treated as a separate entry. This is because the wrappers have independent knowledge-bases, and if one wrapper ceases to report an instance whilst another does not, the entry will remain in the RAs content index and available for query answering.

5.4.2 Creation of New RouterAgents

As previously outlined, RouterAgents are created in two situations: when an existing RA determines that the RA for the concept within a content notification does not yet exist; and when the NotificationAgent on a platform receives an ontology publication notification that is marked for action by that NA. When the notification message regards a concept for which there is no pre-existing RA, i.e., this is the first time resources annotated with instances of this concept have been notified to the system, the semantic routing mechanism will cause the message to arrive at the existing RA whose concept is most closely related. This RA can then determine whether a new RA should be created to handle the newly notified content, by examining the *ontology* and *implied* neighbours in its routing index. If the newly notified concept occurs as an *ontology* neighbour, or if it is 'bridged over' by an *implied* neighbour, the RA knows that the relevant agent has not yet been created. The RA then forwards the content notification to the local RPA, as a *RouterCreationRequestMessage*, requesting creation of a new RA for that concept.

The *RouterCreationRequestMessage* send to the RPA contains the following information:

- The relevant contents of the original *ACLContentNotificationMessage*, namely the URI of the concept and the URL of the concepts ontology, and the resource and metadata file URIs.
- The Jade global unique identifier (GUID) of the RA sending the creation request.
- The *routing index* of the RA sending the creation request – subsequently referred to as the 'donor' routing index.

On receipt of a *RouterCreationRequestMessage* the RPA firstly checks its recent action history log, to determine if a RA has recently been created for the concept referred to in the message. This check is intended to avoid the situation where multiple RAs are created for a concept, as a result of a subsequent *RouterCreationRequestMessage* being created (as the result of a subsequent notification) whilst the RPA is in the process of creating the required RA. The length of time for which the RPA stores recent actions is determined by a system parameter, itself based on consideration of the time delay occurring between generation of a *ACLContentNotificationMessage* and creation of the RA (and its subsequent integration into the RA network)[2]. If the action history does not indicate a repeated request the RPA continues with the RA creation process, otherwise the repeated *ACLContentNotificationMessage* is ignored.

The RouterPlatformAgent then creates the RouterAgent, using facilities within the JADE libraries, and passes a number of data structures as parameters:

[2]In the current implementation, based upon experimentation and evaluation of the system, the action history timeout parameter is set at two seconds.

- Concept URI – the URI of the ontological concept that this RA is being created to handle.
- Ontology URL – the URL of the OWL ontology file in which the concept is defined.
- Content notification – the details of the content acquisition notification that triggered this RA creation process. Specifically, the URL of the wrapper metadata file, the wrapper ID, and the URLs of the annotated resources.
- Donor routing index – the routing index of the RA (*A*) that requested the creation of this RA (*B*). This routing index records the semantic neighbourhood of *A*, listing the GUIDs of the neighbour RAs and the URIs of the concepts they are responsible for. This information is intended to facilitate the process of integrating the new RA into the existing agent network.
- Available ontologies – a list of the URLs of all the ontologies known to SERSE at this time.

Once the new RA has been created it undertakes three tasks in parallel in order to begin functioning as a part of the distributed index. The RA handles the content notification in the normal way, reading the RDF metadata file generated by the wrapper, and then adding the metadata and resource knowledge to its content index. The new RA makes itself available for message routing, by starting the cyclic message handling behaviour described in Section 5.1 above. Finally, the RA acts to integrate itself into the existing network of RAs by identifying those available concepts that are semantic neighbours of its own concept, and then seeking to locate the RAs that handle these concepts. The process of creating and maintaining the semantic overlay network for the RAs is described in the following sub-section.

5.4.3 Semantic Overlay Network Creation

The semantic overlay network that interconnects the RAs within SERSE is fundamental to the system's operation. The creation and maintenance of this overlay is intended to ensure that:

- Concepts (and thus RAs) are linked, by semantic neighbourhood connections, to those other concepts known to the system that are most strongly related to them.
- RAs are not created without semantic neighbourhood connections.
- Closely related concepts are not handled by unconnected RAs.
- Missing RAs do not create 'gaps' within the network.

Once a new RA has been created it must integrate itself into the existing semantic overlay network. An individual RA only has knowledge of a small section of this network, representing the RA's semantic neighbourhood, that it records in its routing index. In order for a RA to identify its semantic neighbourhood, and so build its routing index, it must perform the following process:

- Identify those concepts, from all those available in all of the ontologies that are currently known to SERSE, that are sufficiently closely related to the RA's concept to be included in its semantic neighbourhood.
- For each semantic neighbour concept identified, the RA then attempts to locate the RA that is responsible for that concept, thus completing the semantic neighbourhood connection.

The identification of those concepts that should be in the semantic neighbourhood is determined using the list of available ontologies and the ontological definition of the RA's own concept. The RA systematically examines each ontology and compares the concepts defined within them with its own – using the SRMetric described in the previous chapter. Those concepts that are determined to be sufficiently related to the RA's concept, by exceeding a threshold defined on the SRMetric[3], are then added to the routing index. At this point the identified concept is added to the routing index as an 'ontology neighbour' – as it has been determined from the ontology alone, and the GUID of the responsible RA is not yet known.

The RA will then seek to locate the appropriate RA for each of these neighbours, and this is achieved in two ways. Firstly, the new RA will examine the 'donor' routing index and determine if any of the concepts it has determined to be semantic neighbours are present. This donor routing index is used because the RA that donated this routing index had been identified as the RA responsible for the concept most closely related to that in the original content notification message – indicating that some of the potential semantic neighbours of the notified concept may also be neighbours of its most closely related existing neighbour. If any of the new RAs ontology neighbours are found within the donor routing index as actual or implied neighbours, the GUID of the responsible RA is extracted and added directly into the new routing index, upgrading the routing index entry from an ontology neighbour to an actual neighbour. Any neighbours located in this way are then contacted by the RA to inform them of their new neighbour, and to complete the bi-directional neighbourhood connection. This is achieved by sending an *ACLNeighbourNotificationMessage* to each of these neighbours, notifying them of the concept and GUID of the new RA. This use of the donor routing index is intended to accelerate the integration of the new RA into the overlay network, and reduce the total number of ACL messages exchanged to achieve

[3]Set at a level intended to ensure that the total number of neighbours remains in proportion to the total number concepts indexed within the system. However, further control over the total number of neighbours may be required.

this.

The RA then seeks to locate the remaining ontology neighbours by sending out *finder* messages, that are addressed with the required neighbour concept and sent into the RA network, via the local PA, to be semantically routed to the appropriate RA (if it exists). These *ACLNeighbourLocationMessages* are sent to each of the remaining ontology neighbours, and contain the URI of this RA's concept, the URL of the ontology defining the concept, and this RA's GUID. The result of the semantic routing process for each of these messages will either be that the message successfully reaches the appropriate RA, or that it fails to do so because this RA has not yet been created. In the first case the located neighbour replies to the *ACLNeighbourLocationMessages* by sending an *ACLNeighbourNotificationMessage* back to the originating RA, and both RAs add the new semantic neighbour GUID information to their routing indexes and update the neighbour status. The initial *ACLNeighbourLocationMessages* contains the same GUID, concept and ontology information as an *ACLNeighbourNotificationMessage*, allowing the located RA to obtain this required neighbour information from the initial message, and so avoiding the need for an additional *ACLNeighbourNotificationMessage* from the seeking RA to complete the connection.

This process is illustrated in the sequence diagram presented in Figure 5.1. The sequence depicted follows RouterAgent 1 receiving the initial content notification message, for which there is no existing RA to handle the concept being notified. RouterAgent 1 then sends an *ACLRouterCreationRequestMessage* to its local RouterPlatformAgent, containing the original *ACLContentNotificationMessage* and a copy of RouterAgent 1's routing index. The RouterPlatformAgent then creates the new RA – RouterAgent 2 – passing in the donor routing index from RouterAgent 1. This new RouterAgent 2 then seeks to locate those neighbours it has determined from the ontologies known within the system (as provided by the creating RPA), that it cannot find within the donor routing index. For each of these neighbours RouterAgent 2 sends out an *ACLNeighbourLocationMessage*, via its local PortalAgent, to be then semantically routed to the appropriate RA. The sequence diagram shows the resulting message sequence for one of these neighbour RAs. Once the intended RA – RouterAgent 3 – receives the *ACLNeighbourLocationMessage*, it adds RouterAgent 2 to its own routing index. Finally, RouterAgent 3 replies directly to RouterAgent 2 with an *ACLNeighbourNotificationMessage*, and RouterAgent 2 then completes its routing index entry for RouterAgent 3 – upgrading it to an actual neighbour and recording the agent's GUID.

In the case where the sought neighbour RA does not yet exist, no further action will be taken by the originating RA and the routing index entry will remain as an ontology neighbour. The decision to take

no further action in this case is based upon the fact that when the un-locatable neighbour RA is created, it will seek out its semantic neighbours using the process described. Given that the determination of semantic neighbours is based upon the same ontological information using the same SRMetric implementation, then the originally sought neighbour connection will be created at this time.

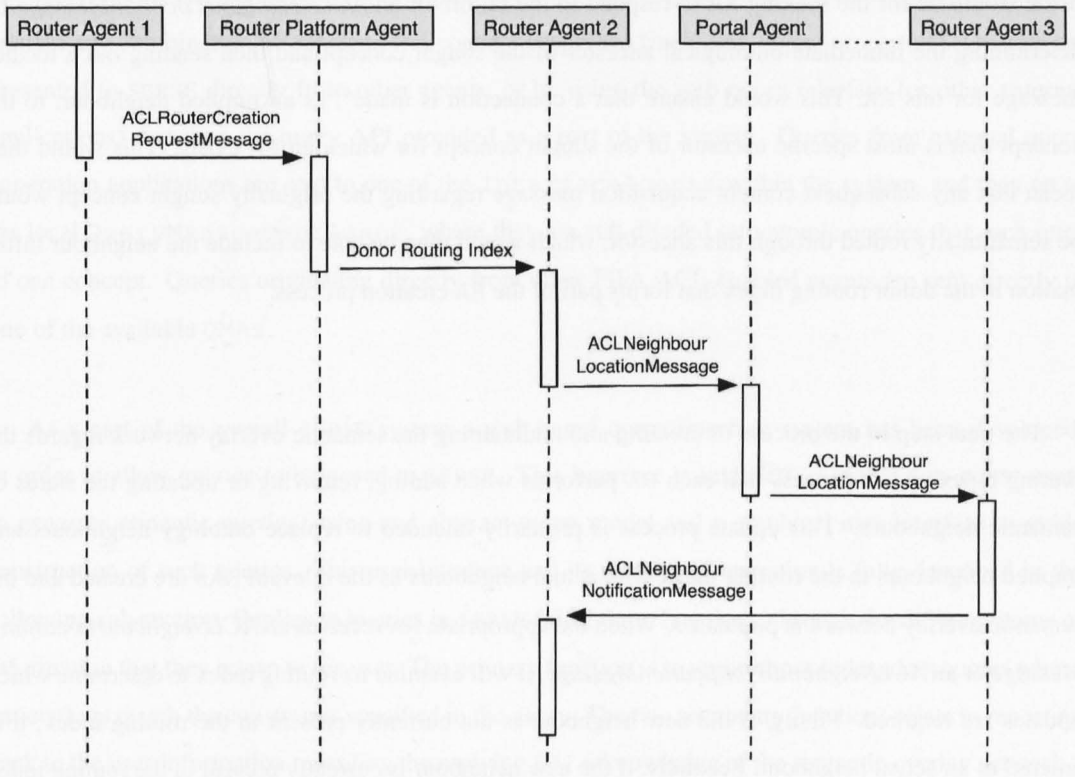


Figure 5.1: Sequence diagram illustrating neighbour location.

This process of RAs autonomously seeking out their own semantic neighbours also ensures that any new ontological information (due to the notification of the availability of new ontologies) is integrated into the the existing semantic overlay network, that has been generated on the basis of the previously available ontologies. This integration is ensured by the fact that any RAs created after the ontology notification will have this ontology included in the list of available ontologies it is provided with on creation. The determination of this RA's semantic neighbourhood then includes concepts from the new ontology, and the new RA will then seek to make connections with these concepts. Furthermore, once RAs for these new concepts are created, they will seek out their semantic neighbours on the basis of all the ontological information, and so connect themselves into the semantic overlay network.

If the system were configured differently, and the RAs had access to different ontological information or used a different means to determine semantic relatedness, there would be no such guarantee that

the connection would be made once both RAs existed. In this case, there are a number of different approaches that might be pursued to ensure that these semantic neighbour connections are completed. Firstly, the RA seeking out a neighbour that does not yet exist could revert to a neighbour location behaviour, in which it periodically re-sends the *ACLNeighbourLocationMessage*, and so will eventually discover its neighbour, after that neighbour is created. An alternative means to ensure the connection is made would be for the seeking RA to respond to the failure of an *ACLNeighbourLocationMessage* by determining the immediate ontological ancestor of the sought concept and then sending out a locator message for this RA. This would ensure that a connection is made[4], as an implied neighbour, to the concept that is most specific ancestor of the sought concept for which an RA exists. This would then mean that any subsequent content acquisition message regarding the originally sought concept would be semantically routed through this ancestor, which would then be able to include the neighbour information in the donor routing index that forms part of the RA creation process.

The final step in the process of creating and maintaining the semantic overlay network regards the routing index update process that each RA performs when adding, removing or updating the status of semantic neighbours. This update process is primarily intended to replace ontology neighbours and implied neighbours in the routing index with actual neighbours as the relevant RAs are created and the semantic overlay network is populated. When the appropriate RA receives an *ACLNeighbourLocationMessage* or an *ACLNeighbourNotificationMessage*, it will examine its routing index to determine which updates are required. Firstly, if the new neighbour is not currently present in the routing index, it is entered as an actual neighbour. Secondly, if the new neighbour is currently present in the routing index as an ontology neighbour, the GUID of the agent is recorded and its status is upgraded to that of an actual neighbour. Finally, if the new neighbour is determined to occupy a position in its ontology hierarchy that places it between the RA performing the update, and one of this RA's implied neighbours, i.e., the implied link 'bridges over' the new neighbour, then this implied neighbour is removed and an actual neighbour connection to the new neighbour is entered into the routing index. The other part of this discarded implied neighbour connection is then replaced by the new neighbour RA then seeking out its own semantic neighbours, which will include this previously implied neighbour.

[4]Through a process of iteratively moving up the concept hierarchy until an existing RA is found, which in turn is guaranteed because the root node concepts of all available concepts are always created within SERSE at the time the new ontology is notified to it.

5.5 Answering Queries

The final core function of SERSE is to answer semantically specified queries over annotated resources that have been indexed within the RA network. Such queries are intended to return the URLs of web resources whose recorded annotations match the constraints expressed in the query. Queries to SERSE are expressed in RDQL, defining constraints upon the annotations of the resources sought, and can contain any combination of concepts and concept properties (including object properties). Queries are presented to SERSE directly from other agents, or by using the web query interface (or other external applications) that uses the query API provided as a part of the system. Queries from external query generation applications are sent to one of the `InterfaceAgents` within the system, and then on to the local `QueryManagementAgent`, where they are sub-divided into atomic queries that each refer to one concept. Queries originating directly from other FIPA ACL enabled agents are sent directly to one of the available `QMAs`.

As a part of the overall SERSE system a web-based query interface system has been developed, in order to allow queries to be posed to SERSE. This interface is intended to enable non-expert users to generate semantic queries, using and abstract query model and a graphical user interface to guide construction of such queries. This user interface and its method of operation is fully described in the following sub-section. Replies to queries in SERSE fulfill three functions, through the different items of information that they return to the user. The primary function is to report those indexed resources whose annotations match the constraints specified in the query. The two secondary functions relate to reporting back to the user information regarding the replying `RA's` knowledge of the semantic overlay network – that is, the neighbourhood information contained in their routing indexes. This neighbourhood information is, firstly, intended to enable the user to modify and recompose the original query using the closely related concepts. An additional purpose of the exposure of this knowledge is to support the user in a 'semantic browsing' mode of resource discovery. The way in which queries are answered, the information returned to the user, and the intended purposes of the information are fully described in Section 5.5.3. Finally, the responses from the various atomic queries, that were generated by the QMA from the original complex query, must be recombined by the QMA to provide an answer to the original query. This reply re-aggregation process, and associated 'cleaning' processes, are described in Section 5.5.4.

5.5.1 Formulating Queries in the Web Interface

The SERSE multiagent system accepts queries for annotated resources specified in RDQL, and such queries can enter the system in one of two ways. Firstly, queries can be sent directly to any of the `QMAs`

within SERSE, using the defined *ACLComplexQueryMessage* message format. This entry route is intended for agents outside of the SERSE system to interact directly at an agent communication level. The second entry route for queries is via the `InterfaceAgent`, and this is intended for non-agent systems to interact with the SERSE agent system. The `IA` exposes a socket interface that other applications, user interfaces, etc. can then connect to[5]. Communication of queries and replies over this interface is achieved by use of pre-defined query and response messages, and a provided interface client API that is utilised by the external application. Using this client, different query interfaces can be developed to suit particular needs.

The primary means provided within SERSE for the expression of semantic queries is through the web-based query interface. This interface is intended as a user interface that enables users to visually construct a semantic query, on the basis of an abstract query model. This interface does not require the user to have knowledge of RDQL, and encapsulates a *guided* query construction process where the user selects the concepts, properties, values, etc. in the query from drop-down selection lists, themselves generated from the available ontologies - see Figure 5.2. In defining a query the user is effectively specifying (sets of) instances of the concepts in the available ontologies to be matched against those instances recorded, by the `RAs`, as annotating particular resources. The query model used based on the same RDF semantics as RDQL, constructing the query in terms of RDF triples that may contain variables. The user selects an ontology, then a concept, then a property of this concept, and finally defines the value of the property – as a datatype for datatype properties or as another RDF triple specifying a concept instance. Any one of the concept, property, and value in these triples can be defined as a variable, and for those variables referring to datatype properties further constraints on the value can be defined (minimum and maximum numeric values, etc.).

The interface was developed as a Java applet, in order to have the necessary interaction with the user and dynamic communication with the multiagent system, via the socket interface managed by the servlet component of the interface. On activation the applet sends a request to a known `IA`[6] to obtain an up to date list of the available ontologies currently known to SERSE. Once this list has been obtained, the following process is used to define each triple within the query:

- The applet presents the user with the list of the available ontologies, prompting the user to select an ontology from which to select the definitions to be used in the current query triple.
- On selection of an ontology, the applet communicates with the `IA` to then fetch a list of the

[5]However, this could also be implemented by other means, such as by exposing a web service.

[6]Each interface deployment may be aware of the existence of a number of `IAs` on different agent platforms within the SERSE system, and the servlet must implement its own policy to select between multiple available `IAs`. The currently implemented policy is to make a random selection between the available agents.

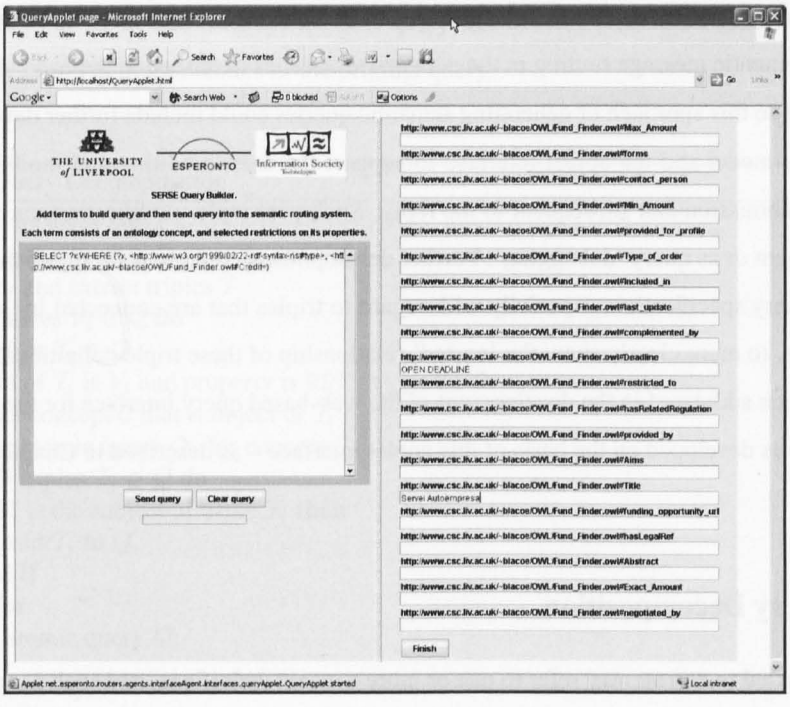


Figure 5.2: Web-based query interface.

concepts defined within the selected ontology, and once returned this list is displayed in a drop-down selection box.

- The user then selects one of these concepts, and, following communication between the applet and the IA, is presented with a list of properties that specify that concept within their domain.

- The user then selects a property whose value they wish to restrict, and the applet again queries the IA for the appropriate range of the selected property. The property range may be that of an XML datatype value, or of a set of concepts whose instances are acceptable values for the selected property. If the property is an object property, the value is defined using one or more additional triples, constructed using this same process nested with one iteration of itself.

- On completion of the definition of a triple, the query is in a state in which it may be submitted to the system. Alternatively, the user may chose to add an additional triple to the query, attaching it to the existing query specification using boolean logical connectives (*AND* and *OR*).

Following one or more iterations of this triple-based query specification process, the user can elect to submit the query to SERSE. At this point the applet automatically generates the RDQL query, using the abstract query model and user's triple definitions. The generated RDQL query is then submitted to the IA, and then sent on to the QMA, in the same manner as for the simple query interface. On receipt of

an *ACLComplexQueryMessage*, from its local IA, the QMA then acts to manipulate the query into a form suitable for semantic message routing in the network of RAs, as described in the following sub-section. Improvements to this approach of generating semantic queries could include further development of the abstract query model and the applet interface to support the user in editing and modifying the query, both before submission and subsequent to the reply. Such post-reply modification may be intended as query refinement or as query 'debugging'. Further developments could improve upon the graphical display of the query specification, especially with regard to triples that are connected by properties in the query – that is, to more clearly show the 'nested' relationship of these triple definitions. Some of these issues have been addressed in the development of the web-based query interface for the QuestSemantic system, that was developed on the basis of this applet interface – as described in Chapter 7.

5.5.2 Query Decomposition

Queries submitted to SERSE may refer to one or more concept definitions, and such queries are, respectively, denoted 'atomic' and 'complex' queries. The semantic message routing process utilised in SERSE requires each message within the network of RAs to be expressed with regard to one concept definition alone, i.e., all messages must be 'atomic'. Therefore, all complex queries submitted to SERSE must be decomposed into a set of atomic queries in order to be semantically routed within the RA network. However, the decomposition should preserve the semantics of the original complex query, and also provide the means to re-compose the set of atomic query replies into a single reply that represents a valid reply to the complex query. This process of query decomposition and subsequent reply re-combination is performed by the QueryManagementAgent – as its primary role in the system. RDF query languages, like RDQL, are intended to function using a centralised repository of meta-data. The sub-division of complex queries into sets of atomic queries, that each relate to only one of the concepts referred to in the original query, is a further unique feature of SERSE. This process enables an individual RA to answer each of the simple queries, using only 'local semantic knowledge to determine each of the replies.

Decomposition of complex queries is achieved by the QMA by first creating an abstract model of the query, that will be used both to guide the query decomposition, and subsequently to guide the reply-set re-composition process. This abstract model of the query is formed by determining which triples within the query refer to concept URIs, and which triples utilise variables that refer to other triples within the query. The query is then syntactically parsed and consistent 'blocks' are identified that group together those triples that make direct or indirect reference to one concept URI. This process may result in some triples being included in more than one of the atomic queries, and this fact is recorded in the abstract query model. At the end of the query decomposition process the QMA has the original complex query, an

abstract model of this query that recorded how the query was decomposed, and a set of atomic queries, that collectively preserve the semantics of the original query.

Algorithm 5 Query Decomposition

Require: QMA receives query $\mathcal{Q}$
Ensure: All atomic queries $\mathcal{Q}_1 \cdots \mathcal{Q}_n$ are sent
1: Parse query and extract triples $\mathcal{T}$
2: **for all** variables $\mathcal{V}_i \in \mathcal{Q}$ **do**
3: Select triple $\mathcal{T}_i \in \mathcal{Q}$
4: **if** subject of $\mathcal{T}_i$ is $\mathcal{V}_i$ and property is `RDF:typeOf` **then**
5: Extract concept $\mathcal{C}$ that is object of $\mathcal{T}_i$
6: Create atomic query $\mathcal{Q}_c$ for concept $\mathcal{C}$
7: **for all** triples $\mathcal{T}_j \in \mathcal{Q}$ **do**
8: **if** $\mathcal{V}_i$ is the subject of triple $\mathcal{T}_j$ **then**
9: Add $\mathcal{T}_j$ to $\mathcal{Q}_c$
10: **end if**
11: **end for**
12: Send atomic query $\mathcal{Q}_c$
13: **end if**
14: **end for**

The query decomposition algorthim can be seen in Algorthim 5. This algorithm shows that the decomposition process first separates the query into the component RDF triples. It then operates upon each of the variables found within the query, as these represent the elements that the individual atomic queries must collectively retrieve. For each of these variables it is first determined whether that variable refers to an ontological concept – as opposed to a data-type object, or an instance URI – by searching through all of the query triples and finding that triple $\mathcal{T}_i$ where the subject of the triple is the variable $\mathcal{V}_i$ and the property is `RDF:typeOf` [7]. For each of the identified 'concept variables', the algorithm then generates an atomic query for that concept $\mathcal{C}$, by querying for the current variable $\mathcal{V}_i$ and inserting into the RDQL triple pattern the triple $\mathcal{T}_i$ that stated that the sought variable was an instance of the concept. For each atomic query, the algorithm then searches through all other triples of the original query and identifies those where the variable sought is the subject of the triple, and adds each of these triples to the current atomic query. In this way the self-consistent 'blocks' that each define the required properties of one variable within the query are identified and re-formed as atomic queries. The issue of the concept hierarchy is handled by, for each concept within the complex query, generating atomic queries for that specific concept and also for each of the concepts subsumed by it – as defined in the ontology. In this way atomic queries are generated for each concept whose instances can be classified as instances of the sought concept. In order to generate atomic queries for all concepts subsumed by the sought concept, the process must utilise additional information from the ontology – restrictions on class membership that

[7] This means that for each concept referred to in the query there is an explicit concept-instanciation statement, however, this is handled in the RDQL generation process within the query interface.

may exclude certain sub-classes, class equivalency statements, etc. Following the query decomposition, the QMA sends each atomic query to the local PA, which forwards each of them to the most *appropriate* RA known to it – that is the RA listed in the PA's routing index that has the highest similarity score with the concept that is the subject of the atomic query.

In addition to forming a set of atomic queries to represent the original complex query, the query decomposition process also records information about the decomposition actions taken. This Query Model records three key sets of information about the decomposition, which are subsequently used to assist in the reply re-combination process. Firstly, the model records the original query and all of the generated atomic queries. Secondly, the model records which triples make reference to other triples in the query by the use of query variables. Finally, the model records the position and type of any logical connectives, from the available set *AND*, *OR*, *NOT*, present in the complex query – enabling the re-combination process to correctly re-apply them to aggregation of the reply-sets.

This query decomposition approach also requires that the QueryManagementAgent must not only create and send the atomic queries, but it must also perform some management tasks upon these atomic queries as part of the overall process. This management includes:

- Seeking to ensure that the replies to all atomic queries forming one complex query have been received before re-combining them. That is, the QMA will wait until all relevant replies have been received[8] before attempting to combine them into a composite reply – using the unique ID labels of the atomic queries recorded in the generated query model.

- Re-sending atomic queries that have failed – due to timeout, temporary router unavailability, etc. The QMA will continue to seek replies to unanswered atomic queries until the timeout period for the original complex query is reached, at which point it will attempt to construct a partial reply using the atomic replies received by that point (see Section 5.5.4). The re-sending of atomic queries has the potential to generate multiple atomic replies where only one is required. This situation is handled by utilising the first received reply to any atomic query in the composite reply generation, and simply ignoring any subsequent atomic replies that have the same query ID.

5.5.3 Answering Atomic Queries

When an RA receives an atomic query, it extracts the query constraints expressed in RDQL and consults its content index to determine whether it indexes instances of the queried concept. If this RA does not

[8]Unless a pre-defined timeout period is exceeded, in which case the QMA will generate a partial composite reply using those atomic replies that have been received within this time.

handle the required concept, the query is routed to the neighbouring RA whose concept is most similar to that queried. Once an atomic query is received by the appropriate RA, the RA matches the RDQL expression against its RDF content index and identifies any instances satisfying the expression. These instances are then matched against resources in the RAs annotation index (within the content index), and the reply to the query is formed by the set of resources annotated with instances matching the query, along with the URIs of the instances themselves. Included in the reply to the query is information about the concepts handled by neighbours of the replying RA. The included neighbour information consists of, for each neighbouring RA, the concept indexed by that neighbour and the RouterAgent GUID. This information can then be used in follow-up queries and enables users to semantically browse from one concept to another closely related concept, using knowledge about the semantic neighbourhood of the replying RA that is revealed by this original query. The query reply is then returned directly to the QMA that dispatched the query, using the return address specified in the query message. The query answering algorithm performing this process can be seen in Algorithm 6.

Algorithm 6 Atomic Query Answering

Require: Query $\mathcal{Q}_s$ received by $\mathtt{RA}_i$ where $\mathcal{C}_s = \mathcal{C}_i$
Ensure: Query result $\mathcal{RS}$ returned to QMA
1: Triple pattern of $\mathcal{Q}_s$ matched against RDF model of $\mathtt{RA}_i$
2: Resulting instances $\mathcal{I}_1 \cdots \mathcal{I}_n$ stored in set $\mathcal{S}$
3: **for all** instances $\mathcal{I}_s \in \mathcal{S}$ **do**
4: **for all** instances $\mathcal{I}_i \in$ content index of $\mathtt{RA}_i$ **do**
5: **if** instance $\mathcal{I}_s$ = instance $\mathcal{I}_j$ **then**
6: Retrieve pointer τ_i to linked web resource $\mathcal{R}_i$
7: Add tuple $\langle \mathcal{I}_i, \tau_i \rangle$ to result set $\mathcal{RS}$
8: **end if**
9: **end for**
10: **end for**
11: Return $\mathcal{RS}$ to QMA

A reply to a query consists of a list of web resources, identified by their URLs, whose annotations match the query constraints, together with the URIs of the relevant annotation instances - see Figure 5.3 - with users being able to access both the returned resources and the meta-data annotations presented. In addition, these replies also contain a list of the concepts that form the semantic neighbourhoods of each the responding RAs.

In addition to the query management actions performed by the QMA, the RAs also behave in a manner to support the query management processes:

- By re-sending atomic queries to a neighbour that is the target of the query when its unavailability is possibly only temporary – as indicated by the warning status indicator for the neighbour in the

RA's routing index (see Section 4.2.3).

- Immediate neighbours of absent query-target RAs answering in their place with an empty reply.

5.5.4 Reply Re-aggregation

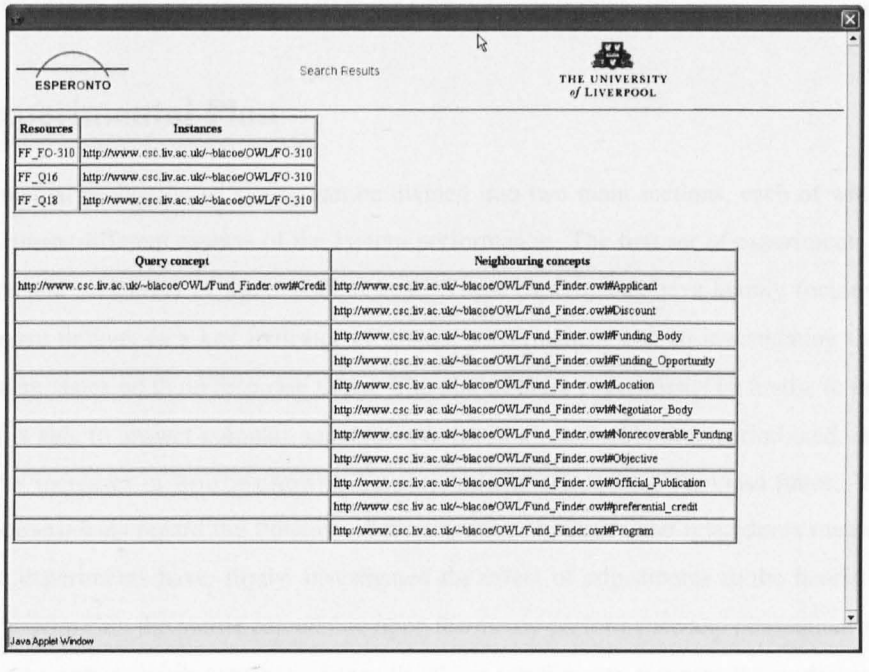
When the QMA has received a reply for each of the atomic queries sent out as part of an original complex query, it re-combines the replies by re-applying the cross-reference connections between them, that were determined when decomposing the complex query. Duplicate resources are also removed during this process. In those cases where the QMA does not receive replies to one or more of the atomic queries, due to the temporary unavailability of the RA handling the relevant concept (or where the relevant RA does not yet exist because there is no annotation data for this concept present within the system), it acts to address the problem. Firstly, the atomic query can be re-sent, and if this also fails (or a query timeout period is reached), the QMA forms a partial reply to the query using that information available from those replies received. This reply is then sent from the QMA to its local IA, and from there to the originating servlet for presentation to the user.

Algorithm 7 Reply Re-combination

Require: QMA receives all atomic query results $\mathcal{RS}_1 \cdots \mathcal{RS}_n$ for query $\mathcal{Q}$
Ensure: Final result set $\mathcal{RS}_f$ sent to the IA that submitted $\mathcal{Q}$

for all result variable $\mathcal{V}_a \in \mathcal{Q}$ **do**
  Retrieve $\mathcal{RS}_b$ for concept $\mathcal{C}_c$ identified by $\mathcal{V}_d$
  for all variables $\mathcal{R}$ in $\mathcal{Q}$ referring to another concept $\mathcal{C}$ **do**
    Retrieve instance $\mathcal{IA}_e$ from $\mathcal{RS}_f$ identified by $\mathcal{R}_g$
    Retrieve $\mathcal{RS}_h$ for concept $\mathcal{C}_i$ identified by $\mathcal{R}_g$
    for all instances $\mathcal{IB} \in \mathcal{RS}_j$ **do**
      if $\mathcal{IA}_e = \mathcal{IB}_l$ **then**
        Retrieve URL (τ_m) of resource $\mathcal{RES}_n$ linked to $\mathcal{I}_o \in \mathcal{RS}_p$
        Add tuple $\langle \mathcal{IA}_e, \tau_m \rangle$ to working set $\mathcal{WS}_q$
      end if
    end for
  end for
  for all $\mathcal{WS}$ **do**
    for all resources $\mathcal{RES}_r$ in result set $\mathcal{WS}_s$ for $\mathcal{V}_a$ **do**
      if $\mathcal{RES}_r \in$ any other result set $\mathcal{WS}_t$ **then**
        Extract instance $\mathcal{I}_u$ linked to $\mathcal{RES}_r$ in $\mathcal{WS}_s$
        Add tuple $\langle \tau_i, \mathcal{I}_1 \cdots \mathcal{I}_n \rangle$ to $\mathcal{RS}_f$
      end if
    end for
  end for
end for
Send $\mathcal{RS}_f$ to IA

The query reply re-aggregation algorithm can be seen in Algortihm 7. This algorithm demonstrates the process by which the cross-references between the query variables are resolved with respect to the corresponding reply-sets. This re-combination uses the information recorded in the relevant query model, to guide the cross-reference resolution and avoiding the need to repeatedly search through the reply-sets for the variable references. The re-combination process also re-applies any logical connectives present between the triples in the complex query, as also recorded in the query model. Once the re-aggregated reply has been generated by the QMA, it is then returned to the IA and on to the originating client servlet. This servlet then generates the query response page, as shown in Figure 5.3. Responses to queries are displayed as lists of web resources, identified by URLs, that match query constraints together with the URIs of the instances that annotate them. In addition, query replies also contain a list of the concepts that are neighbours of each the responding agents. This enables follow-up queries in which the original query is modified by changing the property values of concepts, exchanging one concept for a similar one, broadening or narrowing a query by substituting ontological ancestors or descendents of a concept, etc. This combination in one system of support for both direct querying for web resources and browsing of available resources, both using the semantic of the resources and queries, is a novel approach to locating and retrieving semantic web resources. At present such query modification, or 'semantic browsing' across neighbourhoods, is applied manually via the query interface, but this information could be applied to automatic query modification procedures.



Resources	Instances
FF_FO-310	http://www.csc.liv.ac.uk/~blacoe/OWL/FO-310
FF_Q16	http://www.csc.liv.ac.uk/~blacoe/OWL/FO-310
FF_Q18	http://www.csc.liv.ac.uk/~blacoe/OWL/FO-310

Query concept	Neighbouring concepts
http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Credit	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Applicant
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Discount
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Funding_Body
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Funding_Opportunity
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Location
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Negotiator_Body
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Nonrecoverable_Funding
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Objective
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Official_Publication
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#preferential_credit
	http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Program

Figure 5.3: Query results display.

Part III

Evaluation

Chapter 6

Experimental Evaluation

This chapter presents an experimental evaluation of the SERSE *system, investigating both the overall function of the query answering process, and focusing on the operation of the underlying semantic similarity and relatedness metric, and its comparison with related systems.*
Section 6.1 describes the intentions of the experiments conducted, and then section 6.2 describes the general experimental proceedure, and provenance of the data used in the experiments. Section 6.3 details each of the different specific experiments conducted, and presents their results along with a brief analysis and discussion of the meanings of these results. Finally, Section 6.4 presents an overview of the main conclusions that can be drawn from the experimental results.

6.1 Experimental Plan

The experimental evaluation of SERSE can be divided into two main sections, each of which investigate and evaluate different aspects of the system performance. The first set of experiments regard the performance and scalability of the overall system. These experiments have mainly focused upon the query response timings as a key indicator of system performance, and on investigating the effect of various system states on these response times. The aim of these experiments is, firstly, to demonstrate that SERSE is able to answer complex semantic queries in a reasonable time period, and, secondly, to evaluate how increases in `RouterAgent` network density affect these response times. The second section of experiments regard the function of the semantic similarity and relatedness metric (SRMetric). These experiments have, firstly, investigated the effect of adjustments to the heuristic element weightings used within the metric calculation upon the metric performance and consequent effect upon the `RouterAgent` network topology. Secondly, these experiments compare the performance of the SRMetric with other systems that perform calculations of semantic similarity and relatedness between ontological concepts.

6.1.1 System Performance and Scalability

The two main variables that can directly affect the performance of SERSE, in terms of response times to queries, are: the degree of complexity present in the semantic query, and the density of the the RouterAgent (RA) network. In the first case, a semantic query expressed in RDQL can vary in complexity in a number of different ways, and each of these will have differing effects on the performance of SERSE, particularly in terms of effects on the processes to decompose complex queries and to re-aggregate the results from a number of atomic sub-queries. In the second case, the number of RAs present within the system will affect the query response times, both in terms of the number of RAs a query will pass through on a semantic routing path, and in terms of the number of semantic neighbours that each RA on the routing path has to consider when performing the routing process.

The first section of three experiments were intended to investigate the different effects of these variables on the query response time of the system:

- Experiment 1 examines the query response times achieved when varying the query complexity, in a number of pre-defined ways.
- Experiment 2 examines the query response times achieved when varying the semantic message routing path-length.
- Experiment 3 examines the query response times achieved when varying the semantic neighbourhood density of the RAs on a message routing path.

6.1.2 Semantic Relatedness Metric Evaluation

The key element of the entire SERSE system is the Semantic Relatedness Metric, which underlies both the formation of the semantic neighbourhoods (and consequently the entire RA network topology), and the semantic routing of queries and other messages within the RA network. The second section of three experiments were intended to investigate, firstly, how variation of the metric's parameters affect its function, and, secondly, how the function of the SRMetric compares with other systems performing a similar task. Experiment 4 examines the effects of varying the internal, heuristic weighting parameters of the SRMetric upon the network formation effectiveness and query response times. The results of these evaluations were then used to 'tune' the metric performance by adjusting the relative values of the weighing factors, in order to achieve an optimum metric performance. Experiment 5 compares the performance of the SRMetric with two other systems that measure semantic similarity between concept definitions (i.e.,

ontology alignment systems). This experiment compares the semantic neighbourhoods (within a single ontology) that are determined by the different systems, and measures these against a manually calculated 'Gold Standard'. This 'Gold Standard' determination of the semantic neighbourhoods of each of the concepts within the ontology was determined by an independent third party to represent an optimal set of neighbourhood connections based upon the concept semantics. Finally, Experiment 6 examines how the SRMetric compares with an ontology alignment tools when determining neighbourhood links between two heterogeneous ontologies: that is, how well does the SRMetric perform when comparing concepts defined in different ontologies, when compared to ontology alignment systems.

6.2 Experimental Procedure

Before proceeding with the descriptions of the experiments themselves, it is necessary to briefly cover the three vital points of experimental procedure: one, what systems were experiments performed upon; two, where the test-data used in the experiments comes from; and, three, how the result measurements are obtained from the system during the evaluation. Therefore, the next sub-section will describe the experimental environments, in terms of the hardware, software and network setups of the experimental systems. The following sub-section describes the provenance of the various sets of data used in the different experiments, and the reasons for selecting these data-sets. The final sub-section then describes how the different experimental measurements were obtained from the system, and what these figures represent.

6.2.1 Experimental environments

The environment for an experiment consists of the hardware, software and network conditions applying during the execution of the experiment. The different experiments presented in this chapter were performed in different two experimental environments, with all the experiments utilising timed results being performed within one environment, and the other non-performance related experiments being executed within a different environment.

The first experimental environment was used for Experiments 1, 2 and 3 – so that the timing results for these experiments were comparable. This environment consisted of:

1. Hardware – Asus Laptop PC with Intel Pentium M 2.0GHz. CPU, 1Gb RAM, and supporting 100Mb/s Ethernet networking.
2. Operating System – Microsoft Windows XP, upgraded with Service Pack 1.

3. Other Software – Resin Web Application Server / Servlet container. Sun Java SE SDK 1.4.

4. Network – 100Mb/s Ethernet network carrying IP/TCP messages.

The second experimental environment was used for Experiments 4, 5 and 6 – as these experiments did not report performance results that required comparison with results from the earlier experiments. This environment consisted of:

1. Hardware – Apple iBook G4 with PowerPC 1.3GHz. CPU, 1Gb RAM, and supporting 100Mb/s Ethernet networking.

2. Operating System – Apple OS X 10.4 (Tiger).

3. Other Software – Tomcat Web Application Server / Servlet container. Sun Java SE SDK 1.4.

4. Network – 100Mb/s Ethernet network carrying IP/TCP messages.

6.2.2 Input Data-Sets

Given the specific task of SERSE – to index and retrieve semantically annotated resources – it is necessary to have test data-sets that contain three types of data: an ontology encoded in OWL or RDFS, a set of RDF instances of concepts from that ontology (a knowledge-base), and a set of web resources, each annotated with one or more instances. This requirement means that there have been limited options for test data-sets to use with SERSE, as the availability of such data is severely limited.

Manual creation of test data-sets was not feasible due to limited time and resources, and would not have significantly have improved upon the test data-sets available within the Esperonto project – – both of which have been utilised. Consideration was given to the use of either partially or entirely synthetic data-sets obtained through a process of automatic generation of ontologies, concept instances, and resource annotations. Limited use of synthetic data contributed to the Galen data-set (described below) by providing randomly generated resource URLs as placeholders for resources that actually embody the semantics expressed in their annotations. However, the decision was taken not to make further use of synthetic data-sets because of the potential skewing of the semantics expressed within the ontologies, instance-sets and resource annotations. In this context, 'skewing the semantics' would include any ontological or metadata statements that did not reliably reflect the semantics implied by their context and usage, and which disrupted the semantic integrity of related statements. Any such disrupted semantics could then lead to an anomalous topography of the semantic overlay network inter-connecting SERSE which relies on these semantics. Such anomalous topographies could severely interfere with the

operation of SERSE, and this would invalidate any experimental results obtained in this way. There are a number of ways in which, wholly or partially, synthetically generated data-sets could skew or disrupt the overall semantics of the information available to SERSE. It is possible, with varying degrees of success, to generate each of the three required elements of a SERSE data-set – but each of these synthetic elements could produce 'skewed' semantics in different ways.

- **Ontologies** – Ontologies should seek to express the semantics of the entities they describe, both in the individual concept descriptions and in the context provided by the concept's position in the ontology hierarchy. That is, an ontology should be internally consistent. Synthetic generation of ontologies therefore requires some means to artificially generate meaningful semantic descriptions, that are inter-related in a way consistent with those descriptions. Clearly, any ontology generation process requires a source of knowledge upon which to base the ontology construction, in order to provide the required semantics. The learning of ontologies from, usually large collections of, knowledge sources is a research area in itself, but even the best performing approaches are not able to produce sufficiently rich and precise semantic descriptions to enable SERSE to operate as intended.

- **Instances** – As with ontological descriptions of concepts, instances should accurately reflect the intended semantics of the concept they instanciate. Specifically, the label attributed to the instance and the values attributed to the appropriate properties should 'make sense' in terms of the described semantics, so that an indexing system can make use of them. SERSE, like any other system seeking to index semantically annotated resources, must consider the entire index population when determining inter-connections between annotations, and so the entire semantics of the metadata annotations should be self-consistent. Therefore, synthetic generation of instances in isolation is not feasible, they should reflect some underlying domain of knowledge, rather than simply being a collection of un-related instances consisting of random properties and values. A realistic knowledge-base of instances could be generated using real knowledge sources, and is usually considered to form a part of ontology learning. However, as noted above, current approaches do not produce very precise or detailed semantics. In addition, the algorithms employed in the ontology learning process also have the potential to introduce some skewing of the semantics, that would then affect the operation of SERSE.

- **Annotations** – Semantic annotations attached to digital resources, that reference instances of ontological entities in their statements, should also accurately reflect real world semantics in order to behave 'realistically' within a semantics-based system such as SERSE. That is, resources often refer to the instances of a number of ontological concepts, and, in order to make full use of

the semantic query processing abilities of SERSE, this sort of cross-referencing should remain intact. For example, a web-page describing a university building could be annotated with metadata statements that reference instances of the concepts UNIVERSITY and BUILDING. A semantic query can require that matching resources refer to instances of both these concepts, and that those instances have properties that meet certain specified criteria. Therefore, any synthetic annotation generation process for SERSE should generate a consistent collection of annotations that reflect the domain of knowledge required. A sufficiently complex synthetic annotation generation process was beyond the scope of this thesis, and would amount to creating an automatic processes to semantically annotate resources on the basis of given ontologies and their extensions, such as the Annotation System component of Esperonto. The data-sets available within the Esperonto test-cases contain collections of annotations generated by the Annotation System, and these have been used within these experiments. However, the annotation process requires the creation of specialised resource wrappers, and there are only two sets of case-study wrappers available from the Esperonto project. Finally, a limited form of random generation of synthetic annotations was used within the Galen data-set (described below), but the resulting collection of metadata reqired extensive modification to create appropriate instance co-occurence information, for subsequent use by SERSE – which would be impractical for large-scale annotation generation.

Therefore, out of the limited choice available, different real data-sets have been utilised in different experiments, dependent on the experimental aim. Furthermore, in some cases it has been necessary to utilise partial data-sets, with placeholders used as substitutes for resource URIs, but only when this would not effect experimental results. These composite data sets, consisting of ontology, knowledge-base and annotated resources, are:

1. *FundFinder* – Test-case data-set from the Esperonto project, developed to represent public funding opportunities for SMEs in Catalunya. Consists of 49 concepts and 118 instances.

2. *CulturalTour* – Test-case data-set from the Esperonto project, developed to represent a collection of Spainish cultural artefacts and information resources. Consists of 58 concepts and about 61000 instances.

3. *Galen* – A data-set generated using the OWL translation of the full Galen ontology from the CO-ODE project[1]. The Galen ontology represents large-scale medical and clinical terminologies, and consists of around 10,000 defined concepts. The existing Galen data-set already contained an ontology and a set of instances, with only the annotated resources element not present. Therefore,

[1] www.co-ode.org/galen/

to complete the data-set, random URIs were generated to represent web resources annotated with instances of the ontology's concepts.

4. *UKResearchGroup (UKRG)* – An ontology and knowledge-base, created at the University of Liverpool, to represent the members, projects and publications of an academic research group. Consists of 38 concepts and 173 instances.

5. *OAEI* – Consists of the benchmark ontology from the 2005 Ontology Alignment Evaluation Initiative (OAEI). This ontology was constructed as the reference ontology for the ongoing evaluation effort, and is used as the basis of all the evaluation tests and benchmarks.

6. *QOM* – Consists of two ontologies, with overlapping domains, selected from the Quick Ontology Mapping (QOM) system evaluation test-data: Russia1A and Russia1B. Both of these ontologies represent the domain of tourism in Russia, but in significantly different ways. Usage of the QOM evaluation data provided both a pair of ontologies concerning the same domain, and enabled direct comparison of the results with those of QOM.

6.2.3 Result Data Capture

The different experiments conducted required differing types of data to be collected from SERSE in order to quantify the results. Experiments 1,2 and 3 all required query response times to be recorded from a functioning SERSE system. Experiment 4 also required query response timings to be recorded, and required the recording of average neighbourhood sizes for all `RouterAgents` present in a SERSE system. Experiments 5 and 6 required the recording of individual `RouterAgents'` neighbourhoods, specifying which concepts were determined to be within the neighbourhoods of other concepts, and then comparison of these neighbourhood sets with other such sets – generated by other means. These different types of data were collected from the SERSE system by different means, as follows:

- Query response timings were generated from the system logs, after the experimental runs were completed. The logging process (implemented using *log4j*) was specifically adapted for the experiments to log data from all agents in the system into a single composite log file. This file then recorded details of all queries posed to the system, and tracked the transmission of all messages (sub-queries, sub-query responses and query response) resulting from receipt of a query. The actual timing results were semi-automatically extracted from the logs by use of special XML tags indicating which log entries contained timing data, and to which query these times applied.

- Neighbourhood sizes were also recorded in the system logs. The logging system was again specifically adapted for this experiment, recording every addition to an `RA's` neighbourhood, and these

log entries were identified with XML tags that enabled semi-automatic extraction, and subsequent calculation of average neighbourhood sizes.

- Neighbourhood compositions were determined by use of the SRMetric operating within a 'test harness'. This enabled the specification of two source ontologies, from which the concepts were extracted. Each of these concepts was then semantically compared with all of the other concepts from both ontologies, and for each concept a semantic neighbourhood was determined and recorded.

6.3 Experiments

In this section we provide the detailed experimental descriptions for the six different experiments performed upon SERSE and the SRMetric as a part of this evaluation. Each experiment is described in terms of the experimental aim, the experimental method employed, and the specific experiment results, including pictorial representations of these results. An analysis and discussion of the various experimental results is presented in the following section.

6.3.1 Experiment 1: Query response times.

The primary purpose of this experiment is to demonstrate that SERSE is able to answer queries within an acceptable time, for a end-user facing query engine. A secondary purpose is to show that process of decomposing queries, obtaining replies from different RAs and then re-aggregating replies does not impose an unreasonably high overhead on the query answering function. Therefore, the primary hypothesis being tested is that responses to queries are received within 1 second of query submission from the user interface. The rationale for this time limit is that 1 second clearly represents a timely response from the point of view of the end-user, and up to 5 seconds could be deemed acceptable[2]. The limit selected is towards the lower end of this acceptable range because the experiment is being performed upon a relatively small distributed index, and response time would be expected to increase with index size – though only in an approximately linear manner (see Experiments 2 and 3). The secondary hypothesis is that the increasing complexity of the queries does not increase the response time when compared to separately querying the individual indexes involved in the response. That is, a complex query that requires responses from 7 RAs (such as query 20 in the FF data-set) should not take longer that 7 queries to a single RA (as with queries 1 to 5 in each data-set). This is intended to demonstrate that the cost of decomposing the query and recomposing the reply is not greater than the time gained by submitting the

[2] As supported by comments from independent end-users of QuestSemantics (see Chapter 7).

queries in parallel when compared to executing them in sequence against a single index (with similar communication overheads).

The conditions under which the experiment was carried out were as follows:

- System – Asus Laptop PC with Intel Pentium M 2.0GHz. and 1Gb RAM, Microsoft Windows XP, running SERSE within a Sun Java SE SDK 1.4 JVM and using the Resin Web Application Server for hosting the web interface.
- Dataset – Experiment 1 used the FundFinder and CulturalTour data-sets, developed as test-case demonstrations within the Esperonto project. Experiment 1 used all three elements of these data-sets, plus a set of 20 pre-defined queries specified for each data-set.

Aim – To determine the response time of the system when answering queries for annotated resources, by recording the average round-trip reply time for a set of twenty pre-defined queries. This experiment also tests the performance of the query management processes, as the queries increase in complexity (in terms of the number and types of constraints they contain). Finally, the experiment provides a base-line comparison for the results, by executing the same queries over the same knowledge-base when it is stored in a single RDF model – as opposed to being distributed across a number of `RouterAgents`. This comparison thus offers some insight into the overall processing costs involved in the use of the multiagent system.

Method – The annotated resources, from the FF and CT data-sets, are notified to SERSE, which results in the construction and population of a network of `RAs` to index these resources. A single experimental run consists of each of the twenty pre-defined queries being input to SERSE, with each separate query and reply cycle performed consecutively and independently. For each of the data-sets a total of one thousand experimental runs were conducted, each recording the time taken to perform every query and response cycle, and then calculating the average time taken for each of the queries. For comparison purposes the same set of queries were posed to a single RDF model (built in Jena) containing all of the instance meta-data in the data-set. As before, the queries were repeated 1000 times, and the average response time recorded.

The twenty pre-defined queries were designed to vary in complexity by changing the types and numbers of constraints present in the queries. The four main ways in which the types of constraint can vary, in order of computational complexity, are:

1. querying for a single variable with no other variables present in the query;

```
SELECT ?x, ?z WHERE
(?x, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#Discount>)
(?x, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Aims>, ?y)
(?y, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#Objective>)
(?y, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#objectiveName>, "Company_Creation")
(?x, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#negotiated_by>, ?u)
(?u, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#Negotiator_Body>)
(?u, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#actsForBody>, ?t)
(?t, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#State_Funding_Body>)
(?z, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#subvention>)
(?z, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Deadline>, "30-juny-2005")
(?z, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#Aims>, ?w)
(?w, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#Objective>)
(?w, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#objectiveName>, "Quality")
(?z, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#hasRelatedRegulation>, ?v)
(?v, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, <http://www.csc.liv.ac.uk/~blacoe/OWL/
Fund_Finder.owl#Diari_Oficial_de_la_Generalitat_de_Catalunya>)
(?v, <http://www.csc.liv.ac.uk/~blacoe/OWL/Fund_Finder.owl#date>, "26/04/1996")
```

Figure 6.1: Query 20 from FundFinder dataset.

2. querying for multiple variables with no cross-reference between the variables;

3. querying for a single variable with cross-reference to one or more non-result variables;

4. querying for multiple variables with cross-references to one or more non-result variables.

Within each of these four groups of queries, five separate queries are defined, each progressively increasing the number of query constraints. In this way, we obtain a set of twenty queries that represent an ordered spectrum of query complexity, in order to fully test the performance of the query decomposition and reply re-aggregation processes. Figure 6.3.1 illustrates the most complex query we posed for the FF data-set. This query demonstrates the multiple result variables (x and z), the non-result variable cross-references (y, u, w and v), and the multi-layer cross-references (x to u and then to t) that all contribute to the high complexity of this query.

Results – Figures 6.2 and 6.4 show the average response time for each of the twenty queries, using the two data-sets FF and CT respectively. As would be expected, the response time increases significantly with the complexity of the queries - given that query 1 involves only one responding RA and query 20 involves seven RAs in the response. Comparing these results with those obtained by querying the single RDF model, shown in Figures 6.3 and 6.5 for the FF and CT data-sets respectively, we can see that the relative increase in response time due to increasing query complexity is similar (though with interesting exceptions, as discussed below). In addition, comparing the actual response time for the same queries using SERSE and a single RDF model, we can see that the cost of distributing the semantic resource

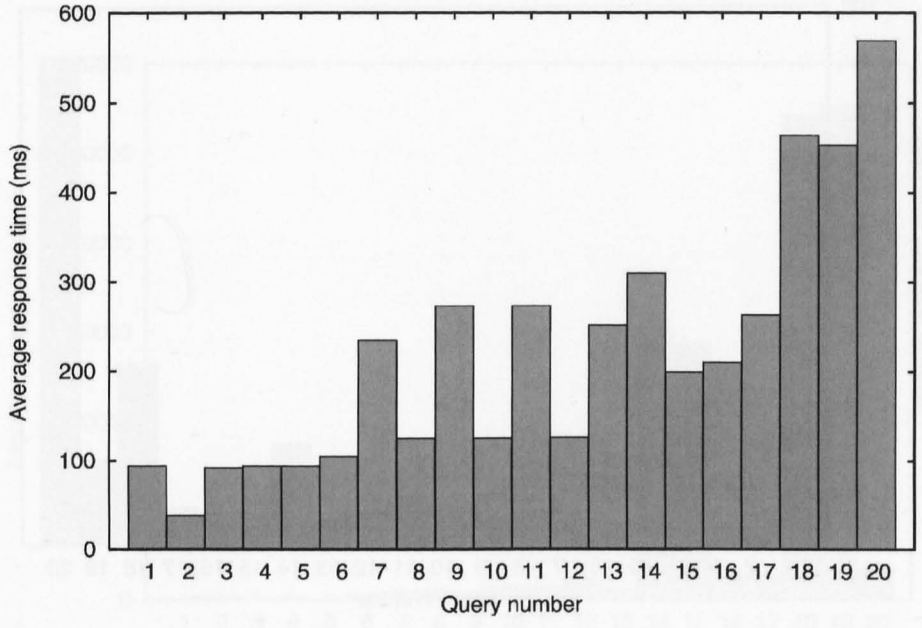


Figure 6.2: Average query response times for FF dataset in SERSE.

index across a multiagent system varies between 44 ms and as much as 483 ms. Nevertheless, the response times recorded demonstrate that SERSE is able to provide timely answers to queries, despite the performance overhead of the multiagent system.

Overall the results of Experiment 1 concerning the response times for different queries shows that SERSE requires more time to answer the queries, when compared with executing the same queries against the same meta-data stored in a single RDF model. This is the predicted result, as SERSE adds a number of different overheads to the query answering process, such as the messages exchanged to enable the semantic message routing and the system's self management, and the query decomposition and reply re-aggregation processes. It should be noted that the results for querying the single RDF model are not intended as a "control" comparison for SERSE. The single RDF model is strictly limited in size, based upon the memory capacity of the host machine, whereas SERSE is specifically intended to be distributed across a number of host machines in order to be able to manage an RDF model that is orders of magnitude larger.

However, within these results, there are some anomalies with queries numbered 14, 15, 18, and 19 for the Cultural Tour data-set. We have identified a number of factors contributing to these anomalies, largely relating to the large number of resources and instances returned by each atomic query, requiring cross-matching and duplicate detection, and increasing message transmission time. These issues are directly related to the specific ontology being handled, and in particular with the relative instance

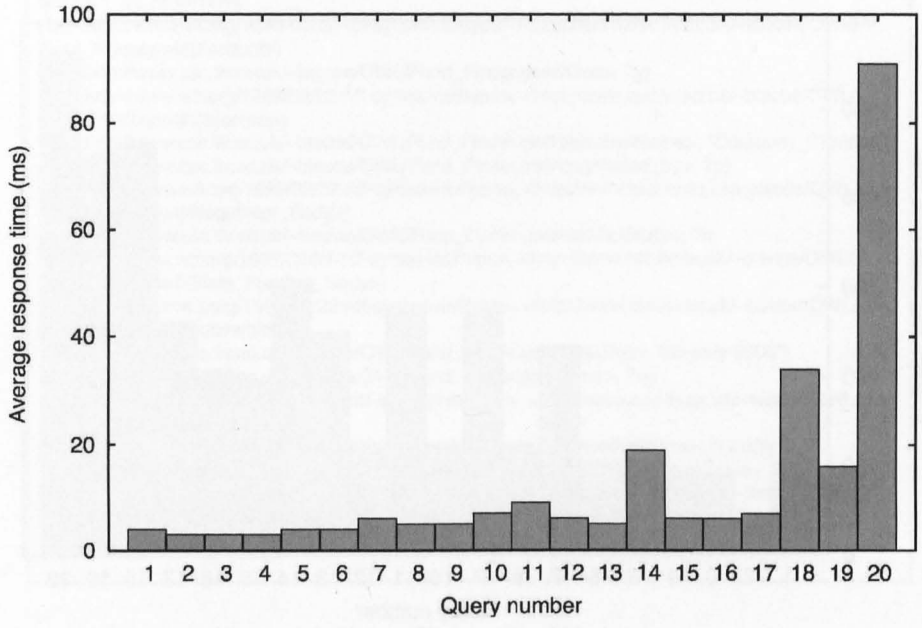


Figure 6.3: Average query response times for FF dataset in Jena.

distribution within them. The Cultural Tour data-set has some unusual features in this respect: around half of the instances instantiate only three concepts, while the other half populate the remaining classes, and this unusual distribution is the primary cause of the anomalies. By creating a network of RAs that are each responsible for one concept out of all those known to the system, SERSE is splitting the *global* RDF model into smaller components, whose size depends on the number of instances for each concept. This can have a significant effect on response time for queries requiring cross-referencing of atomic query results from different RDF models. Normally such queries would not cause efficiency issues, as shown by the response times obtained for the Fund Finder data-set (Figure 6.2). However, when the RAs involved in the query answering handle particularly large extensional models (as in the Cultural Tour data-set where three classes each have around 10000 instances), then the performance degrades significantly.

Conclusion – The experiment broadly supports the primary hypothesis, that responses to queries are received within 1 second of query submission, except under certain conditions described above. In general, the query response time is of the order of 0.1 sec to 1 sec – with the exception of a number of queries in the CT data-set. Of these, queries 12, 16, 17 and 20 have response times in the order of 1 to 4 seconds, which, although outside the strict bounds set by the hypothesis, is still within the upper identified limit of 5 seconds. Queries 14, 15, 18 and 19 have response times that are significantly greater than the hypothesis limit, being in the order of 60 to 110 seconds. The primary reason for this, as identified

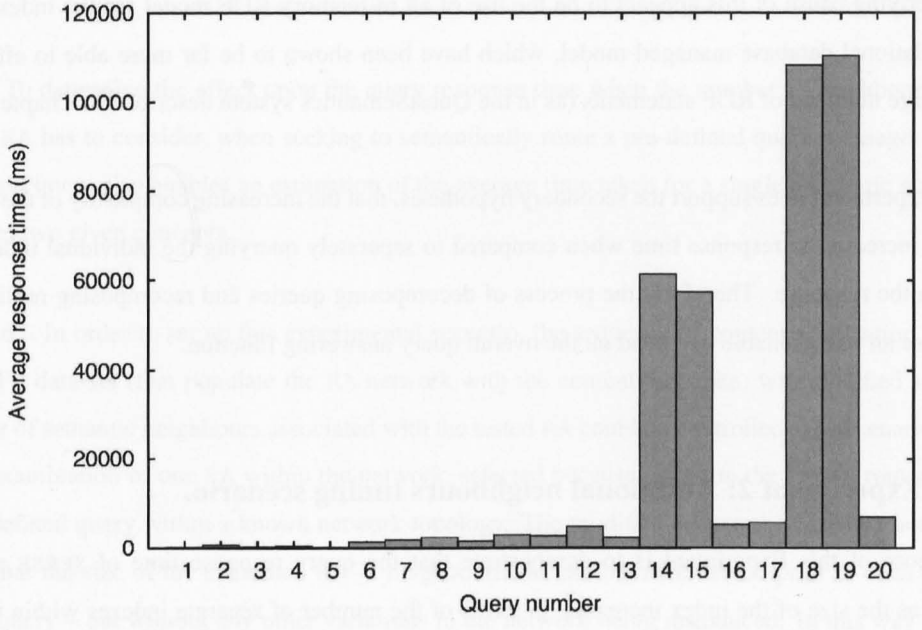


Figure 6.4: Average query response times for CT dataset in SERSE.

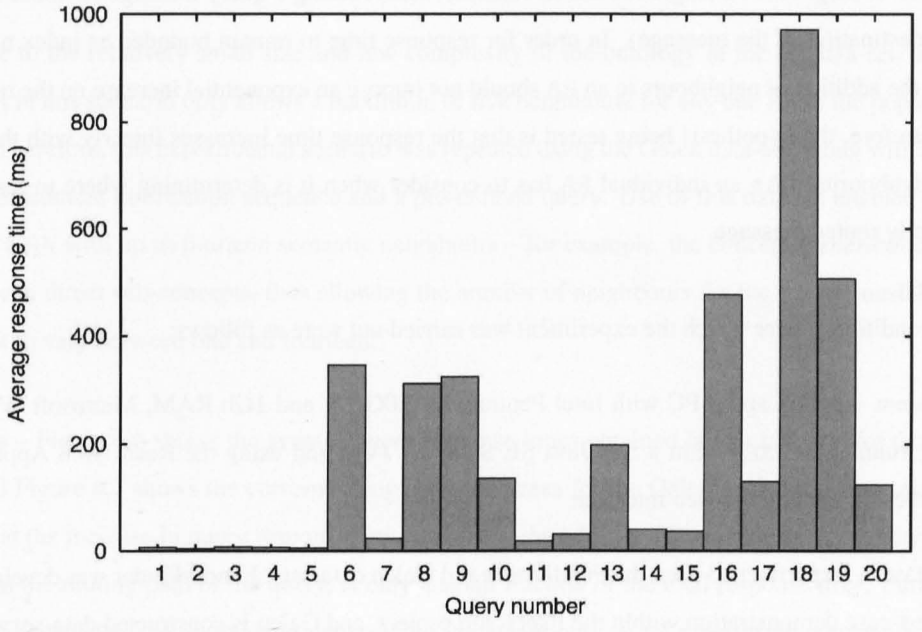


Figure 6.5: Average query response times for CT dataset in Jena.

above, regards inefficiency in the RDF content index handled by each RA – which appears to be overwhelmed by the numbers of instances recorded in the indexes for four of the concepts in the data-set[3]. The underlying cause of this appears to be the use of an in-memory RDF model for the index, rather than a relational database managed model, which have been shown to be far more able to efficiently handle large numbers of RDF statements (as in the QuestSemantics system described in Chapter 7).

The experiment does support the secondary hypothesis, that the increasing complexity of the queries does not increase the response time when compared to separately querying the individual indexes involved in the response. Therefore, the process of decomposing queries and recomposing replies does not impose an unreasonable overhead on the overall query answering function.

6.3.2 Experiment 2: Additional neighbours timing scenario.

The purpose of this Experiment is to demonstrate that the query response time of SERSE remains bounded as the size of the index increases, in terms of the number of separate indexes within the network. That is, more indexes within the system means that, on average, each RA within the system has more neighbours. This in turn means that, on average, each RA has to consider more potential targets for a message, thus adding to the query processing time overhead at each RA on the route of a message. The additional cost imposed can be examined by considering the effect on query response time of varying the number of neighbours a single RA has to consider when routing a query message (without altering the final destination of the message). In order for response time to remain bounded as index numbers increase, the addition of neighbours to an RA should not impose an exponential increase on the response time. Therefore, the hypothesis being tested is that the response time increases linearly with the number of neighbouring RAs an individual RA has to consider when it is determining where to forward a semantically routed message.

The conditions under which the experiment was carried out were as follows:

- System – Asus Laptop PC with Intel Pentium M 2.0GHz. and 1Gb RAM, Microsoft Windows XP, running SERSE within a Sun Java SE SDK 1.4 JVM and using the Resin Web Application Server for hosting the web interface.

- Dataset – Experiment 3 used the FundFinder and Galen data-sets. FunderFinder was developed as a test-case demonstration within the Esperonto project, and Galen is constructed data-set consisting of a fragment of the full Galen ontology and knowledge-base along with a semi-synthetically

[3]Compounded by the large numbers of properties (greater than 10) for many of the concepts in the ontology.

generated annotation set. Experiment 3 used all three elements of these data-sets, plus a pre-defined query and content notification schedule for each data-set.

Aim – To determine the effect upon the query response time when the number of neighbours an individual RA has to consider, when seeking to semantically route a pre-defined query message, is varied. The experiment also enables an estimation of the average time taken for a single SRMetric comparison between two given concepts.

Method – In order to set up this experimental scenario, the sequence of content notification messages in the FF data-set (that populate the RA network with the content metadata) was modified so that the number of semantic neighbours associated with the tested RA could be controlled. The scenario consists of an examination of one RA within the network, selected because it lies on the known routing path of a pre-defined query within a known network topology. The modified sequence of content notifications mean that the size of the examined RA's neighbourhood can be incremented prior to each execution of the query – but without any other variations in the network being introduced. In this way the query response times associated with each of the neighbourhood sizes is recorded, which demonstrates the effect of varying the number of neighbours (that a RA must consider in order to semantically route a message) upon this response time. The entire experimental run, including all content notifications and queries, is then repeated one hundred times to obtain an average set of results.

Due to the relatively small size and low complexity of the ontology in the FF data-set, use of this data-set in this scenario only allows a maximum of five neighbours for any one RA in the populated network. Therefore, this experimental scenario was repeated using the Galen data-set, along with a suitably modified content notification sequence and a pre-defined query. Use of this data-set enabled examination of a RA with up to fourteen semantic neighbours – for example, the concept *GenericBodyProcess* has fifteen direct sub-concepts, thus allowing the number of neighbours for the RA responsible for this concept to vary between one and fourteen.

Results – Figure 6.6 shows the average query response times obtained in this scenario for the FF data-set, and Figure 6.7 shows the corresponding response times for the Galen data-set. These results illustrate that the increase in query response time caused by the addition of neighbours to an individual RA, lying on the routing path of the query, is only a small fraction of the total response time. Furthermore, the addition of RAs to an existing network, as the neighbours of RAs performing semantic message routing, produces only a small, and approximately linear, increase in query response time – which, in turn, indicates that the system as a whole is scalable. The results of this experiment also enable the

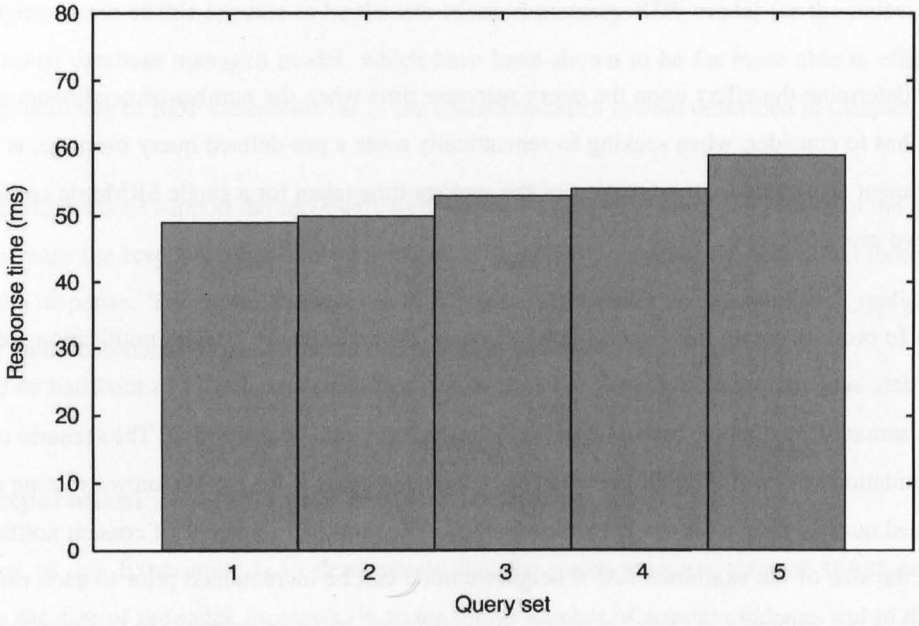


Figure 6.6: Average query response time in Experiment 2 using FF dataset.

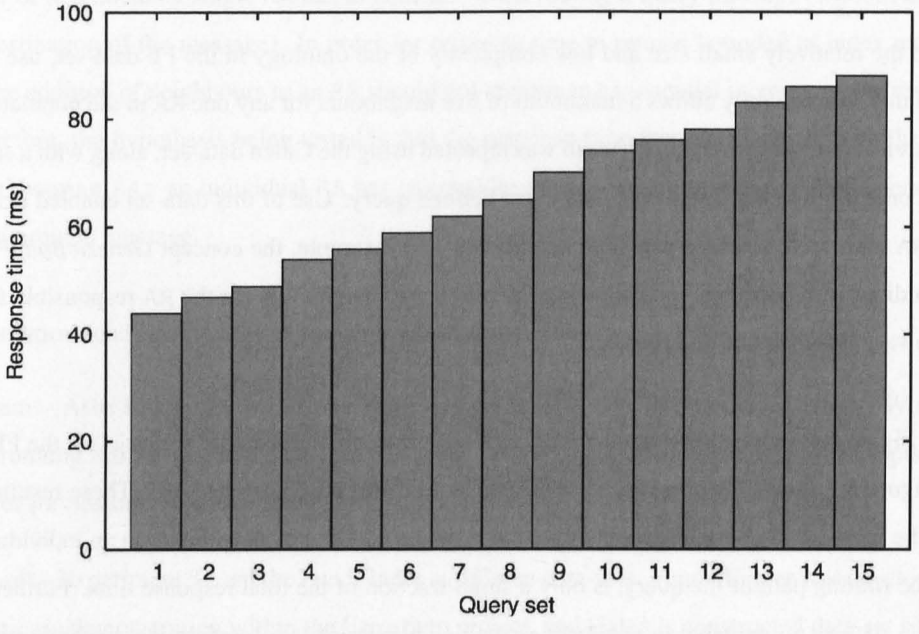


Figure 6.7: Average query response time in Experiment 2 using Galen dataset.

calculation of the average time required for a single SRMetric calculation, and this can be estimated as approximately 3ms. Overall, these results demonstrate that SERSE is able to dynamically adjust its network of RAs, in order to cope with the notification of new content and the consequent addition of new RAs into neighbourhoods, without significantly degrading performance in terms of query response time.

Conclusion – The experiment supports the hypothesis, that the response time increases (approximately) linearly with the number of neighbouring RAs an individual RA has to consider when it is determining where to forward a semantically routed message. That is, the additional cost in response time of additional neighbouring RAs within the system increases by approximately the same amount for each addition. Furthermore, this increase represents only a small fraction of the total response time – about 4 to 6%. This adds supporting evidence that the distributed index in SERSE is able to scale to handle very large numbers of nodes, given that the average number of neighbours for an RA only increases as a fraction of the total number of concepts indexed.

6.3.3 Experiment 3: Additional agents timing scenario.

The purpose of this Experiment is to demonstrate that the query response time of SERSE remains bounded as the size of the index increases, in terms of the number of separate indexes within the network. That is, more indexes within the system means that, on average, each query message will have to traverse more RAs within the system in order to reach its target RA. The message will, on average, need to be received, semantically routed and re-transmitted more times in a larger network, thus adding to the overall query processing time overhead over the whole query routing process. The additional cost imposed can be examined by considering the effect on query response time of varying the number of RAs along the routing path of a single query message (without altering the final destination of the message). In order for response time to remain bounded as index numbers increase, the addition of RAs to the routing path of a message should not impose an exponential increase on the response time. Therefore, the hypothesis being tested is that the response time increases linearly with the number of RAs an individual query message has to traverse in order to be semantically routed to its target.

The conditions under which the experiment was carried out were as follows:

- System – Asus Laptop PC with Intel Pentium M 2.0GHz. and 1Gb RAM, Microsoft Windows XP, running SERSE within a Sun Java SE SDK 1.4 JVM and using the Resin Web Application Server for hosting the web interface.

- Dataset – Experiment 3 used the FundFinder and Galen data-sets. FunderFinder was developed as

a test-case demonstration within the Esperonto project, and Galen is constructed data-set consisting of a fragment of the full Galen ontology and knowledge-base along with a semi-synthetically generated annotation set. Experiment 3 used all three elements of these data-sets, plus a pre-defined query and content notification schedule for each data-set.

Aim – To determine the time effect upon query response time when the number of RAs traversed during the semantic routing of a query message are varied. As in Experiment 2, the aim is to see how the system responds to an increase in workload due to introducing additional RAs along the routing path of a query. However, in this case the experiment shows the additional cost of routing the message through an increasing number of RAs.

Method – In order to setup this experimental scenario, the sequence of content notification messages in the FF data-set (that populate the RA network) was modified so that the number of RAs lying on the known routing path of a pre-defined query could be controlled. The scenario consists of examining one RA within the network, selected because it lies on the known routing path of a pre-defined query within a known network topology. The modified sequence of content notifications mean that each time the query is posed to the system, it has to be routed through an additional RA (created due to an intervening content notification) is order to reach the intended RA - but without any other variations in the network being introduced. In this way the query response times were recorded for each of the different number of RAs on the message routing path, and so would demonstrate the effect on query timings of varying the number of RAs that message must traverse when being semantically routed. The entire experimental run, including all content notifications and queries, is then repeated one hundred times to obtain an average set of results.

However, as in Experiment 2, the FF data-set does not allow much variation of the number of RAs on the path of any one query. Therefore, this experiment was also repeated using the larger Galen data-set. This allows up to fifteen RAs on any individual query path - for example, the concept *TopCategory* has a chain of fifteen successive decendents that allows the number of RAs along the path of a query for *AcuteErosionOfStomach* to vary between one and fifteen.

Results – Figure 6.8 shows the average query response times obtained in this scenario for the FF data-set, and Figure 6.9 shows the corresponding response times for the Galen data-set. As in Experiment 2, these results illustrate that the increase in query response time due to the addition of new RAs along the routing path of a query is only a small fraction of the total response time. Furthermore, the addition of RAs to an existing network, as additional nodes to be traversed during a semantic message

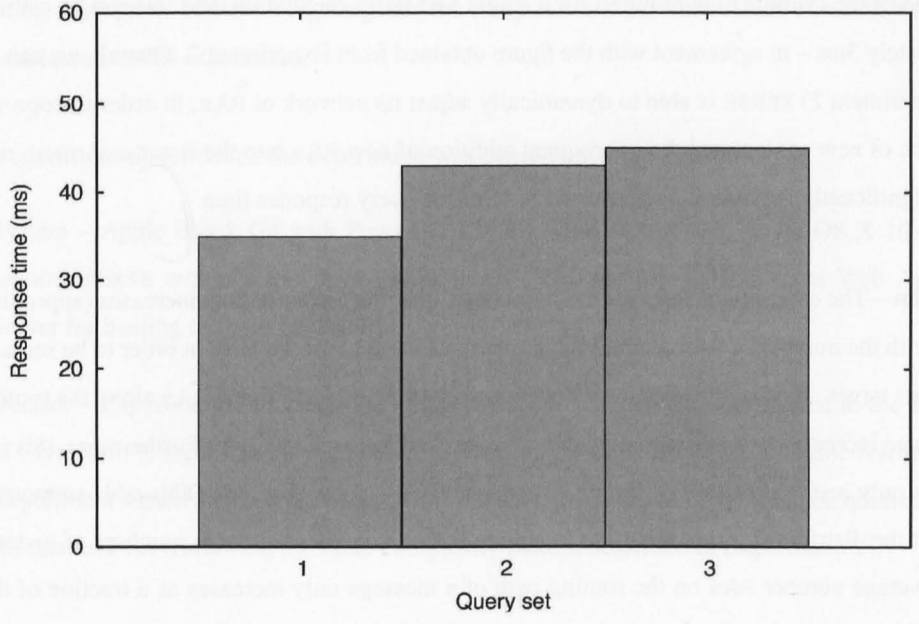


Figure 6.8: Average query response time in Experiment 3 using FF dataset.

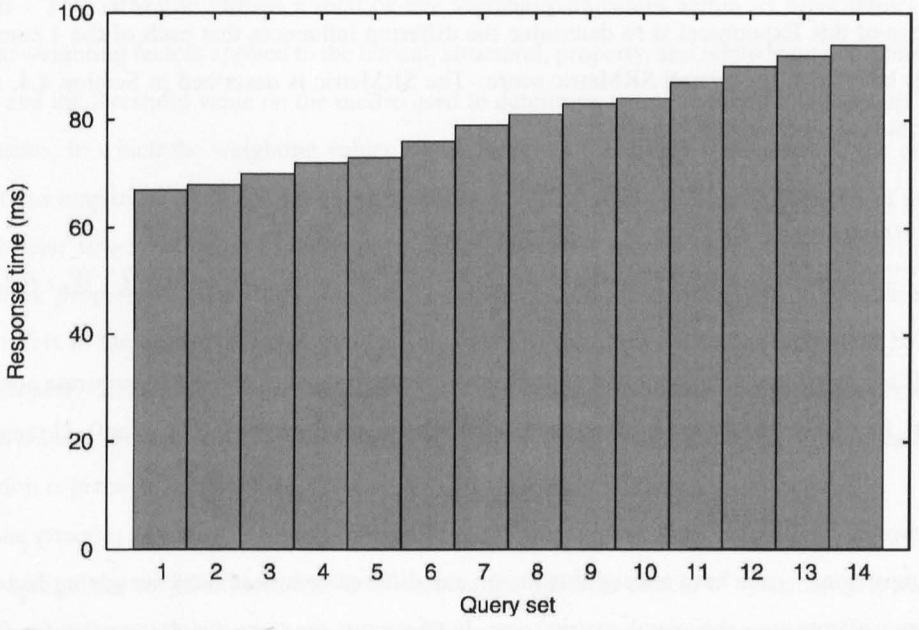


Figure 6.9: Average query response time in Experiment 3 using Galen dataset.

routing, again produces only a small, and approximately linear, increase in query response time – which further indicates that the system as a whole is scalable. The results of this experiment also enable the calculation of the average time required for a single SRMetric calculation, and this can be estimated as approximately 3ms – in agreement with the figure obtained from Experiment 2. Overall, we can see that (as in Experiment 2) SERSE is able to dynamically adjust its network of RAs, in order to cope with the notification of new content and the consequent addition of new RAs into the RouterAgent network, without significantly degrading performance in terms of query response time.

Conclusion – The experiment supports the hypothesis, that the response time increases (approximately) linearly with the number of RAs an individual query message has to traverse in order to be semantically routed to its target. That is, the additional cost in response time of additional RAs along the routing path of a message increases by approximately the same amount for each addition. Furthermore, this increase represents only a small fraction of the total response time – about 3 to 7%. This adds supporting evidence that the distributed index in SERSE is able to scale to handle very large numbers of nodes, given that the average number RAs on the routing path of a message only increases as a fraction of the total number of concepts indexed (given the initial routing action of the PortalAgent).

6.3.4 Experiment 4: Effects of SRMetric parameter variation.

The purpose of this Experiment is to determine the differing influences that each of the 4 component weightings have upon the overall SRMetric score. The SRMetric is described in Section 4.4, and the metric formula is repeated here for reference:

$$SRMetric(\alpha, \beta) =$$
$$max(\mu_s(\sum_{i=1}^{n} W_l \cdot L_i(\alpha,\beta) + \sum_{i=1}^{n} W_s \cdot S_i(\alpha,\beta) + \sum_{i=1}^{n} W_p \cdot P_i(\alpha,\beta)), \mu_r(\sum_{i=1}^{n} W_r \cdot R_i(\alpha,\beta)))$$

where: W_x are individual weighting factors for each of the components, and μ_x are separate normalisation factors for similarity and relatedness, used to express the result of the metric in a [0–1] range.

As shown in the formula, the 4 components of the SRMetric – lexical, structural, property and relatedness comparisons – each have a weighting factor, and different values of these weighting factors will have differing effects upon the overall metric score. Furthermore, the same weighting value for different components will have differing effects upon that overall score. The hypothesis being tested is that each of the 4 metric components produce different inputs to the overall SRMetric score for the same concept

comparison and same component weighting, and so these components differ in their effectiveness of determining semantic relatedness. This differing effectiveness is then utilised as an input into the process of selecting appropriate relative weightings for the components – the final values of which are reported in the conclusion to this experiment.

The conditions under which the experiment was carried out were as follows:

- System – Apple iBook G4 with PowerPC 1.3GHz. and 1Gb RAM, Apple OS X 10.4 (Tiger), running SERSE within a Sun Java SE SDK 1.4 JVM and using the Tomcat Web Application Server for hosting the web interface.

- Dataset – Experiment 4 used the UKResearchGroup (UKRG) dataset, created at the University of Liverpool, to represent the members, projects and publications of an academic research group. Experiment 4 used all three elements of this data-set, plus a set of 20 pre-defined queries specified for this data-set (following the same complexity pattern described in Experiment 1).

Aim – To determine the effect upon the semantic neighbourhoods formed by the RAs, and the subsequent effect on query response times, of systematic variation of the individual component weighting values used within the SRMetric.

Method – The SRMetric utilises a total of five variable parameters within its algorithms – the four heuristic weighting factors applied to the lexical, structural, property, and relatedness components of the metric, and the threshold value on the metric used to determine neighbourhood membership. The four components, to which the weighting values apply, represent the different elements of the ontological descriptions employed when comparing two concepts: *lexical* refers to the comparisons of entity (and other) labels; *structural* refers to the comparison of concepts' local position within their ontological hierarchies; *property* refers to the comparison of concepts' object and datatype properties; and *relatedness* refers to the degree to which two concepts may be explicitly related to each other by a mutual object property. Each of the weighting values and the threshold value are expressed as a decimal value between 0 and 1. Further explanation of these heuristic weighting factors and their role in the SRMetric calculation is given in Section 4.4.

The aim of this set of experiments is to observe the different effects of systematically varying the weighting and threshold values. In order to achieve this, the experiments were performed by varying each of the weighting parameters in turn over a specified set of values – 0.0, 0.25, 0.50, 0.75, and 1.0. At each of these values the average neighbourhood size and average query response time over all concepts

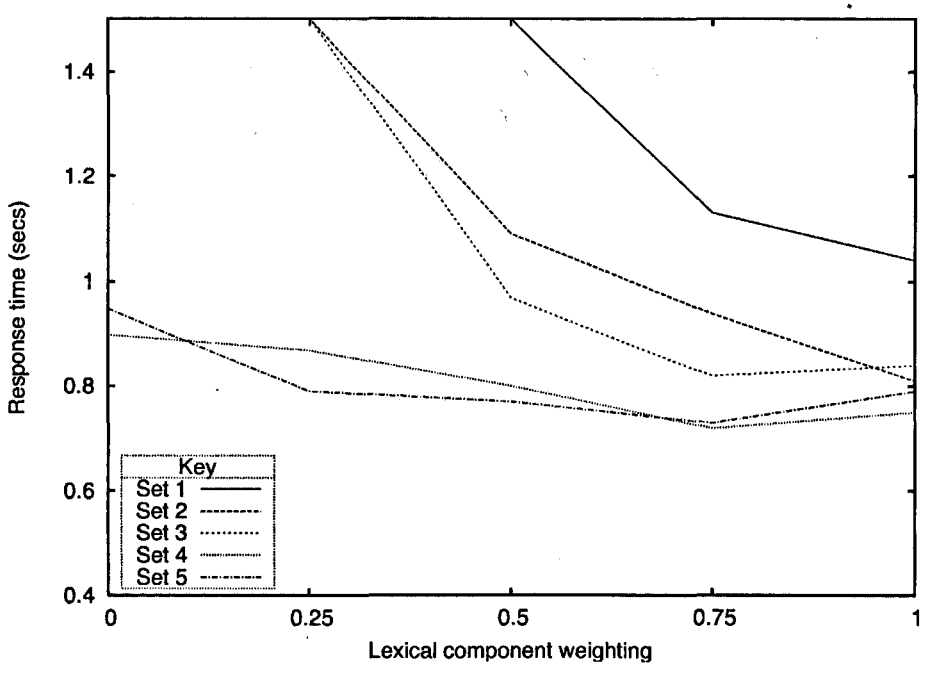


Figure 6.10: Average query response time vs lexical weighting value.

in the ontology is recorded – using the UKRG data-set. The query response time is determined in the same manner as in Experiment 1, using a pre-defined set of twenty queries of varying complexity. In order to observe the interaction between the different metric components, each experiment upon one of the parameters is repeated using differing values of the other weighting parameters. However, in order to keep the volume of experimental results to a manageable size this variation of the other values is also limited to the same set of five specified values. In addition, the threshold value is also varied across the same range of values. Therefore, the results of a single parameter-variation experiment consist of two graphs: one showing the average query response time, and one showing the average neighbourhood size for the different values of the investigated parameter. In each of these graphs there are five sets of results, each representing a different set of values for the other parameters – i.e., in *Set 1* the three remaining component weightings and the threshold value are set at 0.0, in *Set 2* at 0.25, and continuing to *Set 5* at 1.0. Overall, the experiment produces four sets of results, one for each of the SRMetric weighting values, showing how both the average neighbourhood size and average query response time vary as the investigated parameter is varied, and showing how the effect of this parameter variation changes as the other parameters and the threshold value are also varied.

Results – Figures 6.10 to 6.17 show the results for systematic variation of each of the four component weighting values (lexical, structural, property and related) when using the UKRG data-set. For each of the examined parameters there are two graphs of results – one showing how the average query response

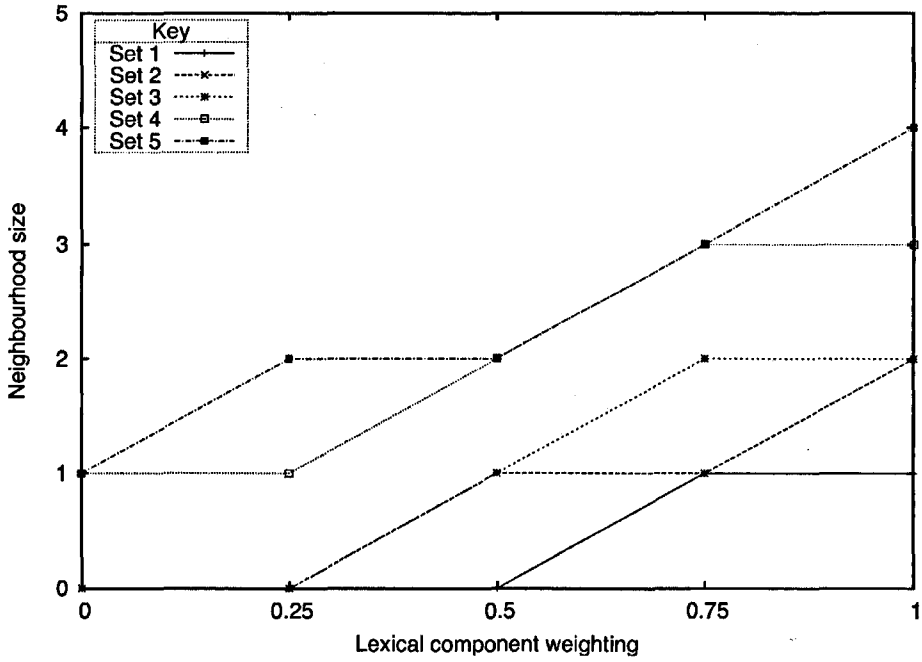


Figure 6.11: Average neighbourhood size vs lexical weighting value.

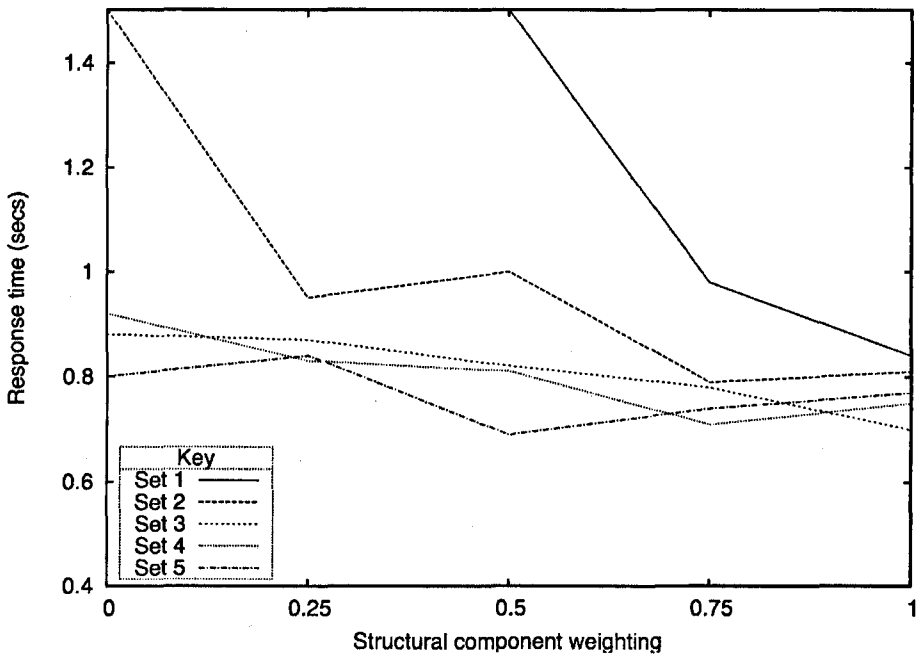


Figure 6.12: Average query response time vs structural weighting value.

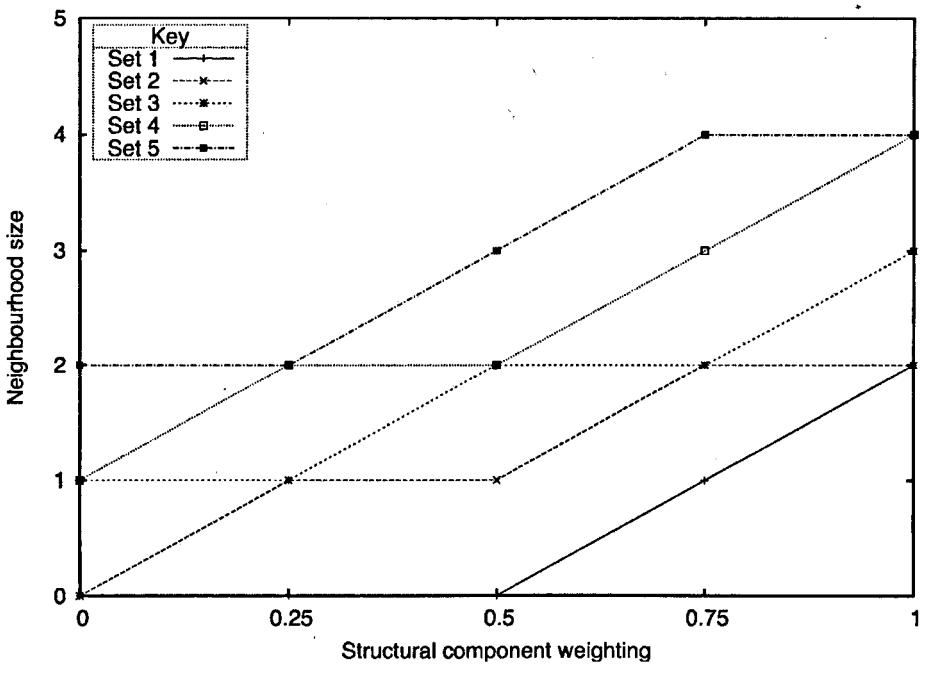


Figure 6.13: Average neighbourhood size vs structural weighting value.

time varies against the component weighting; and one showing how the average neighbourhood size varies against the the component weighting. In each figure there are five sets of results, each showing the differing effect of the values of the remaining parameters within the metric, when they are set at one of the five specified values (i.e., 0.0, 0.25, 0.50, 0.75, or 1.0).

These results demonstrate a number of points regarding the functioning of the SRMetric, and the relative contributions of the different components of the metric score:

- Firstly, there is a clear inverse relationship between average neighbourhood size and average query response time. Larger average neighbourhoods mean that each message is required to pass through fewer RAs before reaching its goal – that is, larger neighbourhoods mean that each message 'hop' between RAs enables the message to traverse more of the overall RA network. This is despite the additional processing overhead in message routing caused by the presence of more RAs within each RA's neighbourhood, as demonstrated in Experiment 2. However, the effect of this additional processing can be seen in the query response timings, especially for the lexical and structural components, when the average neighbourhood size exceeds 3.

- Secondly, the performance of the semantic routing mechanism in terms of query response time is significantly reduced when considering low parameter values. That is when all the parameter values are below 0.5, the query response time varies greatly and is generally high. However, when all parameter values are above 0.5 there is far less variation in response times, and they

are generally low. This demonstrates that the metric relies on input from all four components, and that it is the combination of the different elements of semantic relatedness that enables the metric to function. Furthermore, these experimental results also demonstrate that there is a range of parameter values, between 0.6 and 0.85, that enable the greatest degree of discrimination in the overall metric result.

- Thirdly, significant differences can be observed between the relative effects of the different parameters on the overall function of the metric. Variation in the lexical component weighting clearly has the greatest influence on the SRMetric scores – which matches with our intuition that a significant fraction of the meaning of an ontologically defined concept is encoded in the concept and property labels. The structural component weighting has the second most significant effect on the metric, indicating that significant evidence of similarity is obtained from the comparison of ontology hierarchies. The property component weighting has the least influence on the metric score, which indicates that similar concepts do not necessarily have similar properties, as the defined properties usually reflect the differing conceptualisations underlying the different ontologies. The effect of the relatedness component upon the overall SRMetric score must be considered in light of the fact the fact the SRMetric will only return a score based on the relatedness component when the concept similarity score (resulting from the other three components) is below a pre-defined threshold – i.e., the metric will only return a relatedness score when the concepts are determined to be sufficiently dissimilar. Given this, the relatedness component can be seen to have only a minimal influence on the overall metric score, which matches the intention of this metric component, that was to provide additional connections in the RA network, cutting across the similarity connections and providing support to the 'semantic browsing' form of resource navigation.

Conclusion – The experiment supports the hypothesis, that each of the 4 metric components produce different inputs to the overall SRMetric score for the same concept comparison and same component weighting, and so these components differ in their effectiveness of determining semantic relatedness. Experiment 4 shows the specific effects of varying the weighting values on the average size of the RA neighbourhoods formed, and the consequent effects this has on query-response timings. Firstly, response times are generally high when all parameter values are below 0.5, that is the metric requires inputs from all components and their weightings to be proportional to the neighbourhood threshold value – which in this experiment was set at 0.75. The lexical mapping is most significant input to the SRMetric, because a high proportion of a concept's semantics are in its label and those of its properties. The structural mapping is the next most significant input to the metric, as ontology hierarchies – in terms of the semantics of the ancestor, child and sibling concepts – also encode a lot of a concept's semantics. The property mapping has the least effect on the metric score, which indicates that similar concepts do not

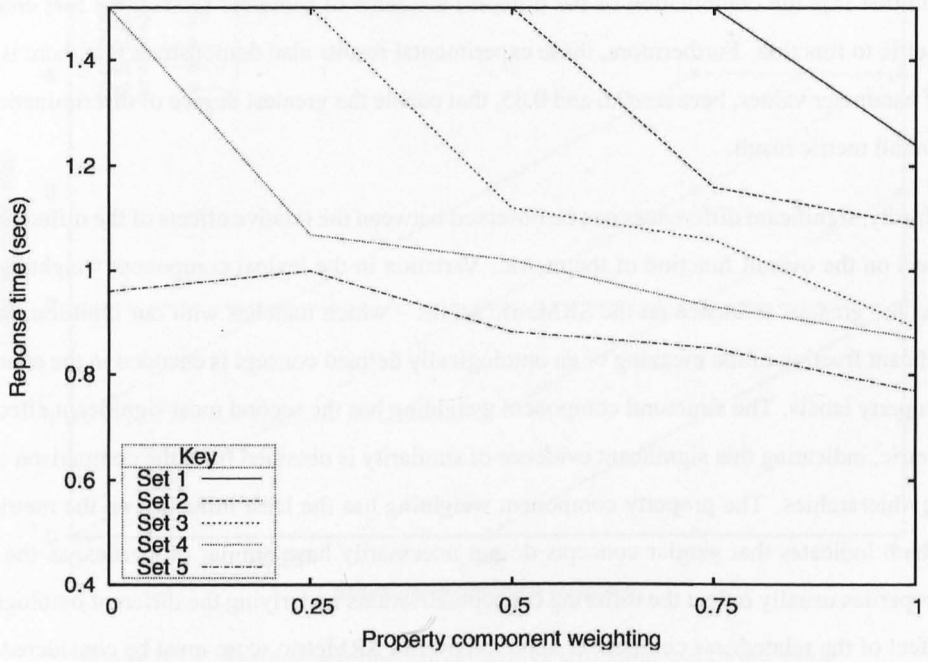


Figure 6.14: Average query response time vs property weighting value.

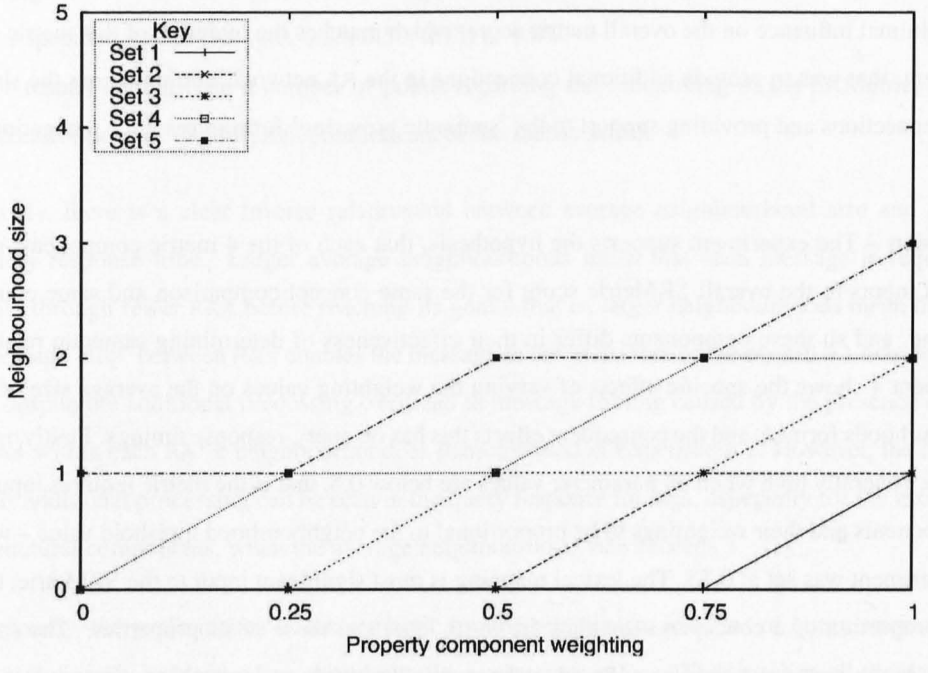


Figure 6.15: Average neighbourhood size vs property weighting value.

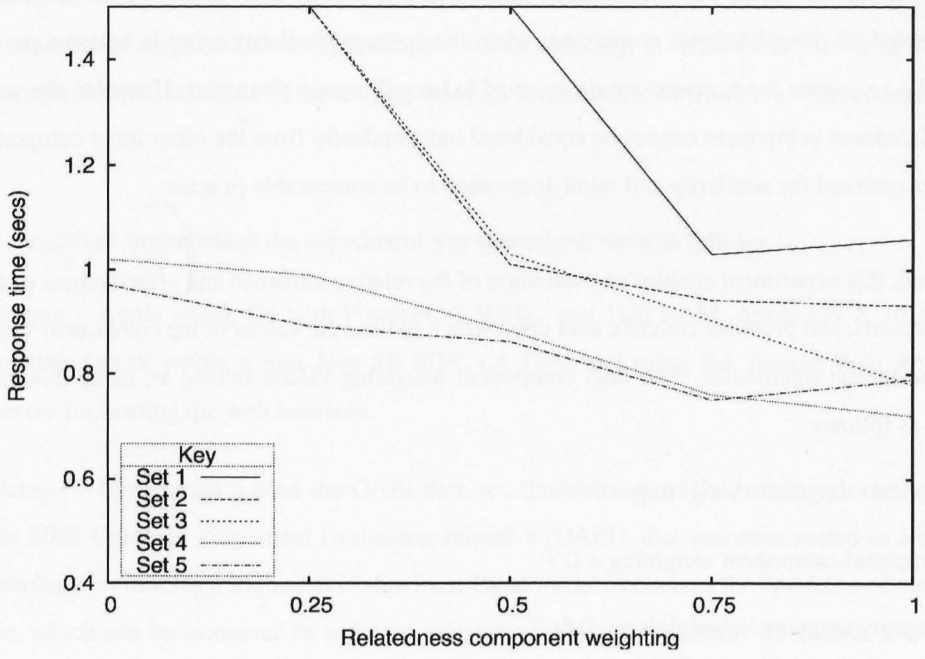


Figure 6.16: Average query response time vs relatedness weighting value.

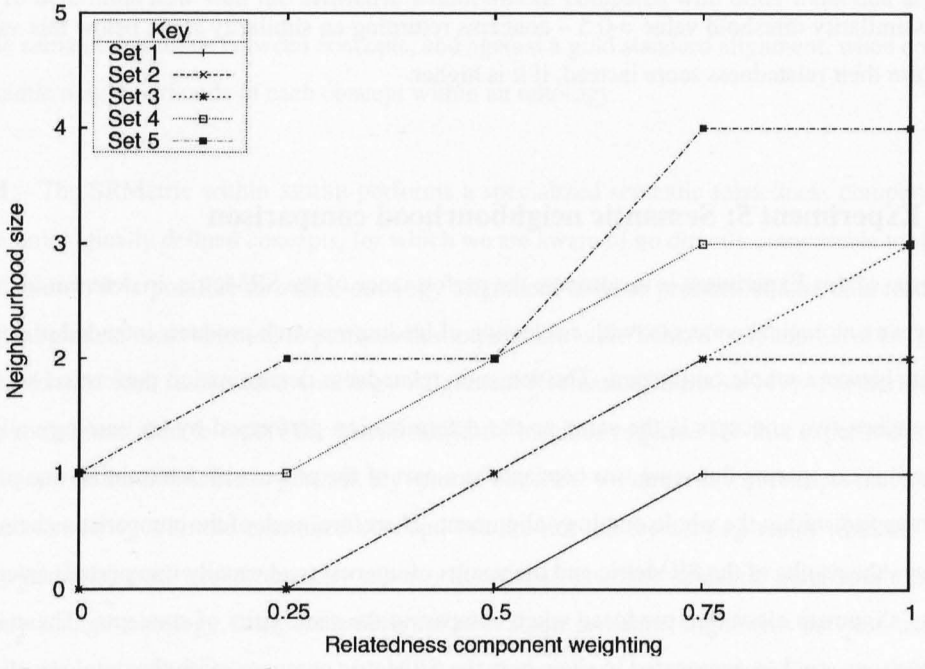


Figure 6.17: Average neighbourhood size vs relatedness weighting value.

necessarily have similar properties, as the defined properties usually reflect the specific needs of the differing conceptualisations underlying the respective ontologies. Finally, the SRMetric will only return a score based on the relatedness component when the concept similarity score is below a pre-defined threshold – i.e., when the concepts are determined to be sufficiently dissimilar. However, the weighting of the relatedness component cannot be considered independently from the other three components, as the scores returned for similarity and relatedness need to be comparable in scale.

Overall, this experiment enables an evaluation of the relative influence and effectiveness of the metric components, and provides concrete data upon which to base the values of the component weightings within the metric calculation. The final component weighting values arrived at, using this and other data, are as follows:

- Lexical component weighting = 0.85
- Structural component weighting = 0.7
- Property component weighting = 0.6
- Relatedness component weighting = 0.7
- Neighbourhood threshold value = 0.75 – used to select only those concepts returning an overall metric score over this value as semantic neighbours
- Dissimilarity threshold value = 0.5 – concepts returning an similarity score below this value can return their relatedness score instead, if it is higher.

6.3.5 Experiment 5: Semantic neighbourhood comparison

The purpose of this Experiment is to compare the performance of the SRMetric, in determining relatedness between ontological concepts, with a selection of leading research products intended to determine alignments between whole ontologies. The semantic relatedness determination performed by the SRMetric between two concepts is the same as the determination performed by an ontology alignment tool when it is comparing the same two concepts as a part of the process of determining one pair-wise concept mapping within the whole ontology alignment. Therefore, each of the comparisons being made are between the results of the SRMetric and the results of internal (and usually unreported) calculations within the alignment algorithm produced when comparing the same pairs of concepts. These individual comparisons are then aggregated to show how the SRMetric compares with the ontology alignment tools when determining the semantic neighbourhood of a concept within this concept's source ontology. In addition, a 'Gold Standard' calculation of the appropriate semantic neighbourhood of each concept

is provided to act as a baseline comparison for the performance of all of the tools. This alignment was provided by an independent third party. The hypothesis being tested is that the SRMetric is able to determine semantic neighbourhoods of concepts with an accuracy, in terms of the declared semantics, that is comparable to a selection of the leading ontology alignment tools available within Semantic Web research.

The conditions under which the experiment was carried out were as follows:

- System – Apple iBook G4 with PowerPC 1.3GHz. and 1Gb RAM, Apple OS X 10.4 (Tiger), running SERSE within a Sun Java SE SDK 1.4 JVM and using the Tomcat Web Application Server for hosting the web interface.

- Dataset – Experiment 5 used the OAEI data-set, that consists of the benchmark ontology from the 2005 Ontology Alignment Evaluation Initiative (OAEI), that was constructed as a reference ontology for ontology alignment evaluations. Experiment 5 concerns the operation of the SRMetric, which can be examined in isolation, outside of the SERSE system. Therefore, it is only the ontology element that is required, as the SRMetric uses only the ontological definition to calculate similarity and relatedness between concepts.

Aim – To determine how well the SRMetric within SERSE compares with other tools that are able to calculate semantic relations between concepts, and against a gold standard alignment, when computing the semantic neighbourhoods of each concept within an ontology.

Method – The SRMetric within SERSE performs a specialized semantic relatedness comparison task between ontologically defined concepts, for which we are aware of no directly comparable tools or systems. However, it is possible to utilize ontology alignment tools to produce comparable results. The ontology alignment tools selected to perform this experiment were FOAM [42] and Crosi [84], both of which performed well in the 2005 Ontology Alignment Evaluation Initiative [49]. Furthermore, both of these tools were able to report the partial semantic matches required for this experiment, and for comparison with the SRMetric in general. By 'partial matches' we mean those concepts that have some calculated similarity with the examined concept, but are not the top-scoring result which is what an ontology matching tool would normally report. In order to generate semantic neighbourhoods when using only a single ontology, using tools designed to evaluate semantic alignments between ontologies, the tools were used to map an ontology to itself – and the semantic neighbourhood for a concept is then formed by those potential matches for each concept (excluding the concept itself, which is usually the best match), above a specified threshold value. In this way the ontology alignment tools can be used

to generate the member concepts of semantic neighbourhoods, in the same way as performed by the SRMetric, and thus making the results comparable.

As is common in state-of-the-art ontology alignment tools, both FOAM and Crosi expose parameters that affect the semantic matching algorthim at the core of each tool. In FOAM the parameters determine a wide range of factors, such as the the number of comparative iterations, confidence (threshold) values, use of alignment strategies, use of existing alignments and external rules or classifiers, and the general efficiency of the alignment process. In Crosi the parameters determine which of the available matching algorithms are employed, and what relative weighting is applied to each of these individual matchers. For FOAM two sets of parameters were defined for this experiment – one representing the simplest alignment calculation (Efficient), and one representing the most complex and computationally expensive alignment calculation (Complete) that the tool will perform whilst not considering any other information in addition to the ontology itself. Two sets of parameters were also defined for Crosi in this experiment, each of them using equally weighted component algorithms – one using all matchers, except those three that employ the WordNet [108] (WN) online lexical database (without WN); and one using all fourteen available matchers (with WN).

Therefore, for each ontology considered there are five sets of results – one for the SRMetric, two for FOAM and two for Crosi. Each of these results contain two sets of graphed measurements that are produced as the tool threshold is varied. The first is the size of the semantic neighbourhood averaged over all concepts in the ontology, and the second is the average precision and recall of these neighbourhoods when compared with an ideal, human-judged neighbourhood for each concept. These human-judged neighbourhoods were determined by an independent third-party on the basis of the considered ontology alone.

Results – Figures 6.18 to 6.27 show the results for both average recall and precision of the calculated semantic neighbourhoods and average neighbourhood size for all five experimental runs – using the UKRG ontology. The experiment was also performed using the OAEI ontology, producing largely similar results that are omitted here for space-saving reasons. When examining the results it is necessary to note that both the SRMetric and FOAM utilize similarity score values that vary between 0 and 1, however, Crosi does not normalise results, and the maximum value can vary up to an observed maximum of 4.8. Furthermore, it is believed that no fixed maximum value exists for Crosi similarity scores, and the calculated maximum score is largely dependent on the amount of (particularly lexical) information encoded within the ontology.

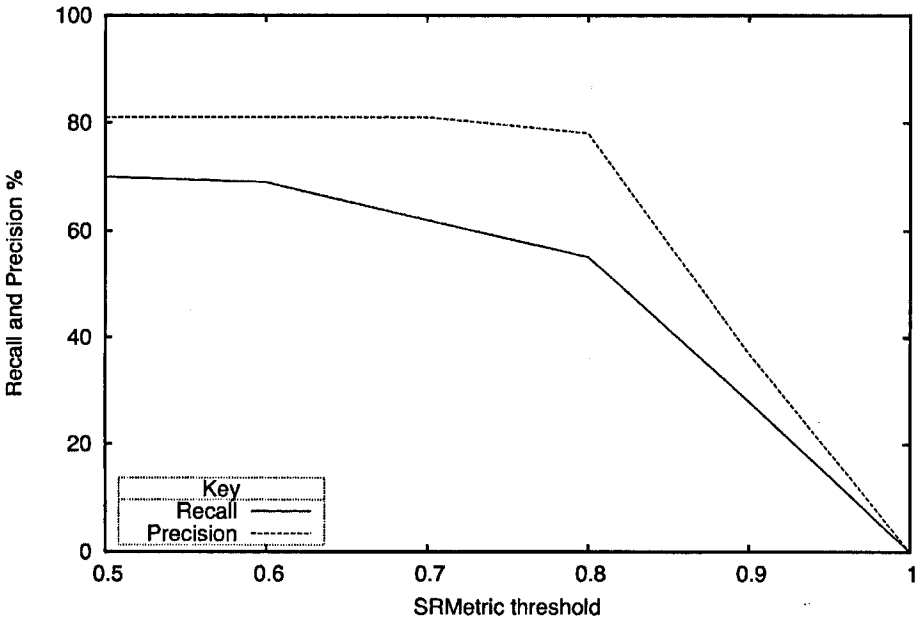


Figure 6.18: Average recall and precision over neighbourhood for SRMetric.

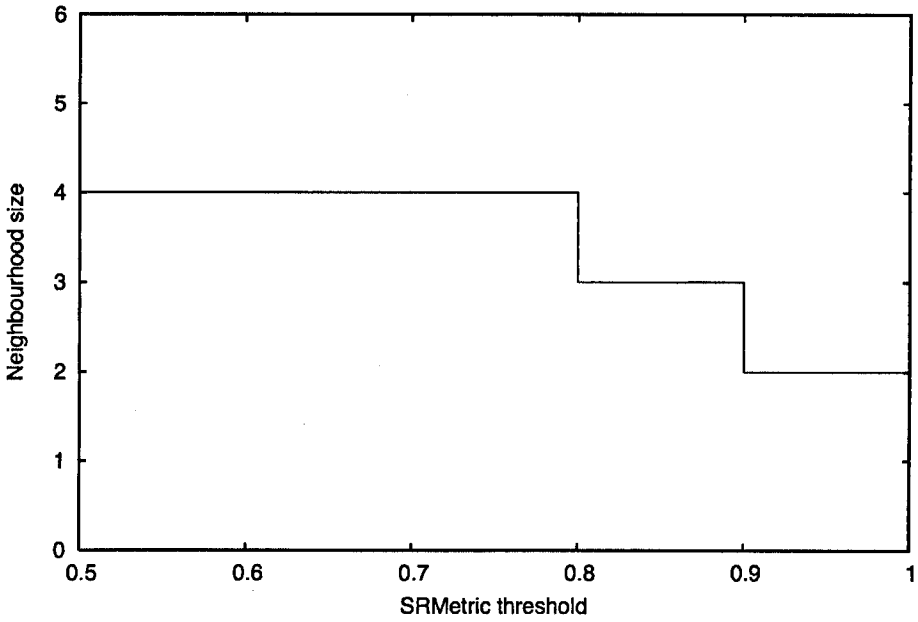


Figure 6.19: Average neighbourhood size for SRMetric.

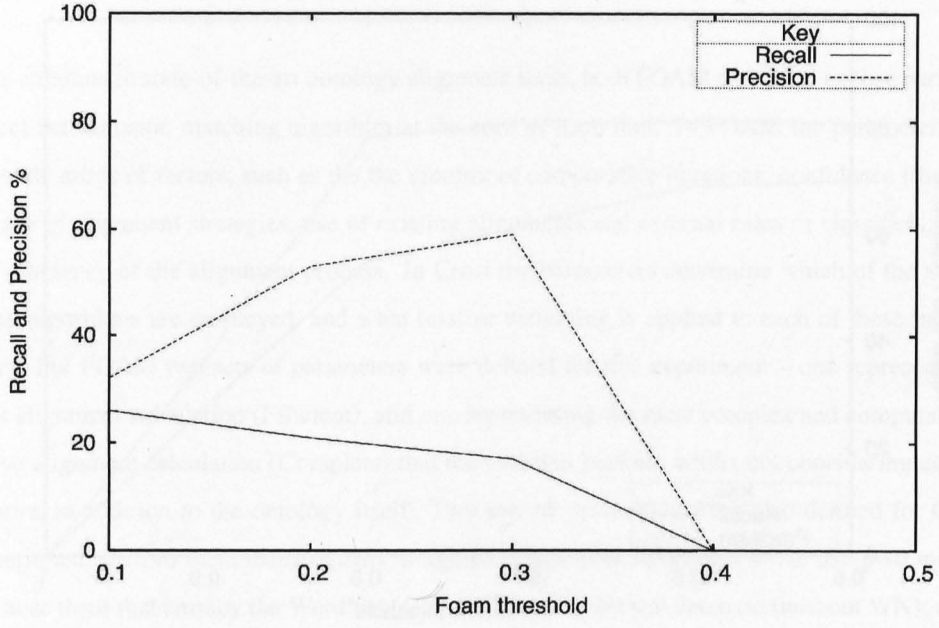


Figure 6.20: Average recall and precision over neighbourhood for FOAM (efficient).

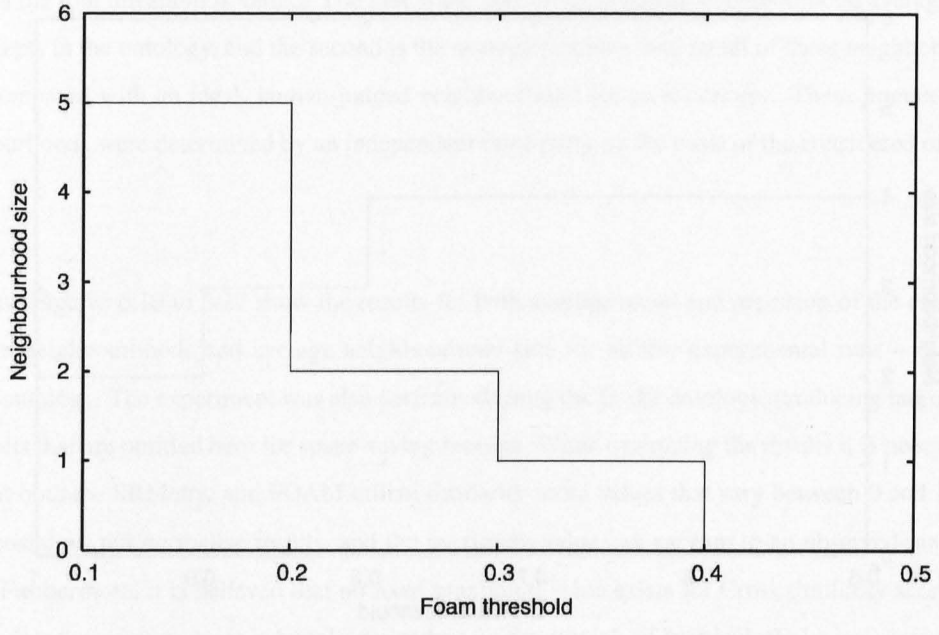


Figure 6.21: Average neighbourhood size for FOAM (efficient).

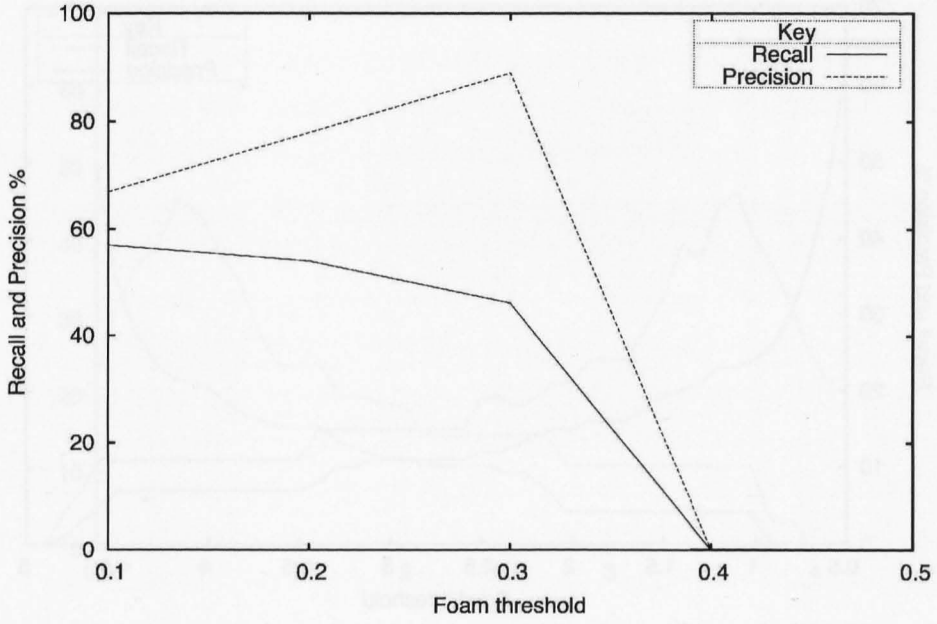


Figure 6.22: Average recall and precision over neighbourhood for FOAM (complete).

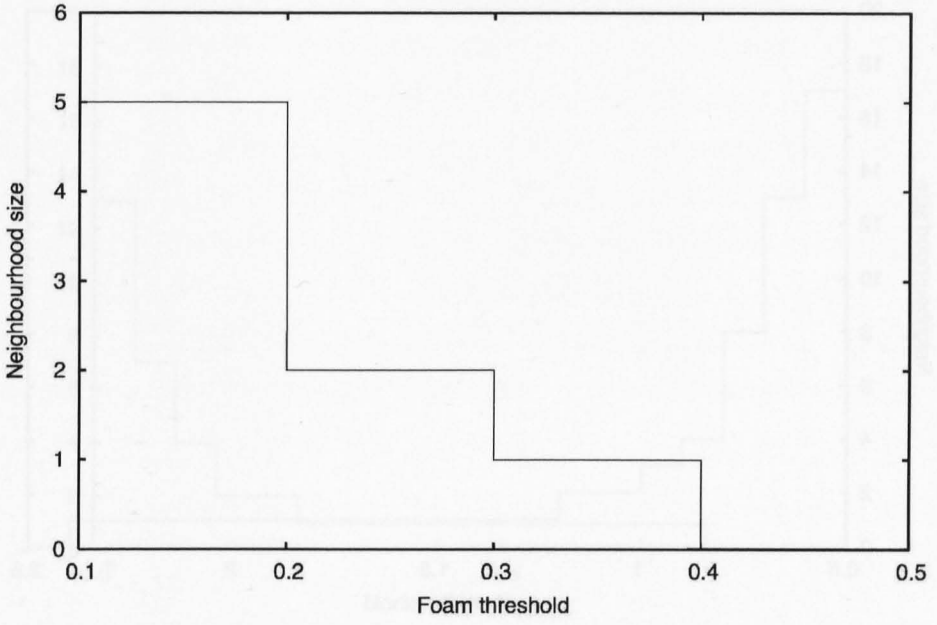


Figure 6.23: Average neighbourhood size for FOAM (complete).

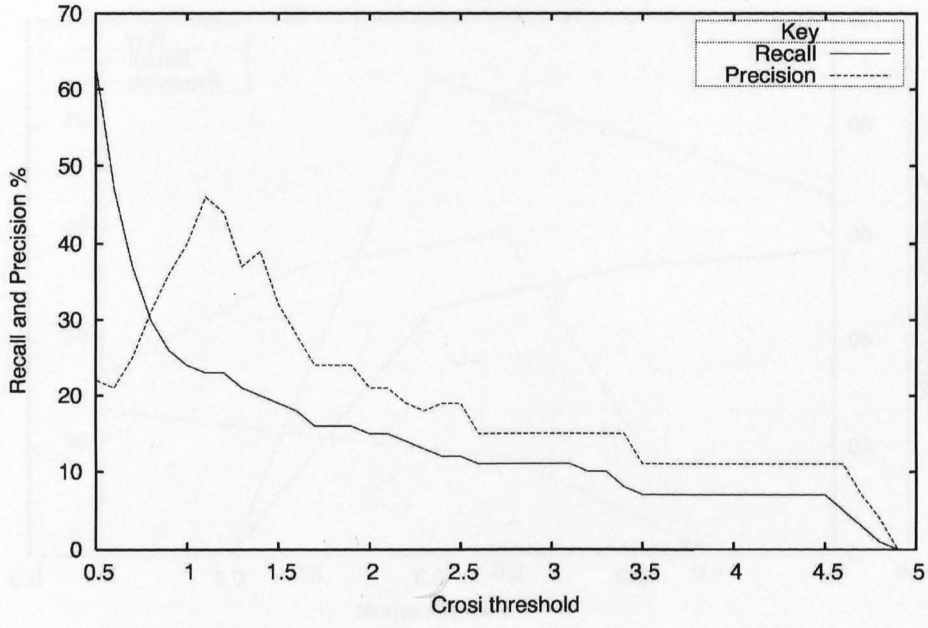


Figure 6.24: Average recall and precision over neighbourhood for CROSI (without WN).

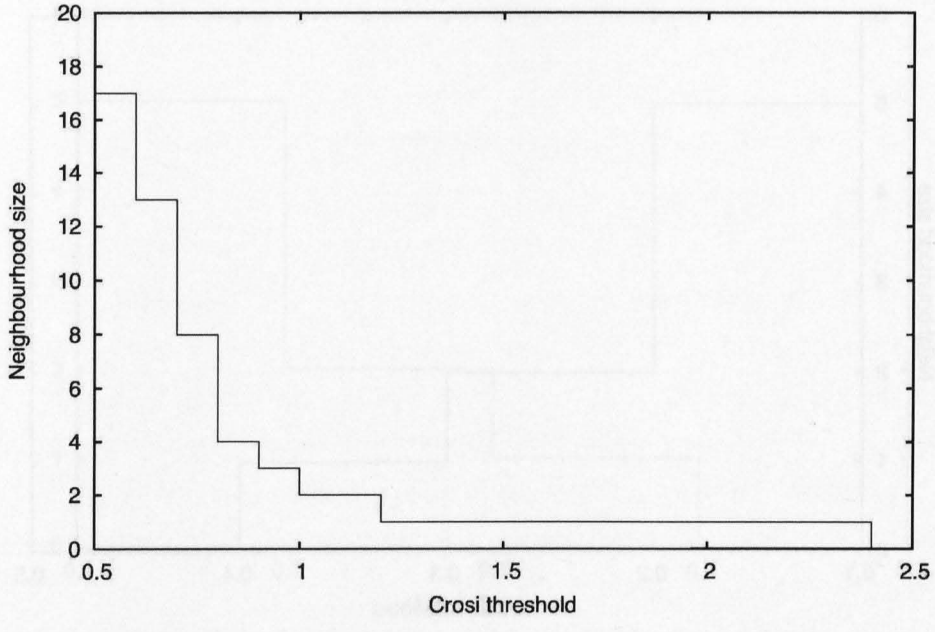


Figure 6.25: Average neighbourhood size for CROSI (without WN).

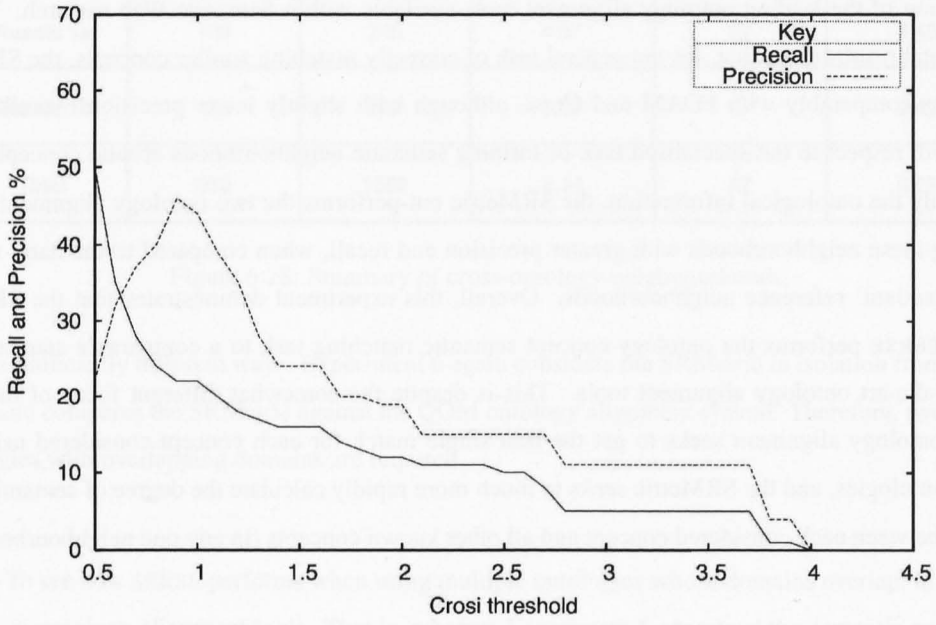


Figure 6.26: Average recall and precision over neighbourhood for CROSI (with WN).

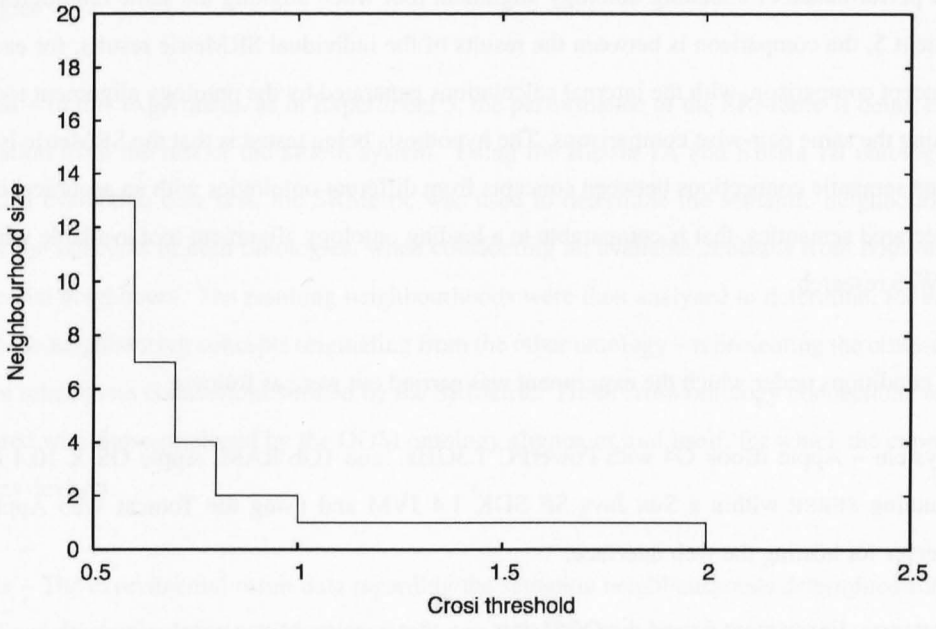


Figure 6.27: Average neighbourhood size for CROSI (with WN).

Conclusion – The experiment supports the hypothesis, that the SRMetric is able to determine semantic neighbourhoods of concepts with an accuracy, in terms of the declared semantics, that is comparable to a selection of the leading ontology alignment tools available within Semantic Web research. The experimental results show that, for the general task of correctly matching similar concepts, the SRMetric performs comparably with FOAM and Crosi, although with slightly lower precision overall. However, with respect to the specialised task of forming semantic neighbourhoods around concepts based upon only the ontological information, the SRMetric out-performs the two ontology alignment tools – forming these neighbourhoods with greater precision and recall, when compared to the hand-mapped 'gold-standard' reference neighbourhoods. Overall, this experiment demonstrates that the SRMetric within SERSE performs the ontology-concept semantic matching task to a comparable standard with state-of-the-art ontology alignment tools. This is despite the somewhat different focus of the tools, where ontology alignment seeks to get the best single match for each concept considered using two whole ontologies, and the SRMetric seeks to much more rapidly calculate the degree of semantic relatedness between each considered concept and all other known concepts (in any one neighbourhood).

6.3.6 Experiment 6: Cross-ontology semantic neighbourhood formation

The purpose of this Experiment is to compare the performance of the SRMetric when determining semantic connections between concepts from different ontologies, that represent overlapping domains, with the performance of a leading ontology alignment tool when aligning the same ontologies. As in Experiment 5, the comparison is between the results of the individual SRMetric results, for each pair-wise concept comparison, with the internal calculations generated by the ontology alignment tool when performing the same pair-wise comparisons. The hypothesis being tested is that the SRMetric is able to determine semantic connections between concepts from different ontologies with an accuracy, in terms of the declared semantics, that is comparable to a leading ontology alignment tool available within Semantic Web research.

The conditions under which the experiment was carried out were as follows:

- System – Apple iBook G4 with PowerPC 1.3GHz. and 1Gb RAM, Apple OS X 10.4 (Tiger), running SERSE within a Sun Java SE SDK 1.4 JVM and using the Tomcat Web Application Server for hosting the web interface.

- Dataset – Experiment 6 used the QOM data-set, that consists of two ontologies, with overlapping domains, selected from the Quick Ontology Mapping (QOM) system evaluation test-data: Russia1A and Russia1B. Both of these ontologies represent the domain of tourism in Russia, but in

	Concepts	Total neighbours	Average neighbourhood size	Total cross-ontology neighbours	Average cross-ontology neighbours
Russia 1a	149	740	4.97	67	0.45
Russia 1b	161	852	5.29	67	0.42
Total	310	1592	5.14	67	0.22

Figure 6.28: Summary of cross-ontology neighbourhoods.

significantly different ways. Experiment 6 again considers the SRMetric in isolation from SERSE, and compares the SRMetric against the QOM ontology alignment system. Therefore, two ontologies with overlapping domains are required.

Aim – To see how SERSE performs when using multiple ontologies whose domains overlap, in comparison with ontology alignment tools. That is, whereas Experiment 5 examined the semantic neighbourhoods formed by the SRMetric, FOAM and Crosi when considering only one ontology, this experiment is intended to determine whether the SRMetric produces the expected connections between closely related concepts from heterogeneous ontologies. This better reflects the situation that would pertain in an open web environment, where similar concepts are likely to be defined in a number of entirely separate ontologies.

Method – In this experiment, as in Experiment 5, the performance of the SRMetric is being examined in isolation from the rest of the SERSE system. Using the Russia 1A and Russia 1B ontologies from the QOM evaluation data sets, the SRMetric was used to determine the semantic neighbourhoods of each of the concepts in both ontologies, when considering all available concepts from both ontologies as potential neighbours. The resulting neighbourhoods were then analysed to determine, for each concept, those neighbouring concepts originating from the other ontology – representing the cross-ontology concept relatedness connections formed by the SRMetric. These cross-ontology connections were then compared with those produced by the QOM ontology alignment tool itself, for which the experimental data was devised.

Results – The experimental result data regarding the semantic neighbourhoods determined for each of the concepts in the two ontologies is shown in Figure 6.28. The two ontologies define a total of 310 concepts, and, as determined by the SRMetric, each concept had an average of 5.14 semantic neighbours – of which 0.22 were drawn from the 'other' ontology. That is, 67 neighbourhood connections were

formed between concepts from the two heterogeneous ontologies. Using the same two ontologies the QOM system generated 70 alignments between concepts from the different ontologies, and of these 70 alignments, 52 were replicated by the neighbourhood connections formed by the SRMetric. 18 of the alignments generated by QOM were not reproduced – mainly due to the fact that the SRMetric does not take into account the known extension of concepts (for performance reasons), whereas QOM does. 15 cross-ontology neighbourhood connections were created that were not present in the QOM alignment, which is explained by the fact that the SRMetric can create more than one cross-ontology neighbour for any given concept – due to an underlying algorithm that attempts to generate all connections for sufficently related concepts, compared to ontology alignment algorithms that seek to chose only the best of the potential mappings. Overall, Experiment 6 demonstrates that the SRMetric is able to generate concept neighbourhood links across heterogeneous ontologies with overlapping domains that are comparable with those generated through widely accepted ontology alignment techniques.

Conclusion – The experiment supports the hypothesis, that the SRMetric is able to determine semantic connections between concepts from different ontologies with an accuracy, in terms of the declared semantics, that is comparable to a leading ontology alignment tool available within Semantic Web research. This is not entirely surprising given that the SRMetric design was based upon the theory behind QOM, and uses a similar comparison process. However, to reduce computation time the SRMetric does not utilise a number of the comparisons in QOM, and does not perform multiple iterations of comparison refinement like QOM. Therefore, simply matching the performance of QOM, which is a reasonable representation of state-of-the-art in ontology alignment, represents a good result for the SRMetric. Most importantly, the result ensures that SERSE is able to generate the appropriate cross-ontology mappings, so that SERSE can operate as intended using multiple ontologies with partially overlapping domains.

6.4 Summary

This final section of this chapter is intended to provide an overview of the main experimental results, and outlines the conclusions that can be drawn from each of the experiments. A number of key points are raised by the results of the experiments, and there are a number of implications raised for the operation of SERSE as a large-scale, distributed indexing system. The section presents an overview of the results of each experiment and the conclusions that can be drawn from them, focusing upon the implications for the functioning of SERSE.

Firstly, Experiment 1 shows that SERSE is able to effectively answer complex queries, by subdividing the complex query, routing the atomic queries to individual RAs and then recombining the

replies. In addition, this experiment shows that the overhead of the query decomposition and reply recombination processes, even when considering very complex queries, does not outweigh the cost of the individual queries themselves. However, the results of experiment 1 also highlight a specific issue regarding the handling of large, in-memory RDF models, which clearly indicates that for indexing efficiency reasons RAs should utilise database-managed RDF models. Experiments 2 and 3 show that SERSE is able to handle additional RAs within the index network without significantly increasing query response time. Specifically, when answering a query, the addition of RAs to the routing path of the query or the addition of neighbouring RAs to the semantic neighbourhoods of RAs on the routing path only increases response time by a small proportion of the overall response time. Furthermore, this increase in response time increases in a linear manner as additional RAs are added to the routing path or neighbourhoods along it. These experiments demonstrate that SERSE is able to manage the large-scale deployments intended for it whilst retaining an acceptable response time. This is underlined by the fact that these two experiments only consider additional RAs that have a direct bearing on a specified query, and do not consider the potentially vast numbers of RAs added to SERSE that do not take part in the routing or answering of any given query.

Experiment 4 shows that each of the 4 metric components produce different inputs to the overall SRMetric score, and so differ in their effectiveness of determining semantic relatedness. The experiment shows the specific effects of varying the weighting values on the average size of the RA neighbourhoods formed, and the consequent effects this has on query-response timings. The relative importance of the different metric components can be summarised as follows: the lexical mapping is most significant input to the SRMetric, because a high proportion of a concept's semantics are in its label and those of its properties; the structural mapping is the next most significant input to the metric, as ontology hierarchies also encode a lot of a concept's semantics; the property mapping has the least effect on the metric score, which indicates that similar concepts do not necessarily have similar properties, due to differing conceptualisations underlying the respective ontologies; finally, the SRMetric will only return a score based on the relatedness component when the concepts are determined to be sufficiently dissimilar, and so requires a weighting proportional to the other components. The results of this experiment provided the primary input into the determination of the appropriate weighting values for each of these components in the overall metric calculation.

Experiment 5 shows that the SRMetric performs comparably with leading ontology alignment tools when matching similar concepts. Furthermore, the SRMetric out-performs the ontology alignment tools in the determination of semantic neighbourhoods for concepts when using only ontological information – forming these neighbourhoods with greater precision and recall, based on the 'gold-standard'

reference. That is, although the SRMetric aims to more rapidly calculate semantic matches by abandoning some of the matching components used in the ontology alignment tools, it performs the general semantic matching task to a comparable standard, and its particular matching task slightly better, than state-of-the-art ontology alignment tools. This demonstrates that the SRMetric within SERSE is suitable for its intended tasks of semantic matching within the system. Finally, Experiment 6 shows that the SRMetric determines semantic connections between concepts from heterogeneous ontologies as well as the leading ontology alignment tool, QOM, upon which it is based. That is, despite removing a number of the comparisons used in QOM to decrease execution time, the SRMetric matches the performance of a state-of-the-art ontology alignment system in generating cross-ontology mappings. This demonstrates that SERSE is able to generate the appropriate cross-ontology mappings for SERSE to operate using multiple ontologies with partially overlapping domains.

Overall, the experiments performed show that, firstly, SERSE is able to correctly answer queries, of any complexity, within an acceptable response time. Secondly, SERSE scales well as additional information, and so `RouterAgents`, are added – which supports the claim that SERSE can operate as a large-scale, distributed index. Thirdly, SERSE utilises a process of semantic concept matching, in the SRMetric, that employs tuned and weighted matching components. This SRMetric is shown to be suitable for its intended task within SERSE by performing the semantic concept matching task, both within one ontology and across heterogeneous ontologies, to a comparable standard with that of leading ontology alignment systems.

Part IV

QuestSemantics

Chapter 7

QuestSemantics

This chapter presents a specific set of developments of the SERSE *system itself, and of the ideas and approaches within* SERSE*, that were undertaken in order to investigate the application of resource location using semantic indexing and querying within real-world business scenarios. The system developed from* SERSE *in this context is known as QuestSemantics, and here we describe both the major extensions and the significant modifications made to the* SERSE *approach, in order to address the specific constraints of the application context.*

Section 7.1 describes the background to the QuestSemantics system, its similarities and differences with SERSE*, and the intended applications for the system. Following this, section 7.2 presents an overview of the design and architecture of QuestSemantics, and then details the main system components and their functions. Section 7.3 describes the deployment of QuestSemantics within two different commercial test-cases, and presents an initial evaluation of the performance of QuestSemantics in these applications. Finally, section 7.4 outlines the main lessons learnt in the development and deployment of QuestSemantics.*

7.1 Development from SERSE

Before moving into a detailed description of the design, development and deployment of the QuestSemantics platform, we must first review the relationship with SERSE. SERSE was designed to provide a scalable indexing and retrieval system for semantically annotated resources. The requirement for a scalable mechanism to index the semantic metadata used to annotate resources was predicated on an expected explosion in semantic annotation of existing and newly created web resources [11]. However, four years after the conception of the SERSE architecture there is still relatively little semantic metadata to index, as shown by the statistic gathered on websites like Swoogle[1]. Furthermore, SERSE was

[1] http://swoogle.umbc.edu/index.php?option=com_swoogle_stats

intended to demonstrate the power of semantic search over annotated resources – leveraging the formally defined metadata to perform precise queries over knowledge-bases of aggregated metadata, and leading to the retrieval of resources on the basis of the *meaning* of its content. However, the relatively limited volume of publicly available semantic metadata meant that there were few opportunities to do so, and in particular there was insufficient volume of heterogeneous ontologies, metadata and resources regarding overlapping domains to properly demonstrate the power of SERSE to index, aggregate and retrieve this information in a logically connected, but physically distributed indexing system. Therefore, we determined that in order to work to towards a significantly populated Semantic Web, and to be able to practically demonstrate the potential of semantically enhanced information search and retrieval, we should develop a system that was able to semantically annotate resources, index the generated metadata, and then present a user-friendly and intuitive user interface to enable the construction and execution of semantic queries over this metadata. In addition, we intended that this system should be used to pursue test-case deployments in real commercial environments to explore the application of semantic annotation and search techniques to real business problems.

The pursuit of test-case deployments in commercial environments was undertaken for two reasons. Firstly, working with commercial partners enabled us to focus on a specific business problem, that is rooted in the problems associated with precisely identifying and retrieving information resources. This focus on a specific issue, and the availably of the partner as the domain expert, meant that we were able to quickly develop ontologies and populated knowledge-bases to describe the required domain, and task, entities and relations. The second reason was that the commercialisation of semantic web technologies had not developed as quickly as initially envisaged, partially due to the problems of annotating the vast and diverse resources available. Therefore, we sought a means by which these technologies could be introduced to small enterprises, to annotate a very specific sub-set of available resources for a clearly defined purpose, thus enabling us to construct a detailed ontological model for the domain of knowledge and the task at hand. This fine-grained business knowledge then enabled the precise annotation and subsequent retrieval of the required resources using semantic queries that are constructed on the basis of this encoded knowledge. Such commercial test-cases would then provide concrete data on the advantages that the adoption of semantic web technologies can bring to classical information retrieval problems, and demonstrate their value in realistic situations.

However, this focus upon small-scale problems in commercial environments had two significant implications for the development of the QuestSemantics platform. The initial intention was that SERSE should be employed as the semantic indexing and retrieval component of QuestSemantics, including a re-developed and significantly enhanced search interface, along with a newly developed annotation

component. The first issue regarded the fact that SERSE was intended to handle large quantities of heterogeneous information, whilst the test-case contexts provided relatively small quantities of fairly homogeneous information. Therefore, SERSE could not be used to aggregate heterogeneous information, and, more importantly, was not required to handle the volume of metadata and resources involved. In fact, the small scale of these test-case deployments meant that the communication and processing requirements of SERSE were significantly higher than those needed for handling the metadata within a single RDF model (as demonstrated in Experiment 1 in Chapter 6). The second issue regarded the use of agent technology to underpin the distributed indexing mechanism in SERSE, and various problems with deployment of such systems within the IT infrastructure of the commercial partners. Not only does the agent system require significant processing and memory resources to operate, it also represents a potential IT security risk, by providing an open communication channel, that the partners were not willing to take on.

Therefore, it was determined that, whilst the option remained to utilse SERSE in this role, a specialised search-engine component should be developed for the QuestSemantics platform. The functioning of this component closely mirrors the main functions of a `RouterAgent` within SERSE – indexing resources and their associated metadata, and executing semantic queries over the metadata in order to retrieve the relevant resources. This specialised component would enable a much easier deployment of the QuestSemantics system onto other IT systems, requiring far fewer processing and memory resources from the host system than SERSE. However, despite the fact that SERSE is not suitable for the deployments undertaken to date, this does not mean that SERSE will not be deployed as the search engine component of QuestSemantics in any future deployment. In a situation where the volume of resources require a more scalable solution, or where heterogeneous information (annotated using heterogeneous ontologies) needs to be aggregated and integrated, SERSE would then become the more viable option for the search engine component. It is our intention to seek a test-case deployment project that requires the particular abilities of SERSE, in order to demonstrate its value in this situation. Furthermore, it should be noted that whilst the search engine component is not a direct development of SERSE, it is based upon the design of an individual `RouterAgent`, and the search interface in QuestSemantics is a direct development of the ideas and approaches employed in the SearchApplet of SERSE. Overall, QuestSemantics continues the focus upon semantic-based search and retrieval begun in SERSE, but does so in a way that may assist in the task of boot-strapping the generation of semantic web content, and attempts to disseminate these technologies to industry and commerce.

7.1.1 Annotation and Search

As previously described in Section 3.2, in a Semantic Web context, semantic annotation of resources is normally achieved by using or creating metadata items (as instances of ontology concepts) to represent specific entities recognised within information resources, and then linking this metadata description to the resource. However, one of the key problems of applying automatic annotation in unrestricted knowledge domains like the web is that it becomes extremely difficult to define a conceptualisation that captures sufficient knowledge to enable detailed descriptions of resources to be constructed. That is, despite efforts such as CyC [98] and SUMO [114], it has proved difficult to define 'general' upper-level ontologies, and, furthurmore, such ontologies cannot provide the level of detailed knowledge required. This lack of more 'general' ontologies has held back any large-scale efforts to generate general-purpose semantic annotations for web (or other) resources.

Using current approaches, the intelligence applied in search tasks, as well as the assessment of the relevance of retrieved pages, is mainly human, with limited support from software [154]. Whilst this type of processing is still adequate for domestic users, it cannot scale to the volume of information available to business, where the vast amount of data available on the web is coupled to company documents and databases. Current keyword based search engines present limitations in that they cannot fully capture the the richness intrinsic in natural language – examples are the problems caused by synonymy and polysemy to the search task. Enhancing search engines with lexicons such as WordNet [108] can help to relieve the problem, but is not sufficient to identify and resolve more complicated types of ambiguity. Furthermore, keyword-based search engines make little provision for the formulation of very specific queries, particularly those that make use of relationships between entities. However, as described in Section 3.4, by application of Semantic Web technologies (such as RDF and SparQL) very precise queries over metadata annotations can be executed. Such queries are usually expressed using the same ontology entities as are used to annotate relevant resources, thus making search over annotated resources very precise and accurate.

In order to generate semantic annotations that capture sufficient detail about the resources to enable precise semantic queries, we have focused on applying knowledge representation and manipulation within restricted and clearly delimited domains of knowledge. This enables the construction of detailed conceptualisations that provide the means to discover, annotate, filter and search for resources on the basis of fine-grained business knowledge. Once a suitable ontology, and relevant extension thereof, of the domain knowledge has been constructed, it can be used to identify and semantically annotate relevant resources. This ontology also underpins the provision of fine-grained search-based access to the identified knowledge resources on the basis of the domain knowledge. Thus, through the application of

knowledge representation and manipulation techniques, within specific contexts, we can apply a degree of 'intelligence' to the problem of access to information. This work has focused on providing a full annotation and search platform that can be applied to information retrieval in a variety of task and domain scenarios. In the annotation stage, we use both domain knowledge (encoded as ontologies) and task specific knowledge (e.g., layout specification, annotation and filter rules) in order to create semantic metadata about the information sources we want to exploit. This metadata then forms the knowledge-base for the search process, where the construction of user queries is guided by the domain knowledge, and these queries are then answered against the annotation knowledge-base.

The remainder of this Chapter presents the QuestSemantics software platform for semantic annotation and search, which is comprised of two main parts:

- Firstly, a general framework for the (semi-) automatic annotation of resources based upon a detailed ontological model of the domain – generating metadata representing knowledge about these resources.
- Secondly, a search interface for the user friendly formulation and execution of knowledge-based queries over the generated metadata – that utilises the represented knowledge without requiring users to handle the constructs necessary to formulate complex queries.

It is this search component that represents the development of SERSE – with the search interface being a direct development of the query applet interface, and the query answering being performed over a metadata model and linked resource index in the same manner as by an individual RA in SERSE. In addition, QuestSemantics was designed in order to maximise the separation between the different types of knowledge represented, that is domain versus task specific, and application versus generic knowledge. This separation is aimed at achieving reusability, and easy customisation of the various architectural components, thus allowing semantic based retrieval in a wide range of tasks and domains.

7.2 Platform Design

QuestSemantics is a generic framework for the automatic annotation and retrieval of semi-structured information sources based on semantic queries – by which we mean queries that are aware and make use of knowledge about the application domain. The QuestSemantics framework is designed for use within applications that aim to leverage different information sources in order to provide searchable knowledge. This overall requirement is often accomplished by means of process steps that only differ slightly

between different applications and different domains. The aim of our framework is to enable applications to abstract from all the details that are common, so that application specific code is reduced and simplified. In this way, the platform components are designed to be customisable, and can be adapted dependent upon the specific requirements of the domain it is applied to. Therefore, the main concern in the platform design is that its customisation only regards the domain-related aspects, i.e., an application of the framework is the result of providing *application specific knowledge* to the general framework. The application of the framework to annotation and search in a specific context then becomes a matter of generating the correct model representations, that encode the application and domain specific knowledge – i.e., the ontologies.

In our design we make a distinction between *domain knowledge* and *task knowledge*. Domain knowledge is the description of all relevant entities in a specific domain. Task knowledge, in general, uses the domain knowledge, and describes all the relevant entities with resepct to the tasks an application executes [159]. These two types of knowledge, though separate, are dependant on each other. In fact, the representation of domain knowledge cannot be entirely independent from the task the knowledge will be used for – this problem has been described in [21] as the *interaction problem*. Due to the modularity and abstraction of our approach, the only decisions taken at the platform design level regard the formalisms adopted for representing this domain and task knowledge. The domain knowledge represents a particular state of affairs and constrains the possible states it can evolve into – that is, static knowledge. Therefore, a domain ontology needs a formalism that allows to easily express taxonomical and non-taxonomical relationships among entities, and other properties of these entities. Task knowledge, on the other hand, describes the ways to perform useful changes to the domain states. Therefore, a task ontology requires a means to represent dynamic operations like sequences, selections and iterations.

The Semantic Web standard for representing ontologies is the Web Ontology Language (OWL) [106]. This Description Logic-based language (as described in Section 3.3), provides the majority of the necessary constructs to represent domain knowledge. However, OWL is not itself able to express the dynamic operations required to represent some relationships, and so, in these situations rules are added on top of OWL ontologies, represented using OWL's standard candidate extension SWRL [75, 74]. One example of such expressive extension was used to describe meronomic relations [168], that were required to be expressed in a domain, and Description Logic was not expressive enough to formalise them. The task (or procedural) knowledge, on the other hand, was represented utilising a combination of declarative rules and procedural definitions in a traditional programming language (in this case Java). Tasks then are represented by clauses (i.e., a set of premises in conjunction and a single consequence) whose consequence is represented by a block of code to be executed (as described in Section 7.2.2).

In the reminder of this section we discuss the main aspects underlying the design and implementation of the QuestSemantics framework.

7.2.1 System Tasks and Components

The issue that QuestSemantics is trying to solve is how to annotate and subsequently retrieve information from, possibly heterogeneous[2], resources that are described with respect to an ontology that formalises the application domain. QuestSemantics addresses this issue by providing a framework for the semi-automatic annotation and retrieval of knowledge regarding a specific business application. As described above, the QuestSemantics framework is based around two software components: an annotation engine to analyse, annotate and filter the retrieved documents; and a semantic search engine to provide fine-grained access to the annotated resources. In the annotation component, we use both domain knowledge (encoded as ontologies) and task specific knowledge (e.g., layout specification, annotation and filter rules) in order to create semantic metadata about the information sources we want to exploit. This metadata is then used in the search component, where specific queries from the application user are answered using the domain knowledge to guide the query process.

Figure 7.1 depicts the general architecture of the system, showing the annotation and search components, the specialised user interfaces, and the various knowledge sources that are utilised within the system. The diagram also shows the system's shared store component, that is responsible for all system data storage, including resource pointers, ontologies and generated metadata, as well as the intermediate results created by the analysis and annotation processes. In the following sub-sections we describe in detail the function of the annotation and search components, and how they interact with each other.

7.2.2 Annotation Component

The annotation component retrieves information resources and then analyses, annotates and filters them based on the application requirements – as encoded in the ontology and rules. Each of these functions is performed by a specific decoupled element within the annotation component. The annotation component utilises both the task-specific knowledge and domain knowledge, but strictly distinguishes between their use by the different elements. For example, the analyzer element uses the task specific knowledge to find relevant information within a resource, whilst the annotator element uses domain knowledge in

[2]Despite the focus upon restricted domains of knowledge in order to reduce knowledge heterogeneity, the different resources will remain hetergeneous in terms of format, layout, content, etc.

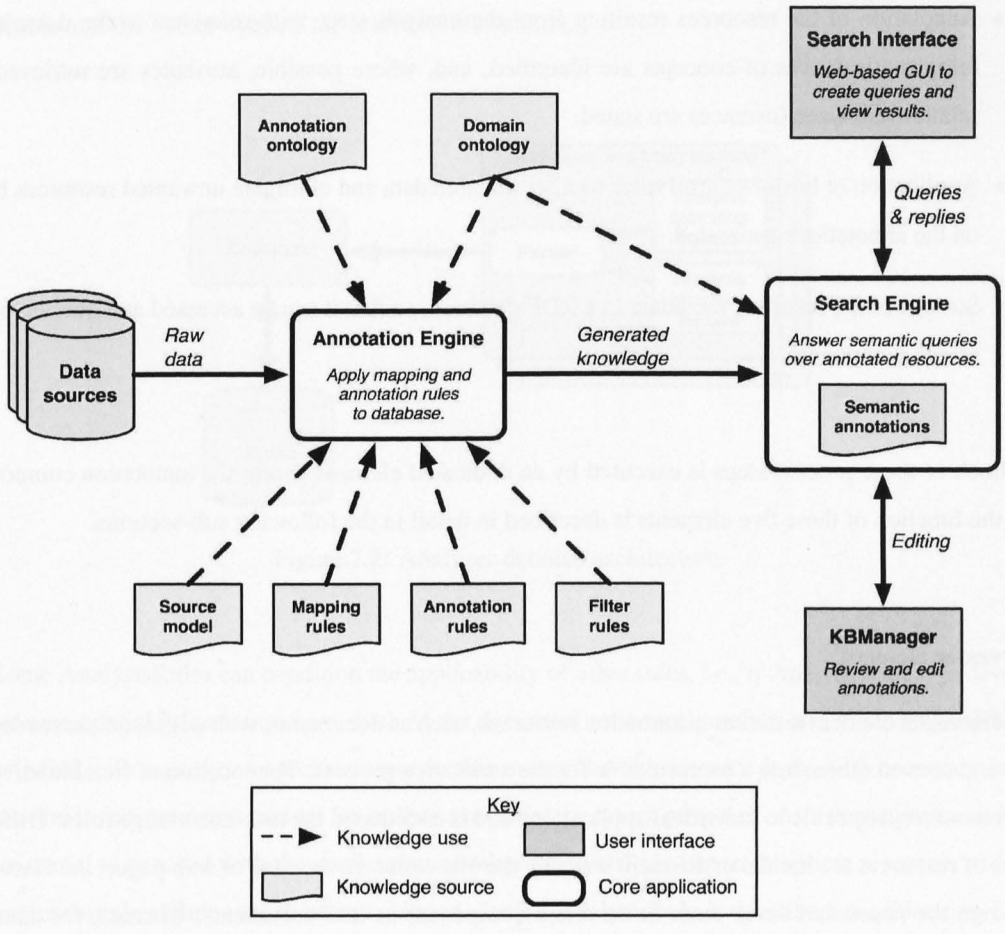


Figure 7.1: QuestSemantics system architecture.

order to create the actual metadata. The intention here was to develop independent system elements by leveraging the distinctions between the knowledge needed for each function, so that changes in task or domain can be performed independently, and have predictable implications for each element – simplifying the design of new applications. Moreover, confining the task specific knowledge to the annotation component makes the search component completely agnostic with regard to the way information is obtained, enabling the use of multiple, heterogeneous information sources to answer users' queries.

The Annotation process is composed of five main steps:

- Harvesting of live information resources, with no intermediate storage required, so retrieved information is up to date with the latest available.
- Analysis of the retrieved resources using the task knowledge encoded in the heuristic rules that specify which resources, and which parts of those resources would be of interest to the annotator element.

- Annotation of the resources resulting from the analysis step, with reference to the domain ontologies: instances of concepts are identified, and, where possible, attributes are retrieved and relations between instances are stated.
- Application of business-level rules to filter the metadata and eliminate unwanted resources based on the annotations generated.
- Storage of the resulting metadata in a RDF database, so that it can be accessed and queried by the search component.

Each of these process steps is executed by an dedicated element within the annotation component, and the function of these five elements is described in detail in the following sub-sections.

Harvester element

The Harvester element retrieves information resources, such as documents, web-pages, database records, etc., and converts them into a form suitable for the annotation process. The location of suitable information resources is specific to individual applications and is encoded in the task-knowledge rules. Different types of resources are handle in different ways by the Harvester. In the case of web pages, the Harvester retrieves the pages and saves them in the Store component as text documents, whereas, for database sources, it first retrieves the database schema and then the table contents, and saves them in an XML format.

Analyzer element

The Analyzer element performs operations to extract relevant information from an input resource, and to then store this information in an intermediate format suitable for use by the Annotator element. The general architecture of this element is sketched in Figure 7.2. The Analyzer element identifies relevant content within resources by use of layout information. This information is encoded in the form of regular expressions or with specialized Java code, where no regular expression could be devised, into an implementation of the MatchingPattern interface. A set of such MatchingPattern implementations is used by a Parser implementation, where the Parser is responsible for extracting the matched content. The combination of a Parser and its MatchingPattern elements forms an AnalysisRule. AnalysisRules, in turn, are considered as atomic objects, meaning that the relevant information found by the MatchingPattern elements inside a AnalysisRule are only extracted if all the MatchingPatterns are found to be satisfied in the input resource – when all MatchingPatterns fire on a resource, the AnalysisRule is

applicable and the Parser extracts the relevant information.

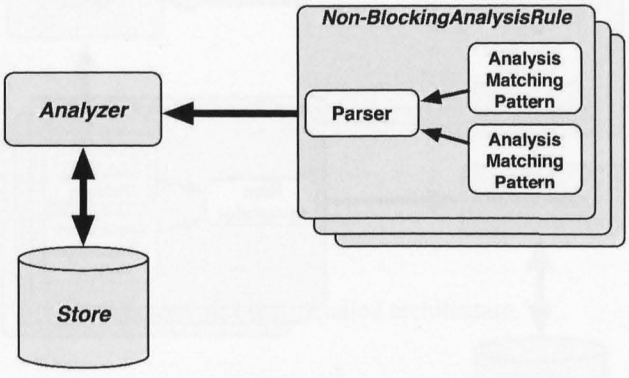


Figure 7.2: Analyzer detailed architecture.

Some AnalysisRules can condition the applicability of other rules, i.e., if AnalysisRule *A* is devoted to find an essential information item, such as the document reference number, and this AnalysisRule is not satisfied with regard to a specific resource, this means that the resource is missing necessary information and is thus rendered useless for further annotation or search purposes. As a consequence, it is no longer useful to proceed with the application of other AnalysisRules upon this same resource, and any subsequent AnalysisRules are therefore skipped. Such a AnalysisRule is described as a Blocking Rule, whilst other AnalysisRules that do not act is this way are described as Non-Blocking Rules. When using the Analyzer over the XML structure extracted from a database, its primary function is to translate from XML into the system-internal intermediate format.

Annotator element

The Annotator element is intended to create the RDF models containing the relevant items of information extracted by the Analyzer, thus building resource metadata according to the domain and application specific ontology(s). The general architecture of this element is sketched in Figure 7.3. In analogy with the internal structure of the Analyzer element, annotation is performed by means of AbstractDocumentMatchingPattern implementations. Each implementation extracts a specific piece of information from the Analyzer output, and then individual ItemAnnotator implementations utilise these items to generate and formalise the relevant metadata in an RDF model. This generated RDF model for an annotated resource will then form part of the overall RDF knowledge base upon which the Search component will operate. As with the Analysis element, ItemAnnotators and AbstractDocumentMatchingPatterns are grouped into AnnotationRules, which, as before, can be Blocking or Non-Blocking. The Annotator

element is the first step in the annotation process where the form of the source information becomes unimportant, i.e., it is agnostic with respect to whether the framework is used to retrieve information from web-pages, documents, databases, etc.

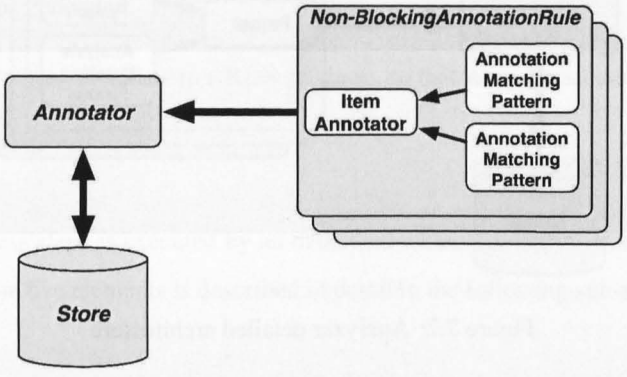


Figure 7.3: Annotation Engine detailed architecture.

Filter element

The role of the Filter element is to apply some pre-defined FilterRules over the generated metadata to determine whether a specific annotated resource is suitable for use by the Search component. One example of the uses of the FilterRules is the removal from the knowledge-base of resources that are no longer up to date – e.g., some information can expire after a certain amount of time, like a call for papers which is not useful after the submission date has passed. As with the AnalysisRules and AnnotatorRules, the FilterRules form part of the application-specific knowledge utilised by the framework. However, whereas the other rules encode information about how to process the source material, FilterRules regard application-level information about specific exceptions to the annotation process results. The general architecture of this element is sketched in Figure 7.4.

Store element

Each step of the annotation process, as well as the search process itself, produces data that must be saved persistently – both for performance reasons, such as storing harvested resources so that they are available for analysis, and to keep track of connections between information items, such as the source resource for a specific annotation. The Store element enables an application to save and retrieve different types of data, with each data item identified by a URI. The different data-types employed include byte streams (typically containing text documents such as HTML pages), Java maps containing intermediate

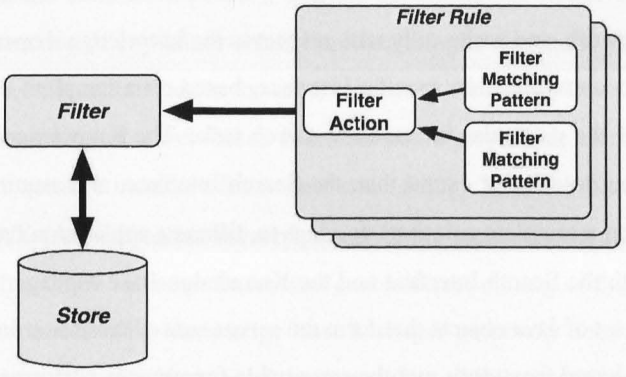


Figure 7.4: Filter detailed architecture.

mapping results, and RDF models containing completed annotations. In addition, the Store element is designed to enable the storage of specific information relationships, such as the fact that a given URI is an alternate name for another resource – that is, in OWL terms, the two resources are OWL:SAMEAS each other. This feature is particularly useful when a single conceptual resource is described by different physical resources that are available at different locations – the reference to any of these resources enable the annotation rules to retrieve all the available information for the resource being annotated, and thus aids in overcoming the problem of information logically related but phisically disconnected. In the current implementation of the framework, it is possible to use either a file based persistence implementation (using files for byte streams, Java Properties files for Java maps and RDF/XML serialization for RDF models), or a database implementation, where all the information items are stored within database tables.

7.2.3 Search Component

The Search component is responsible for executing queries over the resource knowledge generated by the Annotation component. Queries are expressed using the same domain ontology(s) and initial knowledge-base as used in the annotation process, in order to impose constraints upon potentially matching metadata elements. If a query matches one or more concept instances within the generated metadata the search engine then retrieves the annotated resource(s) represented by the matching instance(s). In this way the search engine component uses the domain ontology and knowledge base, and the generated annotation metadata and resource index, to enable access to the annotated resources on the basis of the encoded domain knowledge.

The Search component is divided into three distinct modular elements – the Search Engine, the

Search Interface, and the Knowledge-Base Manager. The Search Engine is a fixed component of the QuestSemantics framework, and varies only with respect to the knowledge it operates upon. In contrast, the Search Interface is an application-specific interface, based on a template solution that is adapted to the requirements of the particular information search task. The Knowledge-Base Manager is also application-specific, but to a lesser extent than the Search Interface, and requires only minor modifications (again based on a template solution) to adapt to different application requirements. The basic architecture of the both the Search Interface and the Knowledge-Base Manager is that of a web-based client interface, and a set of Java objects that form the server-side of these user interfaces. Communication between the web-based front-ends and the server-side functions is performed over HTTP using an AJAX implementation. Each of these user interfaces also communicate with the Search Engine in order to retrieve specific items of metadata. These three components are further described in the following paragraphs.

The Search Engine maintains and utilises a number of different knowledge structures to provide semantics-based retrieval of annotated resources. The domain ontology(s) and knowledge-base, composed of pre-defined instances of domain concepts, enable queries to be expressed using the same vocabulary as the annotations. Such queries are expressed in the current Semantic Web standard query language for RDF – SparQL [121] (as previously described in Section 3.4). SparQL was adopted for QuestSemantics, as a replacement for RDQL used in SERSE, following its submission to the W3C as the proposed standard RDF query language. However, SparQL is very similar to the preceding RDQL in both syntax and semantics, but has some more expressive constructs available. The SparQL queries impose constraints upon the properties and/or values of potentially matching instances within the generated metadata, by specifying one or more RDF statements in which any element of each statement may be a variable. SparQL queries are able to retrieve URIs for any of the three elements of RDF statements – subject, property and object, however, within QuestSemantics queries are always expressed in terms of retrieval of RDF object URIs representing individual instantiations of domain concepts. These URIs are then matched against the annotation index to retrieve those resources annotated with the matched instance. Thus, in response to a SparQL query, the Search Engine will return a set of pointers to matching resources. The type of pointer retrieved is dependent on the resource type, with the most common types being URLs for web resources and pointers to record-sets in databases.

The Knowledge-Base Manager component is intended to enable the users of a particular QuestSemantics application to review and edit the domain knowledge-base used within the system. This knowledge-base includes both the extension of the domain ontology, enabling the addition and removal

of information used in the annotation process, and the generated knowledge-base of resource metadata, to enable the correction of any minor errors introduced in the automatic annotation process. The Knowledge-Base Manager is architected in the same manner as the Search Interface (described below), with the component consisting of a web-based front-end and a server-side element that interacts with the Search Interface to retrieve metadata for review and editing. The web-based front-end is largely constructed using AJAX (Asynchronous Javascript and XML) technologies to provide an interactive interface consisting of multiple selection boxes that enable direct editing of the properties and values of existing metadata instances. The Knowledge-Base Manager purposely does not allow end users to edit the domain ontology, primarily because of the complexity of this process and the potential for introducing far-reaching errors into the knowledge model. However, this separation of concerns reflects the intended roles of ontology and knowledge-base, with the ontology providing the (relatively static) abstract model of entities in the domain, and the knowledge-base representing the actual entities within the domain, that are far more dynamic and may change regularly.

The Search interface provides the means by which final users (i.e., people who are not knowledge engineers nor experts in Semantic Web technologies) will access the resources annotated and filtered by the annotation component. The design of the Search Interface reflects two main concerns: firstly, the user should be presented with an clear abstract query model, in order to hide as much of the underlying representational complexity as possible; and secondly we want to use powerful semantic queries, that use as much of the available information as possible. Therefore the search component presents a specialised interface that enables users to develop a semantic query in an intuitive and non-technical manner. This is, in part, achieved by two key restrictions on the expressiveness of the queries being generated:

- All queries are for RDF objects that represent a specific instance of a domain concept, and queries can only return a single result set – i.e., queries can have only one result variable. This is in line with the annotation approach that generates a single domain concept instance to represent each annotated resource. Therefore, the object element of each RDF statement in the query is the result variable of the query. This approach was taken to simplify the query construction process, as users were far more comfortable defining only the properties and values of a sought resource, as compared to the use of multiple variables and instances to match resources.

- The use of boolean connectives between query triples is restricted, to simplify query construction and to avoid creation of unintended query configurations that do not express the intended semantics. This means that an *OR* or *NOT* connective is only applicable to the immediately preceding property and value specified – i.e., full use of the three connectives would also require the introduction of brackets to define their scope. However, users again found this level of complexity

confusing and were largely unable to employ it correctly.

The web-based interface itself is specialised with respect to the intended application and the type and available knowledge about the resources being searched over. However, this specialisation largely regards the presentation of the search results, and the query generation process is very similar across different domains. This generation process consists of the user defining a number of properties and values, using drop-down selection lists of available properties and object values, and direct input of data-type values. Once the user submits the query, the interface uses the abstract query model to automatically generate a SparQL representation of the specified query. This query is then submitted to the Search Engine, and any result subsequently returned to the Search Interface. The presentation of the query results may vary significantly depending on the type of resources being retrieved, and a single application of the framework may incorporate multiple result layouts to handle different types of available resources. A typical results presentation would consist of a results list in which brief details of all the matching resources are listed, and selection of any of these results would display further details regarding the result. Such further details might consist of displaying the annotation metadata regarding the result, linking directly to the resource (web-page, document, etc.) itself, linking to other resources relevant to the resulting resource (e.g., a map location of a retrieved address), or any combination of these approaches.

The interface also supports query modification, enabling users to edit, add and delete query elements within the graphical interface. Such modification can be performed at any time prior to submission of a query, enabling users to correct mistakes without having to discard an entire query. Furthermore, users can link back to a query from the results produced by that query, enabling users to interact with the knowledge base in order to fine tune the query. In this way a user can iteratively view the results of a query and then modify it so that the results of subsequent searches may become more accurate.

7.3 Development and Deployment

In this section we will first briefly review the main implementation details of the QuestSemantics framework. Following this, we will move on to describe in detail each of the two test-case commercial deployments of the framework.

7.3.1 Development

The QuestSemantics framework is primarily built using Java Standard Edition (version 1.5). The Semantic Web knowledge representation languages employed are RDF for metadata representation, and OWL [106] as the ontology specification language. In order to access the RDF models and OWL ontologies, we used the Jena Semantic Web Framework (version 2.4) from Hewlett Packard Labs [104]. In order to construct and refine the ontologies themselves, we utilised the SWOOP ontology editor[3]. Furthermore, the Pellet DL reasoner[4] was also used for ontology reasoning and consistence checking purposes. The language used to specify semantic queries over the knowledge base is SparQL, which is currently a W3C Candidate Recommendation. The implementation of the SparQL language used is the ARQ engine – developed by the Jena team and included as part of the Jena framework. In addition, XML [18] is used as intermediate format, in both the Store component, and as an intermediate result representation for the database-oriented implementation of Harvester. In order to process the XML data, the Xerces API for XML[5] was used, which is included in Java SE 1.5.

The framework development proceeded in a structured manner, and in parallel with the test-case deployments. The development began with knowledge modeling and construction of the ontologies and initial domain knowledge-base. Secondly, the ontological knowledge was then leveraged to construct both the Annotation Engine component and the annotation and filter rules used within it. Finally the ontologies and annotation meta-data form the knowledge-base over which the Search Engine component executes queries and retrieves results. However, there was some degree of overlap between these tasks in order to ensure a smooth development progression.

7.3.2 Deployment

The QuestSemantics framework has been deployed in two different commercial test-cases, both of which had a requirement for more 'intelligent' searches over business information. In the first case the requirement was to search web documents for appropriate commercial opportunities, based on specified areas of business interest. QuestSemantics was applied to this task of information retrieval, to enable domain-specific identification and filtering of the available documents. In this case the domain ontology and knowledge-base represent the relevant areas of business (i.e., specific sectors, markets, activities, etc.), and knowledge about the source material regarding how to select, annotate and filter those resources on the basis of the business knowledge. Using QuestSemantics this search now produces more accurate results – returning smaller sets of documents that are potential matches for the business criteria.

[3] http://www.mindswap.org/2004/SWOOP/
[4] http://pellet.owldl.com/
[5] http://xerces.apache.org/

The increased accuracy aids identification of suitable resources that may be over-looked in the current human-filtering process due to information-overload. In addition, providing fine-grained access to resources through an advanced search interface enables on-demand search, on the basis of the business knowledge, for resources matching specific criteria.

The second commercial test-case concerns knowledge-based search over database information resources – enhancing an existing facility to search for companies in the aerospace domain. The knowledge represented in this ontology is a conceptualistion of the aerospace domain in terms of the features, capabilities and business relationships applying to companies within that domain. This conceptualisation is instanciated to describe the specific companies and ancillary information, by gathering information from existing database resources – applying annotation rules, layered on top of the ontology, specify how the existing information is automatically mapped into the new representation. The primary benefits of the enhanced search facility are that it produces more accurate results for all types of search over the company information, leading to more efficient identification of potential partner companies.

These two commercial deployments are described in more detail in the following sub-sections.

Vectra Group Ltd.

The first commercial partner was Vectra Group Ltd. Their problem was one of information overload – they need to search documents, published on the web, for appropriate commercial opportunities, based on their areas of business interest. However, their current search service only matches a set of keywords, representing these areas of interest, against the entire publication set. Therefore, there are too many resulting documents, many of which represent false-positive matches, and the documents then need to be human-filtered to determine if they do represent suitable commercial opportunities.

The aim of this project was to apply QuestSemantics to this task of information retrieval, to enable more domain-specific identification and filtering of the published documents. The primary benefits to Vectra are that the system produces more accurate results from the resource filtering process, producing smaller sets of web-pages considered to be potential matches for Vectra's business criteria. This allows Vectra to concentrate efforts on a more precise set of results, reducing the time spent human-filtering of documents to determine which of the identified possible matches are actual matches. The increased result accuracy also aids identification of suitable resources, that may be over-looked in the current human-filtering process due to information-overload. In addition, providing fine-grained access

	Contracts	British Contracts	Returned Matches	True Matches
Total	34285	2894	199	7
Daily average	836.22	70.59	4.83	0.17

Table 7.1: Summary of Vectra test-case evaluation.

to potentially matching resources through an advanced search interface enables Vectra to perform on-demand search, on the basis of the business knowledge, for resources matching specific criteria, rather than having to determine this by a manual search of all resources. For Vectra the knowledge representation formalisms are used to encode knowledge about the areas of business in which they are interested (i.e., specific sectors, markets, activities, companies, locations, etc.), and knowledge about the source material and how to select, annotate and filter those sources on the basis of the business knowledge.

The performance of the annotation and filtering system was evaluated in the first test-case, when applied to the problem of identifying web resources that match specified business interests. The evaluation examined a large-scale harvest of 34285 published documents, determining how many were returned by the QuestSemantics system, and, of these, how many are of genuine business interest to the customer, see Table 7.1. These results demonstrate that only a very small fraction of the published documents are of genuine interest – which matches with expectations. Furthermore, the results show that the semantic annotation and filtering process is performing well, eliminating over 93 percent of unmatched British documents, and these results for QuestSemantics compare very well with the current service. A full comparative evaluation is still ongoing in this regard, however, spot-check comparisions over individual daily returns show an average reduction in non-relevant returns of 71 percent.

North West Aerospace Association

The second commercial test-case concerns knowledge-based search over database information resources. The North West Aerospace Association (NWAA) maintains a database of its member aerospace companies, giving details of these companies, their areas of expertise, specific capabilities, etc. Access to this database is provided through the NWAA website, enabling interested parties to search for aerospace companies. However, the current search facility is inflexible – there is only a basic categorization of activities, capabilities and approvals, and there is no facility to combine search features. This means that searches can only be approximate and do not allow identification of companies exhibiting specific combinations of features, unless manual cross-referencing of search results is performed.

The application of QuestSemantics enables creation of a knowledge-base, based on an ontology of

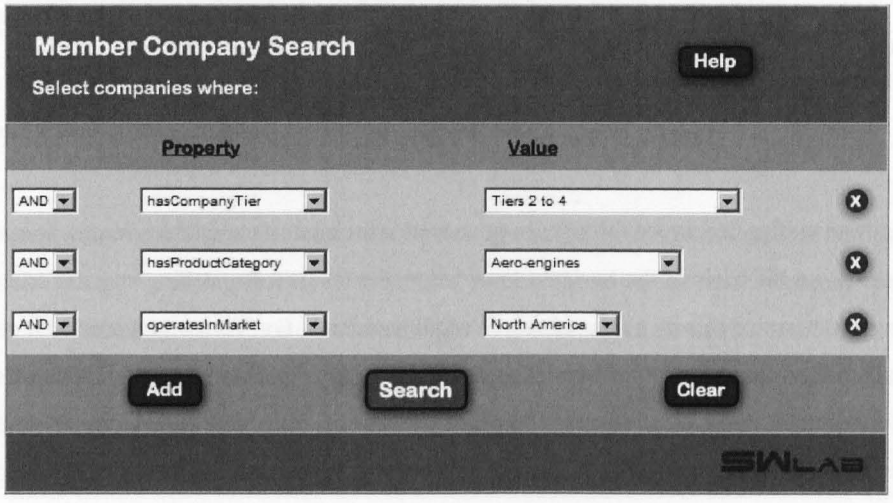


Figure 7.5: NWAA search interface.

the domain, using the company data currently held in NWAA's database, and provides a semantic search facility that allows the knowledge-base to be searched by constructing specific queries based upon the ontological model. The semantic search offers search over multiple, hierarchically structured categorisations and features, combination of categorizations using boolean logic, aggregation of results over similar categories, reference to specific company features within search constraints, etc. The knowledge represented in the ontology is a conceptualistion of the aerospace domain in terms of the features, capabilities and business relationships applying to companies within that domain. This conceptualisation is then instanciated to describe the specific companies and ancillary information, gathered from existing database resources. A detailed presentation of the NWAA domain ontology and initial knowledge-base are presented in Appendix B. Finally, the annotation rules, layered on top of the ontology, specify how the existing information is automatically mapped into this knowledge representation. The primary benefits to NWAA and its members of the enhanced search facility are that it produces more accurate results for all types of search over company information, leading to a saving of company time analysing search results in order to identify potential partner (customer, supplier, etc.) companies.

The Search Interface element for NWAA can be seen in Figures 7.5, 7.6, and 7.7. Figure 7.5 shows the query generation screen, where users construct semantic queries within an abstract query model, using a set of drop-down selection boxes to specify the required properties and values of the sought resources. Figure 7.6 shows the results list screen, displaying the full set of query results along with brief details, and a link to further detail regarding each resource. Figure 7.7 shows the result detail screen, in which the metadata information is displayed and categorised according to the ontology.

Figure 7.6: NWAA search results list.

7.4 Lessons Learned

This section provides a summary of the main lessons learned in the development and deployment of QuestSemantics, both with regard to the current system and general observations about the use of knowledge-based search in restricted domains. As can be seen from the evaluation in the Vectra test-case (see Section 7.3.2), the application of knowledge representation methodologies to intelligent data capture and access can produce very successful results. The significant reduction in false positive returns produces savings in company time and effort expended to identify opportunities, and helps to reduce the likelyhood that suitable opportunities are missed due to information overload. In addition, as shown with both Vectra and NWAA, the facility to access the data resources on the basis of the encoded business knowledge enables users to identify useful resources in a way that is tailored to their needs and experience. Furthermore, our focus upon limited and clearly defined domains of knowledge enables the business partners to specify the conceptualisation needed to apply their implicit knowledge about their business to the problem tasks in an automated manner.

As a result of the two test-case applications of the QuestSemantics system, including feedback from the business partners, a number of lessons have been learnt regarding the application of knowledge representation and manipulation techniques within commercial scenarios.

- Initial feedback from the companies approached regarding development of QuestSemantics, was that they require end-to-end solutions that solve specific business problems, and it is then our task to develop an integrated system of knowledge representation and other technologies to solve the whole of that problem. Many previous semantic web tools in this area provide solutions for only a part of the task – which is useful for research purposes but does not present a commercial IT

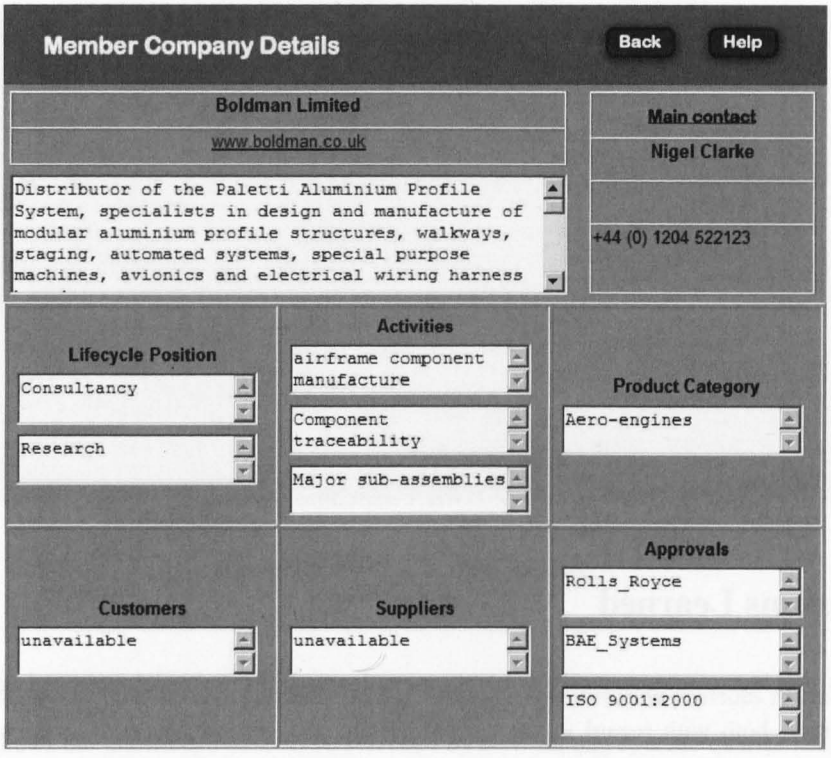


Figure 7.7: NWAA detailed company results.

solution.

- A significant element of the task of recruiting business partners was the need to 'sell' Semantic Web technologies, and in particular the specific business benefit of utilising Semantic Web technologies in preference to more established technologies, such as databases for storage of categorised data, and XML as a metadata representation. One of the key lessons learnt was that in order to pursuade companies about the potential of the Semantic Web, it was necessary to focus upon the ease of knowledge representation and manipulation using SW languages, rather than on the more typical arguments that leverage the shared knowledge and large-scale semantic markup aspects of the envisioned SW.

- The knowledge elicitation process requires significant time and effort, but, in our experience, the rich expressivity of the description logic and rule formalisms employed provides a straightforward means to encode the knowledge required. However, it is usually the case that a knowledge engineer would be required in addition to the domain experts to accurately encode this knowledge – that is the company experts needed the assistance of a knowledgeable user of SW knowledge

representations. It was clear from the company experts engaged with that the SW languages employed provide some assistance here by allowing concepts, properties and values to be represented in a natural way that supports an expressive but clear representation.

- To enable the business partners to make full use of the application, the presentation of the knowledge is as important as its representation. The business partners needed the entire display of the search interface and results to be customised to meet their needs, along with specific formats of result presentation that only exposed precisely those features of the data that were required by their specific task. Furthermore, the most appropriate results presentation also differed for different types of knowledge and resource. These requirements lead to the conclusion that each application of QuestSemantics would require a specialised user interface, but based upon a common core layout and functionality.

- Despite the relative ease with which Semantic Web languages can be used to encode knowledge, the majority of users encountered still required most of the knowledge representation detail to be hidden from them to enable them to easily make use of the system. Where possible the user interface should guide and constrain the possible actions of the user to assist them in generating accurate queries.

- Application of semantically enabled annotation and search requires both a detailed domain model and a set of rules or policies to guide the process. Within strictly delimited domains of knowledge it is relatively easy to construct these models and rules, but the larger the domain the more difficult it becomes to create a model that provide both the scope and detail required to support semantic annotation and search.

Finally, the focus upon maintaining a strict separation between the various different types of knowledge represented, both problem-specific and generic, underpins the flexiblility of the approach, and enables its application to almost any domain, given a sufficiently detailed ontology and annotation rules.

Part V

Synopsis

Chapter 8

Conclusions

This chapter summarises the work presented in this thesis, examines the extent to which the research objectives presented in the Introduction have been achieved, and presents possible avenues for ongoing research and development based around the two existing implementations.
Section 8.1 presents a brief analysis of the design and implementation of the SERSE *and QuestSemantics systems, and reviews the unique contributions of the work presented in this thesis. Section 8.2 then discusses how both systems have satisfied their use-case requirements, and summarises the results of the experimental evaluation of* SERSE *and the evaluation of QuestSemantics. Finally, Section 8.3 presents a summary of the ongoing and possible future developments of* SERSE *and QuestSemantics.*

8.1 Summary of Completed Work

This thesis began by presenting relevant background material to support the subsequent presentations of SERSE and, to a lesser extent, QuestSemantics. This background material introduced the fundamental technologies of the Semantic Web and gave details on core tasks within the domain – such as semantic annotation, semantic querying, and ontology alignment. This material also described the concepts of autonomous agents and multiagent systems, and went on to explain the ideas underlying agent-based information systems in order to contextualise SERSE as a cooperative information system. In addition, the background describes the basis of peer-to-peer systems, their relationship to multiagent systems, and the means by which queries can be routed through distributed indexes.

The main presentation of this thesis has been SERSE – SEmantic Routing SystEm – a distributed multi-agent system composed of specialised agents that provides robust and efficient indexing and retrieval of semantically annotated, heterogeneous resources. The agents composing SERSE use concept

descriptions represented in ontological models to search for and retrieve semantically annotated knowledge sources, by cooperatively maintaining a *semantic index* of the relevant instances of those ontologies utilised in the annotations. This distributed semantic index is then used to answer semantically specified queries over the total set of annotations, and thus retrieving resources on the basis of their metadata. The robust nature of the system is supported by the implementation of autonomic behaviour techniques, characterised by self-management and self-healing capabilities, aimed at permitting the system to manage the failure of one or more of its agents and ensure continuous functioning.

The primary aim of the system design was to enable the system to handle large volumes of annotation metadata though sub-dividing and distributing the global index into a number of interconnected indexes, managed by specialised agents. Each agent in SERSE creates and maintains a local *semantic index* of the instances of the concept from the annotation ontologies for which they are responsible. Efficient retrieval is made possible through the semantic routing mechanism, which permits identification of the agent indexing the resources requested by a user query without having to maintain a central index or resorting to broadcast messages. The semantic routing mechanism and the determinations of interconnections between agents are both based on the specialised Semantic Relatedness Metric (SRMetric) developed within SERSE. The SRMetric utilises the ontological definitions of concepts to determine the degree to which two concepts are *related*, either in terms of their *similarity* or other defined relationship.

The main contributions of the research and development of SERSE regard a number of different issues. The use of autonomous agents to manage elements of a sub-divided semantic index that are each indexes regarding a specific ontological concept, and interconnecting these agents on the basis of the relationships between the concepts they manage is, as far as we are aware, a unique approach to indexing Semantic Web metadata. In addition, the SRMetric is unique in its run-time evaluation of the degree of *relatedness* between concepts using only the ontological definitions, and its application to the problem of message routing in a distributed system. Finally, SERSE illustrates the application of techniques from autonomic systems within the multiagent system in order to support the fault-tolerance and self-maintenance of the system as a whole.

A number of important lessons were learnt in the design, development and evaluation of SERSE. Firstly, it is clear that the potential power of semantic querying over annotated resources offered significant benefits in terms of provision for precise query formulation and the consequent improvement in the precision of results. However, it is also clear that such precise querying was dependent upon a sufficiently detailed model of the relevant domain of knowledge, and that this model needs to be constructed

with the search task in mind. The various 'general knowledge' ontologies available do not provide sufficient detail to enable any large-scale annotation of web resources, and many existing ontologies are unsuitable for the task. These and other issues argue against the original conception of a 'bottom-up' construction of the Semantic Web, where metadata is generated on a wide-scale, representing heterogeneous resources that relate to diverse knowledge domains, and, thus, requires a large-scale indexing system that could inter-relate the represented domains. Furthermore, the initially anticipated volumes of Semantic Web metadata have not (yet) been forthcoming, and so the task for which SERSE was designed is not required at the present time – although we still anticipate that such distributed indexes that can determine relationships between entities from heterogeneous ontologies will soon be needed. Therefore, we re-established our focus upon the semantic annotation and search task employing a more 'top down' view of Semantic Web development in which knowledge is represented in strictly delimited domains and applied to specific tasks, with the intention of developing a large number of independent semantic webs that can be usefully applied without requiring the representations and systems to handle the openness, heterogeneity and complexity of the global Semantic Web. Finally, the development of SERSE demonstrated the difficulties inherent in designing a generic semantic query interface, and showed that that query and results presentation and workflow require adaptation to the specific details of the search task undertaken. Many of these lessons then lead to the development of the QuestSemantics framework.

This thesis has also presented the QuestSemantics framework, which is designed to enable automatic resource annotation and subsequent semantic search in specified domains. QuestSemantics provides a generic framework for semantic annotation of resources, based upon a set of models that encode the knowledge applied by the annotation process. This enables the framework to be applied to a wide range of sources, knowledge domains, etc. Furthermore, the modular construction of the system means that the functionality is provided by an extensible and modifiable set of tools, which enables the framework to be adapted to specific annotation tasks.

QuestSemantics offers many of the key benefits of searching over semantic metadata, whilst being tailored to work in specific domains of knowledge based upon a relatively simple ontological model. The primary reason for the focus being upon straight-forward semantic models that are applied in tightly restricted task contexts is that QuestSemantics is intended to provide a self-contained solution for semantic annotation and search in commercial environments. Therefore, one of the key objectives is to reduce the time and effort required for knowledge – elicitation and encoding – enabling the system to be applied relatively quickly and easily. This is assisted by restricting the search task undertaken, meaning that the variation in what can be searched for and retrieved is kept to a minimum. Both of these approaches are illustrated in the two commercial test cases:

- The Vectra ontology simply models the business activities, companies, etc. that the company is

interested in identifying in published tenders, and the search task is restricted to one of retrieving tenders exhibiting specified features.

- The NWAA ontology models the capabilities, activities, etc. of their member companies, and the search task is restricted to retrieving details member companies exhibiting specified features.

The annotation process performed by QuestSemantics is based upon multiple knowledge models. These models are comprised of the domain ontology (and its known extension), plus a number of supplementary models encoding other aspects of the process. These supplementary models include:

- Annotation ontology – encoding the relationships that can exist between the annotated resources and the automatically generated metadata.
- Source models – encoding the internal structure and organisation of the type of resources that are to be annotated.
- Annotation rules – encoding the conditions under which an annotation is created for a resource.
- Mapping rules – encoding the links between different input sources that relate to the same concept instance.
- Filter rules – encoding the conditions under which an annotated resource is accepted into or rejected from the generated knowledge-base.

The search facilities provided by the QuestSemantics framework enable the retrieval of both the semantic metadata and the annotated resources, using semantic queries that employ the same ontological vocabulary as used in the annotation. The construction of such semantic queries is facilitated by the system's query interface that provides a simplified means to select the constraints expressed to be in a query, which is based upon an abstract query model that slightly reduces the expressiveness allowed in such queries in order to reduce the complexity of the query expression. Once completed the abstract query models constructed in the query interface are translated into the semantic query language RDQL and then executed by the search engine component over the stored annotation metadata. Presentation of search results will usually differ dependent upon the specific search task and the information contained within the results. Each of the deployments of QuestSemantics have provided a specialised results presentation format, tailored to the task and results involved.

The development within the QuestSemantics system of many of the approaches utilised in SERSE, demonstrates that the application of knowledge representation methodologies to intelligent data retrieval

and access can produce very successful results. The deployments of QuestSemantics within small and medium enterprises (SMEs) to solve specific information access problems is an early-stage exploration of the potential value of Semantic Web technologies in commercial environments, and strongly indicates that their use can enable valuable functionality provided that the inherent complexity of the representations employed is sufficiently hidden from the end users. The key contribution of QuestSemantics is the demonstration that the knowledge representation and manipulation technologies can be sufficiently 'packaged' into solutions to specific commercial problems, based upon an extensible framework that can be relatively easily adapted to a wide range of annotation and retrieval tasks within any knowledge domain (given a suitable domain model).

As a result of the two test-case applications of the QuestSemantics system a number of lessons have been learnt regarding the application of knowledge representation and manipulation techniques within commercial scenarios. Companies usually require end-to-end solutions that solve specific business problems, and require an integrated system of knowledge representation and other technologies to solve the *whole* of that specific problem. The knowledge elicitation process requires significant time and effort, but, in our experience, the rich expressivity of the description logic and rule formalisms employed provides a straightforward means to encode the knowledge required. To enable the business partners to make full use of the application, the presentation of the knowledge is as important as its representation. The languages employed provide assistance here by allowing concepts, properties and values to be represented in a natural way that supports an expressive but clear presentation. Finally, our focus upon maintaining a strict separation between the various different types of knowledge represented, both problem-specific and generic, underpins the flexiblility of our approach, and enables its application to almost any domain, given a sufficiently detailed ontology and set of annotation rules.

8.2 Satisfaction of Requirements

In this section we examine the extent to which both SERSE and QuestSemantics have satisfied their design requirements, and thus how well they perform their intended tasks. The requirements for the SERSE system are detailed in the Introduction to this thesis. The primary requirements were to have a scalable and robust system, that was able to make full use of the semantic information available, and in which the use of intelligent agents offers significant support to these features.

- **Scalability** – The scalability (i.e the ability to cope well with increased system size and demands) of the system is based upon the use of specialised agents to manage the sub-divided index. The agents are able to be distributed over different host systems by virtue of the facilities provided

by the FIPA-compliant agent platform. There is no central index to effectively limit the overall volume of information indexed within the system. Furthermore, the fact that the agents composing SERSE can be hosted on any agent platform on any host system means that the system can be distributed over as many hosts as are required to provide the computing and storage resources needed.

- **Robustness and fault-tolerance** – The are a number of features within SERSE that underlie the system's robustness and fault-tolerance (i.e the ability of the system to cope well with and recover from individual component failure). Firstly, the sub-division of the semantic index ensures that the unavailability of any one `RouterAgent` will only affect the fragment of the index handled by that agent, leaving the remainder of the system fully operational. Furthermore, the system is able to recover from such localised failures by re-activating `RAs` and restoring their index knowledge from saved states (see Section 4.2.3). These features are enabled by the autonomic system processes that allow `RouterAgents` to determine when one of their neighbours has become unavailable, and to re-route neighbourhood interconnections accordingly.

- **Use of semantics** – SERSE is intended to make full use of the semantic information available in the resource annotations and the relevant ontologies. This has been achieved in three main ways within the implemented system: in the semantic indexes, in the use of semantic queries, and in the Semantic Relatedness Metric. The agent-managed indexes make use of the semantics by organising the entries on the basis of the concept instances, and by making explicit use of the instance inter-connections. The generated queries utilise the semantics by employing the RDF query language RDQL, which enables the queries to make specific reference to the properties and values of the concept instances present in the resource annotations. Finally, the SRMetric utilises the semantics encoded in the relevant ontologies to determine the similarity and other connections between the defined concepts.

- **Use of agents** – The use of autonomous agents to manage the sub-divided semantic index enables the system control mechanisms to be distributed on the same basis as the index itself. The individual `RouterAgents` exert local control over their own index fragment, and over their inter-connections with other `RouterAgents` This application of local control over each index fragment and neighbourhood inter-connections is a key feature of SERSE, and underlies both the scalability and fault-tolerance of the system as a whole.

SERSE underwent experimental evaluation of various aspects of its functionality. This evaluation consisted of a number of detailed experiments, each examining different features and functions of the system. The evaluation firstly tested the performance of the search and retrieval capabilities of the

system, and the experimental results show that SERSE generally maintains response times in an usable range, demonstrating that the computational and communication overheads due to the index distribution do not have a significant adverse impact on the system performance. However, this experiment also showed that the system is sensitive to the relative instance distribution, when answering specific types of queries, as we outlined when discussing the experimental results. Further experiments were conducted regarding the autonomic behaviour, and the results show how the system is able to efficiently re-configure itself as agents are added to and removed from the `RouterAgent` network. Finally, a number of experiments were conducted to evaluate the performance of the SRMetric underlying the semantic routing mechanism, particularly in order to compare its performance to other systems able to determine semantic relations between ontological entities, such as ontology alignment tools.

The requirements for the QuestSemantics framework are outlined in Chapter 7. The main requirements were to develop a system for semi-automatic semantic annotation and search, that could be applied in commercial environments to solve specific information access problems. Furthermore, the system was intended to be easily applied to different resources, domains, tasks, etc. by encoding these knowledge items in independent semantic models. Finally, QuestSemantics should require only relatively simple domain models, reducing the knowledge elicitation task, and, thus, meaning that the framework could be relatively easily applied to different problems, without needing prior extensive and time-consuming knowledge engineering.

These requirements drove the design of QuestSemantics as a modular framework of specific tools in which all the required knowledge is stored in ontological models and rules. The annotation process is supported by how the domain knowledge is represented in the relevant ontology, in particular in the way that each type of resource examined can be represented as an instance of an ontological concept, with the values of the properties being determined from further examination of the resource. The stipulation on the use of only relatively simple domain ontologies is adhered to through a number system features. Firstly, QuestSemantics is only able to use ontologies specified in OWL lite and OWL DL, thus keeping the language expressiveness to a manageable level. Secondly, the nature of the knowledge categorisation task at hand often leads to straightforward taxonomic hierarchies that are simply encoded. Thirdly, the required detail of the domain knowledge can usually be encoded within such categorisation hierarchies by use of extensive use of properties, in particular object properties that reference an instance within the ontology's known extension.

As can be seen from the evaluation in the Vectra use-case, the application of knowledge representation methodologies to intelligent data capture and access can produce very successful results. The

significant reduction in false positive returns produces savings in company time and effort expended to identify opportunities, and helps to reduce the likelyhood that suitable opportunities are missed due to information overload. In addition, as shown with both Vectra and NWAA, the facility to access the data resources on the basis of the encoded business knowledge enables users to identify useful resources in a way that is tailored to their needs and experience. Furthermore, our focus upon limited and clearly defined domains of knowledge enables the business partners to specify the conceptualisation needed to apply their implicit knowledge about their business to the problem tasks in an automated manner.

The evaluation of QuestSemantics consisted of both a limited experiment, conducted within one of the commercial test-cases, and a more subjective review of the operation of the system by both of the commercial partners. The annotation accuracy of the QuestSemantics framework was evaluated in the context of the Vectra Group test-case deployment, in which the framework was applied to the annotation and retrieval of contract tender documents. This evaluation demonstrated that the application of Semantic Web technologies provided the means to accurately annotate and subsequently retrieve the tender documents in terms of Vectra's specified business interests. When compared to the simple keyword-based selection process that was then employed by Vectra, the QuestSemantics deployment was able to demonstrate an approximately five-fold increase in precision, whilst retaining an equivalent level of recall. In addition, the operation of the QuestSemantics system was reviewed by both of the commercial test-case partners independently regarding how well they felt the system had addressed their particular information access problems. In both cases the feedback was mainly positive, and, in particular, highlighted the fact the the users found it intuitive to retrieve resources on the basis of specified properties and values. However, this feedback also highlighted the fact that further software support was required for the knowledge elicitation process, to enable this to be be largely achieved by the users independently without requiring repeated consultations with a knowledge engineer.

8.3 Ongoing and Future Development Directions

Finally, to complete the presentation of SERSE and QuestSemantics, we briefly overview the ongoing and potential development directions for both of these systems.

The SERSE system has a number of possible development directions, encompassing both revisions and extensions to existing features and the addition of new features and functionality. The elements of the system that are the primary candidates for further development are the Semantic Relatedness Metric, the inter-agent communication messaging, and the query generation and results interface:

- There are a number of possible refinements and extensions to the SRMetric that may improve the way in which it determines the degree of relatedness between ontological concepts. One of these is the use of flexible threshold values and weighting parameters for each of the metric components, which would enable the `RouterAgents` to exert autonomy over these parameters and thus have greater control over their role within the system. Another improvement would be to separate the semantic similarity and semantic relatedness elements of the metric, thus enabling each of these different estimations of semantic connections to be applied only where required. Finally, the efficiency of the existing algorithm implementation – in terms of the time required to execute a concept comparison, and, to a lesser extent, the computational costs (processor load, memory use) – can be improved upon. Any reduction in the time taken for an average semantic relatedness calculation would have a significant positive effect on the usability of SERSE as a large-scale indexing system.

- The messages exchanged between the agents within the system are currently implemented as ACL messages containing XML encoded information, that have different implied handling procedures dependent on the message type. A more open solution would be to utilise an existing content expression language, so that the implied handling procedure is instead explicitly expressed as an action requested by the sender of the recipient. This would also require the development of an ontology to encode the vocabulary require to express the requested actions.

- It is intended to refine and extend the query interface along the same lines as explored within the QuestSemaantics system. However, although many of the developments to the QuestSemantics interface can be re-applied here, significant differences must remain given that SERSE must perform a more generic search task, over a wider range of resources and using heterogeneous ontologies.

In addition to further development of the system functionality, further experimentation and evaluation work is planned, in order to more fully test the scalability and robustness of the distributed indexing system. Such evaluation will require the availability of large volumes of semantically annotated resources, that have been annotated using a number of independent ontologies that represent overlapping domains of knowledge. At the time of writing such large-scale data-sets are not readily available, particularly with regard to the use of multiple ontologies with overlapping domains.

As with SERSE, the development of the QuestSemantics system is continuing on a number of fronts. There are many ways in which the currently deployed applications could be improved upon, both in the existing tasks of annotation and search, and in extensions to the current system to address areas such as

knowledge management and business intelligence. The primary development directions of this future work are:

- Development of search interface – Further development of the user search interface to provide a simple, powerful and intuitive query construction and results presentation interface. This is currently focussing upon refinement of the query construction and editing methodology, enabling a more intuitive and flexible workflow, and permitting in-line editing of search constraints and data-type values. Work on the presentation of search results is focussing upon extending the search result ordering to allow a variety of rankings, based on different criteria, to be applied. Furthermore, we are exploring the potential of a declarative layout system for results presentation, so that the required layout for any specified set of results can be encoded in an ontological model, and thus improve the flexibility of the framework by removing the need to adapt the search interface software to suit result presentation.

- Development of annotation component – The accurate and detailed annotation of resources is the most important feature of the current system, and we are continuing to develop the annotation component to improve the results of this process. Specific developments currently ongoing are improvements to the use of natural language processing in the analysis phase, to provide further means by which sought information can be recognised and extracted. Specifically, we intend to use lexical engineering tools to apply grammatical and language rules to provide greater 'understanding of the text within resources, and use electronic dictionaries to supplement search term sets with synonyms, enabling matching of search terms when the expected words are not used. In addition, we are developing extensions to the use of filter rules to consider specific rule-exceptions, and thus allowing more flexible application of filters. Finally, we are enhancing the annotation lifecycle management system to revise and remove annotations describing resources, when required on the basis of various criteria, in a fully automated manner.

- Use of SERSE as SearchEngine component – As further suitable ontologies, metadata, resource sets, etc. become available, particularly where different resources are annotated using two or more ontologies regarding overlapping domains of knowledge, the need grows for an indexing and retrieval system that can generate interconnections between such heterogeneous ontologies. Our intention is to employ the multiagent system of SERSE as the Search Engine component of QuestSemantics, enabling the system to both index large volumes of metadata and inter-relate different annotated resource sets. This synthesis of the two systems presented here would enable the application of the new system to large-scale semantic annotation and retrieval tasks, where the knowledge is encoded in multiple ontologies representing the differing viewpoints of the different stakeholder groups within the scope of the application.

Finally, commercial partners continue to be sought in order to apply QuestSemantics to other information access problems. The understanding is that there are a significant number of commonalities in the various information categorisation, access and manipulation tasks that have been presented, and that by developing solutions based upon QuestSemantics to address different specific domains and tasks these commonalities can be elicited and leveraged. This would then enable reductions in the burden of knowledge elicitation and in development of specialised knowledge manipulation processes, and further simplify the adaptation of the framework for different tasks and domains. The long-term development target for QuestSemantics is to develop a comprehensive semantic modeling, annotation and search system that can be applied across a wide range of domains and sources.

There are a number of possible directions of future research under consideration, in relation to both SERSE and QuestSemantics, that have a wider scope than the (re-)engineering issues outlined above. Prime examples of these preliminary research ideas, that mainly involve how the kind of semantic annotation and search described in this thesis can relate to and benefit from other current web technologies, include:

- Since the initial design of the SERSE architecture there have been a number of significant developments in Semantic Web, distributed systems, and other technologies that offer the possibility of both refining the current architecture and entirely replacing it. Potential refinements range from simple new features like using a (semantic) web service as the access point to the `InterfaceAgent`, to more significant changes like utilising a web interface constructed using up-to-date web development technologies (as in the QuestSemantics interface). The more fundamental re-architecting of SERSE, fulfilling the same basic requirements for semantics-based, scalable and robust resource indexing and retrieval, could be based upon technologies such as semantic web services [53] – with each seperate concept-based resource index managed by a different web service process on, potentially, different hosts. The system could make use of the service discovery and location functionalities offered with current semantic web service platforms, and, in an extension to SERSE, could make use of the semantic service description formalisms available to identify the service both as a resource index that accepts semantic notifications and queries, and that the index handles resources relating to a specified concept. This would open the index network up to operation as individual indexes, as well as a part of one or more general indexing systems.

- A significant extension to the annotation component of QuestSemantics would be to find a means to include the now widespread use of folksonomic tagging. Use of such textual labels that exhibit

informal and implicit semantics is clearly disjoint from the Semantic Web approach to formal and explicit knowledge representation. However, due to the accepted semantics of tags within their specific community of use (fulfilling the SW requirement for shared semantics), we can attempt to generate mappings to SW representations, that then allow us to take advantage of the considerable volume of web resources that have been tagged. Initial ideas on how this may be achieved include: straightforward use of the lexical information in the tag label to map to concepts, plus use of online dictionaries / thesuri to support such mapping; use of the tag contexts, such as the contents of tagged documents, and commonly co-occuring tags to provide additional information to assist the mapping process; and leveraging the interest within communities of use to provide human-generated tag to concept mappings.

- Ranking of search results is clearly a vital feature within large-scale search tasks, as demonstrated by the role of the pagerank algorithm in the success of Google [71][117]. As mentioned above, there is existing work in the QuestSemantics system addressing result ranking based on the number of search clauses matched. However, an another approach would be to produce an analogy of the well-known pagerank that performs the same ranking task over concept instances. That is the relative 'strength' of different instances of the same (or potentially just significantly similar) concepts could be estimated on the basis of the usage of and references to that instance. Therefore, an instance that is widely used within RDF statements annotating resources, and is widely referred to in RDF statements in general, would rank above one that only appears in a few RDF statements. A search system could then use these instance rankings to inform search result ranking, with the rankings of the each of the instances referred to in the annotations of each resource being combined to contribute to the overall rank of the resource in the result-set. The issues to address in such ranking are similar to those faced by Google – primarily the vast size of the required index and the processing needed to continually update rankings. In addition, such instance ranking information would have to be globally available within the search system, which would be difficult within a distributed system like SERSE due to the additional communication, or replicated comparison processing that would result. Therefore, as a research topic this approach would best be developed within a system like QuestSemantics, that focusses on a small domain of knowledge represented in a centralised knowledge-base. However, this approach would also require a generally high-density of semantic annotations, just as page-rank needs lots of hyper-links, and is therefore not yet applicable to the wider web where semantic annotations are generally scarce.

- Both SERSE and QuestSemantics are able to index and retrieve URLs for any type of web resource that is semantically annotated, that is, has attached RDF statements describing properties of the resource in terms of ontological concepts. Technologies are available to attach such semantic annotations to a wide range of resource types, including semantic web services [116], HTML and

XML[65], images, sound clips, etc. Therefore, both systems are able to address indexing for the currently important research area of semantic web services. However, within a specialised web service application built upon QuestSemantics, additional resources could be brought to bear to extend the coverage of the resource index. A key example here would be to make use of the UDDI web service index[153], which would probably be addressed using a mapping between the existing descriptive elements used in the UDDI index and a Semantic Web representation of a service. In this way, such a web service index could contain both those services semantically annotated by their authors (or a third party), and those services described within existing indexes like UDDI.

Part VI

Appendix

Appendix A

SERSE Message Types

SERSE utilises many different message types, with each one being specifically tailored to suit an individual system function. However, wherever possible the same message type has been utilised for more than one function to reduce the number of message types required. These messages can be divided into two main groups: those messages that involve communication between agents in SERSE and software external to the agent system – external messages; and those messages exchanged between agents within the system itself – internal messages. An additional group of system messages are those utilised in the system testing only, and do not play a part in the functioning system. These three sets of messages are detailed in the following three sub-sections.

A.1 External Messages

External messages are those exchanged between agents in SERSE and other systems outside of the multiagent system – i.e., systems not hosted within a FIPA agent platform. SERSE has connections with two external systems – the Notification Server component of Esperonto, from which SERSE can receive messages regarding content acquisition, ontology publication and ontology modification; and the web-based query interface, from which SERSE receives queries and to which the replies are returned. The details of these messages and the ways in which they are handled are as follows:

Content Acquisition notifications

These messages originate from the Annotation system, and are generated when a wrapper newly acquires and annotates web resources. A content acquisition message is sent from the Notification Server, via the Notification Mediator, to the `NotificationAgent` on one of the agent platforms comprising SERSE. The `NotificationAgent` parses the XML message, and extracts the URIs of the notified

concepts, and the URL of the RDF file containing the generated metadata. For each of the concepts in the notification the NotificationAgent generates an ACLNotificationMessage, which are then sent into the RouterAgent network via the PortalAgent, and semantically routed to their destination RAs.

Ontology Publication notifications

These messages originate from the Ontology Server, and are generated when a new ontology is made available for resource annotation. An ontology publication message is sent from the Notification Server, via the Notification Mediator, to the NotificationAgent on all of the agent platforms comprising SERSE. The NotificationAgents each parse the XML message, and extract the ontology name and URI. This information is then sent to the local RouterPlatformAgent and InterfaceAgent as an ACLAvailabilityMessage. The RouterPlatformAgent adds the newly published ontology to its list of available ontologies, that is used by RouterAgents to determine their semantic neighbourhoods from all of the ontologies. The InterfaceAgent passes the information to the external web interface to update the ontology list, so that queries can be formed using all of the available ontologies.

Ontology Modification notifications

These messages originate from the Ontology Server, and are generated when an existing ontology (i.e., one already published by the Ontology Server) is modified. An ontology modification message is sent from the Notification Server, via the Notification Mediator, to the NotificationAgent on one of the agent platforms comprising SERSE. The NotificationAgent parses the XML message and extracts the ontology name and URI, and the set of declarative mappings from the original ontology to newly modified version. The NotificationAgent then forms an ACLOntologyModificationMessage for each of the concepts referred to in the original message, inserting the mappings concerning different concepts into the relative message. This set of messages are then sent into the RouterAgent network via the PortalAgent, and semantically routed to their destination RAs.

Queries

These messages originate from the web-based query interface, and are generated when a query is created and submitted by a user. Query messages are sent from the query interface, via the server-side interface component and a socket interface, to the local InterfaceAgent. The InterfaceAgent parses the XML message, extracting the RDQL query contained within it. This RDQL query is then placed in an ACLComplexQueryMessage and sent to the QueryManagementAgent.

Replies

These messages are generated by the `InterfaceAgent` following receipt of an `ACLQueryReplyMessage` from the `QueryManagementAgent`. The `InterfaceAgent` extracts the reply information from the ACL reply and forms these into an XML message. This XML message is then sent across the socket connection to the web interface.

A.2 Internal Messages

Communication between agents within SERSE, and therefore all existing on a FIPA agent platform, is achieved through use of FIPA Agent Communication Language messages. FIPA ACL messages are chosen from a set of performatives, which have fixed protocols for actions and responses regarding each different performative. SERSE uses only `Inform-ref` and `Query-ref` messages, and the contents of these FIPA ACL messages vary dependent on the purpose of the message. The messages utilised within SERSE are contained within the $\langle Content \rangle$ field of the FIPA ACL messages, as string serializations of XML messages. For each message type there is an XML Document Type Definition and a corresponding Java class used to represent the message. The type of a message is recorded in a user-defined parameter of the FIPA ACLMessage class, and is one of the following:

ACLAvailabilityMessage

Uses the FIPA `Inform-ref` message performative. These messages are generated by the `NotificationAgent` when it receives a notification message regarding the publication of a new ontology on the Ontology Server. This message is sent to the local `RouterPlatformAgent` to make it available to all `RouterAgents`, and to the `InterfaceAgent` to inform it of the availability of this ontology for the formation of queries. This message type defines a field for $\langle OntologyURL \rangle$ that contains a pointer to the newly published ontology.

ACLCreateRouterMessage

Uses the FIPA `Inform-ref` message performative. These messages are sent by `RouterAgents` to the local `RouterPlatformAgent`, when they determine that a received content acquisition notification refers to a new concept that is not handled by any existing `RouterAgent`. This message type defines fields for:

- $\langle ConceptURI \rangle$ and $\langle OntologyURL \rangle$ that identify the indexing responsibility of the new `RouterAgent`;
- $\langle DonorRI \rangle$ that holds the senders routing index;

- $\langle WrapperID \rangle$ that holds the ID of the wrapper that acquired and notified the resource.

ACLComplexQueryMessage

Uses the FIPA `Query-ref` message performative. These messages are generated by the `InterfaceAgent`, following reception of a user-generated query, to represent the query within an ACL message. These messages contain a string serialization of the RDQL query that is output from the web-based query interface, and are sent to the local `QueryManagementAgent`.

ACLComplexQueryReplyMessage

Uses the FIPA `Inform-ref` message performative. These messages are sent by a `QueryManagementAgent` to the local `InterfaceAgent` as a response to a complex query. They are generated by the `QueryManagementAgent` following re-aggregation and optimisation of the results of the atomic queries, contained in the relevant `ACLSimpleQueryReplyMessages`. This message type defines fields for:

- $\langle ReplyTo \rangle$ – holding the message ID of the query to which this message is the response;
- $\langle Results \rangle$ – holding the result set for the query concept. This result set is formed by a list of resources, each coupled with a list of instance URIs that caused the resource to be retrieved;
- $\langle Neighbours \rangle$ – holding an aggregated list of the semantic neighbour concepts of all of the replying `RouterAgents`.

ACLFailureMessage

Uses the FIPA `Inform-ref` message performative. These messages are generated by `RouterAgents` when a query or notification message has exceeded a set number of re-transmissions without finding the appropriate agent – as indicated by the $\langle Deathclock \rangle$ within these messages. The failure message is sent to the `PortalAgent`, and then forwarded to either the `QueryManagementAgent` or `NotificationAgent` depending on the type of message that has failed. These messages have one field for $\langle FailedMessageID \rangle$ that contains the ID of the message that has failed.

ACLHeartbeatMessage

Uses the FIPA `Inform-ref` message performative. These messages are generated by `RouterAgents` in order to 'ping' their semantic neighbours, if they have not successfully communicated within a pre-determined time-frame.

ACLNeighbourLocatorMessage

Uses the FIPA `Query-ref` message performative. These messages are sent by `RouterAgents` following their creation, in order to discover the `RouterAgent` responsible for handling a neighbouring concept, for which the sending `RouterAgent` does not yet have an agent GUID. These messages are sent from a `RouterAgent` to its local `PortalAgent` to then be semantically routed to the appropriate `RouterAgent` for the neighbouring concept. In addition to fields identifying the sought concept, the message has a $\langle NeighbourGUID \rangle$ field, identifying the sending `RouterAgent`.

ACLNeighbourLocatorReplyMessage

Uses the FIPA `Inform-ref` message performative. These messages are sent by `RouterAgents` following receipt of a `ACLNeighbourLocatorReplyMessage` referring to the concept for which they are responsible. These messages are sent directly to the sending `RouterAgent`, and contain fields identifying the concept and the agent GUID of the sender.

ACLNeighbourNotificationMessage

Uses the FIPA `Inform-ref` message performative. These messages are sent by `RouterAgents` following their creation, to those `RouterAgents` that handle neighbouring concepts, and for which the `RouterAgent` already has a GUID – obtained from the donor routing index of the `RouterAgent` that requested the creation.

ACLNeighbourShutdownMessage

Uses the FIPA `Inform-ref` message performative. These messages are generated by `RouterAgents` as part of their shutdown process, and are intended to inform the neighbours of a `RouterAgent` that it will no longer be available for semantic message routing.

ACLNotificationMessage

Uses the FIPA `Inform-ref` message performative. These messages are generated by the `NotificationAgent` following receipt of a content acquisition message from the Notification Server, and contain the information required by the appropriate `RouterAgent` for it to retrieve the metadata for the newly annotated resources. The `NotificationAgent` sends these messages to the local `PortalAgent` to be semantically routed to the appropriate `RouterAgent`. This message type defines fields for:

- $\langle Deathclock \rangle$ – recording the number of re-transmissions between `RouterAgents` that the message has traversed;
- $\langle Concept \rangle$ and $\langle Ontology \rangle$ – identifying the subject concept of the notification;

- $\langle WrapperID \rangle$ – holding an identification of the wrapper that is the source of the notification;
- $\langle DataURL \rangle$ – holding the location of the appropriate RDF file;
- $\langle AcquisitionDate \rangle$ – containing the date (and time) that the relevant resources were acquired by the Annotation System.

ACLOntologyModificationMessage

Uses the FIPA `Inform-ref` message performative. These messages are sent by the `NotificationAgent` following receipt of an ontology modification notification from the Notification Server. Each of these messages represents one of the ontology mappings contained in the original notification message, and are sent to the local `PortalAgent` to be semantically routed to the appropriate `RouterAgent`. This message type defines fields for:

- $\langle Concept \rangle$ and $\langle Ontology \rangle$ – identifying the subject concept of the modification;
- $\langle Mappings \rangle$ – containing a number of OWL mappings between the old and new definitions of a concept defined in the ontology.

ACLSimpleQueryMessage

Uses the FIPA `Query-ref` message performative. These messages are generated by the `QueryManagementAgent` following decomposition of a received `ACLComplexQueryMessage`. They are sent to the local `PortalAgent` to be semantically routed to the appropriate `RouterAgent`. This message type defines fields for:

- $\langle Deathclock \rangle$ – recording the number of re-transmissions between `RouterAgents` that the message has traversed;
- $\langle Concept \rangle$ and $\langle Ontology \rangle$ – identifying the subject concept of the query;
- $\langle ReplyGUID \rangle$ – holding the GUID of the sending `QueryManagementAgent`;
- $\langle Query \rangle$ – holding the RDQL atomic query.

ACLSimpleQueryReplyMessage

Uses the FIPA `Inform-ref` message performative. These messages are generated by `RouterAgents` in response to received `ACLSimpleQueryMessages` that refer to their concept. The results are generated by the `RouterAgent` by executing the atomic RDQL query over its stored metadata index, and then cross-referencing results with its stored resource index. This message type defines fields for:

- ⟨*Concept*⟩ and ⟨*Ontology*⟩ – identifying the subject concept of the reply;
- ⟨*ReplyTo*⟩ – holding the message ID of the query;
- ⟨*Results*⟩ – holds the result set for the query concept. This result set is formed by a list of resources, each coupled with a list of instance URIs that caused the resource to be retrieved;
- ⟨*Neighbours*⟩ – holding a list of the semantic neighbour concepts of the replying `RouterAgent`.

A.3 Test Messages

In addition to the internal system messages used in normal operation, there are a number of additional internal messages that are only used for performing tests upon the system as a part of the development process. These messages are:

ACLClientReadyMessage

These messages are used to indicate to the `TestAgent` that the *NotificationClient* class within the `NotificationAgent` has successfully registered with the Notification Server and is ready to receive notification messages.

ACLCreationTestMessage

These messages are sent by a newly created `RouterAgent` to the `TestAgent` to confirm that it has been created. The message contains the initial routing index and content index of the `RouterAgent` to enable the `TestAgent` to check that it has been created correctly.

ACLRoutingTestMessage

These messages are sent by a `RouterAgent` on receipt of a message referring to the concept for which it is responsible. This message is sent to the `TestAgent` to confirm receipt of the semantically routed message.

ACLUpdateTestMessage

These messages are sent by a `RouterAgent` following an update of its content index, caused by receipt of a `ACLNotificationMessage`. This message is sent to the `TestAgent` to confirm update of the content index, and the message includes a serialisation of this index so that the update can be checked.

Appendix B

NWAA Ontology and Knowledge-Base

In this appendix we present the details of the ontology and knowledge-base constructed for the Quest-Semantics framework deployment with the North West Aerospace Association (NWAA) – as referred to from Chapter 7. The ontology and knowledge-base were constructed using information from the current database schema, existing capability search categories and input from NWAA as the domain experts.

The ontology has been produced to represent the 'top level' abstraction for the restricted domain of describing NWAA's member companies for the purposes of search and identification. Therefore, the ontology has MemberCompany as its central concept, and a number of subsidiary concepts used to describe the properties of a MemberCompany. The following description of the ontology divides into 4 sections, each describing different groups of concepts: companies, contact details, business categories, and approvals. In each section there is both a textual and an abstract diagrammatic description of the concepts involved.

The ontology description is accompanied by the description of the initial knowledge-base of company details, consisting of instances of the concepts and properties defined in the ontology. The majority of the knowledge-base consists of MemberCompany instances – detailing the information about individual companies in the domain. However, an important pre-requisite is the definition of instances of concepts used to describe the features of these companies. In each of the following sections, for each ontology concept described we detail the concept instances required to populate the knowledge-base with sufficient detail to fully describe the companies within the domain. These are then used in an automatic annotation process upon the current member company database content.

In the descriptions provided below the representation of concepts and their properties (including the range and cardinality of the property) is as follows:

Concept name

– property name – range of property – (cardinality)

B.1 Companies

Primary items of information to represent are member companies. They should sub-class a more general Company concept so that non-member companies (as suppliers, customers, etc.) can also be represented. The majority of the information in the domain is represented in terms of MemberCompany instances and their properties. Population of the knowledge-base with MemberCompany instances will be performed through automatic mapping of the information in the current database into instances of MemberCompany, using the other concept instances described below as values of many of the instanciated properties.

Company

– hasName – `string` – (1)

NonMemberCompany

MemberCompany

– hasDescription – `string` – (1)

– hasWebsite – `string` – (1)

– hasTurnover – `int` – (1)

– hasAerospaceTurnover – `int` – (1)

– hasNumEmployees – `int` – (1)

– hasAddress – Address – (1)

– hasContact – Person – (n)

– operatesInSector – Sector – (n)

– operatesInMarket – Market – (n)

– hasLifecyclePosition – LifecyclePosition – (n)

– hasCompanyTier – CompanyTier – (n)

– hasActivity – Activity – (n)

– hasProductCategory – ProductCategory – (n)

– hasApproval – Approval – (n)

– hasCustomer – Company – (n)

– hasSupplier – Company – (n)

– hasParentCompany – Company – (1)

– hasSubsidiary – Company – (n)

N.B.: the hasParentCompany and hasSubsidiary relationships are defined as a sub-properties of the partOf / hasPart relationships described in Section B.5.

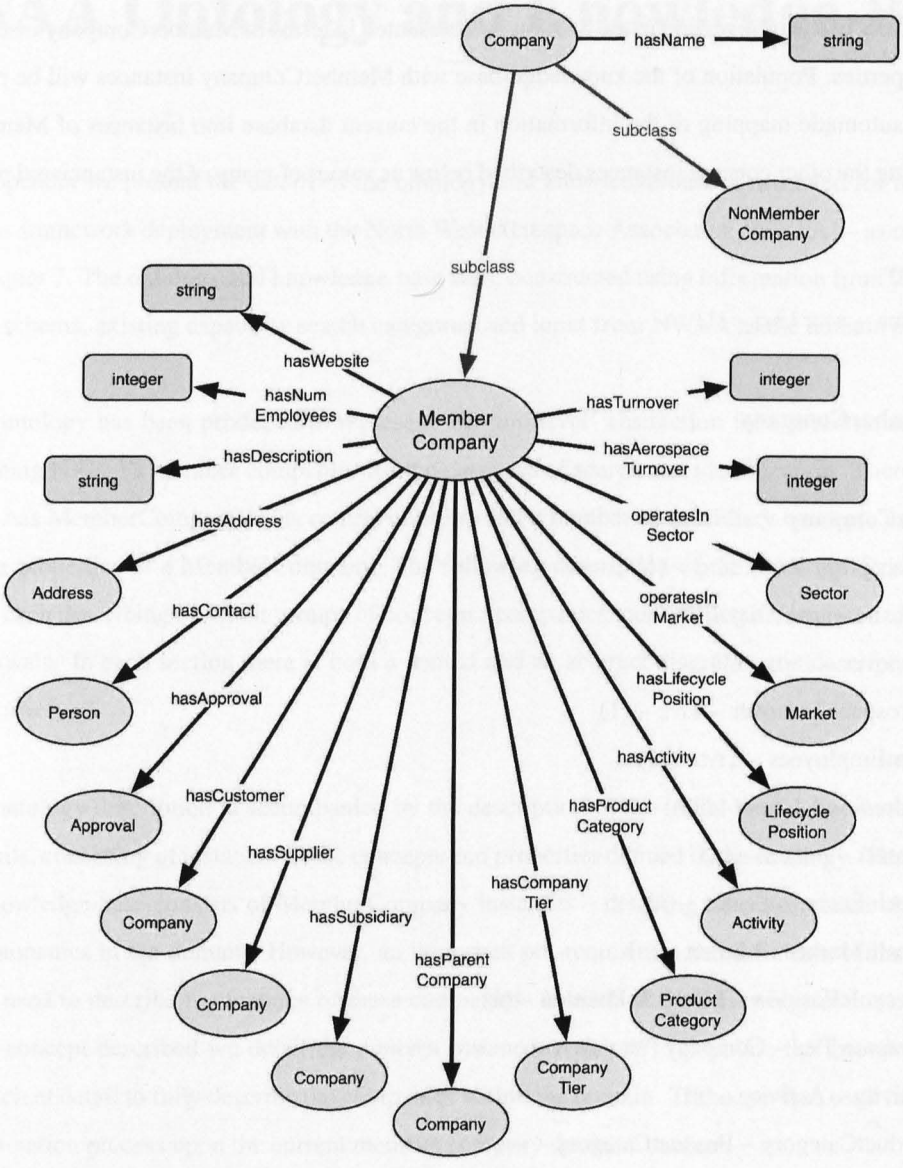


Figure B.1: Company concepts.

B.2 Contact Details

In the company information we need to represent people and addresses. People are described for the purpose of providing contact details for various personnel within companies. Addresses are described for the purpose of providing a company address, and address details for the represented personnel. It is intended that this information will be extracted from the current database as part of the automatic annotation process.

Person

– hasName – `string` – (1)

– hasPosition – `string` – (1)

– hasPhoneNumber – `string` – (1)

– hasMobileNumber – `string` – (1)

– hasFaxNumber – `string` – (1)

– hasAddress – Address – (1)

Address

– number – `string` – (1)

– road – `string` – (n)

– town – `string` – (1)

– region – `string` – (1)

– country – `string` – (1)

– postcode – `string` – (1)

B.3 Business Categories

A company's business activities can be represented using concepts for sector, market, and company categorisations based on activity, tier, lifecycle position and product category. Sector to represent sectors other than aerospace that the company operates within. Market is used to represent general geographic areas in which the company is active. CompanyTier and LifecyclePosition describe the general position of a company within the aerospace domain and supply chain. Activity instances describe the specific activities a company engages in. ProductCategory describes the category of the main product(s) of the company. These CompanyTier, LifecyclePosition, Activity and ProductCategory instances can be hierarchically structured – as shown in the concept definitions and, where appropriate, in the instance

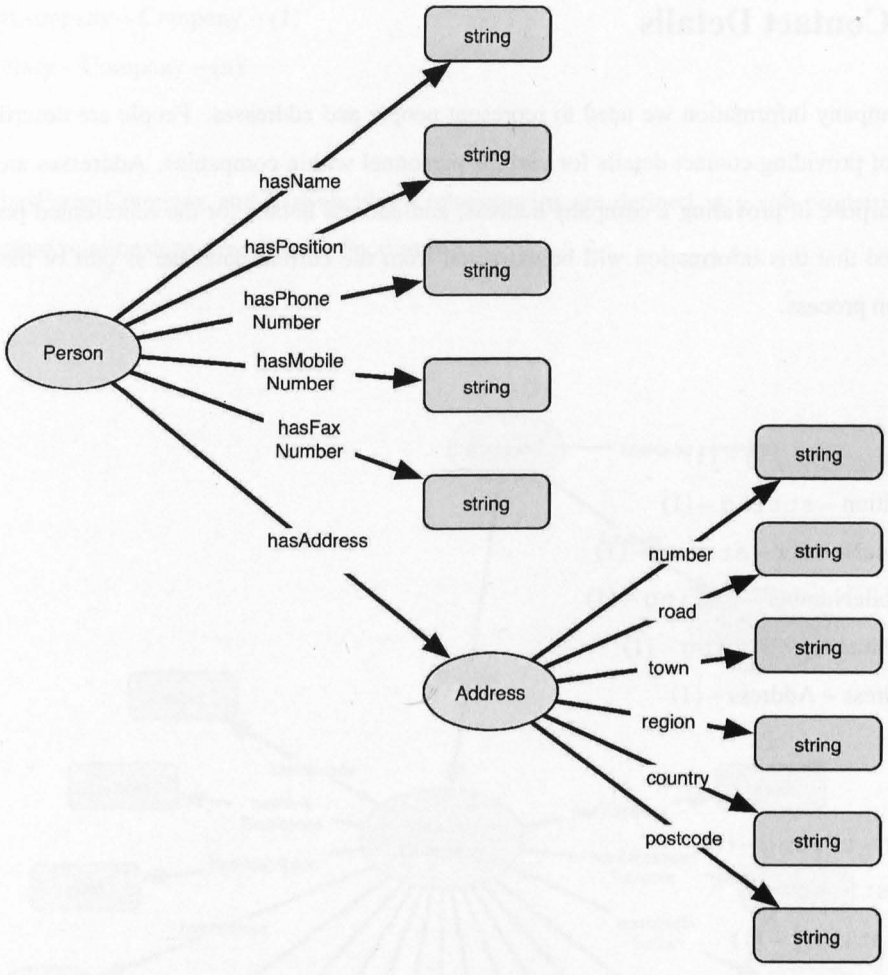


Figure B.2: Contact detail concepts.

descriptions. These hierarchically structured categorisation relationships are defined in terms of a general partOf / hasPart pair of relations, described in Section B.5.

Also, the Support instance (and all those instances that form partOf Support) defined under both CompanyTier and LifecyclePosition are instances of both these concepts – i.e., the same entities are intended in both cases, and so are defined only once, but belong to both categorisation hierarchies.

Sector

– hasName – `string` – (1)

– hasDescription – `string` – (1)

The required instances of the Sector concept for the initial knowledge-base are as follows: Aerospace, Automotive, Energy, Marine, Medical, Oil, Textiles, Services, Defence, and OtherSector.

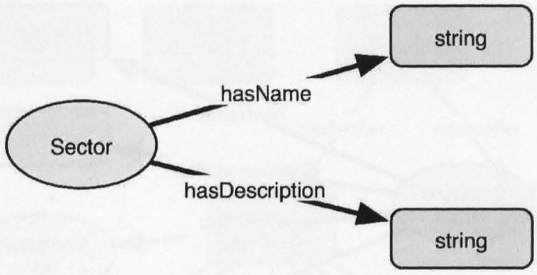


Figure B.3: Sector concept.

Market

– hasName – `string` – (1)

– hasDescription – `string` – (1)

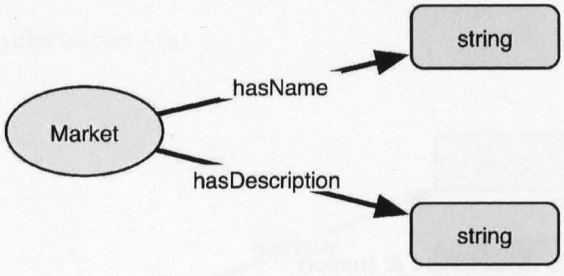


Figure B.4: Market concept.

The required instances of the Market concept for the initial knowledge-base are as follows: Europe, North America, South America, Asia, Worldwide, Australia, UK, Africa, and OtherMarket.

CompanyTier

– hasName – `string` – (1)

– hasDescription – `string` – (1)

– subTierOf – CompanyTier – (1)

– hasSubTier – CompanyTier – (n)

The required instances of the CompanyTier concept for the initial knowledge-base are shown in the following list. The indentation within the list indicates the containment of tiers by other tiers, which is represented in the instantiations by use of the `hasSubTier` property, itself a sub-property of the `hasPart` property described in Section B.5.

- OEM

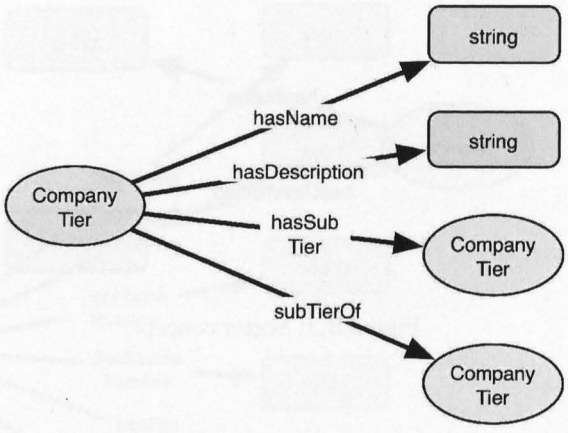


Figure B.5: CompanyTier concept.

- Tier 1
- Tier 2 – 4
- Support
  - Tooling
  - OtherSupport
    * CAD / CAM software & support
    * T / IS services
    * Consultancy
    * Universities
    * Local Government
    * Trade unions
    * Airport services

N.B.: the Support instance, and all of those instances that form part of Support, are instantiations of both CompanyTier and LifecyclePosition. These instances appear in both definitions for the purpose of clarity, not due to their replication.

LifecyclePosition

– hasName – `string` – (1)

– hasDescription – `string` – (1)

– subPositionOf – LifecyclePosition – (1)

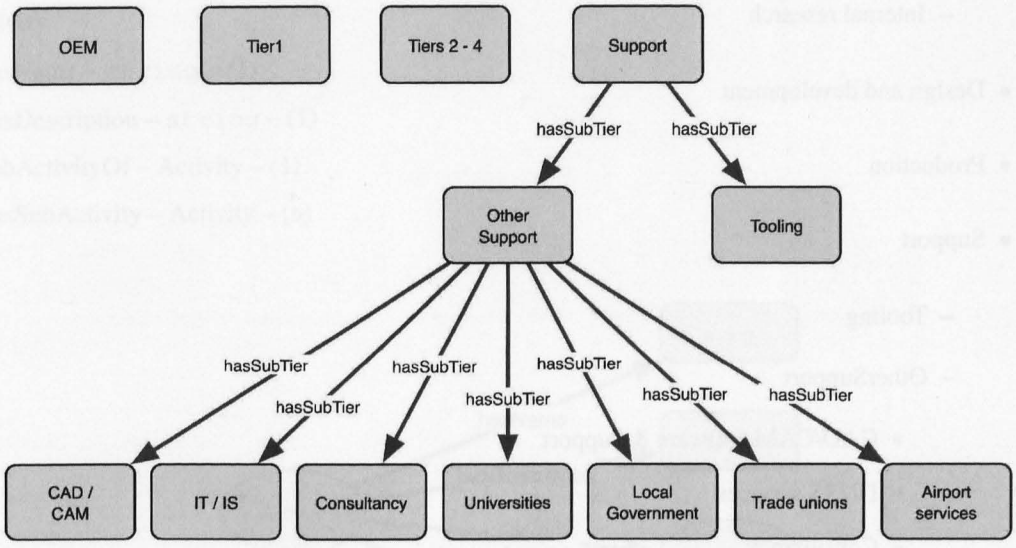


Figure B.6: CompanyTier instances.

– hasSubPosition – LifecyclePosition – (n)

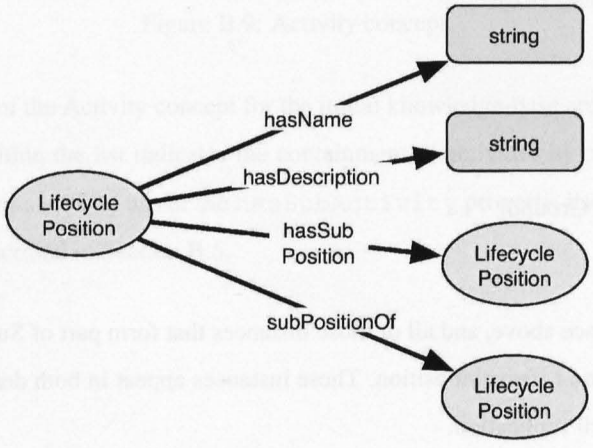


Figure B.7: LifecyclePosition concept.

The required instances of the LifecyclePosition concept for the initial knowledge-base are shown in the following list. The indentation within the list indicates the containment of positions by other positions, which is represented in the instantiations by use of the `hasSubPosition` property, itself a sub-property of the `hasPart` property described in Section B.5.

- Research
  - Academic institutions
  - Commercial research organisations

  - Internal research
- Design and development
- Production
- Support
  - Tooling
  - OtherSupport
    * CAD/CAM software & support
    * IT / IS services
    * Consultancy
    * Universities
    * Local Government
    * Trade unions
    * Airport services
- Aftermarket
  - M.R.O.
  - Aircraft On Ground

N.B.: the Support instance above, and all of those instances that form part of Support, are instantiations of both CompanyTier and LifecyclePosition. These instances appear in both definitions for the purpose of clarity, not due to their replication.

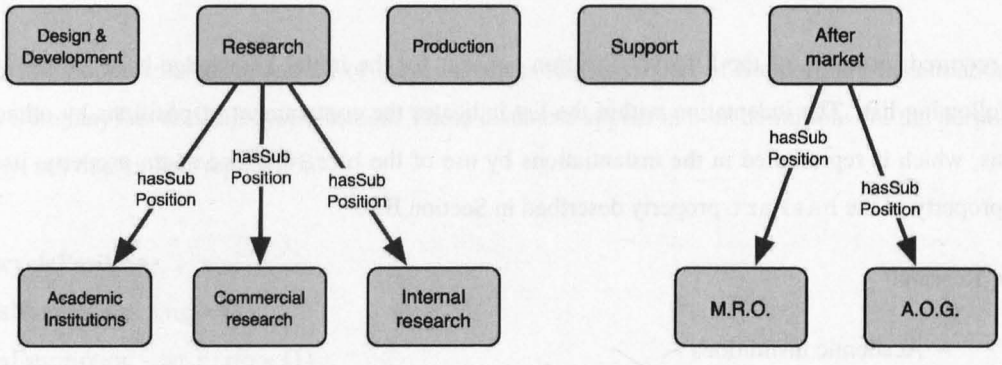


Figure B.8: LifecyclePosition instances.

Activity

– hasName – `string` – (1)

– hasDescription – `string` – (1)

– subActivityOf – Activity – (1)

– hasSubActivity – Activity – (n)

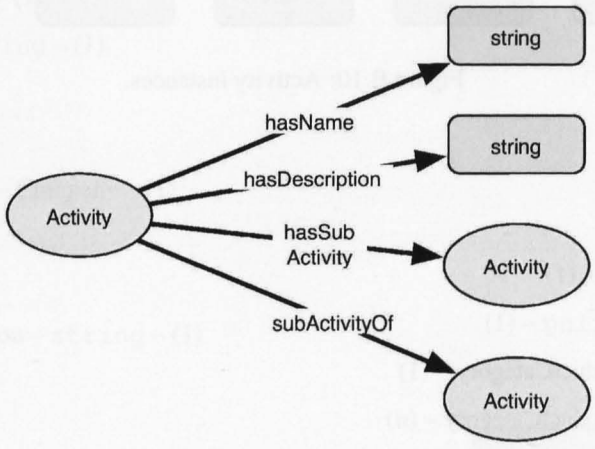


Figure B.9: Activity concept.

The required instances of the Activity concept for the initial knowledge-base are shown in the following list. The indentation within the list indicates the containment of activities by other activities, which is represented in the instantiations by use of the `hasSubActivity` property, itself a sub-property of the `hasPart` property described in Section B.5.

- Own product
- Major assemblies
- Build-to-print
  - Fabrication / machining
  - Material
  - Treatments / NDT
  - Testing
  - Integration
- Major sub-assemblies
- Material

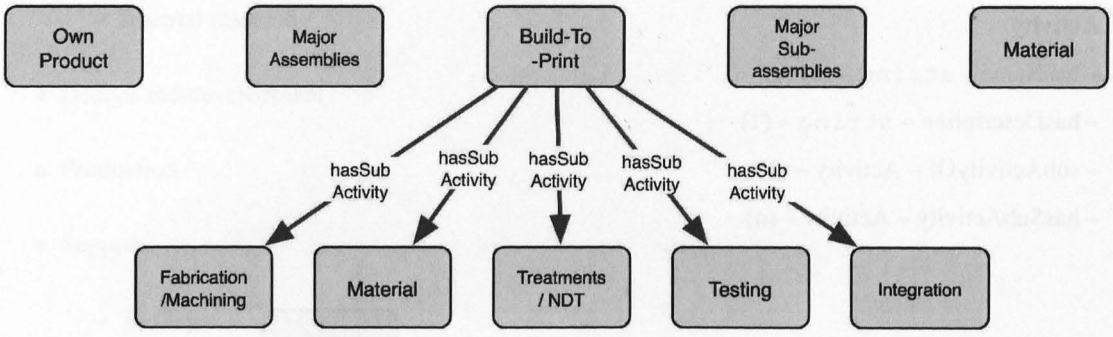


Figure B.10: Activity instances.

ProductCategory

– hasName – `string` – (1)

– hasDescription – `string` – (1)

– subCategoryOf – ProductCategory – (1)

– hasSubCategory – ProductCategory – (n)

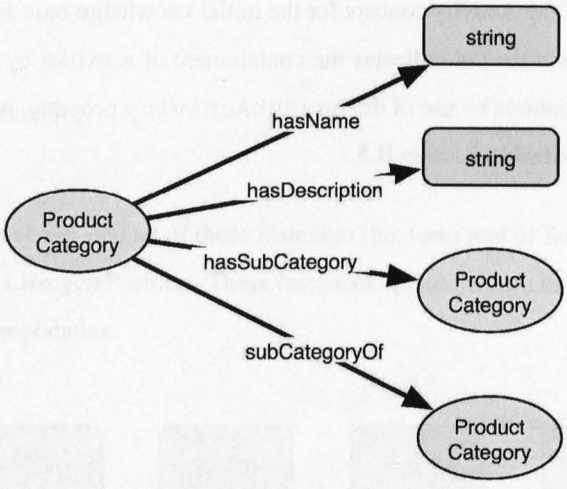


Figure B.11: ProductCategory concept.

The required instances of the ProductCategory concept for the initial knowledge-base are as follows: Aero-engines, Aero-structures, Major aircraft systems, Aircraft interiors, Aircraft maintenance, Aircraft construction, Ground support, and Electrical / electronic.

B.4 Approvals

Approval is used to represent the different national and company approvals that may have been awarded to a company. The two sub-classes CompanyApproval and NationalApproval represent the two types.

Approval

– hasName – `string` – (1)

– hasDescription – `string` – (1)

CompanyApproval

– awardedByCompany – Company – (1)

NationalApproval

– awardedByOrganisation – `string` – (1)

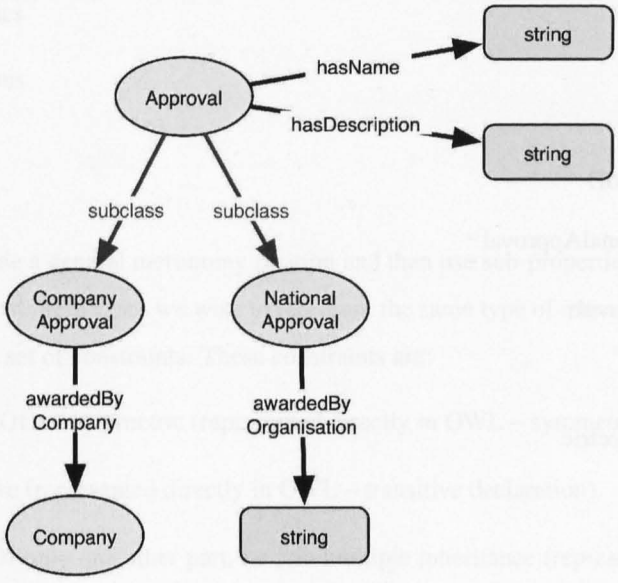


Figure B.12: Approval concepts.

The required instances of the CompanyApproval and NationalApproval concepts for the initial knowledge-base are as follows.

- **NationalApproval:**
  - AS/EN/JISQ 9100
  - AS/EN/JISQ 9110

- AS/EN/JISQ 9120
- ISO 9000:2000
- ISO 9001:2000
- ISO 9001:TickIT
- ISO 14001:2004
- ISO 17025
- EASA Part 145 – Approved maintenance organisations
- EASA Part 147 – Training organisation
- EASA Part 21 – Design organisation approval
- EASA Part 21 – Production organisation approval
- OHSAS 18001
- CAA
- FAA
- UKAS
- NADCAP
- NATO / MoD
- OtherNationalApproval

- **CompanyApproval:**
  - Raytheon
  - General Electric
  - Avio
  - Thales
  - Cessna
  - Bombardier
  - Goodrich
  - Snecma
  - Fokker
  - Airbus
  - Rolls Royce

- BAE Systems
- Smiths
- Boeing
- GKN Westland
- OtherCompanyApproval

B.5 Part-of Relation

There are five situations within this ontology where we wish to present a meronomy hierarchy, i.e., represent entities as being part of other entities. The cases we wish to represent are:

- Parents and subsidiaries of companies
- Activities
- ProductCategories
- LifecyclePositions
- CompanyTiers

Therefore, we define a general meronomy relation and then use sub-properties of this in each of the specific cases. In each of these cases we wish to represent the same type of meronomy relation between entities, with the same set of constraints. These constraints are:

- hasPart and partOf are symmetric (represented directly in OWL – symmetric declaration)
- both are transitive (represented directly in OWL – transitive declaration)
- any part is partOf only one other part, i.e., no multiple inheritance (represented directly in OWL – functional declaration)
- parts can have multiple sub-parts (represented directly in OWL – inverse functional declaration)
- parts cannot be partOf themselves or their sub-parts (represented as a rule – using SWRL)
- disjointness of sub-parts from their siblings only (represented as a rule – using SWRL)

This encoding of the required part-whole relations has been constructed with appropriate reference to the W3C guidelines on meronomy within OWL ontologies [126].

Bibliography

[1] J. D. Anderson and J. P. Carballo. The nature of indexing: how humans and machines analyze messages and texts for retrieval - Part I: Research, and the nature of human indexing. *Information Processing Management*, 37(2):255–277, 2001.

[2] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 2004.

[3] Aristotle. *The Categories, On Interpretation, Prior Analytics*. Harvard University Press, Cambridge, MA.

[4] J. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. WebODE: A Scalable Workbench for Ontological Engineering. In *Proceedings of the First International Conference on Knowledge Capture, K-CAP 2001*, pages 6–13. ACM-Sigmod, 2001.

[5] J. L. Austin. *How to Do Things With Words*. Oxford University Press: Oxford, England, 1962.

[6] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In D. Hutter and W. Stephan, editors, *Festschrift in honor of Joerg Siekmann*, Lecture Notes in Artificial Intelligence. Springer, 2003.

[7] F. Baader and W. Nutt. *Basic Description Logics*, pages 43–95. Cambridge University Press, 1993.

[8] H. Balakrishnan, M. F. Kaashoek, D. Karger, and I. Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, 46(2):43 - 48, February 2003.

[9] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. JADE a white paper. *EXP In search of innovation*, 3(3):12–16, September 2003.

[10] V. Benjamins, J. Contreras, M. Blázquez, M. Niñ, A. García, E. Navas, J. Rodriguez, F. Hernandez, C. Wert, and J. Dodero. ONTO-H: A collaborative semi-automatic annotation tool. In *Proceedings of the 8th international Protégé conference*, Madrid, Spain, 2005.

[11] V. Benjamins, J. Contreras, A. Gómez-Pérez, H. Uszkoreit, T. Declerck, D. Fensel, Y. Ding, M. Wooldridge, and V. Tamma. Esperonto application: Service provision of semantic annotation, aggregation, indexing, and routing of textual, multimedia and multilingual web content. In *Proceedings of WIAMSI'03*, 2003.

[12] V. Benjamins, E. Plaza, E. Motta, D. Fensel, R. Studer, B. Wielinga, G. Schreiber, Z. Zdrahal, and S. Decker. Ibrow3: An intelligent brokering service for knowledge-component reuse on the World Wide Web. In *Proceedings of the 11th Knowledge Aquisition for Knowledge-Based Systems Workshop (KAW-98)*, Banff, Canada, 1998.

[13] A. Bernaras, I. Laresgoiti, and J. M. Corera. Building and reusing ontologies for electrical network applications. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 298–302, Budapest, Hungary, 1996.

[14] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proceedings of the Workshop on the Web and Databases (WebDB'02).*, 2002.

[15] M. Boddy. Anytime problem solving using dynamic programming. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, California, USA, 1991.

[16] A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[17] P. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse.* PhD thesis, Centre for Telematica and Information Technology, University of Twente, 1997.

[18] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Y. (Editors). Extensible Markup Language (XML) 1.0 (Fourth Edition), 2006. http://www.w3.org/TR/xml.

[19] P. Bretier and D. Sadek. A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents III Agent Theories, Architectures, and Languages*, volume 1193/1997, pages 189–204. Springer-Verlag: Berlin, Germany, 1997.

[20] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the First International Semantic Web Conference*, Sardinia, Italy, 2002.

[21] T. Bylander and B. Chandrasekaran. Generic Tasks in Knowledge-based Reasoning: The Right Level of Abstraction for Knowledge Acquisition. In B. Gaines and J. Boose, editors, *Knowledge Acquisition for Knowledge Bases*, volume 1, pages 65–77. Academic Press, London, 1988.

[22] S. Castano, A. Ferrara, S. Montanelli, E. Pagani, and G. Rossi. Ontology-Addressable Contents in P2P Networks. In *Proceeding of the First WWW International Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID 2003)*, 2003.

[23] M. Castro, P. Druschel, A.-M. Kermarree, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal of Select. Areas Communication*, 20(8), October 2002.

[24] H. Chalupsky. OntoMorph: A Translation System for Symbolic Knowledge. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning - Proceedings of the Seventh International Conference (KR'2000)*, pages 471–482, San Francisco, CA, 2000. Morgan Kaufmann.

[25] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In *Proceedings of the Fifteenth American Conference on Artificial Intelligence (AAAI-98)*, pages 600–607, Madison, Wisconsin, 1998. AAAI Press/The MIT Press.

[26] I. Clarke, O. Sandberg, and B. Wiley. Freenet: A distributed anonymous information storage and retrieval system. In *Lecture Notes in Computer Science*, volume 2009, pages 46+. Springer-Verlag, 2001.

[27] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. *Intention in Communication*, pages 221–256, 1990.

[28] P. R. Cohen and C. R. Perrault. Elements of a Plan Based Theory of Speech Acts. *Cognitive Science*, 3:177–212, 1979.

[29] A. M. Collins and E. F. Loftus. A Spreading-Activation Theory of Semantic Processing. *Psychological Review*, 82:407–425, 1975.

[30] O. Corcho and A. Gómez-Pérez. A roadmap to ontology specification languages. In R. Dieng, editor, *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management. LNCS*, volume 1937, pages 80–96, Berlin, 2000. Springer Verlag.

[31] O. Corcho, A. Gómez-Pérez, A. López-Cima, and M. Suárez-Figueroa. ODESeW - Automatic Generation of Knowledge Portals for Intranets and Extranets. In *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, LNCS. Springer, 2003.

[32] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

[33] A. Crespo and H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. In *Proceedings of the International Conference on Distributed Computing Systems*, 2002.

[34] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Computer Science Department, Stanford University, October 2002.

[35] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 578–583, Nagoya, Japan, 1997.

[36] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74, 2000.

[37] S. J. DeRose. Expanding the notion of links. In *Proceedings of ACM Hypertext '89*, pages 249–257, Pittsburgh, USA, November 1989.

[38] E. Diday. Knowledge discovery from symbolic data and the SODAS software. http://citeseer.comp.nus.edu.sg/462410.html, 2000.

[39] E. H. Durfee, D. L. Kiskis, and W. P. Birmingham. The Agent Architecture of the University of Michigan Digital Library. *IEEE Proceedings on Software Engineering*, 144(1):61–71, Feb. 1997.

[40] M. Ehrig and S. Staab. QOM Quick Ontology Mapping. In S. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference*, number 3298 in LNCS, pages 683–697, Hiroshima, Japan, 2004.

[41] M. Ehrig and Y. Sure. Ontology Mapping - An Integrated Approach. In C. Bussler, J. Davis, D. Fensel, and R. Studer, editors, *Proceedings of the First European Semantic Web Symposium (ESWS-04)*, pages 76–91, Heraklion, Greece, 2004.

[42] M. Ehrig and Y. Sure. FOAM - Framework for Ontology Alignment and Mapping: Results of the Ontology Evaluation Initiative. In J. E. Benjamin Ashpole, Marc Ehrig and H. Stuckenschmidt, editors, *Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies*, Banff, Canada, October 2005.

[43] D. Engmann and S. Massmann. Instance Matching with COMA++. In *Proceedings of the Workshop of Model Management und Metadaten-Verwaltung*, 2007.

[44] F. Esposito, D. Malerba, G. Semeraro, C. Antifora, and G. de Gennaro. Information Capture and Semantic Indexing of Digital Libraries through Machine Learning Techniques. In *Proceedings*

of Fourth International Conference on Document Analysis and Recognition ICDAR '97, pages 722–727, Ulm, Germany, 1997.

[45] F. Esposito, D. Malerba, and V. Tamma. Dissimilarity Measures for Symbolic Objects. In H.-H. Bock and E. Diday, editors, *Analysis of Symbolic data. Exploratory methods for extracting statistical information from complex data*, volume 15 of *Studies in Classification, Data Analysis, and Knowledge Organisation*, pages 165–185. Springer-Verlag, Berlin, Germany, 2000.

[46] F. Esposito, D. Malerba, V. Tamma, and H.-H. Bock. Classical Similarity and Dissimilarity Measures. In H.-H. Bock and E. Diday, editors, *Analysis of Symbolic data. Exploratory methods for extracting statistical information from complex data*, volume 15 of *Studies in Classification, Data Analysis, and Knowledge Organisation*, pages 139–152. Springer-Verlag, Berlin, 2000.

[47] O. Etzioni, N. Lesh, and R. Segal. Building softbots for UNIX. In O. Etzioni, editor, *Software Agents - Papers from the 1994 Spring Symposium (Technical Report SS–94–03)*, pages 9–16. AAAI Press, March 1994.

[48] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg, Germany, 2007.

[49] J. Euzenat, H. Stuckenschmidt, and M. Yatskevich. Introduction to the Ontology Alignment Evaluation Initiative 2005. http://oaei.ontologymatching.org/2005/results/oaei2005.pdf.

[50] J. Euzenat and P. Valtchev. An integrative proximity measure for ontology alignment. In A. Doan, A. Halevy, and N. Fridman Noy, editors, *Proceedings of the Semantic Integration Workshop at ISWC-03*, 2003.

[51] J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 333–337, 2004.

[52] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, 2001.

[53] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to its full potential*. The MIT Press: Cambridge, MA, 2003.

[54] I. Ferguson and M. Wooldridge. Paying their way: Commercial Digital Libraries for the Twenty-First Century. *dLib Magazine: The Journal of Digital Library Research*, pages 385 – 391, June 1997.

[55] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. *Spring Symposium on Ontological Engineering of AAAI. Stanford University, California*, pages 33 - 40, 1997.

[56] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In *Software Agents*, pages 456–463. ACM, 1997.

[57] A. Formica and M. Missikoff. Concept Similarity in SymOntos: an Enterprise Ontology Management Tool. *The Computer Journal*, 45(6):583–594, 2002.

[58] N. Fridman-Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics (SMI), Department of Medicine, Stanford University School of Medicine, 2001.

[59] N. Fridman Noy and M. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In A. Gómez-Pérez, M. Gruninger, H. Stuchenschmidt, and M. Uschold, editors, *Proceedings of the IJCAI'01 Workshop on Ontologies and Information Sharing*, pages 63–70, 2001.

[60] J. R. Galliers. *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Mulri-Agent Conflict.* PhD thesis, Open University, UK, 1988.

[61] L. Gasser and M. Huhns, editors. *Distributed Artificial Intelligence (Volume II).* Pitman / Morgan Kaufmann, 1989.

[62] M. R. Genesereth and S. P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[63] The genome@home project website. http://genomeathome.stanford.edu.

[64] R. Graham. P2P Computing: Towards a Definition. http://www.itm.mh.se/ rosgra/p2pdefinition.html, 2001.

[65] Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Recommendation., September 2007.

[66] The Groove web-site. http://www.groove.net, 2003.

[67] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Aquisition*, 5(2):199–220, 1993.

[68] N. Guarino. Formal ontologies and information systems. In N. Guarino, editor, *Proceedings of the International Conference on Formal Ontology in Information Systems*, 1998.

[69] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the Third International Semantic Web Conference*, Hiroshima, Japan, November 2004.

[70] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proceedings of the International Conference on Data Engineering (ICDE'03)*, Bangalore, India, 2003.

[71] T. Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, Stanford University – Digital Library Technolgies Project, 1999.

[72] R. V. J. Heflin and J. Dale. Requirements for a web ontology language. Technical report, World Wide Web Consortium (W3C), 2002.

[73] I. Horrocks, P. Patel -Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1, 2003.

[74] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL Rules: A Proposal and Prototype Implementation. *Journal of Web Semantics*, 3(1):23–40, 2005.

[75] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. http://www.w3.org/Submission/SWRL/.

[76] R. Huebsch, J. Hellerstein, N. Lanham, and B. Thau Loo. Querying the internet with PIER. In *Proceedings of the 29th VLDB Conference. Berlin, Germany.*, 2003.

[77] M. Huhns, editor. *Distributed Artificial Intelligence*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.

[78] M. Huhns. Agent Foundations for Cooperative Information Systems. In H. S. Nwana and D. T. Ndumu, editors, *Proceedings of the Third International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology*, London, 1998.

[79] R. Ichise, H. Takeda, and S. Honiden. Rule Induction for Concept Hierarchy Alignment. In *Proceedings of the Workshop on Ontology Learning at the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[80] The Jabber web-site. http://www.jabber.org.

[81] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of 2003 IEEE WET ICE Workshop on Enterprise Security.*, 2003.

[82] N. Jian, W. Hu, G. Cheng, and Y. Qu. FalconAO: Aligning Ontologies with Falcon. In B. Ashpole, M. Ehrig, J. Euzenat, and H. Stuckenschmidt, editors, *Proceedings of K-CAP Workshop on Integrating Ontologies*, 2005.

[83] R. Kahn and R. Wilensky. A framework for distributed digital object services. Technical Report cnri.dlib/tn95-01, Corporation for National Reasearch Initiatives, 1995.

[84] Y. Kalfoglou and B. Hu. CROSI Mapping System (CMS): Results of the 2005 Ontology Alignment Contest. In B. Ashpole, M. Ehrig, J. Euzenat, and H. Stuckenschmidt, editors, *Proceedings of the K-CAP Workshop on Integrating Ontologies, Banff, Canada*, October 2005.

[85] The Kazaa web-site. http://www.kazaa.com.

[86] J. Kephart and D. Chess. The vision of autonomic computing. *Computer magazine*, 36(1):41–51, 2003.

[87] A. Keromytis and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of the ACM SIGCOMM'02 Conference.*, 2002.

[88] M. Kifer, G. Lausen, and J. We. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[89] H. Kitakami, Y. Mori, and M. Arikawa. An intelligent system for integrating autonomous nomenclature databases in semantic heterogeneity. In R. R. Wagner and H. Thoma, editors, *Proceedings of Database and Expert System Applications (DEXA-96)*, pages 187–196. Springer, 1996.

[90] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In A. Gómez-Pérez, M. Gruninger, H. Stuckenschmidt, and M. Uschold, editors, *Proceedings of the IJCAI'01 Workshop on Ontologies and Information Sharing*, pages 53–62, 2001.

[91] M. Klusch, editor. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet.* Springer-Verlag: Berlin, Germany, 1999.

[92] M. Klusch. Information agent technology for the internet: A survey. In D. Fensel, editor, *Journal on Data and Knowledge Engineering. Special Issue on Intelligent Information Integration*, volume 36(3). Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 2001.

[93] M. Klusch and K. Sycara. Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents*, pages 197–224. Springer-Verlag, Berlin, 2001.

[94] G. Klyne and J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, February 2004.

[95] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, S. Gummadi, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS.*, 2000.

[96] R. Larson and C. Carson. Information Access for a Digital Library: Cheshire II and the Berkeley Environmental Digital Library. In *ASIS*, Washington, USA, October 1999.

[97] S. Lawrence and C. Giles. Searching the World Wide Web. *Science*, 280(5360):98-100, 1999.

[98] D. Lenat. Cyc: a Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

[99] M. Lesk. The digital library: What is it? why should it be here? Technical Report TR-93-35, Department of Computer Science, Virginia Tech., VA, 1993.

[100] I. V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybernetics and Control Theory*, 1966.

[101] H. Lieberman. Personal assistants for the web: An MIT perspective. In *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, pages 279–292. Springer, 1999.

[102] C. Lynch, A. Michelson, C. Preston, and C. A. Summerhill. CNI White paper on Networked Information Discovery and Retrieval. http://www.cni.org/projects/nidr/, 1995.

[103] A. Maedche and S. Staab. Measuring similarity between ontologies. In A. Gómez-Pérez and R. Benjamins, editors, *Proceedings of EKAW'02, Siguenza, Spain*, pages 251–263, Berlin, 2002. Springer-Verlag.

[104] B. McBride. JENA: A Semantic Web toolkit. *IEEE Internet Computing*, 6:55–59, Nov-Dec 2002.

[105] D. L. McGuinness. Ontologies come of age. In *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002.

[106] D. L. McGuinness and F. van Harmelen (Editors). OWL Web Ontology Language Overview. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-features/.

[107] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.

[108] G. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[109] L. Miller, A. Seabourne, and A. Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. Tech. Report HPL-2002-110, HP Labs, 2002.

[110] R. Mizoguchi. Ontological engineering: Foundation of the next generation knowledge processing. *Lecture Notes in Computer Science*, 2198, 2001.

[111] J. Nanard and M. Nanard. Using structured types to incorporate knowledge in hypertext. In *Proceedings of the Third Annual ACM Conference on Hypertext*, pages 329–343, 1991.

[112] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.

[113] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the World Wide Web Conference (WWW)*, pages 604–615, Honolulu, Hawaii, USA, 2002.

[114] I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems*, Ogunquit, Maine, USA, October 2001.

[115] M. Nodine, D. Chandrasekara, and A. Unruh. Task Coordination Paradigms for Information Agents. In *Proceedings of Intelligent Agents VII. Agent Theories, Architectures and Languages, 7th International Workshop (ATAL-00)*, Boston, MA, 2000.

[116] OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November 2004 – http://www.w3.org/Submission/owl-s/.

[117] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University – Digital Library Technolgies Project, 1998.

[118] P. Patel -Schneider and I. Horrocks. OWL 1.1 Web Ontology Language Overview. W3C Member Submission 19 December 2006 - http://www.w3.org/Submission/owl11-overview/.

[119] P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language. Semantics and Abstract Syntax. W3C Recommendation, February 2004.

[120] A. Preece, K. Hui, A. Gray, P. Marti, T. Bench-Capon, Z. Cui, and D. Jones. KRAFT: an agent architecture for knowledge fusion. *International Journal of Cooperative Information Systems*, 10(1/2):171–196, March & June 2001.

[121] E. Prud'hommeaux and A. Seaborne. SparQL Query Language for RDF. W3C Recommendation - 15th January 2008. http://www.w3.org/TR/rdf-sparql-query/.

[122] M. R. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, 1968.

[123] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.

[124] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The International Journal on Very Large Databases (VLDB)*, 10(4):334–350, 2001.

[125] A. S. Rao and M. P. Georgeff. Formal models and decision procedures for multi-agent systems. Technical Note 61, Australian AI Institute, Melbourne, Australia, 1995.

[126] A. Rector and C. Welty. Simple Part-Whole Relations in OWL Ontologies. -- http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/. Technical report, W3C, 2005.

[127] M. Rodríguez and M. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 2002.

[128] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 91–99, Los Angeles, CA, 1985.

[129] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany*, 2001.

[130] S. J. Russell, D. Subramanian, and R. Parr. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 338–344, Chambery, France, 1993.

[131] S. J. Russell and E. Wefald. *Do the Right Thing - Studies in Limited Rationality*. The MIT Press: Cambridge, MA, 1991.

[132] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, MA, 1989.

[133] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. In *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing (P2P2002)*, 2002.

[134] D. Schoder and K.Fischbach. Peer-to-peer prospects. *Communications of the ACM*, 46(2):27–29, February 2003.

[135] The seti@home project web-site. http://setiathome.ssl.berkley.edu.

[136] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *Journal of Data Semantics IV*, pages 146–171, 2005.

[137] M. P. Singh and M. N. Huhns. Social abstraction for information agents. In M. Klusch, editor, *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, pages 37–52. Springer-Verlag: Berlin, Germany, 1999.

[138] M. Sintek and S. Decker. TRIPLE - an RDF query, inference and transformation language. *Deductive Databases and Knowledge Management (DDLP)*, 2001.

[139] J. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[140] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of the ACM SIGCOMM'02 Conference.*, 2002.

[141] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[142] R. Studer, V. Benjamins, and D. Fensel. Knowledge engineering, principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[143] M. Sussna. Word sense disambiguation for free-text indexing using a massive semantic network. In *Proceedings of the Second Conference on Information and Knowledge Management*, Arlington, Virginia, 1993.

[144] K. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[145] K. Sycara, M. Klusch, S. Widoff, and J. Lu. LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multiagent Systems*, 5:173–203, 2002.

[146] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record. Special Issue on Semantic Interoperability in Open Information Environments*, 1999.

[147] V. Tamma, I. Blacoe, B. L. Smith, and M. Wooldridge. SERSE: Searching for Semantic Web Content. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, volume 3257, pages 419–432, 2004.

[148] V. Tamma and T. Payne. Toward semantic web agents: AgentLink and Knowledge Web. *AgentLink newsletter*, 17, 2005.

[149] V. Tamma and P. R. S. Visser. Integration of heterogeneous resources: Towards a framework for comparing techniques. In *Proceedings of the "6 Convegno dell'Associazione Italiana per l'Intelligenza Artificiale (AI*IA), Workshop su Stumenti di Organizzazione ed Accesso Intelligente per Informazione Eterogenee"*, pages 89–93, Padua, Italy, 1998.

[150] V. A. M. Tamma. *An Ontology Model Supporting Multiple Ontologies for Knowledge Sharing.* PhD thesis, The University of Liverpool, 2002.

[151] T.Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284:35–43, May 2001.

[152] A. Tversky. Features of similarity. *Psychological Review*, 84(4):327–372, 1977.

[153] Universal Description Discovery and Integration. OASIS Project - http://www.oasis-open.org/.

[154] V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art. *Journal of Web Semantics*, 4(1), 2006.

[155] M. Uschold. Knowledge level modelling: concepts and terminology. *Knowledge Engineering Review: Special Issue on Ontologies*, 13(1):5–29, 1998.

[156] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–136, 1996.

[157] M. Uschold and R. Jasper. A framework for understanding and classifying ontology applications. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods*, Stockholm, Sweden, 1999.

[158] R. van Eijk, J. Hamers, T. Klos, and M. S. Bargh. Agent technology for designing personalized mobile service brokerage. GigaMobile Deliverable D3.8, Telematica Instituut, Enschede, The Netherlands, 2002.

[159] G. van Heijst, A. Schreiber, and B. Wielinga. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, 46(2):183–292, 1997.

[160] C. J. van Rijsbergen. *Information Retrieval.* Butterworth, 1979.

[161] J. van Zyl and D. Corbett. Population of a framework for understanding and classifying ontology applications. In V. R. Benjamins, A. Gómez-Pérez, N. Guarino, and M. Uschold, editors, *Proceedings of the ECAI-00 Workshop on Applications of Ontologies and Problem-Solving Methods*, pages 101–112, 2000.

[162] R. Vanrenesse, K. Birman, A. Bozdog, D. Dimitriu, M. Singh, and W. Vogels. Heterogeneity-aware peer-to-peer multicast. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC2003).*, 2003.

[163] P. Visser, D. Jones, T. Bench-Capon, and M. Shave. Assessing heterogeneity by classifying ontology mismatches. In N. Guarino, editor, *Formal Ontology in Information Systems. Proceedings FOIS'98, Trento, Italy*, pages 148–182. IOS Press, 1998.

[164] V. Vlachos, S. Androutsellis-Theotokis, and D. Spinellis. Security applications of peer-to-peer networks. *Computing Networks Journal*, 45(2):195–205., 2004.

[165] P. Watry and R. Larson. Cheshire 3 framework white paper: Implementing support for digital repositories in a data grid environment. Technical report, University of Liverpool and University of California, Berkeley, 2005.

[166] J. E. White. Telescript technology: The foundation for the electronic marketplace. White paper, General Magic Inc., Mountain View, CA 94040, USA, 1994.

[167] G. Wiederhold and M. Genesereth. The conceptual basis for mediation services. *IEEE Experts*, 12(5):38–47, Sept-Oct 1997.

[168] M. E. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.

[169] W. Woods. What's in a Link: Foundations for Semantic Networks. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 35–82. Academic Press, New York, 1975.

[170] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[171] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering at ICSE 2000*, Limerick, Ireland, 2000.