A SOFTWARE SYSTEM FOR AGENT-ASSISTED ONTOLOGY BUILDING

by

Denish Mumbaiwala

B.Eng. Electronics
The Maharaja Sayajirao University of Baroda, 2007

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

March 2016

Abstract

This thesis investigates how one can design a team of intelligent software agents that helps its human partner develop a formal ontology from a relational database and enhance it with higher-level abstractions. The resulting efficiency of ontology development could facilitate the building of intelligent decision support systems that allow: high-level semantic queries on legacy relational databases; autonomous implementation within a host organization; and incremental deployment without affecting the underlying database or its conventional use. We introduce a set of design principles, formulate the prototype system requirements and architecture, elaborate agent roles and interactions, develop suitable design techniques, and test the approach through practical implementation of selected features. We endow each agent with model meta-ontology, which enables it to reason and communicate about ontology, and planning meta-ontology, which captures the role-specific know-how of the ontology building method. We also assess the maturity of development tools for a larger-scale implementation.

Contents

List of Figures

Acknowledgements

I would like to sincerely express my gratitude to my co-supervisors, Dr. Jernej Polajnar and Dr. Desanka Polajnar, for their invaluable support and suggestions throughout all stages of this thesis. Such a study may not have been possible without their experience, advice, encouragement, and patience.

I am thankful to Dr. Pranesh Kumar for his support and interest in serving on my supervisory committee.

The industry perspective of the research problem and some background work were identified during the internships of Denish Mumbaiwala with a partner company under the Mitacs-Accelerate Graduate Research Internship Program.

The research presented in this thesis was partially supported by the UNBC Research Seed Grant which was awarded to Dr. Jernej Polajnar with an objective to support the prototype implementation.

This Thesis is dedicated to my family

Bipin Mumbaiwala, Dipika Mumbaiwala,

and Mikita Mumbaiwala

who selflessly supported me in every stage of my life,

enabling such a study to happen today.

Chapter 1

Introduction

Computer-based information systems have been progressively accepted and adopted across all disciplines of human knowledge. In applications for various business domains, information systems have been extensively deployed to support the distribution, processing, management, and evaluation of data. Contemporary business organizations increasingly rely on them for decision support, which requires efficient response to flexible, sophisticated queries involving application-domain expertise. The ability of most presently deployed decision-support systems to meet such requirements is limited by the fact that they derive information directly from a traditional data organization, such as a relational database, that lacks explicit application-domain semantics. To overcome that limitation, intelligent decision support systems are being developed that employ specifically designed domain-knowledge representations, allowing direct execution of flexible and complex semantic queries.

An intelligent decision support system can delegate tasks to autonomous intelligent entities, i.e., *software agents* (Wooldridge, 2009). For reasoning and communication, agents require a formal representation of knowledge, i.e., an *ontology*, of the application domain. The methodology, technology, and standardization of ontology construction have been advancing rapidly in recent years in the context of the Semantic Web development (Berners-Lee et al., 2001; W3C, 2016). This includes the tools for automatic generation of an ontology, expressed in a formalism such as the Web Ontology Language (OWL), from an existing normalized relational database (RDB), in support of ontology-based data access (OBDA) (Rodrıguez-Muro and Calvanese, 2012).

The potential for advancement of intelligent decision support through synergy of multiagent and Semantic Web technologies has been recognized and elaborated in the description of the Semantic Query Access System (SQAS) by Polajnar et al. (2012, 2014). It envisions an architecture of a distributed system, integrated through agent-oriented middleware, with server nodes containing reference ontologies constructed on top of relational databases for relevant knowledge domains, and client nodes that provide user-specific access to servers through a layer of locally developed custom ontology. A key aspect of SQAS is that reference and custom ontologies are developed gradually within the system itself, in interactions between human experts and software agent assistants. Expected benefits include: autonomous development of the system within the host organization, with its own resources; its ability to evolve the higher-level ontology according to the organization's needs, without affecting the underlying database structure; and the possibility of incremental deployment that does not disturb the conventional use of the existing RDB.

A cornerstone of intelligent decision support in SQAS is the role of agents in the ontology development; we refer to this paradigm as *agent-assisted ontology building* (AAOB). It should be pointed out that the AAOB concept seems applicable in principle to any system in which software provides intelligent assistance to its user. This gives rise to a variety of questions on how ontology-building agents should be designed and implemented, both in principle and in practice, and what type and level of support they can realistically provide to the ontology development process. The existing literature sources on agents with roles in ontology development, such as SQAS, Dynamo (Ottens et al., 2009), and Wiki@nt (Bao and Honavar, 2004), provide little or no guidance in that regard.

The purpose of this thesis is to advance our understanding of the principles and techniques for construction of AAOB systems, as well as the level of support provided by existing and emerging software components and tools in the rapidly evolving research landscape of MAS and Semantic Web. To this end, I propose the system requirements, a general architecture, and agent team structure for a concrete AAOB system called the Ontology-Building Multiagent System (OBMAS), as well as the detailed design and prototype implementation of its key agent role, the Ontology Builder Agent (OBA). This also presents several research challenges that I have encountered in designing OBMAS, and describes the solutions that I have introduced.

An essential question in designing ontology-building agents is how to endow them with the general knowledge about ontology and its development methods. Following SQAS, I used the term *meta-ontology* to describe the ontology of the ontological knowledge domain (note that this sense of the term differs from its established meaning in philosophy). My design of meta-ontology for OBMAS is based on the observation that agents need meta-ontology for two different purposes. First, meta-ontology enables agents to reason and communicate about ontological concepts such as class, subclass, or data property, both in general and in reference to a concrete knowledge domain, such as e-commerce (which I used as the running example). Second, meta-ontology provides to agents the procedural knowledge, i.e., know-how, of the ontology building process in accordance with a specific methodology, such as Ontology Development 101 (Noy et al., 2001). In OBMAS, these two purposes are met by two separate knowledge representations: the *model meta-ontology*, which is the same for all agents, and enables them to reason and communicate with each other about ontological concepts; and the *planning meta-ontology*, which is specific to each agent's role, and provides the agent with the know-how of ontology building relevant to that role. I provide the designs of both meta-ontologies and an implementation for the latter.

Another research challenge in designing OBMAS has been to identify how the agents can assist in the building of knowledge structures that specifically support advanced semantic queries. In the OBDA approach (using the terminology of SQAS), a *base ontology* is automatically "bootstrapped" from the underlying RDB, and then enriched into *reference ontology* with higher-level entities such as new subclasses, superclasses, data properties, etc. Before such a higher-level entity can be used in semantic queries (which are translated to SQL queries on the underlying RDB), a specific mapping must be introduced that relates the entity to RDB-level concepts. A contribution of OBMAS is that whenever the Ontology Builder Agent (OBA) adds a higher-level entity to the reference ontology, it automatically constructs the appropriate mapping which allows immediate use of the new entity in semantic queries. In this thesis, I introduce a full set of mapping rules that enable the OBA to generate the mappings. Several of the mapping rules have been implemented in the current OBA prototype.

The thesis also summarizes some of my insights from the design and partial implementation of OBMAS in regard to existing software components, utilities, tools, and techniques that belong to the current state of the art in the field, hoping that they may provide helpful pointers for further research.

The remaining chapters present: the background and related work (Chapter 2); the research objectives, challenges, and design questions (Chapter 3); the OBMAS architecture and prototype implementation (Chapter 4); the results on meta-ontology design and mapping rules for ontology-building agents, with comments on tool selection and implementation (Chapter 5); the behavioural aspects of ontology development by agent team (Chapter 6); and conclusions and future work (Chapter 7).

Chapter 2

Background and Related Work

In this chapter, I describe relevant background information and summarize the previous work in the fields of the decision support systems (DSS), Semantic Web (SW), multiagent systems (MAS), and ontology development from relational database (RDB).

2.1 Decision Support Systems (DSS)

The organizations today have an increasing demand to find explicit or hidden knowledge in their data sources for making business decisions, solving problems, and forecasting growth. The systems which fulfill these requirements are known as decision-support systems. In this section, I categorize and describe the approaches for the decision support based on the representations of their underlying data sources.

2.1.1 DS Based on Conventional Representation

The conventional decision support systems generally employ relational databases (RDB) for representing and querying the data. Due to the limitations of the relational model of RDB in terms of expressing the table relations, there may be a semantic gap between a user's understanding of the knowledge domain and the actual representation of the concepts in an RDB. One requirement of the decision-support system is to overcome that semantic gap by means of structured SQL queries on the RDB. Moreover, from an organization's perspective, the RDBs may contain implicit knowledge about the performance of the organization and other patterns or trends that can be retrieved in order to make decisions and support them.

Nowadays, two approaches are widely used for the decision support from RDB. In the first approach, a non-technical user receives technical assistance from other skilled users such as a report writer, database administrator, or application developer in order to generate reports from the RDB. The technical user first understands information requirements of the high-level user, formulates and executes structured queries on the RDB, and then generates reports in the desired format.

In the second approach, enterprises employ data warehousing techniques to discover hidden aspects of their data. The production data, which may be stored at multiple locations or in different databases, is first moved and consolidated in an operational database. From there, it is extracted, transformed, and loaded (ETL) into a data warehouse in terms of facts and dimensions (Olszak and Ziemba, 2007), which is then used for reporting and analysis purposes. In the end, the results are saved in the reports and shared with the non-technical user. It is possible to create multidimensional report with efficient querying with that approach.

Both of the aforementioned approaches employ the highly-skilled technical staff to bridge the semantic gap between the user's requirements and the RDB structure. The technical user must have an expert understanding of the RDB structure in order to formulate queries and generate reports. Therefore, the process of generating reports using these approaches increasingly time-consuming and tedious. An accurate foresight of the future information requests is also required for the data warehousing approach in order to design the data warehouse for the long-term usage without an extra technical oversight.

2.1.2 DS Based on Knowledge Representation

Nowadays, a new area of advanced computing known as intelligent decision support system has emerged to address the issues with the conventional systems. An intelligent decision support system for a particular application domain relies on the formal representation of the domain knowledge, i.e., a domain ontology. The raw data in the relational database (RDB) can be semantically annotated using the domain ontology, which in turn enables intelligent agents to reason with the meaningful information within the application context. The domain ontology can be expanded and enriched with new concepts and relationships for evolving application domain to support increasing complexity of the information requests. The systems which employ ontologies as an abstraction over the database content to query information are known as ontology-based data access (OBDA) systems (Rodrıguez-Muro and Calvanese, 2012). The Optique project (Kharlamov et al., 2013) is an example of a complex OBDA system with wide scope; it aims to provide end-to-end scalable solution to Big Data Integration. The development of a domain ontology to support semantic querying from the RDB is a complex task that requires a specialist user like an ontology developer.

In recent times, an increasing practical relevance of the multiagent systems and Semantic Web technologies has raised the possibility of building intelligent decision support systems that provide intelligent assistance to build a domain ontology from RDB. To this end, Semantic Query Access System (SQAS) (Polajnar et al., 2012, 2014) proposes an innovative distributed agent-oriented architecture with intelligent software agents which develop a domain ontology with their know-how of ontology building process. In SQAS, a user can execute sophisticated semantic queries against an RDB using a customized ontology which is built from the user's definitions of domain concepts. An important aspect of that proposal is that an enterprise decision-support system like SQAS could be developed within the organization itself, and evolve with its needs, without modifying the underlying RDB (Polajnar et al., 2014). Moreover, the development process does not disrupt the conventional access to the RDB.

2.2 The Semantic Web

The Semantic Web (Berners-Lee et al., 2001) is a web of data that enables computer systems to understand the semantics, or meaning, of information that populates the World Wide Web (WWW), i.e., the web (Berners-Lee, 1998). The notion of Semantic Web was proposed to represent a new machine-readable version of the web. The web is also referred as a web of documents because the information is transferred and read mostly in terms of documents. It is largely unstructured and only humans can read and interpret most of the documents on the web. The Semantic Web intends to transform the documents into structured data through a set of new technologies available for that purpose such as Resource Description Framework (RDF) and Web Ontology Language (OWL). The structured representation then enables computer programs to collect, interpret, and synthesize information from the web. The agents can answer questions from its user through interaction with other agents, or by reading, understanding, synthesizing, and analyzing structured information on the Semantic Web. The technologies of the Semantic Web that are relevant for this research are explained next.

2.2.1 The Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a class of specifications used to describe design and structure of the information on the web (Manola et al., 2014). The meaning of data is expressed through a concept model which encodes the data in a set of triples as subject, predicate, and object. The subject is the concept being described, the predicate is an attribute of the concept, while the object holds a value of the attribute for the concept. The subject and predicate have specific resource URIs (Universal Resource Identifier) which uniquely identify them over the web. The RDF follows standard syntax for XML (eXtensible Markup Language) to express the concepts. It plays an essential role for the data representation in the Semantic Web. The Web Ontology Language (OWL) for representing ontologies conforms to the grammar described in the RDF specifications.

2.2.2 The Web Ontology Language (OWL)

The semantics of the knowledge or information on the web can be expressed through an explicit specification of concepts known as ontology. An ontology contains a hierarchical structure of concepts corresponding to a particular knowledge domain, and provides inference rules to reason about the knowledge domain. The Web Ontology Language (OWL) (Hitzler et al., 2012) is a W3C standard and recommended knowledge representation formalism for describing ontologies in the Semantic Web. A concept in the ontology can be expressed with its name, roles (properties), and role restrictions. A property can be associated with one or more concepts. There are two kinds of properties: object property links individuals to other individuals, and data property links an attribute value to an individual. The ontology supports advanced structuring features like property restrictions, property characteristics, and inference rules to reason about the information.

The OWL 2 Web Ontology Language, informally OWL 2, (Motik et al., 2012) describes OWL profiles, i.e., fragment or sublanguage, that support different subsets of full OWL 2 specification in order to reduce expressiveness for the efficiency of the reasoning. The profiles described here are useful in different application scenarios. The choice of the profile to use in practice depends on the structure of the ontologies and the reasoning task.

OWL 2 EL is useful in applications that employ ontologies with numerous classes. It captures the expressiveness required by such ontologies and provides basic reasoning in polynomial time respective to the size of the ontology. The EL acronym reflects the profile's basis in description logic with existential quantification.

OWL 2 QL is aimed at applications that use large volumes of data, and where query answering is the most important reasoning task. The expressive power of the profile is quite limited, although it includes most of the main features of the conceptual models. The QL acronym reflects the fact that query answering can be achieved through rewriting queries into a standard relational Query Language.

OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can accept sound but incomplete reasoning with the ontologies. The RL acronym reflects the fact that reasoning in this profile can be implemented using a standard Rule Language.

2.2.3 The SPARQL Protocol and RDF Query Language (SPARQL)

The SPARQL Protocol and RDF Query Language (SPARQL) (Prud'hommeaux and Seaborne, 2008) describes structured semantic queries for data access from the RDF data store. The queries can be executed across diverse data sources, for instance, the data stored in RDF data store, or the data viewed as RDF but retrieved from the different data store through a translator. For instance, the RDB-RDF translators can execute SPARQL queries on the underlying RDB after translation to its SQL counterpart. The SPARQL queries can return results as RDF triples or graphs.

2.2.4 RDB-to-RDF Translation

The RDBs are widely used data sources for supporting the large number of websites on the web. In order to build a structured web, the data from the RDBs should be translated to the Semantic Web technologies such as RDF. The RDB-to-RDF translation is the process which does that. Two means are important for that process: RDB-to-RDF mappings, and a tool which can employ the mappings for translating the data to RDF triples.

The RDB-RDF translators can generate a basic ontological structure from the RDB schema through bootstrapping. The process also generates RDB-to-RDF mappings which express the relations of classes with underlying tables using a mapping language such as R2RML (Das et al., 2012). The automated process employs a basic or direct mapping rule for conversion from the RDB schema to an ontology in that each table corresponds to a class and its columns to the data properties of the class.

The mappings can also be used to access data from the RDB using a SPARQL query. It is first translated to a SQL query through SPARQL-to-SQL translation process supported by some RDB-to-RDF translators. The SQL query is then executed on the underlying RDB, and results are converted to the RDF triples using the same mappings.

RDB to RDF Mapping Language (R2RML) expresses customized mappings from relational databases to RDF datasets. Each R2RML mapping can be tailored to a specific RDB and target vocabulary. R2RML mappings are expressed as RDF graphs and written in Turtle syntax. An R2RML mapping refers to logical tables to retrieve data from the input database. A logical table can be one of the following: a physical RDB table; an RDB view, or, a valid SQL query.

Virtuoso data server supports R2RML by means of its proprietary mapping language called Linked Data Views (Software, 2014). It can be used to generate default R2RML mappings from the database schema using the basic mapping rule. It also hosts a SPARQL endpoint which enables a user to execute SPARQL queries.

Ontop is an open-source ontology-based data access (OBDA) system which employs domain ontology as an abstraction over an RDB to query information (Rodríguez-Muro et al., 2013). It supports query rewriting from SPARQL-to-SQL like other translators. It uses custom mapping language to represent RDB-to-RDF mappings, which can also be translated to and from R2RML mappings. It employs several structural and semantic optimizations while rewriting queries and supports all major relational databases. Quest (Rodríguez-Muro and Calvanese, 2012) is a core query rewriting engine and reasoner for Ontop which supports OWL 2 QL entailment regime for reasoning with ontologies. It is under active development.

2.3 Multiagent Systems

A multiagent system (MAS) (Wooldridge, 2009) consists of a number of intelligent agents who can interact with each other while situated in a common dynamic environment. The agent is a computer system capable of independent autonomous action on behalf of its user or owner to accomplish its delegated objectives. The agent perceives the environment in real-time manner to capture the sporadic changes. It reacts in a timely fashion to those changes through actions in order to achieve its delegated design objectives. Some agents also possess a goal-directed behaviour (pro-activeness) by taking initiatives to fulfill their design objectives. The agents are believed to play an important role in the Semantic Web as well. They can seek, collect, synthesize, understand, and interpret the structured information. The users can save time of information retrieval and management from the varied web sources, and make quick decisions from the combined information (Berners-Lee et al., 2001).

2.3.1 Practical Reasoning in Multiagent Systems (MAS)

The concept of belief - desire - intention (BDI) model was first formulated by Shoham (1993) while proposing the agent-oriented programming (AOP) techniques. The idea underlying the model is to program computer systems, especially intelligent agents, with these notions in order to support agent reasoning. The notions are defined as follows: beliefs represent the agent's understanding of the state of the world; desires represent all state of the affairs the agent would like to fulfill which can influence the agent's actions; and intentions (i.e., goals) are the state of affairs the agent has determined to achieve. The AOP techniques are envisioned to design the agents with a non-technical declarative style of programming in order to reduce emphasis on control aspects of the agent reasoning. The four main properties of the intentions are: they lead to actions, they persist until achieved or renders unachievable, they constrain agent deliberation to avoid inconsistency, and they are related to the agent's future beliefs.

The model of decision-making to determine actions from the belief, desires, and intentions of the agent is called practical reasoning. It is an action-directed reasoning. Two main activities within the practical reasoning are the deliberation and means-ends reasoning. The process of deliberation determines the intentions of the agent, while the means-ends reasoning decides how to achieve an end (i.e., an intention) with available means (i.e., actions). A planning system employs the agent's beliefs, intentions, and actions repertoire to generate a plan to achieve the desired state of the affairs.

An elaborate reasoning cycle for intelligent agents is implemented in the Jason interpreter (Bordini et al., 2007) for AgentSpeak (Rao, 1996). The agent follows a set of steps within a reasoning cycle to decide a set of actions that it can execute in order to change the state of the environment, i.e., world. The steps are as follows: perceive the environment to identify changes; update the belief base with the changes and create new events for them; read messages from the message queue; select acceptable messages and generate corresponding events; select an event from the pending list of events; select relevant plans for the selected event; determine applicable plans, i.e., desires based on the present state of the world from the relevant plans; select one applicable plan to include in the set of intentions; select an intention for execution from the set of intentions; and, execute one step of an intention.

2.3.2 Platforms for MAS Development

Three known platforms for the research and development of the multiagent systems are as follows:

- Jason is an interpreter and platform for a programming language called 'AgentSpeak' which is useful in building computer programs as *agents* (Bordini et al., 2007). It comes with a rich environment that enables agents to communicate and coordinate with one another in a high-level manner. Each agent follows an elaborate reasoning cycle which can be customized as per the design of the multiagent system.

- Madkit is an implementation of Aalaadin which is a meta-model of artificial organization (Ferber and Gutknecht, 1998). Madkit allows for the several multiagent systems to work concurrently.

- The Agent Factory (af2, 2014) framework is an open source collection of tools, platforms, and languages that support the development and deployment of the multiagent system. The Common Language Framework (CLF) is a part of the Agent Factory and it supports the development of agent interpreters for different Agent Oriented Programming (AOP) languages.

2.3.3 Using Ontologies in MAS

Two or more agents should agree on the *terminology* that they use to describe the domain (Wooldridge, 2009) in order to communicate effectively. The ontology can provide common basis of understanding for the agents in order to remove ambiguity in communication messages. Moreira et al. (2006) propose an approach of agent-oriented programming with underlying ontological reasoning. The four main advantages of that approach are as follows: (i) the expressive queries can be formulated which can take advantage of explicit as well as inferred knowledge from the ontology; (ii) it enables checking consistency of the knowledge, i.e., the ontology for new or updated beliefs; (iii) an agent can follow common plans using subsumption relation between classes through plan trigger generalization; (iv) and the agents can share knowledge about the domain terms with the other agents using OWL ontologies.

JASDL (Jason-AgentSpeak Description Logic) is an extension of Jason agent platform which supports the aforementioned features through Jason customization techniques and the usage of OWL API. It is the first full implementation of an agent-oriented programming language (AOP) using ontological reasoning within a declarative setting (Klapiscak and Bordini, 2009). It annotates the agent beliefs with a specific ontology instance annotation. In addition, the Jason plan library is extended to support enhanced plan lookup using the plan trigger generalization.

Ontology for Agent Plans: The agents in a MAS can possibly benefit from using ontologies as a plan library to influence the agent's behaviour in order to suite different deployment scenarios. To this end, an approach proposed by Freitas et al. (2014) presents a semantic model for planning domains which can be translated to actual agent plans. The semantic model is an OWL planning ontology. In addition, they propose and implement an algorithm to facilitate the conversion from classes in the planning ontology to the agent plans in AgentSpeak language. The planning ontology contains the following major concepts. *Operator* describes how a primitive task can be performed. It contains name, parameters, preconditions, a delete list and an add list of predicates. *Method* indicates how to decompose a compound task into a partially ordered set of subtasks which can be either compound or primitive. It contains three parts: the task, preconditions, and subtasks. *Method Flow* describes how it can be decomposed based on the current state of the world. *Parameter* is a variable symbol used by operators, methods, and predicates.

2.4 Developing Domain Ontology

The ontology development process for any knowledge domain involves a number of iterations for its classification and expansion. It is a fairly complex process by its nature. Generally, an ontology developer or a knowledge engineer who possesses a specialist knowledge of ontology language syntax and development process is actively involved in that process along with domain experts. An ontology development environment such as Protégé (Horridge et al., 2004) or TopBraid Composer (Inc, 2011) can be employed to build an ontology. In addition, the ontology developer accesses and browses other resources such as dictionary, thesaurus, WordNet, or external ontologies to gather, analyze, identify, and reuse existing knowledge of the domain in terms of classes, properties, and their relationships.

2.4.1 Ontology Development Methodology

An ontology development methodology describes a set of principles, methods, and activities to design, construct, build, and share ontologies (Gašević et al., 2006). Noy et al. (2001) states that there is no standard ontology development methodology which can be common to many application domains to build an ontology. However, there are a few guiding steps which can be useful in developing a domain ontology. A methodology describing these steps is termed as Ontology Development 101 methodology and the steps are as follows:

Step 1: Determine domain and scope of the ontology

The process starts with defining domain and scope of the ontology. Some common questions which can be asked are related to the knowledge domain, purpose of ontology, types of questions that the ontology is expected to answer, and prospective users of the ontology.

Step 2: Consider reusing existing ontologies

During this step, an existing ontology for the knowledge domain can be identified and reused after some refinements instead of building an ontology from scratch.

Step 3: Enumerate important terms in the ontology

Create a comprehensive list of domain terms with the help of the questions such as (a) what are the terms a user would like to talk about? (b) what properties these terms have? (c) what is an additional information about a term?

Step 4: Define classes and class hierarchies

During this step, new classes can be introduced in three ways: *top-down* approach involves creating specialized concepts from the general concepts; *bottom-up* approach follows a reverse direction to group common properties in specialized classes to a general class; and *combination* approach which starts with the most notable concepts and then continues to expand through the generalization and specialization to the remaining concepts.

Step 5: Define the properties of the class - slots

The properties provide additional information about a class by describing its internal structure. From the list of important terms, several terms are defined as classes in the previous step, while remaining terms are most likely to be the properties of those classes.

Step 6: Define the facets of the slots

A slot can have facets, i.e., property restrictions such as allowed values and cardinality.

Step 7: Create instances

In the final step, individual instances are created for the classes in the ontology.

The steps can be reiterated a number of times until an acceptable version of the ontology is developed.

2.4.2 Ontology Metamodel

The OWL 2 Metamodel based on the MetaObject Facility (MOF) (Brockmans et al., 2008) enables the applications to represent ontological notions such as class, data property, object property, property restriction, and subclass relation using a metamodel OWL ontology. It describes a machine-readable specification of OWL 2; however, it is not updated to reflect the latest refinements in the OWL 2 specification. Fig. 2.1 highlights a section in the graphical representation of the large OWL 2 Metamodel in an OWL ontology.

2.4.3 Building Domain Ontology from RDB

There has been numerous approaches to creating a domain ontology from the RDB. Most of them propose frameworks with a set of rules and mapping related to the target knowledge domain to create an ontology from the RDB (Trinh et al., 2006; Choi and Kim, 2012; Ramathilagam and Valarmathi, 2013). The approach requires an ontology developer who is adept at the language syntax, semantics, and the development methodology. The advantage of having rules and mappings to create a basic ontology from the RDB is that it provides a significant head start in the ontology development. Hence, allowing for efforts of the ontology developer and domain experts to focus on refining and expanding the basic ontology through new abstractions. The ontology development is an iterative process and there is no correct ontology-design methodology. A best solution for building the ontology depends on the expected usage and extensions of the resultant ontology. The Ontology Development 101 (Noy et al., 2001) methodology can be used to build a domain ontology from the basic ontology.

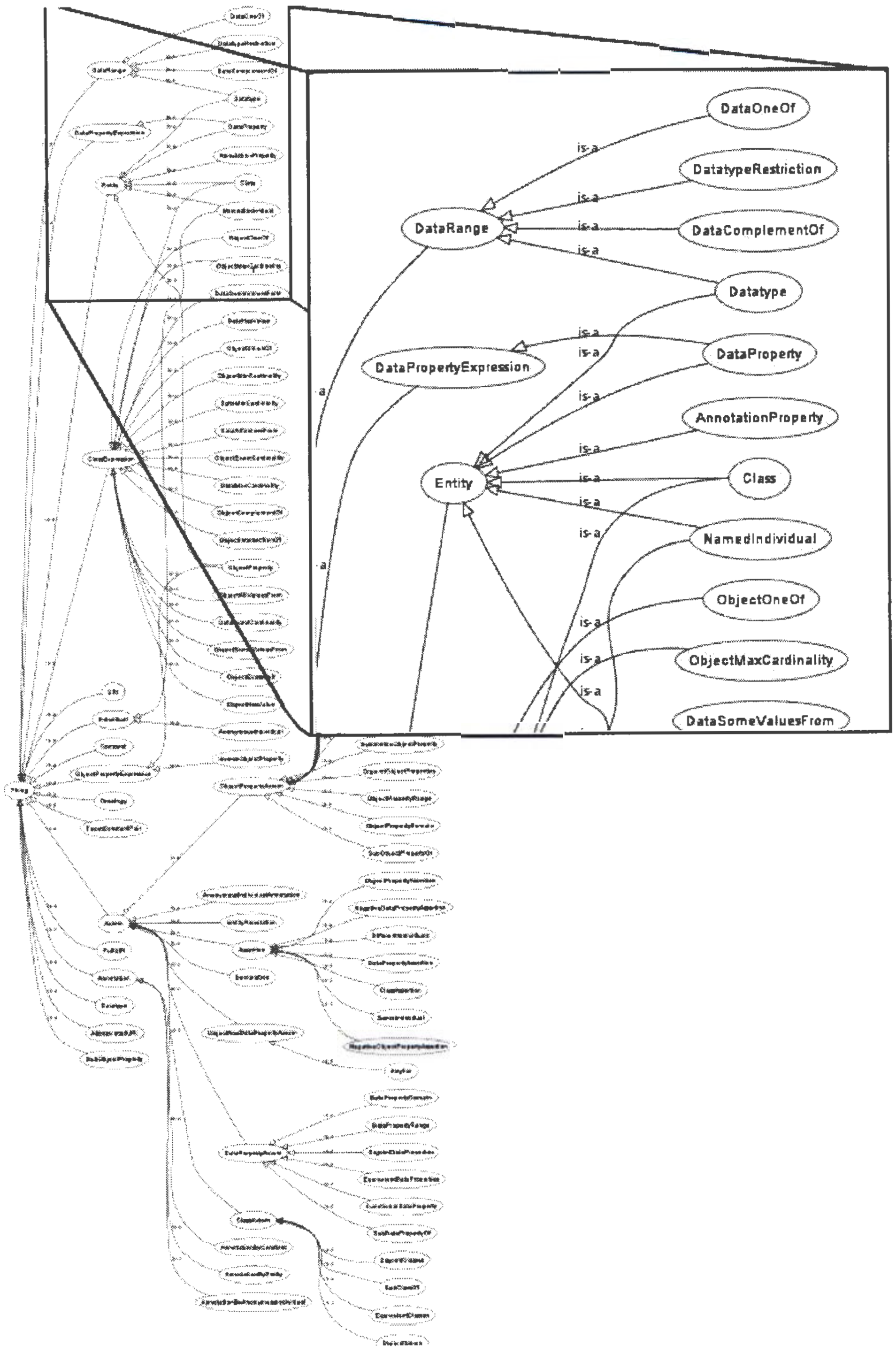


Figure 2.1: The OWL 2 MOF-Based Metamodel

2.4.4 Using MAS for Ontology Development

The semantic capability of Web Ontology Language (OWL) is high due to an availability of various ontology constructs for the formal concept modelling. The complexities of the ontology development methodology make the ontology-building process slow, tedious, costly, and time consuming. As a fully-automated ontology development process is impossible to achieve, the software agents can assist humans in the collaborative ontology development (Bao and Honavar, 2004). The agent-oriented ontology-building systems can be categorized and evaluated as per the following criteria: underlying database technology, number of agents, independent agent roles and capabilities, agent communication mechanism, agent's knowledge of ontology development methodology, and roles of the domain expert in the ontology-building task.

Dynamo Framework is a MAS-based semi-automated approach to learn ontology by analyzing the domain-specific set of texts (Ottens et al., 2009). The approach is focused on the ontology learning using a distributed hierarchical clustering algorithm. It is semi-autonomous because an ontology developer is required to design the ontology with adaptive agents. The framework generates concepts based on several criteria which are derived from the statistical computations on the texts. The agents represent the concepts in the system and are linked by labelled relations. In the process, the agents repeatedly execute the distributed algorithm until the equilibrium state (a binary tree) is achieved. The framework does not define any explicit communication mechanism but it is possible to create a simple communication layer for messaging between agents. The ontology developer can modify the first hierarchy generated by the system or in between the clustering process. The system adapts to the changes depending on the modification request from the user.

Wiki@nt (Bao and Honavar, 2004) supports collaborative ontology development for integration and reconciliation of multiple heterogeneous and inconsistent ontology modules. The ontology modules correspond to semantically heterogeneous and distributed data sources. The environment provides agent-oriented framework which supports collaborative ontology building as a design principle where multiple individuals can communicate and cooperate in editing a shared ontology. The human partner needs to understand the syntax of the ontology language to use the environment. The environment can be set up to enable an ontology developer to build the ontology collaboratively with the software agents. The approach is semi-autonomous owing to the need of human partner in the ontology development. The agents can assist in finding concepts and their relations from original data sources, small pieces of ontologies, and consistent concepts.

Both of the aforementioned approaches employ intelligent agents to build ontologies, but they build ontologies that can not be used for intelligent decision support. The Dynamo framework limits itself to defining taxonomy of terms instead of a domain ontology. On the other hand, Wiki@nt lacks explicit definitions of the ontology-building agents. In addition, both of the approaches use different data sources than an RDB which can provide basic structure for the domain ontology.

SQAS (Polajnar et al., 2012, 2014) is a qualified intelligent decision support system which develop domain ontology with intelligent agents to support semantic queries over the RDB. It employs D2RQ Mapping Language (Cyganiak et al., 2012) to facilitate the conversion from the RDB data to RDF triples. The generate-mapping tool of D2RQ server creates a default mapping from the RDB. The default mapping is referred as a *base ontology.* The autonomous agent endowed with a knowledge of ontology-building techniques assists the database administrator in building the *refer-*

ence ontology. The mapping file generated by D2RQ is considered the base ontology in SQAS.

The reference ontology is developed based on the base ontology, domain knowledge of the user, and related existing ontologies retrieved from the Semantic Web. The agent reads the mapping file, and defines primary classes and properties in the reference ontology. It then assists the user in writing generalized concepts, properties with meaningful names, and relations between classes. The users can customize the reference ontology according to their understanding of the domain in order to create a *custom ontology*. Later, the user submits a request for report in Simplified Natural Language (SNL) which is verified with the help of the custom ontology to create a SPARQL query. The SPARQL query is translated to its SQL counterpart which is then executed on the RDB. The results are translated to the RDB triples, and then formatted and presented to the high-level user in the form of a report.

SQAS consists of two subsystems namely User Subsystem (US) and DataBase Subsystem (DBS). In both subsystems, the agents along with core environmental software components constitute a layer of *intelligent middleware* that can offer support to other external agent-oriented applications. The US runs on the client machine, and hosts the User Interface Agent (UIA) and User Interface Environment (UIE). The DBS interacts with Database Administrator (DBA) and Data Entry Operator (DEO). It hosts Database Interface Agent (DBIA) and Database Interface Environment (DBIE). The agent provides the practical reasoning capabilities to the parent subsystem, enabling it to autonomously resolve arising problems without intervention of the human partner. The agent's knowledge about building ontology is stored as an ontology of the ontology-building knowledge domain which is referred as a *meta-ontology*.

In essence, SQAS introduces a novel approach of employing intelligent software agents to assist their human partner in building domain ontology from basic ontological structure generated from the RDB. That approach is completely original to my knowledge in which the emphasis is on endowing software agents with the know-how of the ontology development in a meta-ontology.

Chapter 3

Research Objectives

In this chapter, we outline the main objectives of the present research. With the background, context, and related work presented in the previous chapter, we now revisit the thesis objectives stated in the Introduction. Essentially, we address the agent-assisted ontology building (AAOB), as introduced in SQAS (Polajnar et al., 2012, 2014), from complete multiagent system design perspective, with prototype implementation of key AAOB functions. The overall design of the multiagent system, including the agent team structure and various supporting modules and utilities, is embodied in the architecture of the Ontology Building Multiagent System (OBMAS). The system retains the ontology-based data access (OBDA), with ontology generation from RDB as in SQAS, as its AAOB application domain. For practical development, testing, and presentation, I use a fragment of the publicly available e-commerce RDB, Opencart (Opencart, 2014). The OBMAS architecture is methodically developed, starting with the formulation of system requirements, and includes all supporting components needed for a complete system. The central AAOB functionality of OB-

MAS is represented in the Ontology Builder Agent (OBA), which is implemented at the prototype level.

Given the novelty of the AAOB system design, the objectives include the investigation of AAOB research challenges, design issues at the OBMAS level, and prototyping issues at the OBA level. We address them in three separate sections below.

3.1 Research Challenges in AAOB

In this thesis, I have identified three research challenges to the agent-assisted ontology development from RDB. First, *how to design meta-ontology* (proposed in SQAS) which imparts the knowledge of an ontology development process and ontological notions to the agents, such that they can offer active assistance in interaction with the user and other agents by reasoning with the meta-ontology. Second, *how an agent can help create advanced support for translating semantic queries* for data access. Third, *how to select tools and languages* from rapidly evolving landscape of tools that are appropriate for practical development of an agent prototype.

Using meta-ontology, we assure that the ontological know-how is not part of the agent design but retrieved from an external knowledge source which can be easily upgraded. The existing approaches to using agents for ontology development such as SQAS, Dynamo (Ottens et al., 2009) and Wiki@nt (Bao and Honavar, 2004) do not support or provide designs for meta-ontologies. In order to design them, we may need to solve problems such as extracting metamodel for ontology formalism, translating concepts from ontology formalism to agent design language, and identify that how the agent team structure will influence the meta-ontology design.

In order to translate semantic queries to RDB-oriented SQL queries, we need mappings which link SQL queries to the concepts in the ontology. Presently, tools such as Ontop (Rodríguez-Muro et al., 2013) support mappings for the query translation, which are manually developed by a user for higher-level concepts in the domain ontology. We intend to delegate that responsibility from the user to the ontology builder agent.

There exists several tools and technologies which are built for the different purposes of the OBDA system development. We intend to identify and employ them as appropriate for the design, implementation, and further research on the proposed system through an analysis based on various usability aspects.

3.2 Design Questions in OBMAS

In order to demonstrate the AAOB approach, a multiagent system should be designed and implemented with a small-scale prototype. Some main questions for its design and our approach to solve them are described next. Those questions are not yet addressed in any research projects from the perspective of the agent-assisted ontology development.

What are the system requirements for the proposed system? In order to answer that question, we will manually build the domain ontology, i.e., *reference ontology* in terms of SQAS, by following a method of ontology development in order to gain experience and identify specific system requirements in the context of OBDA. The process will also give us clear ideas about possible agent tasks. The system requirements will enable us to present complete scope of the proposed system.

What are the individual agent roles and skill profiles? The agent roles will be identified from the various sources such as the system requirements, known tasks for the ontology development, OBDA projects such as Optique (Kharlamov et al., 2013), and approaches to knowledge discovery (Spanos et al., 2012) from external sources such as ontology repositories (Ding et al., 2004; d'Aquin et al., 2007). They will help us to decide the composition of the agent team and skill profiles of the agents.

We intend to design a low-level architecture for the proposed system because the distributed-system model of SQAS cannot be employed here due to its larger footprint for ontology development in an enterprise.

3.3 Prototyping the Ontology Builder Agent

The prototype with an ontology builder agent would demonstrate some key tasks or problems for the ontology development from RDB as well as some of the research challenges. To this end, we will identify specific features or issues while formulating system requirements, some of which will be addressed in the implementation. We expect to demonstrate features which require the agent's assistance such as bootstrapping the basic structures (i.e., base ontology) from RDB, building mappings, following an ontology development methodology, and creating higher-level abstractions in the reference ontology.

The prototype would provide the first implementation of the novel approach of ontology development using intelligent agents. The design of a working agent and its associated components would lay out the groundwork for building other agents and utilities for a larger-scale implementation. It would be used to identify challenges and research problems associated with the development of intelligent decision support systems including SQAS.

Chapter 4

The OBMAS Architecture Model

In this chapter, I identify the system requirements, and present the Ontology Building Multiagent System (OBMAS) architecture with the descriptions of its structural and behavioural aspects. Polajnar et al. (2012, 2014) have identified and analyzed the general requirements for the intelligent agent-oriented middleware in the SQAS architecture. This chapter takes that research further, to identify the composition of the ontology-building agent team, specify individual agent roles, and describe the necessary utilities and interfaces. This results in a new architecture that is used to design and partially implement a functioning OBMAS prototype using existing state-of-the-art technologies and tools, and to lay the groundwork for a more complete implementation in the future. In the following sections, I describe the system requirements (4.1), the system structure (4.2), the system behaviour (4.3), and the current prototype implementation (4.4).

4.1 System Requirements

In this section I describe the initial steps leading to the discovery of system requirements (Subsection 4.1.1), and classify those requirements into three categories: *functional requirements* (4.1.2), *non-functional requirements* (4.1.3), and *domain-specific requirements* (4.1.4) (Sommerville, 2015).

4.1.1 The Requirements Discovery Process

The first step in the requirements discovery process was to gain immediate experience of ontology development in the context of ontology-based data access (OBDA) (Rodrıguez-Muro and Calvanese, 2012). Following the *Ontology Development 101* methodology (Noy et al., 2001), I manually built the reference ontology for the e-commerce domain from publicly accessible Opencart RDB (Opencart, 2014) using the Protégé ontology editor. In order to generate the RDB-to-RDF mappings, I used the Ontop Protégé plug-in (Rodríguez-Muro et al., 2013). Finally, I used the generated mappings to execute SPARQL queries on the Opencart RDB.

The activity diagram in Fig. 4.1 illustrates the main actions in this development process. The manual construction of the reference ontology enabled me to identify a complete set of system requirements. I also gained insights into technical aspects of ontology development from RDB, and learned the state of existing standards and tools. Based on this experience, I adopted the Ontology Development 101 methodology, which provides the generic steps for ontology development independent of any specific process and knowledge domain.

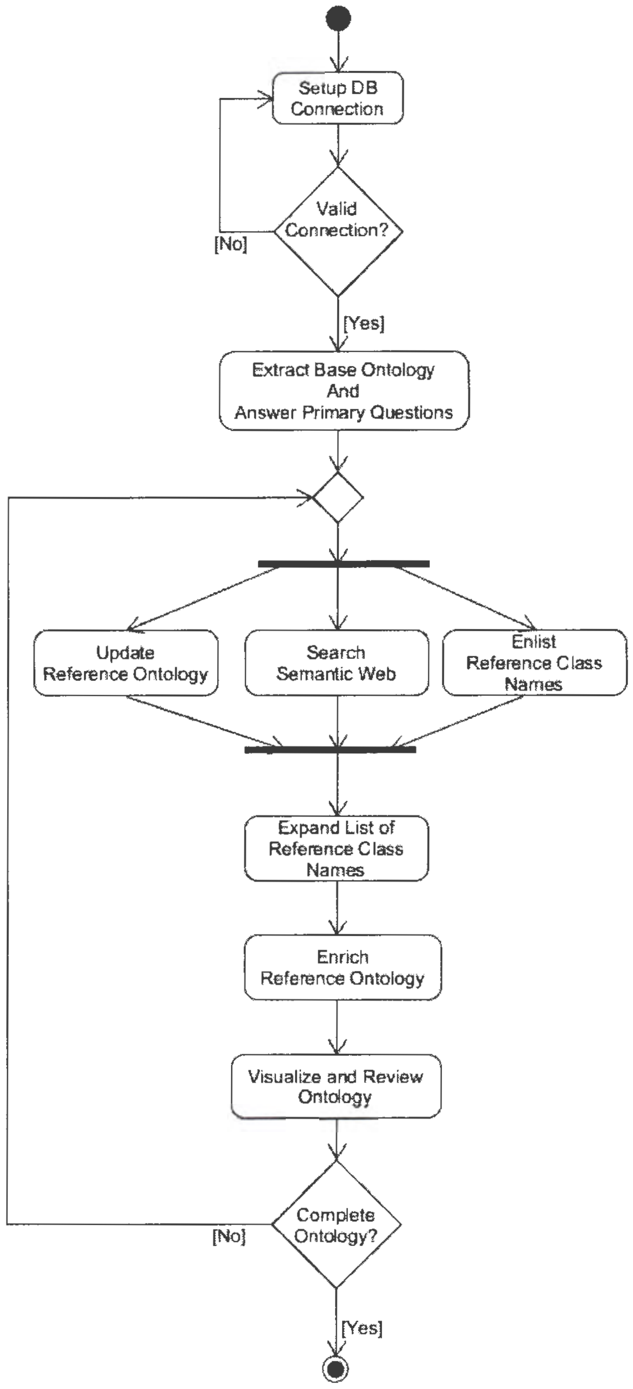

Setup DB Connection
Valid Connection?
[No]
[Yes]
Extract Base Ontology And Answer Primary Questions
Update Reference Ontology
Search Semantic Web
Enlist Reference Class Names
Expand List of Reference Class Names
Enrich Reference Ontology
Visualize and Review Ontology
Complete Ontology?
[No]
[Yes]

Figure 4.1: The reference ontology development for e-commerce domain

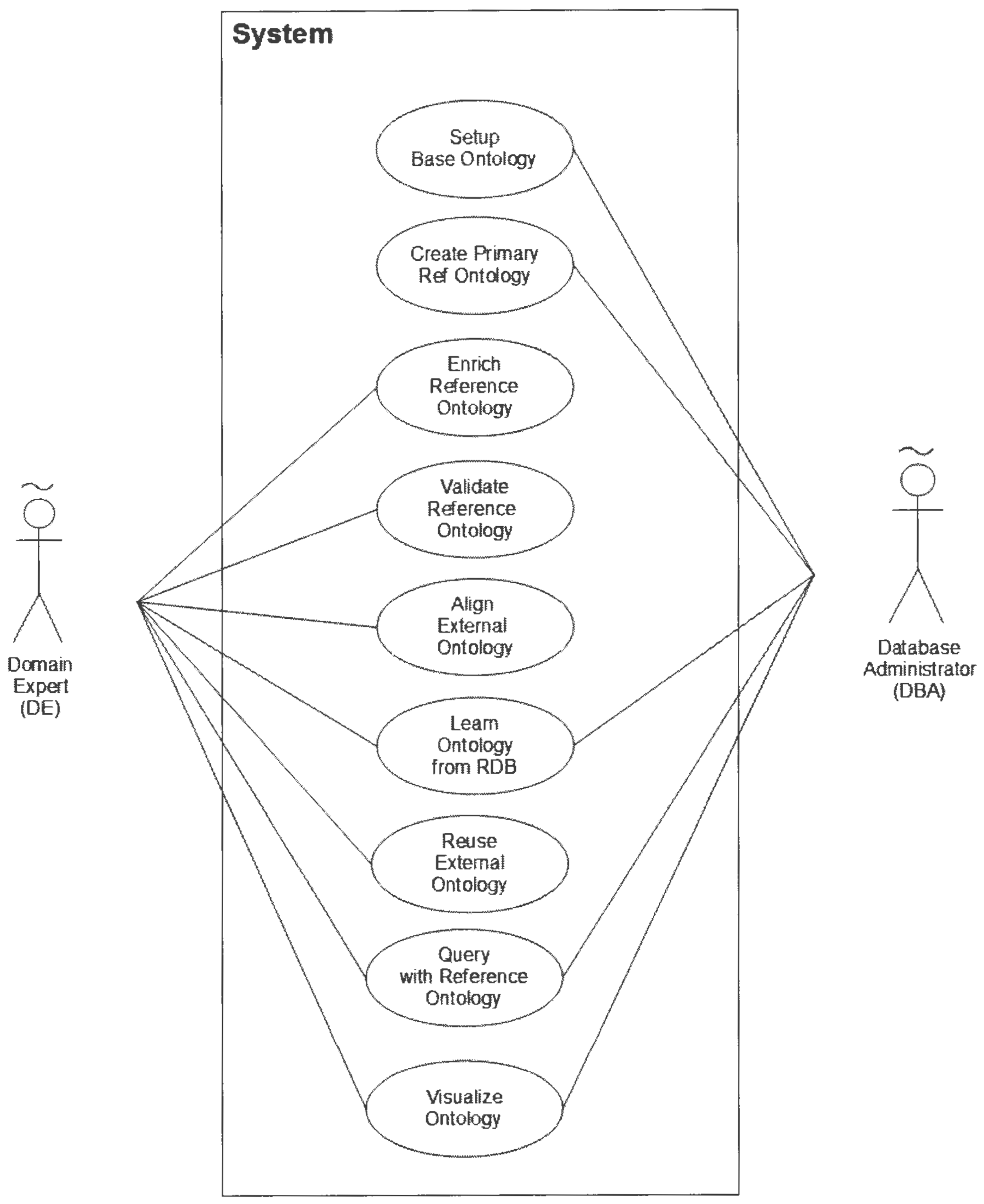
4.1.2 Functional Requirements

The high-level functions of OBMAS are shown in Fig. 4.2. The use case diagram illustrates the actors and use cases that are explained in the rest of this subsection. The actors are: the Database Administrator (DBA), who develops a primary reference ontology using the knowledge of table relations in the RDB schema; and Domain Expert (DE), who enriches the reference ontology using the knowledge of application domain. Fig. 4.3 shows the decomposition of all high-level use cases. The detailed descriptions of all use cases can be found in the OBMAS project report (Mumbaiwala, 2016).

In each of the use case descriptions that follow, it is assumed that the routine functions and lower-level decisions are delegated to the OBMAS agents, or performed by specialized system utilities. The actor initiates the activity, receives information and suggestions from the system, and retains responsibility for higher-level decisions.

In the *Setup Base Ontology* use case, the Database Administrator (DBA) first identifies the RDB and provides it connection details; the system connects to the RDB and bootstraps the RDB schema into the base ontology. Next, the DBA names the knowledge domain of the RDB, enabling the agents to look for existing external ontologies in the same domain. The DBA also selects the OWL sublanguage for the representation of reference ontology.

When the DBA invokes the *Create Primary Reference Ontology* use case, OBMAS creates the initial reference ontology from the base ontology and helps the DBA identify new subclasses, class names, subclass relations, and object property relations (as defined in OWL) between existing classes from the RDB schema. The system then refines the initial reference ontology with these relations to create the primary


System
Setup Base Ontology
Create Primary Ref Ontology
Enrich Reference Ontology
Validate Reference Ontology
Align External Ontology
Learn Ontology from RDB
Reuse External Ontology
Query with Reference Ontology
Visualize Ontology
Domain Expert (DE)
Database Administrator (DBA)

(a)

Figure 4.2: The actors and high-level use cases of OBMAS

reference ontology.

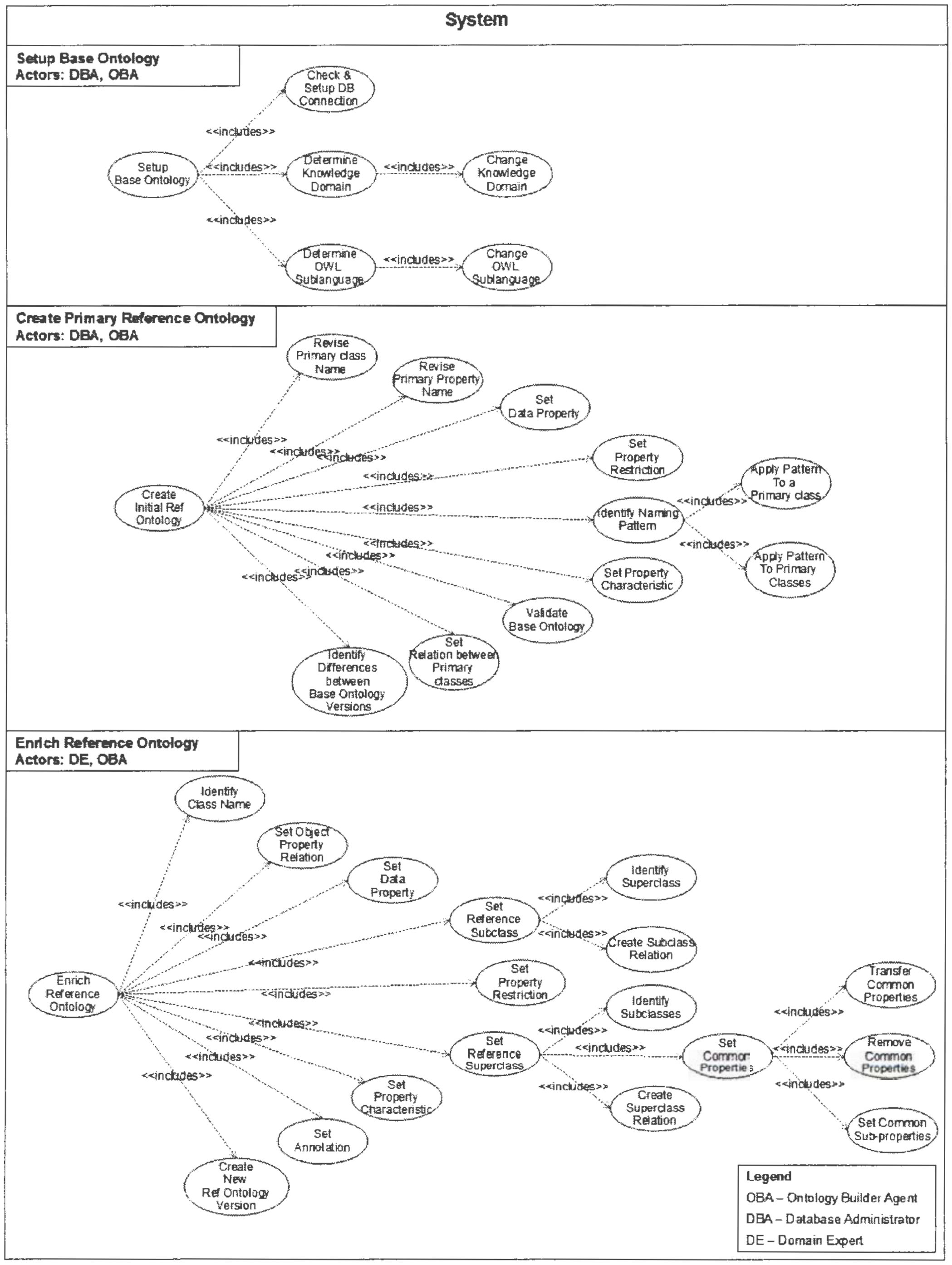
The *Enrich Reference Ontology* use case captures some of the central activities of OBMAS. It is invoked by the Domain Expert (DE), who asks OBMAS to identify new subclasses and superclasses for the primary classes (in the primary reference ontology). The DE then selects and approves the new subclasses and superclasses, and also introduces the desired data properties, object properties, property restrictions, property characteristics, and annotations. This enrichment process can proceed recursively until the DE considers the reference ontology to be complete. The system also generates new reference ontology versions in response to changes in the RDB schema.

When the DE invokes the *Align External Ontology* use case, OBMAS identifies the alignments, i.e., the equivalencies between the classes and properties in the reference ontology and an external ontology. The DE then selects the appropriate alignments and requests to introduce them in an alignment ontology.

The *Validate Reference Ontology* use case is periodically invoked after a time interval or upon the actor's request. It validates the reference ontology in order to identify any logical conflicts or modelling errors. The DE then attempts to resolve the conflicts and errors through reclassification or other modifications in the reference ontology.

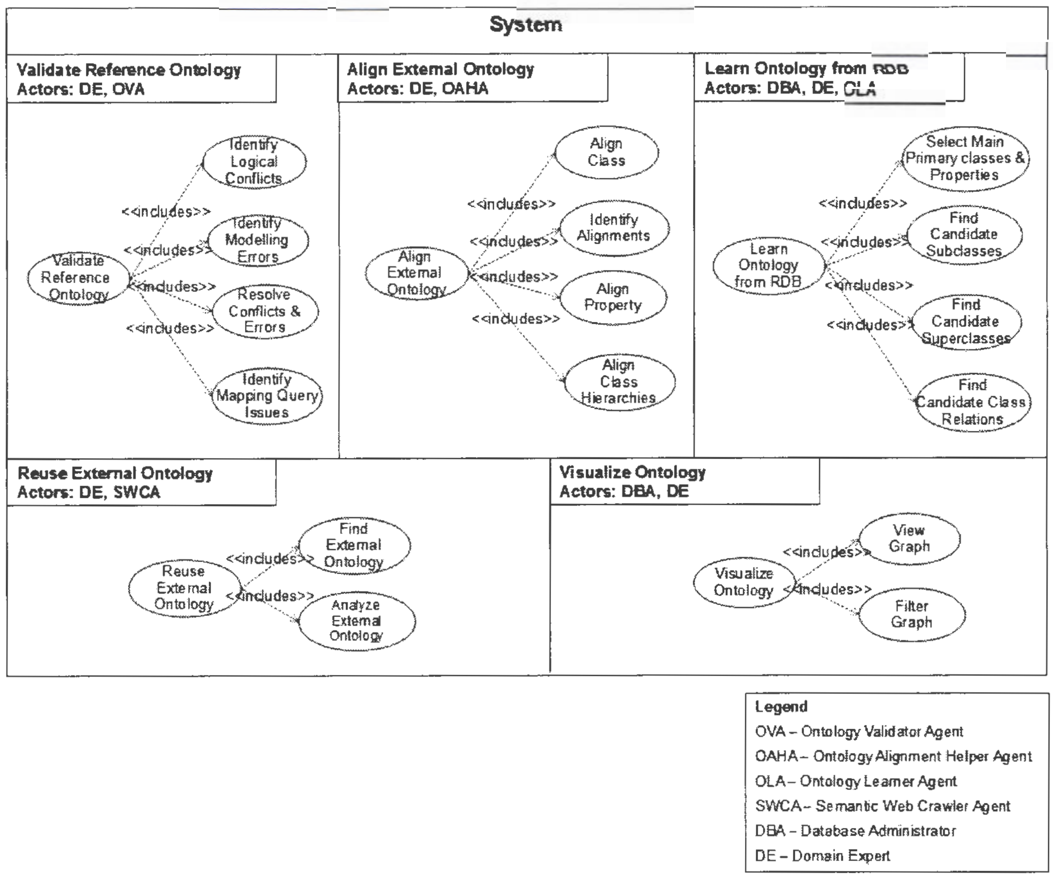
In the *Learn Ontology from RDB* use case, the actor selects the main primary classes and properties for the domain, and the system analyzes both the RDB schema and the RDB data in order to identify new candidate subclasses, superclasses, and relationships among them. Next, the actor selects the appropriate classes and relationships to include them in the reference ontology.

When the DE invokes the *Reuse External Ontology* use case, OBMAS explores


System
Setup Base Ontology
Actors: DBA, OBA
Setup Base Ontology
Check & Setup DB Connection
Determine Knowledge Domain
Change Knowledge Domain
Determine OWL Sublanguage
Change OWL Sublanguage
<<includes>>
Create Primary Reference Ontology
Actors: DBA, OBA
Create Initial Ref Ontology
Revise Primary class Name
Revise Primary Property Name
Set Data Property
Set Property Restriction
Identify Naming Pattern
Apply Pattern To a Primary class
Apply Pattern To Primary Classes
Set Property Characteristic
Validate Base Ontology
Set Relation between Primary classes
Identify Differences between Base Ontology Versions
Enrich Reference Ontology
Actors: DE, OBA
Enrich Reference Ontology
Identify Class Name
Set Object Property Relation
Set Data Property
Set Reference Subclass
Identify Superclass
Create Subclass Relation
Set Property Restriction
Identify Subclasses
Set Reference Superclass
Set Common Properties
Transfer Common Properties
Remove Common Properties
Set Common Sub-properties
Create Superclass Relation
Set Property Characteristic
Set Annotation
Create New Ref Ontology Version
Legend
OBA – Ontology Builder Agent
DBA – Database Administrator
DE – Domain Expert

(a)

Figure 4.3: Decomposition of the high-level use cases


System
Validate Reference Ontology
Actors: DE, OVA
Validate Reference Ontology
Identify Logical Conflicts
Identify Modelling Errors
Resolve Conflicts & Errors
Identify Mapping Query Issues
<<includes>>
Align External Ontology
Actors: DE, OAHA
Align External Ontology
Align Class
Identify Alignments
Align Property
Align Class Hierarchies
Learn Ontology from RDB
Actors: DBA, DE, OLA
Learn Ontology from RDB
Select Main Primary classes & Properties
Find Candidate Subclasses
Find Candidate Superclasses
Find Candidate Class Relations
Reuse External Ontology
Actors: DE, SWCA
Reuse External Ontology
Find External Ontology
Analyze External Ontology
Visualize Ontology
Actors: DBA, DE
Visualize Ontology
View Graph
Filter Graph
Legend
OVA – Ontology Validator Agent
OAHA – Ontology Alignment Helper Agent
OLA – Ontology Learner Agent
SWCA – Semantic Web Crawler Agent
DBA – Database Administrator
DE – Domain Expert

(a)

Figure 4.3: (Cont.) Decomposition of the high-level use cases

the Semantic Web in order to find external ontologies, and analyzes them to gather knowledge about new class hierarchies and relationships. The system presents that information to the DE who decides to include the appropriate classes and relationships in the reference ontology.

In the *Query with Reference Ontology* use case, the actor builds semantic queries using the domain terms from the reference ontology. Next, the system validates, translates, and executes the semantic queries on the RDB. Later, the results are formatted and displayed to the actor.

When the actor invokes the *Visualize Ontology* use case, OBMAS presents the base ontology and reference ontology in a graph and enables the actor to browse specific entities and explore the class hierarchies.

4.1.3 Non-functional Requirements

The non-functional requirements specify the constraints on the structure, features, and services of OBMAS. The first of the two main requirements is the system's independence of the specific RDB-to-RDF translator that it employs. Considering the ongoing research in that area, it is essential to keep OBMAS open to incorporation of more advanced RDB-to-RDF translators as the tools evolve. The second main requirement is to keep the OBMAS architecture compatible with the distributed client-server model of SQAS.

4.1.4 Domain-specific Requirements

The domain-specific requirements for OBMAS specify the details of multiagent systems (MAS) structure, as well as the technologies and constraints in ontology development. The first requirement is that the agents should communicate using a shared ontology model represented in OWL 2 (Hitzler et al., 2012). Second, to keep the choice of methodology flexible, the desired methodology of ontology building should be incorporated into a *planning meta-ontology* which is a plan library for the agents. Third, to enable the agents to reason and communicate about ontology, one must endow them with identical understanding of abstract ontology concepts, such as class or subclass within their belief bases. Fourth, in order to allow semantic queries to refer to the reference ontology concepts, OBMAS should provide a mechanism of *semantic query translation* into the equivalent SQL queries.

4.2 System Structure

The high-level structure of OBMAS is shown in Fig. 4.4. OBMAS has four main components: System Core, Data-Source Mapper, OBMAS RDB, and Graphical User Interface (GUI). System Core includes the agent team and its supporting components. It also provides the access to external services such as WordNet (Miller, 1995). Data-Source Mapper provides automatic extraction of the base ontology from the existing RDB that is commonly known as bootstrapping. OBMAS RDB stores the state of the system and other resources which are persistent across user sessions. GUI provides a friendly interface for user interactions with OBMAS.

The rest of the section explains the major functional components of OBMAS in more detail.

4.2.1 System Core

System Core (Fig. 4.5) is the central component of OBMAS. It contains the agent team and several other subcomponents, discussed next.

System Ontologies is the set of all ontologies in the system, namely the base ontology, reference ontology, and meta-ontology, for each agent. It gives thread-safe access to the latest in-memory copies of the ontologies using the OWL API (Horridge

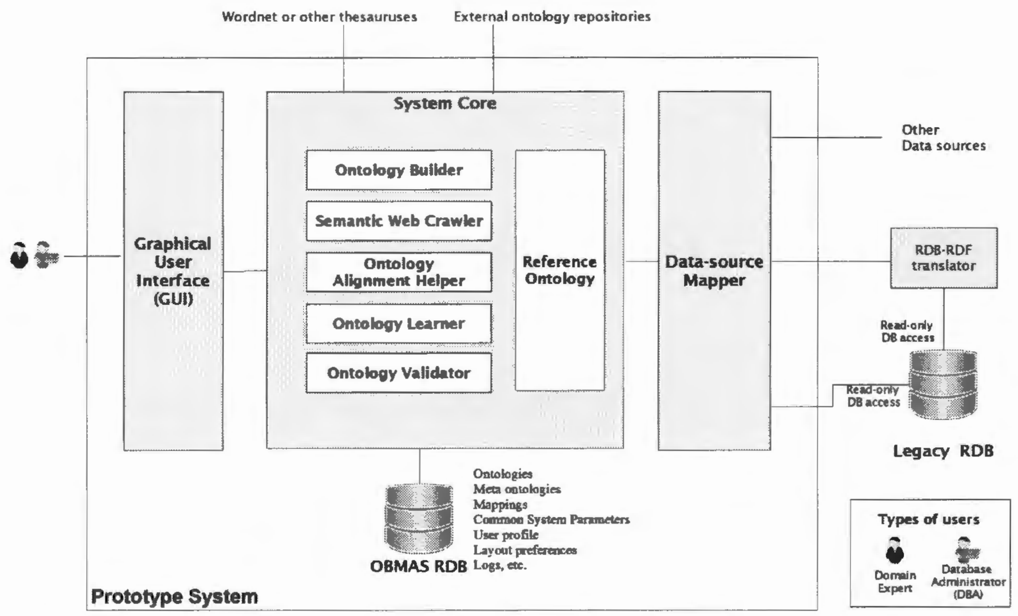


Figure 4.4: The high-level structure of OBMAS and its external relations

and Bechhofer, 2011).

Agent Team is the group of agents which can interact with users and among themselves during different steps of the ontology development process. Ontology development involves multiple distinct and complex tasks such as: building the reference ontology, ontology learning from the underlying RDB, ontology lookup on the Semantic Web, alignment with external ontology, and validation of the reference ontology to identify conflicts and errors. These compound tasks require significant agent reasoning and deliberation along with user interaction. Therefore, it will be reasonable to conceive them as belonging to the agents with individual skill profiles, described next.

In the following paragraphs, I provide short descriptions of roles or objectives of each agent.

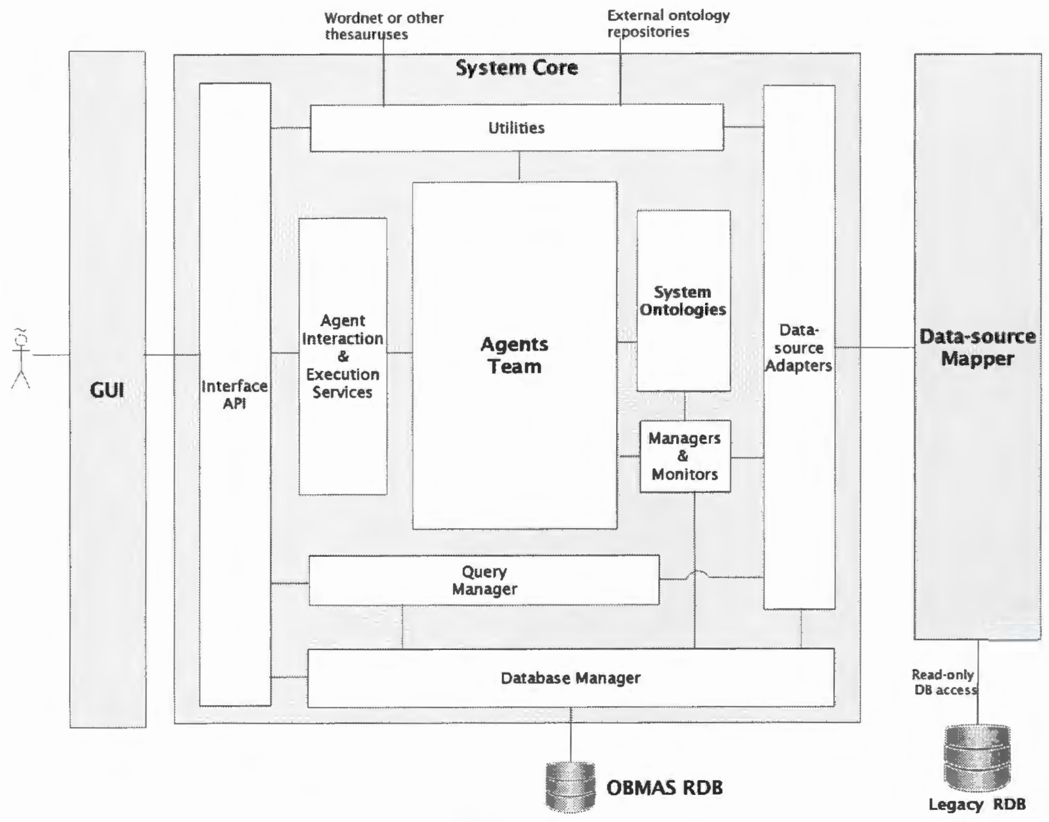


Figure 4.5: System Core

Ontology Builder Agent (OBA) is the chief agent in the OBMAS agent team. Its task is to build the reference ontology in interaction with the Database Administrator (DBA) and Domain Expert. It requests help from the other agents as necessary to fulfill its task. Its roles include: assisting the user in formulating competency questions (Noy et al., 2001) for the reference ontology; handling technical ontology-oriented tasks such as adding or removing classes, properties, and property restrictions; and semantics-oriented tasks such as introducing new classes through generalization or specialization of existing classes. The OBA follows an ontology development methodology, such as Ontology Development 101, in order to assist the user in developing the reference ontology. It communicates with the *Semantic Web Crawler Agent*, *On-*

tology Alignment Helper Agent, and *Ontology Learner Agent* to gain the knowledge of the domain in terms of hierarchies of classes, their properties, and relations between them. The acquired knowledge is used to assist the user in the ontology development process.

Semantic Web Crawler Agent (SWCA) explores the Semantic Web repositories such as Swoogle (Ding et al., 2004), Watson (d'Aquin et al., 2007), and Open ontology repository (Baclawski and Schneider, 2009) for existing known ontologies in the knowledge domain by occasional interaction with the DE. The external ontologies may contain varied degrees of relevant knowledge for the domain. The SWCA assists its human partner in identifying and extracting relevant classes, class hierarchies, properties, and relations from the external ontologies. To this end, the agent employs services from Semantic Web Discovery utilities for the ontology lookup as well as Thesaurus & Dictionary Utilities to identify semantic similarities between domain concepts for knowledge extraction. The acquired knowledge is then shared with the OBA which interacts with the DE to introduce classes or properties in the reference ontology.

Ontology Alignment Helper Agent (OAHA) enables semantic interoperability between the reference ontology and an external ontology by creating alignments between classes and properties using equivalency, and identifies candidate classes for the knowledge domain through an analysis of external ontologies. It interacts with the Domain Expert (DE) to align the classes, properties, and class hierarchies from an external ontology to the classes in the reference ontology. It also analyzes the latest version of the reference ontology to identify new candidate alignments or classes. It shares the learned class hierarchies with the Ontology Builder Agent.

Ontology Learner Agent (OLA) analyzes the data in the legacy RDB, as well as the RDB schema, in order to identify candidate classes for specialization. For an extended analysis, it also applies reverse engineering or other ontology learning techniques (Spanos et al., 2012) to extract the underlying ER (Entity Relationship) model (Calvanese et al., 1999) for the RDB, in order to identify implicit domain knowledge from the conceptual schema of the model. The agent shares the acquired knowledge about the model with the Ontology Builder Agent which interacts with the DE to introduce classes or properties in the reference ontology.

Ontology Validator Agent (OVA) keeps track of updates in the current version of the base and reference ontologies, and initiates the ontology validation process periodically or whenever desired (for example, before creating a new version of the reference ontology). It alerts the Ontology Builder Agent to contradictions or modelling errors in the reference ontology. It can also load plans from the planning meta-ontology and determine a plan for handling inconsistencies in the ontology. Alternatively, it can assist the DE to resolve the problems in the ontology.

Agent Structure is illustrated in Fig. 4.6. Each agent encapsulates five components: *belief loader*, which reads system ontologies, translates their properties into the agent beliefs, and loads them into the agent's belief base; *event generator*, which creates events for the reasoner based on incoming messages or changes in the belief base; *reasoner*, which determines what needs to be done in response to an event, including possible commitments to goals that activate the plan selector; *plan loader*, which loads all available plans for the agent in the plans data structure; and *plan selector*, that chooses a plan from the list of applicable plans or removes an achieved goal from list of goals. In addition, the agent contains three data structures: *belief*

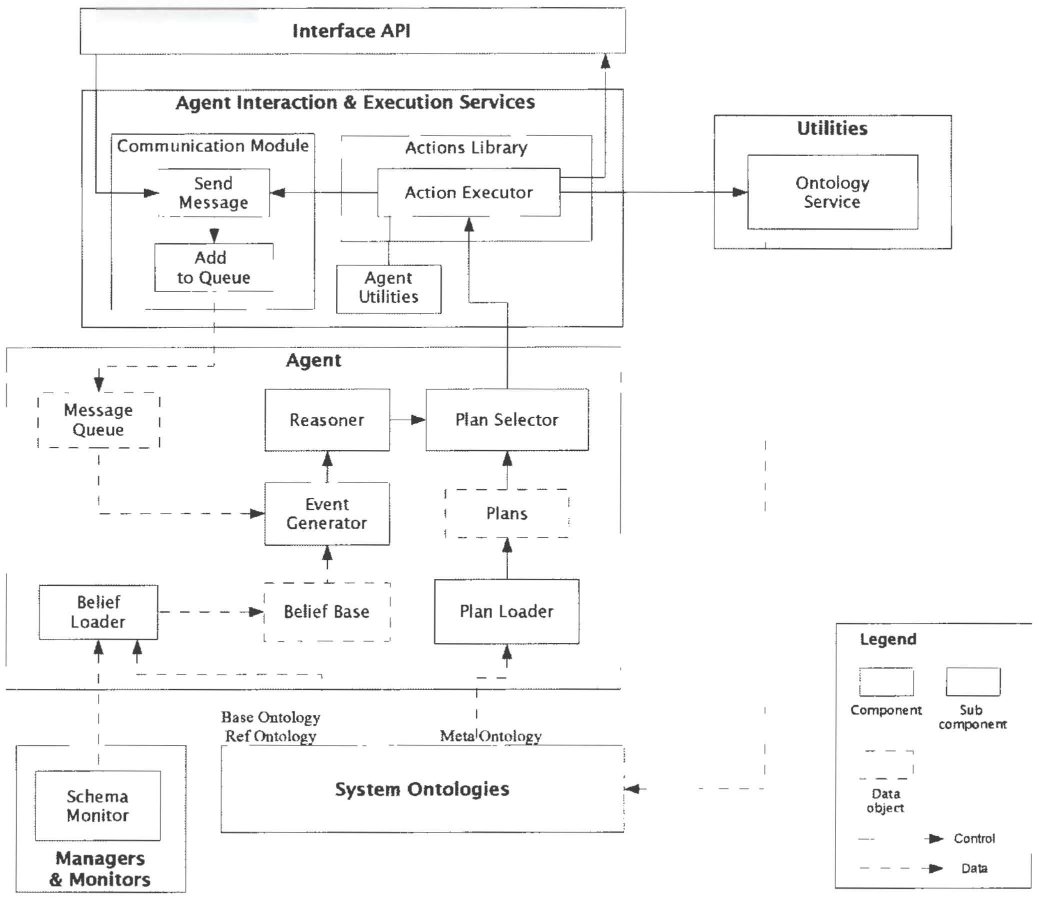


Figure 4.6: The internal structure of an agent

base, which contains the beliefs of the agent about the state of the environment (i.e., ontology); *plans*, which stores the agent plans retrieved from the plan library; and *message queue*, which holds the messages received from other agents or the users.

Agent Interaction & Execution Services component provides a set of three services which are shared between the agents: *communication module*, which forwards asynchronous messages from sender agents to message queues of receiving agents; *actions library*, which provides a range of actions to be selected and executed by the agents such as sending an agent request or response to the interface API, sending or broadcasting a message to agents, or invoking ontology service utilities to modify the ontology; *agent utilities*, which provides a set of general utility functions to read and format action parameters before execution of an agent action.

Interface API facilitates bidirectional interactions between GUI and components of System Core. It employs the publish-subscribe mechanism for services. It offers *UI services*, which deliver UI requests and responses from publishers to subscribers, and *agent services*, which deliver agent requests and responses from publishers to subscribers. GUI publishes to the UI services and subscribes to the agent services while the agents do the reverse. The interface API also offers unidirectional *system services* without subscription which any consumers can use for common system functions.

Utilities five types of services: *Thesaurus & Dictionary Utilities* provide connect and fetch services to find related words from WordNet and other external dictionaries and thesauri; *Ontology Service Utilities* provide technical ontology development and maintenance services; *Semantic Web Discovery Utilities* help find relevant existing domain ontologies from local or external repositories; *Ontology Alignment Utilities* provide technical services for the ontology alignment between classes or properties of an external ontology and the reference ontology; and *Repository for External Ontologies*, maintains a list of external domain ontologies along with their locations and provenance information.

Managers and Monitors provide the following services: *Ontology & Mappings Manager* maintains versions of the reference ontology, base ontology, and their corresponding RDB-to-RDF mappings files for the other system components; *Meta-ontology Manager* manages planning meta-ontologies for the agents and provides services to translate them to agent plans in AgentSpeak; *Schema Monitor* periodically monitors the RDB schema to detect changes, identifies the differences between schema versions, and notifies the Ontology Builder Agent (OBA); *Query Manager* verifies SPARQL queries against the reference ontology and connects to *data-source adapter* to execute SPARQL queries on the RDB; *Database Manager* provides a service to check valid connection to RDB and maintains a list of standard queries for handling data in OBMAS RDB; and *Data-source Adapters* provide access to services of Data-Source Mapper for bootstrapping base ontology, executing semantic query, materializing the reference ontology, and accessing the RDB schema.

4.2.2 Data-Source Mapper

This component presently provides RDB-related services; in principle, it is envisioned to also connect with various file-based or streaming data sources. In the current architecture, it serves as a bridge between OBMAS and an external RDB-RDF translator such as Ontop (Rodríguez-Muro et al., 2013). It provides four interfaces: *Bootstrapper* interfaces to an RDB-RDF translator to automatically generate base ontology and associated mappings from the RDB schema; *SPARQL Query Responder* interfaces to *SPARQL query execution service* of an RDB-RDF translator; *Materializer* interfaces to an RDB-RDF translator to load instances of classes, i.e., the data from the RDB into the reference ontology; and *Schema Loader* reads the RDB schema through direct read-only access, monitors the schema modification events through binary log processing, informs the schema adapter which in turn notifies the schema monitor.

4.2.3 Graphical User Interface (GUI)

GUI enables a user to invoke any of the use cases described in the functional requirements section. Fig. 4.7 illustrates its internal structure.

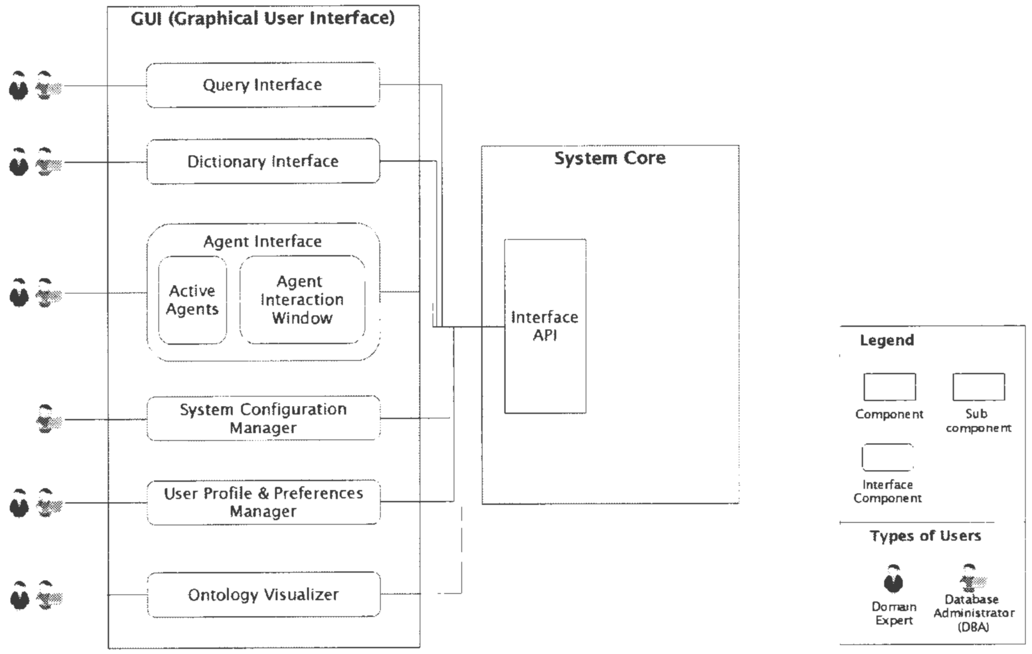


Figure 4.7: Graphical User Interface (GUI)

The Database Administrator (DBA) can access all components of GUI; and the Domain Expert (DE) can access all components except the *system configuration manager*. GUI provides an *agent interface* that lists all active agents in the system, and allows the user to open an *agent interaction window* for each selected agent. The window contains handlers for agent requests and responses. For system configuration, GUI enables the DBA to set up connections to the RDB and OBMAS RDB, and manage the users through the system configuration manager. With the *query interface*, the DBA or DE can build and execute SPARQL queries for information retrieval from the RDB using classes and properties in the reference ontology. The *dictionary* *interface* provides access to WordNet and other dictionaries and thesauri. Users can save their profile information and system preferences using the *user profile & preferences manager*. The *ontology visualizer* renders a selected version of an ontology as a graph. The ontology graph can also be filtered to display a limited set of classes.

4.2.4 OBMAS RDB

OBMAS RDB is used to preserve the state of OBMAS in a database for future user sessions. It also stores resources that persistent across multiple user sessions. It is designed to store the meta-information related to objects such as the base ontology, RDB, external ontology, layout preferences, logs, mappings, meta-ontology, reference ontology, system parameters, user profile, and user roles.

4.3 System Behaviour

In this section, I explain user-agent interaction and communication between agents.

An agent interacts with the user within the context of a dialogue, illustrated in Fig. 4.8. A component such as *GUI* or *agent* executes actions in the direction of control flow in the activity diagram; while the event flow transfers control from the source to destination. A new dialogue is initiated when the user sends a request or the agent decides to perform the next step prescribed by the ontology development methodology. The agent then deliberates, commits to a goal, and executes a plan associated with the goal. A dialogue involves four kinds of messages: *User Request*, *Agent Request*, *User Response*, and *Agent Response*. In each round of reasoning in

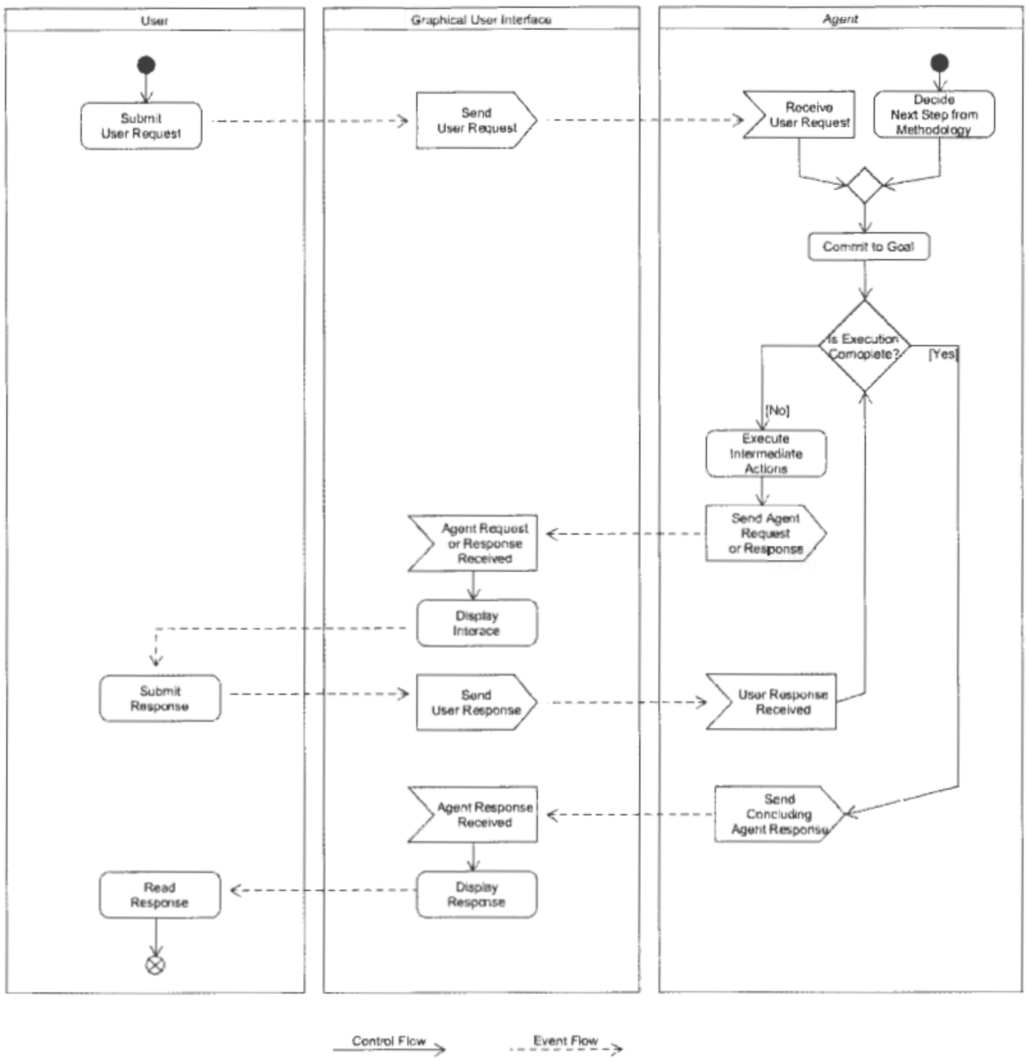


Figure 4.8: The user-agent interaction

the dialogue, the agent commits to new subgoals, or determines and executes a set of intermediate actions based on the user response; in the absence of failure, this continues until either the goal is achieved, or it becomes unnecessary, or it turns unachievable within current conditions. The subgoals are pursued in the same dialogue for simpler goal tracking. At the end of each round, the agent waits for a user response to start a new round.

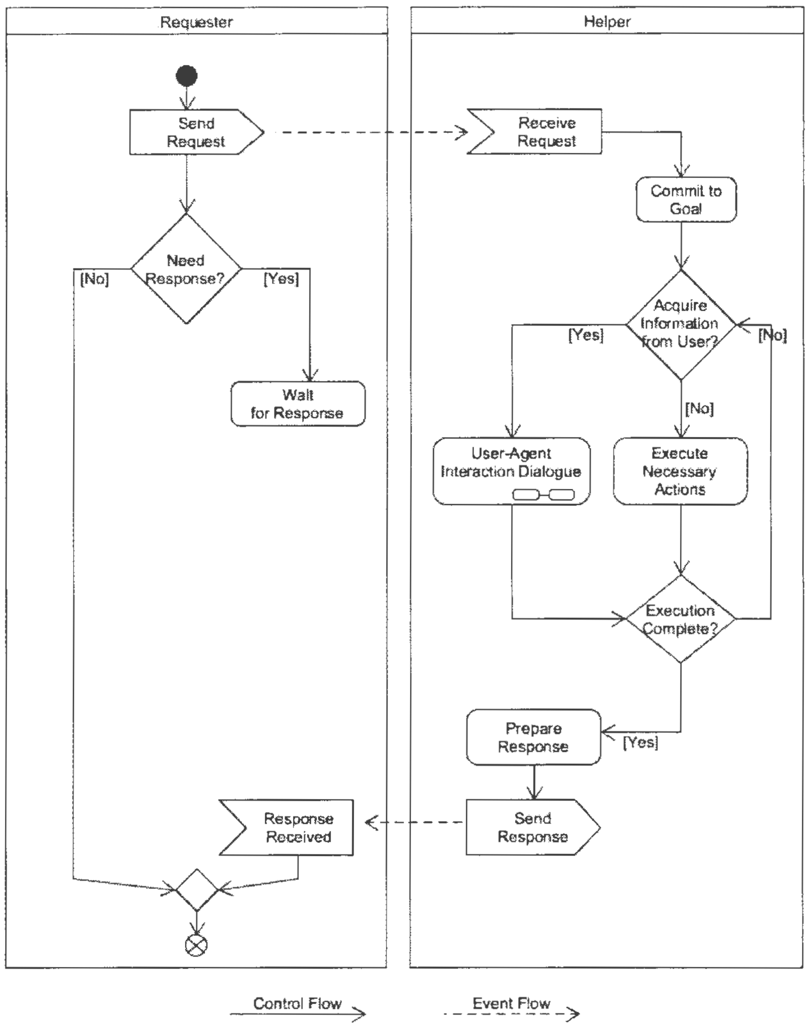


Figure 4.9: The agent communication cycle

Fig. 4.9 illustrates a communication cycle between two agents; here, the *requester* agent initiates communication, and the *helper* agent executes actions or provides information upon request. The helper generates events for the changes in its belief base which may be caused by the content of an incoming message. Each event is stored in the list of pending events, and selected for further processing in order to identify and commit to a goal during subsequent reasoning cycles. If additional information is required in order to achieve the goal, the helper initiates a dialogue with the user (as shown in Fig. 4.8); otherwise, it determines and executes a plan including initiation of communication with other agents as necessary. In the absence of failure, the process repeats until either the goal is achieved, or it becomes unnecessary, or it turns unachievable. The communication cycle ends with the helper sending a response message to the requester.

A request type corresponds to one of the following KQML performatives (Finin et al., 1994): *achieve*, *unachieve*, *tell*, *untell*, *tellHow*, *untellHow*, *askHow*, *askOne*, and *askAll*. The agents use standard ontological notions when communicating about entities in the ontology. For example, the requester can ask the helper about semantic equivalence of two classes:

```
(askOne
  :content equivalentClass(class(product),class(merchandise))
  :receiver helper
  :language AgentSpeak
  :ontology reference-ontology
)
```

4.4 The Prototype Implementation

The primary objective of the prototype has been to demonstrate how the intelligent agents can interact with the user and other agents in order to build the reference ontology using the Ontology Development 101 methodology. To this end, the following components of OBMAS have been designed and implemented: *Ontology Builder Agent*, *Graphical User Interface*, *OBMAS RDB*, *Interface API*, *Thesaurus & Dictionary Utilities*, *Ontology Service Utilities*, *Ontology & Mappings Manager*, *Meta-ontology Manager*, *Data-source Mapper*, *Database Manager*, and *Query Manager*. Fig. 4.10 illustrates a standard GUI for the OBA.

Some components of OBMAS are not present in the prototype. They include: *Semantic Web Crawler Agent*, *Semantic Web Discovery Utilities*, *Repository for External Ontologies*, *Ontology Alignment Helper Agent*, *Ontology Alignment Utilities*, *Ontology Validator Agent*, *Ontology Learner Agent*, and *Schema Monitor*; their functionalities were not essential for the current prototype. The ontology editor such as Protégé can be used for the features currently not present in the prototype such as creating complex class hierarchies and reclassifying them.

The prototype support for each step of Ontology Development 101 is described next.

- *Determine domain and scope of the ontology.* The information about the domain and scope of the ontology is presently provided by the user through GUI and that knowledge is recorded in OBMAS RDB. When the agents such as SWCA, OAHA, and OLA are implemented, the awareness about the domain will help them in searching and acquiring knowledge from external resources.

- *Consider reusing existing ontologies.* Presently, the user identifies domain knowledge from external ontologies and provides the information to the OBA in order to incorporate classes and properties in the reference ontology. When the SWCA is built, it will perform this function in interaction with the user. Further research and development is required in order to equip the agents with an ability to reuse knowledge from external ontologies, based on concepts such as semantic similarity service (Han et al., 2013).

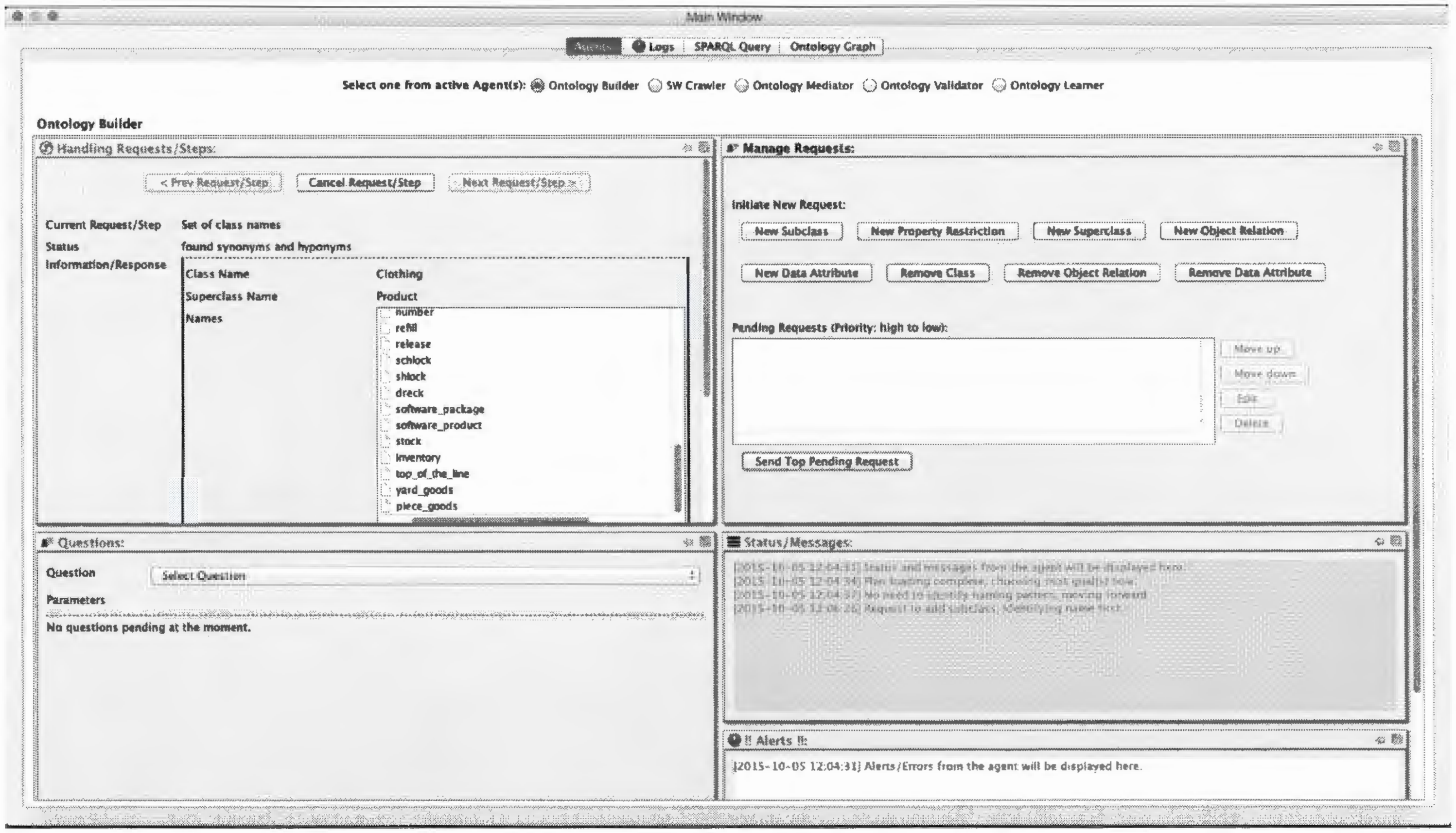
- *Enumerate important terms in the ontology.* A set of domain classes, retrieved from the RDB schema, already exists in the base ontology. The DBA or DE can determine terms for subclasses, superclasses, and properties, that are dependent on and reducible to the primary classes.

- *Define classes and class hierarchies.* The features such as identifying class name, introducing subclasses, adding superclasses, and generating RDB-to-RDF mappings with mapping rules are implemented and demonstrated using the prototype.

- *Define the properties of the class.* During the reference ontology development, the user introduces new data properties, that are replicated from the existing data properties of other classes through a join of underlying tables in order to support querying. The prototype then creates the data property, and generates corresponding RDB-to-RDF mappings using mapping rules. The feature to introduce object properties and associated mappings can be similarly developed.

- *Define the facets of the slots.* In terms of OWL, slots correspond to properties for classes, and facets are restrictions on the slots. The user introduces property restriction, and the prototype creates it in the reference ontology including the associated RDB-to-RDF mapping. The prototype supports different restrictions such as *allValuesFrom*, *hasValue*, *minExclusive*, *minInclusive*, *maxExclusive*, and *minInclusive*.

- *Create instances.* OBMAS does not need populated instances in the reference ontology because they are generated at runtime using on-demand SQL queries from the RDB.



Select one from active Agent(s): [✓ Ontology Builder] SW Crawler Ontology Mediator Ontology Validator Ontology Learner

Ontology Builder

Current Request/Step	Set of class names
Status	found synonyms and hyponyms
Information/Response	

Class Name	Clothing
Superclass Name	Product
Names	number refill release schlock shlock dreck software_package software_product stock inventory top_of_the_line yard_goods piece_goods

No questions pending at the moment.

```
[2015-10-05 12:04:31] Status and messages from the agent will be displayed here.
[2015-10-05 12:04:34] Plan loading complete, choosing most quality flow.
[2015-10-05 12:04:37] No need to identify naming pattern, moving forward
[2015-10-05 12:06:26] Request to add subclass, identifying name first.
```

```
[2015-10-05 12:04:31] Alerts/Errors from the agent will be displayed here.
```

Figure 4.10: Graphical User Interface (GUI) for the Ontology Builder Agent (OBA)

Chapter 5

Designing Ontology-building Agents

In this chapter, I describe three research challenges identified during the course of this thesis, and provide explanations of how I have resolved them. The first two challenges deal with fundamental questions: how to design meta-ontologies that enable the agents to understand ontological notions and follow an ontology-building methodology; and how to translate semantic queries, formulated in terms of reference ontology concepts that may not have direct equivalents within the given RDB schema, into SQL queries for the RDB. The third research problem is practical and concerns the selection of tools that appear the most appropriate for the prototype implementation. To the best of my knowledge, complete solutions to the first two research challenges do not exist in the current literature. Relying in part on existing results and techniques, I introduce my own solutions in sections 5.1 and 5.2. The choices for tools, which impact the current prototype design, are discussed in section 5.3. The final section

(5.4) briefly summarizes the current implementation status.

5.1 Designing Meta-ontologies

The OBMAS design includes two types of knowledge representation for the ontology design domain. The *model meta-ontology* represents the ontological notions, such as classes and properties, that support the agent's reasoning about ontology, and also allow the agent to communicate with other agents using standard ontological terms. The *planning meta-ontology* represents a particular ontology development methodology in terms of agent plans; it allows the agent to deliberate on selection and execution of plans (from the supplied library) that best fit its tasks and its design objectives. These two meta-ontologies are described in subsections 5.1.1 and 5.1.2 respectively.

5.1.1 The Model Meta-ontology

The purpose of model meta-ontology is to enable the agents that are engaged in the building of reference ontology for a specific knowledge domain (we use e-commerce as the running example throughout this thesis) to reason and communicate about ontological concepts. The agents need two kinds of knowledge. First, they need to know the abstract ontological concepts, such as 'class' and 'property', and their mutual relationships, e.g., the fact that a class definition can include properties. The term *metamodelling* as defined by Motik (2007) for the Semantic Web concepts describes the design of the abstract ontological knowledge model – the *metamodel*. Second, the agents must be able to apply the abstract concepts to the concrete knowledge domain for which they are building the reference ontology; for instance, in the case of e-commerce domain, they need to know that 'product' is a class name. Thus the metamodel concepts need to be connected to the reference ontology concepts to produce the *concrete metamodel* for the knowledge domain. The concrete metamodel contains the ontological knowledge that the agents need for reasoning and communication in the ontology-building process. One should note that the concrete metamodel requires dynamic updating as new concepts are added to the reference ontology.

To enable the agents to reason and communicate about ontology, one must also decide how to impart to them the knowledge of the concrete metamodel. One possibility is to design specialized ontology-building agents with hard-coded ability to deal with abstract ontological concepts. We adopt a more flexible approach, in which the agents are enabled for practical reasoning of BDI type, and the knowledge of concrete metamodel becomes a part of their belief base. This approach requires a translation of the original concrete metamodel representation into the representation of agent beliefs in the particular BDI platform. The agents can then use their knowledge of the ontology-building domain in the same way as any other domain knowledge.

The process of creating and maintaining a model meta-ontology for a concrete knowledge domain is shown in Fig. 5.1. It starts with two components: a metamodel representing the abstract ontological concepts; and initial version of the reference ontology, representing the domain-specific concepts. These components must share the same ontology representation language, which in our case is OWL 2. Consequently, we use the OWL 2 Metamodel (Brockmans et al., 2008), which I upgraded to full compliance with the OWL 2 Specification (Hitzler et al., 2012), and build the reference ontology using the OWL 2 API, which is consistent with the metamodel. The two components are then integrated, through a process called *concretization*,

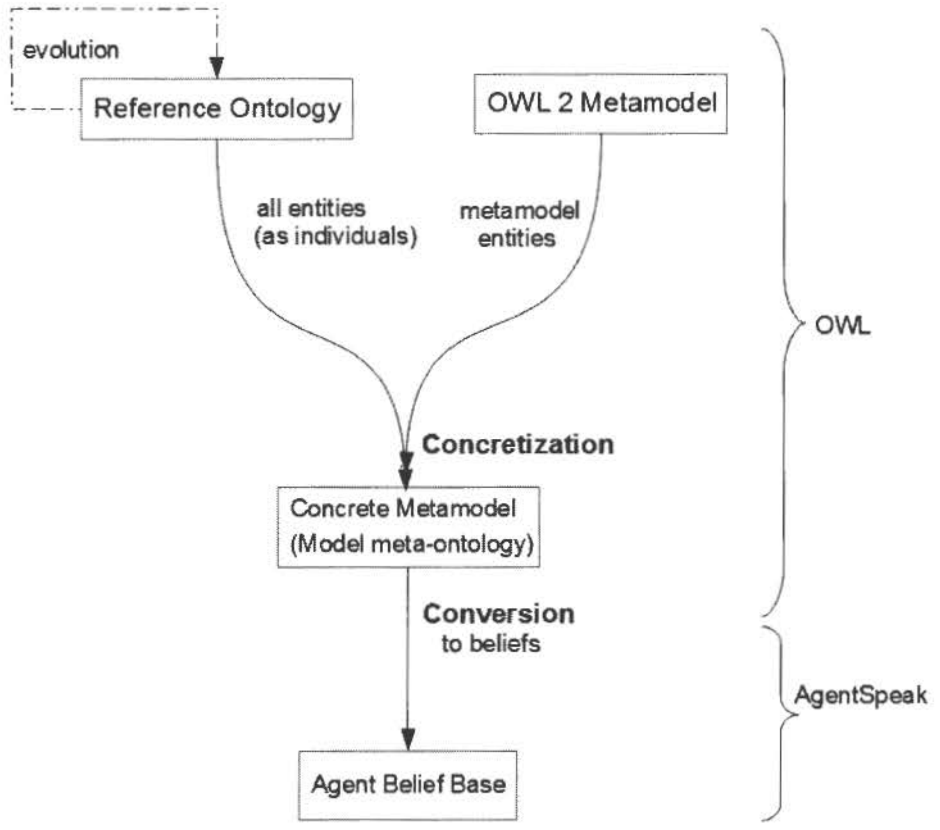


Figure 5.1: The process of developing a model meta-ontology

into a concrete metamodel for the domain, which is subsequently converted to agent beliefs in a suitable BDI formalism, in this case AgentSpeak (Rao, 1996) on Jason platform (Bordini et al., 2007). The initial reference ontology in our case is created by bootstrapping from an existing RDB. It is gradually enhanced with higher-level domain-specific concepts, with the concretization and conversion steps repeated each time as necessary.

The important upgrades to the OWL 2 Metamodel included the following refinements: adding cardinality restrictions on data properties, replacing URI (Universal Resource Identifier) with IRI (Internationalized Resource Identifier), setting property restrictions on DataSomeValues and DataAllValuesFrom classes, and a change in SubObjectPropertyOf axiom.

The concretization process involves metamodelling (Motik, 2007) of entities such as classes and properties in the concrete reference ontology. During the metamodelling, the classes in the OWL 2 Metamodel are instantiated with corresponding entities as individuals in a combined ontology, called the concrete metamodel, which is also our model meta-ontology. The updates in the reference ontology during its evolution are also applied to the concrete metamodel in order to reflect the latest state of the reference ontology and subsequently converted to agent beliefs.

The proposed converter is an upgraded version of the JASDL (Jason AgentSpeakDescriptionLogic) translator (Klapiscak and Bordini, 2009); it forms a part of the belief loader component of the agent. In OBMAS, the agents employ the resultant beliefs from the conversion process in order to deliberate using the meta-ontological knowledge and communicate with other agents using standard ontological terms.

5.1.2 The Planning Meta-ontology

A planning meta-ontology has been designed and implemented as a hierarchical model of ontologies which represent the agent plans at different levels of abstraction. Here, the agent plans correspond to various tasks that depend on the agent's role and skill profile. In this subsection, I outline my design approach, present the hierarchical model structure, and illustrate it with an example.

In general, practical reasoning in agents built according to the BDI model consists of two phases: deliberation, in which an agent forms an intention, and planning, in which the agent determines its action plan to achieve the intention. In most BDI-based systems, agents do not construct plans at runtime but select them from a library of predefined plans (described in subsection 2.3.1). Our current approach also follows that established practice. The planning meta-ontology is a library of plans for the agents.

In our model, an intention corresponds to a composite goal, which can be recursively resolved into subgoals, leading eventually to primitive goals. Composite goals are resolved by activating plans from the library whose preconditions are satisfied by the agent's beliefs, while the primitive goals are executed directly by the agent, as determined by the agent's design. One should note that the agent's beliefs reflect the perceived state of the environment, which in this case is the reference ontology being built, and also contain ontological knowledge, imported as a part of model meta-ontology. Our definitions of aforementioned terms are consistent with the properties (proposed by Erol et al. (1994) and represented by Freitas et al. (2014)) of the Hierarchical Task Network (HTN) planning systems. Here, composite goals correspond to goal tasks and compound tasks while primitive goals correspond to the primitive tasks of HTN. Similarly, the preconditions correspond to constraints to control ordering of tasks and the postconditions [1] correspond to other compound or primitive tasks. The temporal ordering of postconditions can be specified by numerical annotation properties. This solution does not support fully general partial ordering but is satisfactory for prototyping purposes.

In the current design of OBMAS, the agent's role, such as OBA or OLA, determines its skill profile, in terms of both primitive goals that it knows how to execute,

[1]The term postcondition is somewhat misleading because it refers to a partially-ordered set of primitive or composite goals, but is retained here for compatibility with the work of Freitas et al. (2014)

and high-level composite goals that it tends to formulate as initial intentions in its deliberation process. The latter aspect makes the agent 'eager' to perform activities in its role's purview. For instance, the OBA may pick up pending ontology-building tasks whenever it has some idle time, while the SWCA might proactively look for new resources on the Semantic Web. The planning meta-ontology reflects the methodology which the agents follow in pursuing their composite goals.

The selection of a particular plan from the planning meta-ontology is also influenced by the type of event that leads to invocation of the plan. In the terminology of AgentSpeak, the events are classified as test goals (questions from user or other agents), achievement goals (to which the agent commits), and triggering events (such as adding a new belief to agent's belief base).

The hierarchical model of planning meta-ontology

The starting point in our model is the idea of planning ontology in the form of a plan library, introduced by Freitas et al. (2014) (discussed in subsection 2.3.3); their library is developed in OWL 2 and automatically translated into agent plans in AgentSpeak. We introduce a hierarchical model (shown in Fig. 5.2) with four levels of abstraction and three event types, as discussed above. Our model also relies on the model meta-ontology in order to identify the state of the ontology for selecting applicable plans. For each level, the model extends the planning meta-ontology by instantiating classes with different individuals corresponding to the level scope. For each agent, this four-level planning meta-ontology represents its procedural know-how of building an ontology using the desired methodology.

The four levels of planning meta-ontologies and their significance are summarized as follows.

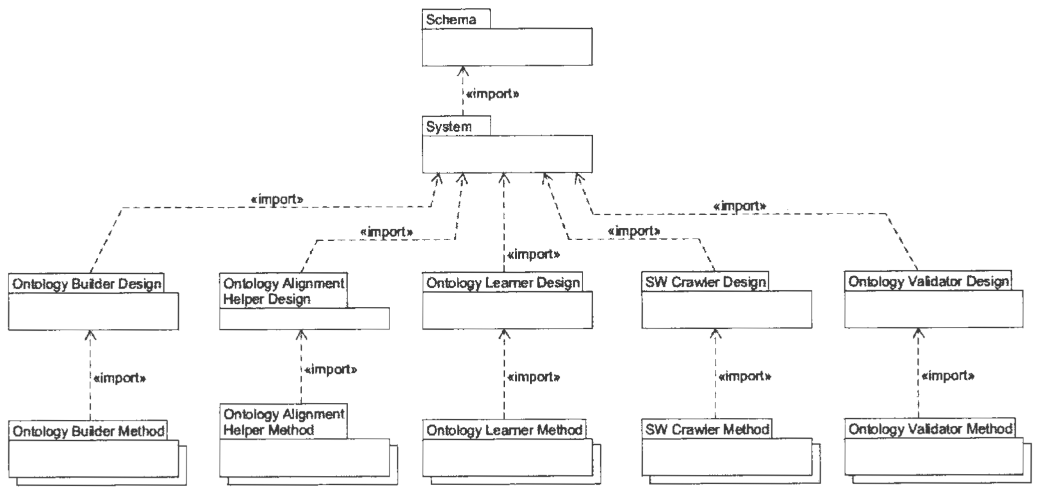
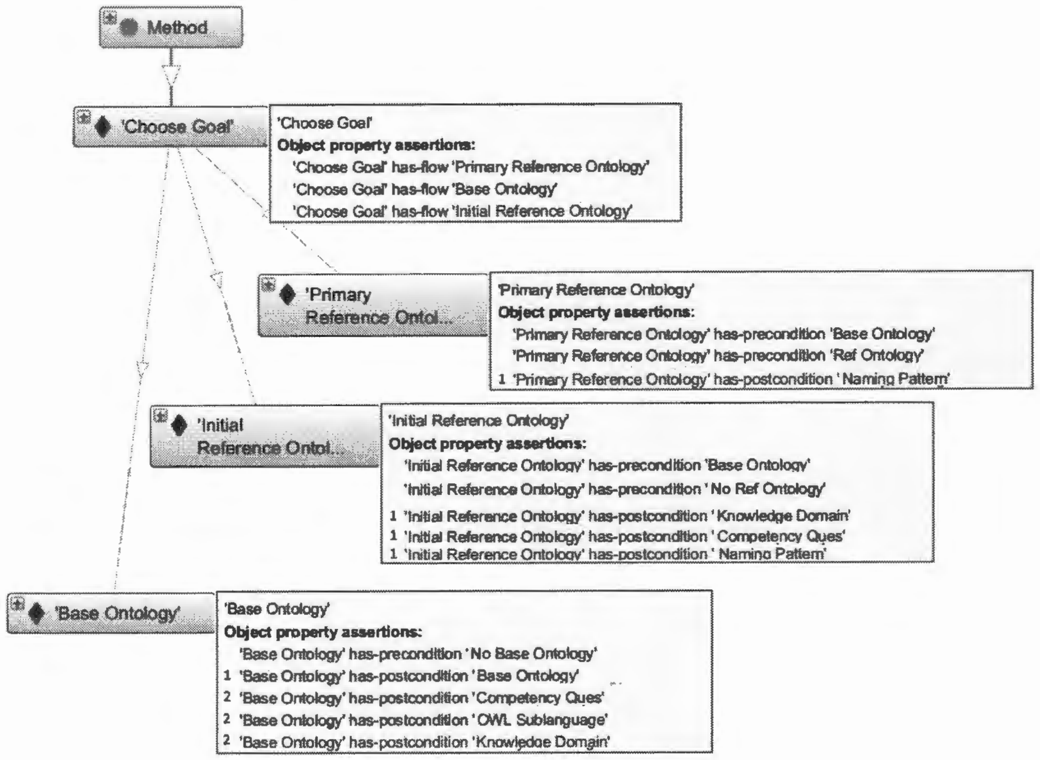


Figure 5.2: The hierarchical model of planning meta-ontology

The top level in the hierarchy is the *Schema*, which represents the common classes and properties for the planning meta-ontology without any instances, i.e., individuals; it has been refined with a new data property called event-type reflecting different triggering events. The *System* level inherits (or 'imports', in OWL terminology) the schema-level ontology and extends it with instances of primitive goals, composite goals, beliefs, and variables that are common to all agents. The *Agent-specific-Design* level imports the system-level planning ontology and extends it with agent-specific instances of the primitive goals, composite goals, beliefs, and variables, that are independent of development methodology. The *Agent-specific-Method* level imports the agent-specific-design-level ontology and extends it with agent-specific instances of the composite goals and plans that are specific to the desired ontology development methodology.

Our model description employs the terminology of Freitas et al. (2014) which corresponds to the AgentSpeak terminologies as follows: *Operator* corresponds to a primitive goal; *Method* corresponds to a composite goal which is decomposed further into other composite or primitive goals; *Method Flow* corresponds to a plan to achieve a goal (method); *Predicate* corresponds to agent belief; and *Parameter* corresponds to a variable for a goal or belief.



'Choose Goal'
Object property assertions:
'Choose Goal' has-flow 'Primary Reference Ontology'
'Choose Goal' has-flow 'Base Ontology'
'Choose Goal' has-flow 'Initial Reference Ontology'

'Primary Reference Ontology'
Object property assertions:
'Primary Reference Ontology' has-precondition 'Base Ontology'
'Primary Reference Ontology' has-precondition 'Ref Ontology'
1 'Primary Reference Ontology' has-postcondition 'Naming Pattern'

'Initial Reference Ontology'
Object property assertions:
'Initial Reference Ontology' has-precondition 'Base Ontology'
'Initial Reference Ontology' has-precondition 'No Ref Ontology'
1 'Initial Reference Ontology' has-postcondition 'Knowledge Domain'
1 'Initial Reference Ontology' has-postcondition 'Competency Ques'
1 'Initial Reference Ontology' has-postcondition 'Naming Pattern'

'Base Ontology'
Object property assertions:
'Base Ontology' has-precondition 'No Base Ontology'
1 'Base Ontology' has-postcondition 'Base Ontology'
2 'Base Ontology' has-postcondition 'Competency Ques'
2 'Base Ontology' has-postcondition 'OWL Sublanguage'
2 'Base Ontology' has-postcondition 'Knowledge Domain'

Figure 5.3: Available plans for the Choose Goal Method (point of entry)

Fig. 5.3 illustrates a fragment of the planning meta-ontology for the Ontology-Builder Agent (OBA) representing the initial steps of the Ontology Development 101 (Noy et al., 2001) methodology. The *Choose Goal* is a composite goal which serves as an entry point for the agent to identify next goal and associated plan. Depending upon the state of the world identified using the preconditions, the agent commits to one of the subgoals from the following: *Base Ontology*, which represents bootstrapped ontology from the legacy RDB; *Initial Reference Ontology*, which represents the initial state of reference ontology with common properties and references to standard ontolo-

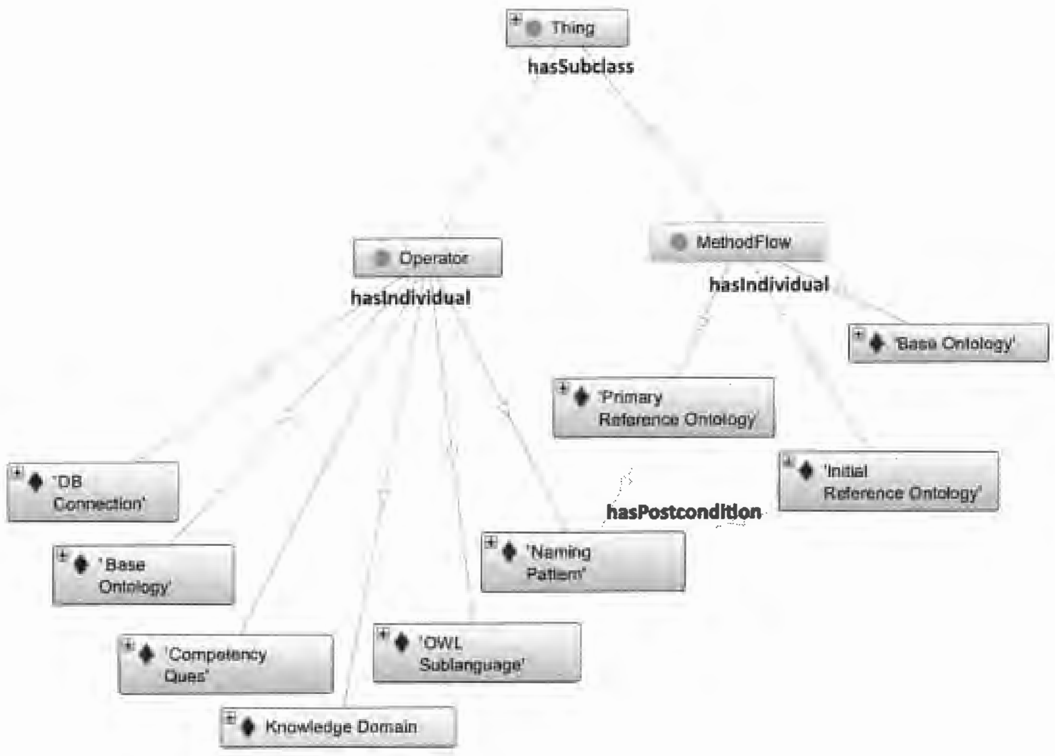


Figure 5.4: A fragment of the planning meta-ontology

gies; and *Primary Reference Ontology*, which represents the reference ontology with correct relations between the primary classes of the base ontology. The preconditions and postconditions for each plan have been shown in Fig. 5.3.

Fig. 5.4 illustrates a fragment of the planning meta-ontology based on which the OBA pursues the *Naming Pattern* goal in order to identify and apply a naming pattern after building the Initial Reference Ontology. This planning meta-ontology has been successfully used to support *determine domain and scope of the ontology* step of the Ontology Development 101 methodology (Noy et al., 2001). It can be similarly extended to support other steps in the methodology. Other agents get involved into the ontology development as and when determined from their planning meta-ontologies from the same methodology. For example, the Semantic Web Crawler Agent (SWCA) initiates the process to explore the Semantic Web after the knowledge domain of the legacy RDB is determined.

5.2 Automated RDB-RDF Mapping Generation

We now address the problem of translating semantic queries over an ontology that was created by bootstrapping from an RDB and enhanced with additional abstractions into an SQL query in the underlying RDB. In order to translate semantic queries, an RDB-RDF translator such as Ontop (Rodríguez-Muro et al., 2013) relies on a set of RDB-to-RDF[2] mappings. Each mapping establishes a correspondence between an element of the ontology and an SQL query in the underlying RDB. During the bootstrapping process, the mappings corresponding to the base ontology elements are created automatically by the translator. In the present OBDA systems, the mappings for higher-level abstractions are constructed manually in an ontology editor such as Protégé.

In our approach, the OBA automatically generates those mappings while creating entities in the reference ontology. The agent relies on a set of *mapping rules* that correspond to specific operations in the construction of higher-level abstractions in the reference ontology. The agent invokes one or more rules when constructing a mapping for a particular higher-level abstraction such as subclass or superclass. Whenever a new abstraction is added to the reference ontology, it can immediately be used in formulating and executing semantic queries.

The process of semantic query translation involves the analysis of the semantic

[2]The term RDB-to-RDF is used here due to its traditional usage in the literature, but in the OBDA context one actually translates RDF-oriented semantic queries into RDB-oriented SQL queries.

query, the matching of ontology-level to RDB-level concepts, and the generation of SQL query. For instance, the primary classes translate directly to the corresponding RDB tables, from which they were originally constructed. The *non-primary* classes (i.e, subclasses and superclasses that were built by the OBA) have no directly matching RDB tables; their correspondence to RDB-level concepts is captured in the mappings that the OBA generated while building them. Apart form class mappings, the translation process also requires mappings for the data properties and object properties in the reference ontology that were not inherited from the base ontology.

We discuss the mapping rules in the subsections below. To simply the representation of the ontology fragments in this section, we use Manchester syntax (Horridge and Patel-Schneider, 2012).

5.2.1 New Subclass

This rule, in association with the mapping rule for property restriction (explained next), generates the mapping for a new subclass. In the first step, the *new subclass* rule creates a new mapping which inherits the IRI for instances as well as the SQL query from the mapping of the parent class. In the second step, the *new property restriction* rule extends the SQL query in the mapping with data filters for the property restrictions that characterize the subclass.

The mapping rule for new subclass is shown in Fig. 5.5. It is expressed in pseudocode based on AgentSpeak (Rao, 1996), as implemented in the Jason platform (Bordini et al., 2007).

The mapping rule creates a new mapping for a `SubClassType` from an existing

```
+!newSubClassMapping(ParentClassType, SubClassType) <-
    mapping(ParentMapID, targetRDF(ParentIRI, "a", ParentClassType), ParentSQL);
    SubClassMapID = actionsLib.generateNewMappingId(SubClassType);
    +mapping(SubClassMapID, targetRDF(ParentIRI, "a", SubClassType), ParentSQL).
```

Figure 5.5: The mapping rule for New Subclass

mapping for its parent `ParentClassType`. The line

```
+!newSubClassMapping(ParentClassType, SubClassType) <-
```

introduces (with `!`) the goal to create the mapping. The activation of this goal (indicated by `+`) results in execution of the plan that follows after `<-`. The line

```
mapping(ParentMapID, targetRDF(ParentIRI, "a", ParentClassType), ParentSQL);
```

retrieves the mapping that matches the `ParentClassType`. Within it, `ParentMapID` uniquely identifies the mapping; `ParentIRI` represents the IRI template for instances of the parent class; `"a"` is a shorthand for RDF type predicate; `ParentClassType` identifies the parent class; and `ParentSQL` is the SQL query for the parent class. The next line invokes a library routine to create the new `SubClassMapID`:

```
SubClassMapID = actionsLib.generateNewMappingId(SubClassType);
```

Finally, the new mapping is generated and added (with `+`) to the agent's belief base:

```
+mapping(SubClassMapID, targetRDF(ParentIRI, "a", SubClassType), ParentSQL).
```

For example, the new subclass rule can be invoked as follows to create the mapping for the subclass *Product of Canada* of parent class *Product*:

```
!newSubClassMapping ("http://www.unbc.ca/.../ref/product",
                     "http://www.unbc.ca/.../ref/product\_of\_canada")
```

The generation of this subclass mapping will be completed with the new property restriction rule discussed next.

5.2.2 New Property Restriction

This rule extends the mapping of a class for a new property restriction on the class. A property restriction is a constraint that specifies allowed values for a data property of the class. A data property of a class in the reference ontology is always reducible to a column in the underlying RDB, even if the class is non-primary. The mapping rule identifies the underlying RDB column for the data property, and updates the existing SQL query in the mapping for the class with a filter condition on the column.

The mapping rule is shown in Fig. 5.6. The property restriction AllValuesFrom prescribes a set of allowed values for the property, while HasValue prescribes a specific value (Hitzler et al., 2012). The remaining cardinality (i.e., number of values) restrictions supported by the reference ontology are ignored, because in the OBDA context every class instance can only have one value of a data property.

```
+!newPropertyRestrictionMapping(ClassType, PropertyType, RestrType, Values) <-

    Column = actionsLib.findColumnName(PropertyType);

    mapping(ClassMapID, targetRDF(ClassIRI, "a", ClassType), ClassSQL)

    if(RestrType == "AllValuesFrom") {
        .concat(ClassSQL, " WHERE ", Column, " in ", Values, ClassSQLNew)
    }
    if(RestrType == "HasValue") {
        .concat(ClassSQL, " WHERE ", Column, " = ", Values[0], ClassSQLNew)
    }

    -mapping(ClassMapID, targetRDF(ClassIRI, "a", ClassType), ClassSQL);
    +mapping(ClassMapID, targetRDF(ClassIRI, "a", ClassType), ClassSQLNew).
```

Figure 5.6: The mapping rule for New Property Restriction

Continuing the example in which we created the *Product of Canada* subclass of the class *Product*, we can now complete the generation of the mapping by adding the relevant property restriction of the subclass, namely that the data property *Location* has all values from the set {Canada, CA}.

The creation of the new subclass in the reference ontology now involves the following steps:

1. **Enrichment of Ontology (Manchester Syntax):**

```
Class: product_of_canada
    SubClassOf: <http://.../ref/product>
    EquivalentTo:
        <http://.../ref/product#location>
            only {"Canada", "CA"}
```

2. **Mapping Rule Invocations (AgentSpeak Syntax):**

```
!newSubClassMapping(
    "http://www.unbc.ca/.../ref/product",
    "http://www.unbc.ca/.../ref/product_of_canada")

!newPropertyRestrictionMapping(
    "http://.../ref/product_of_canada",
    "http://.../ref/product#location",
    "AllValuesFrom",
    ["Canada","CA"]).
```

3. **Addition of New Mapping (Turtle Syntax to SQL):**

```
target     <http://.../ref/product/{product_id}>
                a :product_of_canada .

source     SELECT * FROM oc_product
                WHERE location in ("Canada", "CA")
```

5.2.3 New Object Property

This rule generates a mapping for a new object property of a class. An object property represents a relation which associates an instance of a class (i.e., domain) to the instances of the same class or another class (i.e., range). The mapping of an object property contains a target RDF triple with the domain-class instance IRI as subject, property type as predicate, and range-class instance IRI as object.

The mapping rule is shown Fig. 5.7. It reads the mappings of the domain and range classes. The unnecessary variables such as mapping-id for the mapping are ignored (with _). The rule then extracts RDB column names from the IRI templates for the instances of domain and range classes, and prepares an SQL query that retrieves the RDB column values through a join of the following: tables for primary domain and range classes, or SQL queries for non-primary domain and range classes. The join condition in the SQL query helps in relating an instance of the domain class to the instances of the range class.

```
+!newObjectPropertyMapping(DomainClassType, ObjPropertyType, RangeClassType) <-

    mapping(_, targetRDF(DomainClassIRI, "a", DomainClassType), DomainClassSQL);
    mapping(_, targetRDF(RangeClassIRI, "a", RangeClassType), RangeClassSQL);

    SelectColumns = actionsLib.extractColumnNames
                                ([DomainClassIRI, RangeClassIRI]);

    ObjPropertySQL = actionsLib.getSQLJoinQuery([DomainClassSQL, RangeClassSQL],
                                            SelectColumns);

    ObjPropertyMapID = actionsLib.generateNewMappingId(ObjPropertyType);

    +mapping(ObjPropertyMapID,
             targetRDF(DomainClassIRI, ObjPropertyType, RangeClassIRI),
             ObjPropertySQL).
```

Figure 5.7: The mapping rule for New Object Property

For instance, the new object property rule can be invoked as follows to create the mapping (presented next) for the object property *hasDescription* with the *Product* class as domain and the *Product description* class as its range.

```
!newObjectPropertyMapping("http://.../ref/product",
                          "http://.../ref/product#hasDescription",
                          "http://.../ref/product_description")
```

The Resulting Mapping:

```
target    <http://.../product/{product_id}>
            :product#hasDescription
               <http://.../product_description/{product_id};
                  language_id={language_id}> .

source    SELECT oc_product.product_id,
                 oc_product_description.language_id
                 FROM oc_product, oc_product_description
                 WHERE oc_product.product_id
                       = oc_product_description.product_id
```

5.2.4 New Data Property

This rule generates a mapping for a new data property of a class. A data property for a class (i.e., domain) represents an attribute of the class, for instance, the *price* of a product. All existing columns of RDB tables are associated with the data properties of primary classes while bootstrapping the base ontology, which are then inherited by the reference ontology. In some cases, a new data property can be introduced for a primary class or non-primary class in the reference ontology by referring to a data property from other related primary class (i.e., referred class), using an identical property name. For instance, *name* data property for *product* which is actually part of the *product description* class. The mapping of a data property contains a target RDF triple with the domain-class instance IRI as subject, data property type as predicate, and an RDB column reference as object.

The mapping rule is shown in Fig. 5.8. The rule reads the mappings for the domain and referred classes, extracts RDB column names from the IRI template for the domain-class instances, and identifies the RDB column name for the data property. It then prepares an SQL query that retrieves the RDB column values using a join of the following: the underlying RDB table for the referred class, and the RDB table when the domain class is a primary class or SQL query when the domain class is a non-primary class. In the target RDF triple, the column is referred using a column reference in curly braces.

```
+!newDataPropertyMapping(DomainClassType, DataPropertyType, RefClassType) <-

    mapping(_, targetRDF(DomainClassIRI, "a", DomainClassType), DomainClassSQL);
    mapping(_, targetRDF(RefClassIRI, "a", RefClassType), RefClassSQL);

    SelectColumns = actionsLib.extractColumnNames([DomainClassIRI]);

    DataPropertyColumn = actionsLib.findColumnName(DataPropertyType);

    DataPropertySQL = actionsLib.getSQLJoinQuery([DomainClassSQL, RefClassSQL],
                                                 [SelectColumns,
                                                   DataPropertyColumn]);

    .concat("{", DataPropertyColumn, "}", ColumnRef)

    DataPropertyMapID = actionsLib.generateNewMappingId(DataPropertyType);

    +mapping(DataPropertyMapID,
             targetRDF(DomainClassIRI, DataPropertyType, ColumnRef),
             DataPropertySQL).
```

Figure 5.8: The mapping rule for New Data Property

For instance, the new data property rule can be invoked as follows to create the mapping (presented next) for the data property *name* for the *Product* domain class, that is derived from the reference class *Product description*.

```
!newDataPropertyMapping("http://.../ref/product",
                        "http://.../ref/product#name",
                        "http://.../ref/product_description")
```

The Resulting Mapping:

```
target    <http://.../product/{product_id}>
            :product#name
               {name}.

source    SELECT prod.product_id, prod_desc.name
            FROM oc_product prod, oc_product_description prod_desc
            WHERE prod.id = prod_desc.product_id
```

5.2.5 New Enumerated Data Property

This rule generates a mapping for a new data property which is enumerated from the values of a different data property (i.e., referred property) for its domain class. The enumerated data property uses the same structure of mapping for a new data property. Therefore, their mapping rules have general similarity with exceptions of the following: this rule refers to a different data property from the same class, and the SQL query in the mapping contains *cases* which translate an enumeration index to its corresponding text value.

Fig. 5.9 illustrates the mapping rule. It retrieves the mapping for the domain class, extracts the RDB column names from IRI templates, and finds RDB column names for the reference property and enumerated data property. It then prepares an SQL query by extending the SQL query for the domain class that retrieves values for the enumerated data property by introducing cases on the RDB column for the referred property from a list of value pairs.

```
+!newEnumDataPropertyMapping(DomainClassType, RefDataPropertyType,
                             EnumDataPropertyType, EnumValuePairs) <-

    mapping(_, targetRDF(DomainClassIRI, "a", DomainClassType), DomainClassSQL);

    SelectColumns = actionsLib.extractColumnNames([DomainClassIRI]);

    RefDataPropColumn = actionsLib.findColumnName(RefDataPropertyType);
    EnumDataPropColumn = actionsLib.extractColumnName(EnumDataPropertyType);

    EnumDataPropertySQL
        = actionsLib.getEnumSQLQuery([DomainClassSQL], [SelectColumns],
                        RefDataPropColumn, EnumDataPropColumn, EnumValuePairs);

    .concat("{", EnumDataPropColumn, "}", ColumnRef)

    EnumDataPropertyMapID
        = actionsLib.generateNewMappingId(EnumDataPropertyType);

    +mapping(EnumDataPropertyMapID,
             targetRDF(DomainClassIRI, EnumDataPropertyType, ColumnRef),
             EnumDataPropertySQL).
```

Figure 5.9: The mapping rule for New Enumerated Data Property

For instance, the new enumerated data property rule can be invoked as follows to create the mapping (presented next) for the enumerated data property *Product status* based on *Status* property for the *Product* domain class. The *Product status* data property presents *Active* and *Inactive* as textual representations of status values 1 and 0 respectively.

```
!newEnumDataPropertyMapping("http://.../ref/product",
                            "http://.../ref/product#status",
                            "http://.../ref/product#product_status",
                            [ [1, "Active"], [0, "Inactive"] ]).
```

The Resulting Mapping:

```
target    <http://.../product/{product_id}>
            :product#product_status
              {product_status}.

source    SELECT product_id,
CASE WHEN status = 1 THEN "Active"
     WHEN status = 0 THEN "Inactive"
END AS product_status
FROM oc_product
```

5.2.6 New Superclass

Superclasses do not require their own mappings because an OWL reasoner can infer their instances from the instances of their concrete subclasses.

5.2.7 Equivalent Class or Property

If one of the equivalent classes or properties can be instantiated with existing mapping, the others do not require individual mappings. The reasoner can refer to the class or property with existing mapping in order to fetch the RDF triples. If none of the classes or properties has existing mappings, the new mapping should be created based on the mapping rules for subclass, superclass, data property, or object property.

5.3 Selection of Tools

In order to study the feasibility of designing OBMAS, I studied, analyzed, and identified a set of state-of-the-art tools and technologies which are used to fulfill the design principles and system requirements. The following usability aspects formed the core of my study: generating a basic structure of ontology from the RDB schema; reading, browsing, and modifying an ontology; reasoning with an ontology; translating and executing semantic queries containing domain terms on the RDB; and building a team of agents which can deliberate with ontology-based knowledge for planning and communication. In the following paragraphs, I list and describe a set of tools, languages, and technologies which played important roles in the OBMAS design and implementation.

Ontop RDB-RDF translator: Ontop (Rodríguez-Muro et al., 2013) preference was selected over Virtuoso Universal Server (Erling and Mikhailov, 2007) and D2RQ server (Cyganiak et al., 2012) for the following reasons: (i) Ontop is developed with an objective to support OBDA (Rodrıguez-Muro and Calvanese, 2012) systems such as SQAS while the other two tools publish Linked data (Bizer et al., 2009) over the Semantic Web that does not require extensive transformation in terms of domain concepts; (ii) Ontop supports an OWL 2 QL reasoner called Quest (Rodríguez-Muro and Calvanese, 2012) which allows querying on generalized abstract classes without extra RDB-to-RDF mappings; (iii) it supports R2RML mapping language (Das et al., 2012) which is a W3C recommended specification for greater interoperability with external systems; (iv) it has a performance advantage over other tools as demonstrated by benchmark tests (Rodríguez-Muro et al., 2013).

Jason Platform with AgentSpeak: It is a platform for building multiagent systems (Bordini et al., 2007). It supports an extension of the AgentSpeak agent-oriented programming language and contains an interpreter for the language. The main reasons for its selection are (as explained in the domain-specific requirements 4.1.4) the good support of BDI model for the agent reasoning and declarative-style of programming. Additional reasons include: (i) it supports a well-designed reasoning cycle for agents; (ii) it facilitates reconsideration of agent plans for quickly responding to changes in the environment, i.e., the ontology; (iii) the recent work by Freitas et al. (2014) on semantic representations of agent plans using ontologies, which provides a basis for our planning meta-ontology, also presents an algorithm for the translation of the ontology to AgentSpeak.

OWL API for Ontology Access: OWL API (Horridge and Bechhofer, 2011) is a widely used Application Programming Interface (API) to create and manipulate OWL ontologies. These are the reasons for its inclusion in OBMAS: (i) it supports the several desired features for reading, searching, and modifying ontologies, as well as reasoning with them; (ii) several interfaces have been integrated in OWL API to work with reasoners such as FaCT++, HermiT, Pellet and Racer; (iii) its extensions, which are also employed by Protégé, support thread-safe operation on the ontology, which is quite useful for OBMAS.

OWL Sublanguage: I have used the OWL 2 QL sublanguage as the default language to build ontologies due to the following reasons. (i) The Quest reasoner (Rodríguez-Muro and Calvanese, 2012) for the reasoning with abstractions can only understand ontologies based on OWL 2 QL or simpler ontology sublanguages. (ii) OWL 2 QL is designed to support sound and complete reasoning in LOGSPACE while still allowing several essential features for representing reasonably complex ontologies.

A user can still select a different sublanguage such as OWL Lite, OWL DL, or OWL 2 RL in order to represent more-or-less expressive ontologies.

SPARQL for Semantic Queries: SPARQL (Prud'hommeaux and Seaborne, 2008) is an RDF-oriented language, but it is sufficient to represent the intended test queries for the prototype. It is a W3C recommendation facilitating interoperability with external tools. Nevertheless, it will be interesting to explore any ontology-oriented query language to write expressive semantic queries in the future.

5.4 Current Implementation

In this section, I present the current state of implementation for the proposed solutions of the research challenges addressed in this chapter, designs of meta-ontologies; mapping rules for semantic query translation; and selection of state-of-the-art tools.

In the current implementation of OBMAS, the model meta-ontology and its converter for the agents do not have implemented versions. They are conceptually proposed and designed as a part of this thesis. For the planning meta-ontology, the hierarchical model has been designed, implemented, and instantiated with the Ontology Development 101 methodology (Noy et al., 2001). In addition, a converter of planning ontology to plan library in AgentSpeak, developed by Freitas et al. (2014), has been extended, tested, and used for translation of planning meta-ontology as a part of the research.

The ontology service utilities component has been designed to generate mappings with the proposed mapping rules. The current implementation has the following rules which are functional at present: new subclass, new property restriction, and new data property. The mapping rules are implemented in Java.

The Ontop RDB-RDF translator has been integrated with OBMAS to provide the bootstrapping and semantic query translation services. The reference ontology is developed using the OWL 2 QL sublanguage and the semantic queries were written with SPARQL. The Jason platform has been used to develop the agents with AgentSpeak language, and the Jason's centralized environment is employed during execution. The latest version of the OWL API has been used to read, load, build, and modify the ontologies.

5.4.1 Considerations for Larger-Scale Implementation

In this section, I describe the considerations for a larger-scale implementation of OBMAS such as the performance improvement techniques and known issues with the state-of-the-art tools.

The number of RDB-to-RDF mappings increases as the reference ontology evolves, and they become difficult to maintain. An alternative option, as proposed in SQAS, is to design and build a SPARQL-to-SPARQL query rewriting engine which can reduce reference classes to equivalent primary classes by incorporating intermediate property restrictions in the resultant SPARQL query.

The semantic queries perform slow in the prototype compared to the SQL queries in ETL-based approach of other decision support systems, due to the runtime SPARQL-to-SQL translation process. For performance improvement in a production deployment, the prototype can cache data for frequent queries, or the reference ontology can be materialized through selective pre-loading of important classes.

Ontop (Rodríguez-Muro et al., 2013) is not a complete solution and carries several known issues (Xiao et al., 2015) including: no support for HAVING for aggregates; errors for complex nested FILTERs; missing string and literal equality in FILTER; and no support for inverseExpression in R2RML. The latest version of Ontop provides limited support of aggregate or complex queries which is essential for any decision support system. Nevertheless, it is under active development and future releases may resolve those issues and limitations in order to support a larger-scale implementation.

Chapter 6

Ontology Building Assisted by Agent Team

In this chapter, I describe the behaviour of the OBMAS agents through building the reference ontology from a basic knowledge structure implicit in the RDB schema (6.1) and evolving the reference ontology in order to incorporate its changes (6.2). I demonstrate the execution of semantic queries based on the terms from the reference ontology (6.3). The major behavioural aspects implemented in the prototype include the OBA assisting DE in the processes of generalization and specialization of existing classes, and the execution of semantic queries that rely on a set of mapping rules which automatically generate the required RDB-to-RDF mappings. The behavioural aspects of the other agents which are presented in this chapter have not been implemented in this version of the prototype.

6.1 Building Reference Ontology

In this section, I describe and illustrate the method of ontology development from an existing RDB. For illustration, I use a fragment of the public e-commerce RDB called Opencart (Opencart, 2014) (described in 6.1.1). During the ontology development process for this running example, I have assumed the roles of the Database Administrator (DBA) and Domain Expert (DE). The method involves the following generic steps: extracting the base ontology from the RDB schema, by an automatic process commonly referred to as bootstrapping; generating the initial reference ontology from the base ontology with annotation properties that capture additional information on RDB structure, such as primary keys (6.1.2); creating the primary reference ontology based on insights from the RDB schema by introducing object property and subclass relations between classes (6.1.3); and enriching the reference ontology with new abstractions through generalization and specialization of existing classes (6.1.4, 6.1.5).

6.1.1 The Starting Point: A Relational Database

At the start of the process, the Ontology Builder Agent (OBA) interacts with the DBA to establish a valid connection to the RDB. The RDB schema is assumed to be in third normal form (Kent, 1983). The RDB schema that is used as a running example in this chapter is a fragment of the Opencart schema shown in Fig. 6.1. While the fragment is not entirely independent in that the tables have columns related to other parts of the Opencart schema, we shall refer to this fragment as the RDB schema in presenting the examples.

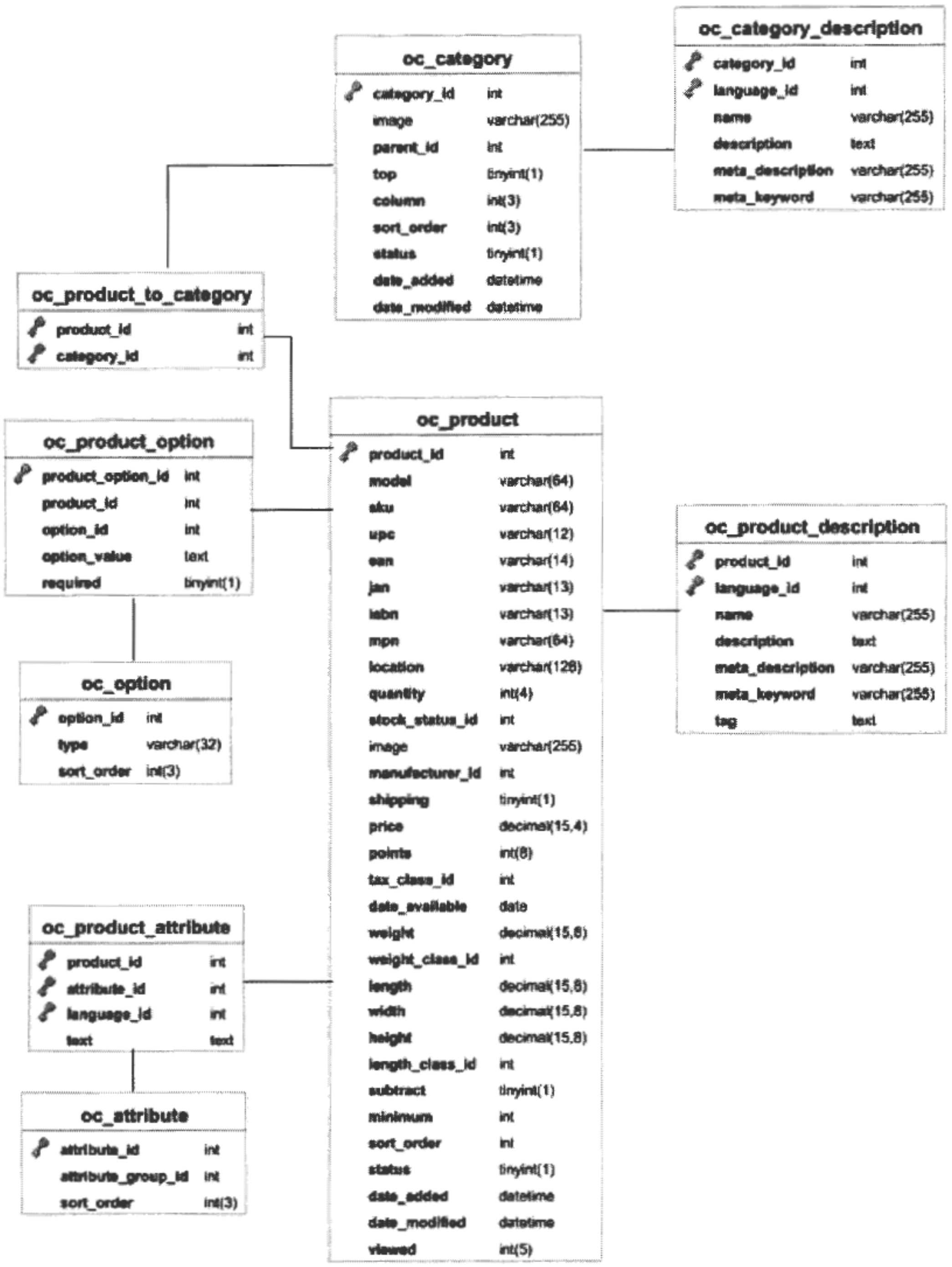


Figure 6.1: The fragment of the Opencart schema used in examples

6.1.2 Generating the Initial Reference Ontology

As a first step in generating the initial reference ontology, the OBA initiates the bootstrapping of the base ontology from the RDB schema using an external RDB-RDF translator called Ontop (Rodríguez-Muro et al., 2013). The translator converts the RDB concepts to ontological concepts in OWL 2 representation (Hitzler et al., 2012). This is achieved by mapping tables to classes, which are referred as primary classes, and columns to data properties, as described in the Subsection 2.2.4. Fig. 6.2 illustrates the base ontology generated from the Opencart RDB schema.

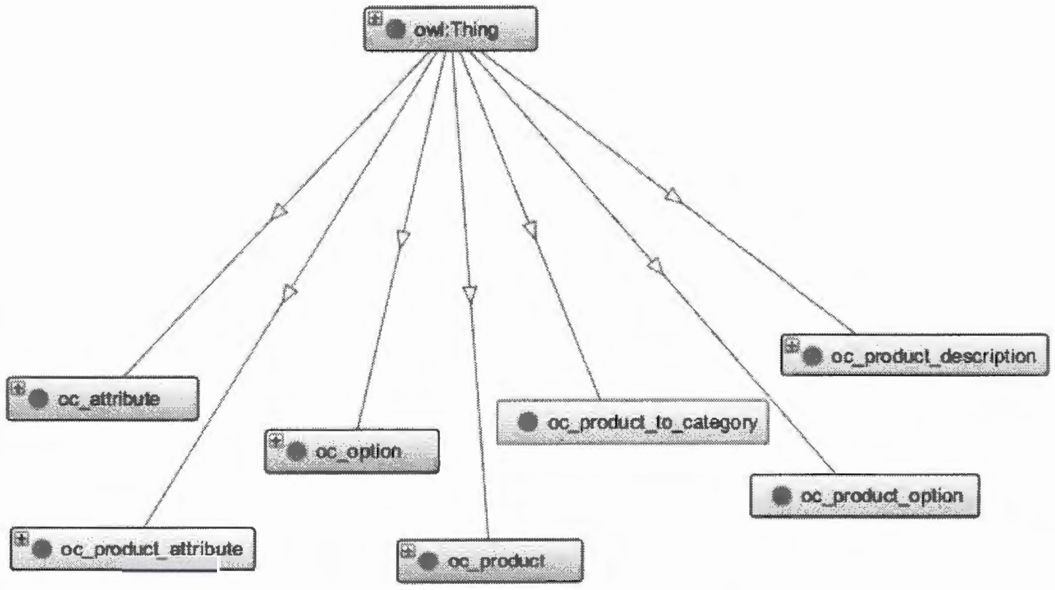


Figure 6.2: The base ontology generated from the RDB schema

After bootstrapping the base ontology, an initial reference ontology is created that inherits the primary classes from the base ontology and includes annotation properties that can help the agents during the later steps of ontology development. In the current example, it includes the properties for naming, such as *prefLabel*, which distinguishes the primary name from its aliases, and *altLabel* for alternate names. In addition, an annotation property *isTempLabel* is introduced to denote classes which have temporary primary names that can be later used to identify pending tasks for the OBA.

6.1.3 Creating the Primary Reference Ontology

The primary reference ontology is built by the OBA from the initial reference ontology in consultation with the Ontology Learner Agent (OLA), which analyzes the RDB schema to find relations, and the DBA, who selects which relations to include in the primary reference ontology. The development process involves the following three stages. First, the RDB schema is analyzed to identify, select, and create subclass relations between primary classes which are implicitly present in the schema. Second, the RDB schema is analyzed to find, select, and create object property relations for each primary class with other primary classes. Third, the names inherited from the RDB schema, such as table names and column names, are revised to improve human readability and filter out irrelevant detail.

Identifying Subclass Relations between Primary Classes

In this stage, the OBA initiates the interaction process by requesting assistance from the OLA to identify candidate subclass relations. If two or more tables in the RDB schema share a column name which is either a primary key or a part of the composite primary key, then the OLA marks them as candidate classes for subclass relations. The OLA tries to identify common superclass among them with up to four different methods specified and employed in the following order. First, if only one table from the list contains a regular primary key while the others have composite primary keys that involve the regular primary key, the OLA marks that table as candidate superclass. Second, the OLA requests the Semantic Web Crawler Agent (SWCA) to find a known superclass for the listed classes by referring to the class hierarchies from similar external ontologies. Third, the OLA connects with the WordNet or other thesauri to find a common hypernym from class names. Fourth, the OLA looks for structural correspondence among the class names containing multiple words to find the most common class name which may be a candidate superclass.

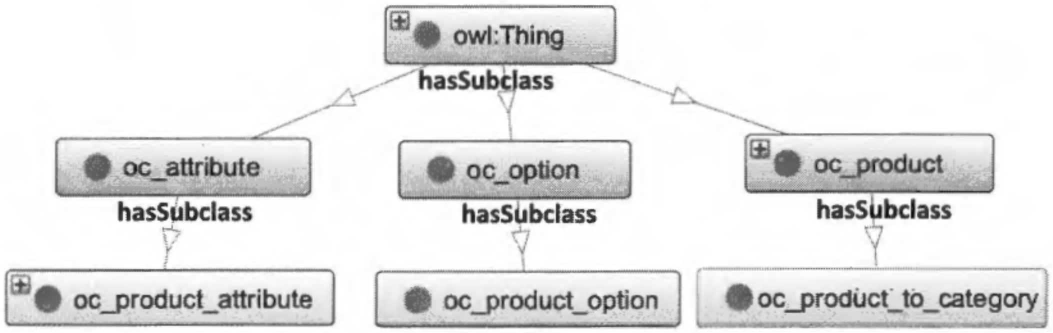


Figure 6.3: The primary reference ontology with subclass relations

For example, the OLA finds a set of primary classes related by the *product_id* column name: *oc_product*, *oc_product_to_category*, *oc_product_description*, and *oc_product_attribute*. In addition, it identifies *oc_product* as candidate superclass because it contains a regular primary key while other tables have composite primary keys (which introduce additional dependencies). From the suggestions, the DBA instructs the OBA to create subclass relations between *oc_product* and *oc_product_to_category*. In a similar manner, the DBA selects appropriate relations from the other sets of primary class relations. Fig. 6.3 illustrates the state of the primary reference ontology at the completion of this stage.

Identifying Object Properties for Primary Classes

In this stage, the OBA initiates an interaction with the OLA by requesting that it identify candidate object property relations between primary classes. The OLA analyzes the RDB schema in order to find foreign key relations between the RDB tables, and sends a set of candidate object property relations between corresponding primary classes to the OBA. The DBA, in consultation with the OBA, selects the desired object property relations and provides new property names. The OBA then creates the object properties in the primary reference ontology.

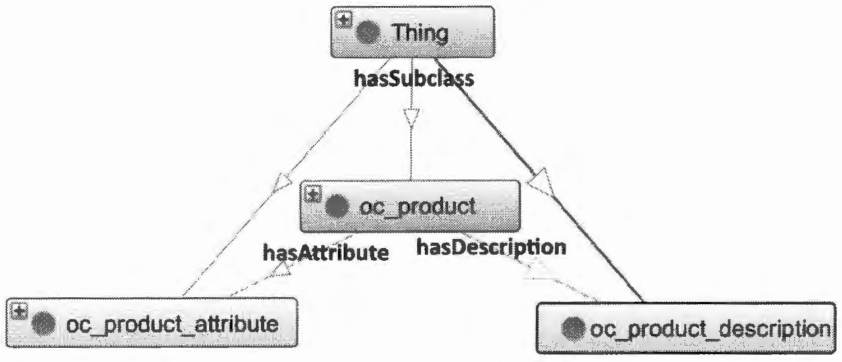
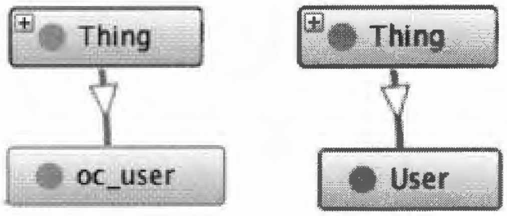


Figure 6.4: The primary reference ontology with object properties

For example, the OLA identifies candidate object property relations between primary classes such as *oc_product* with *oc_product_description*, *oc_product* with *oc_product_option*, *oc_product* with *oc_manufacturer*, *oc_product* with *oc_product_to_category*, and *oc_product* with *oc_product_attribute*. In response, the DBA selects two important relations *hasAttribute* and *hasDescription*, as illustrated in Fig. 6.4.

Revision of Names for Primary Classes

In this stage, the OBA identifies and applies a renaming pattern that filters out the irrelevant detail from the names inherited from the RDB, such as the *oc_* prefix example, shown in Fig. 6.5. The OBA also interacts with the DBA to replace the inherited names with appropriate primary names that reflect the terminology of the knowledge domain. The correspondence of primary names to the original base ontology names is preserved, to be used later during query translations and RDB schema revisions. The primary names can be further revised by the DE, if necessary.



(a) (b)

Figure 6.5: *oc_user* primary class before and after the revision

Fig. 6.6 presents the primary reference ontology after the first three stages.

6.1.4 Looking for New Abstractions

The next stage in ontology development is its enrichment with new abstractions, namely new subclasses and superclasses, which we discuss in subsection 6.1.5. However, this is a creative process for which the initial ideas come from the human partner or from existing knowledge sources, such as external ontologies, with the agents assisting the DE in the elaboration of the idea.

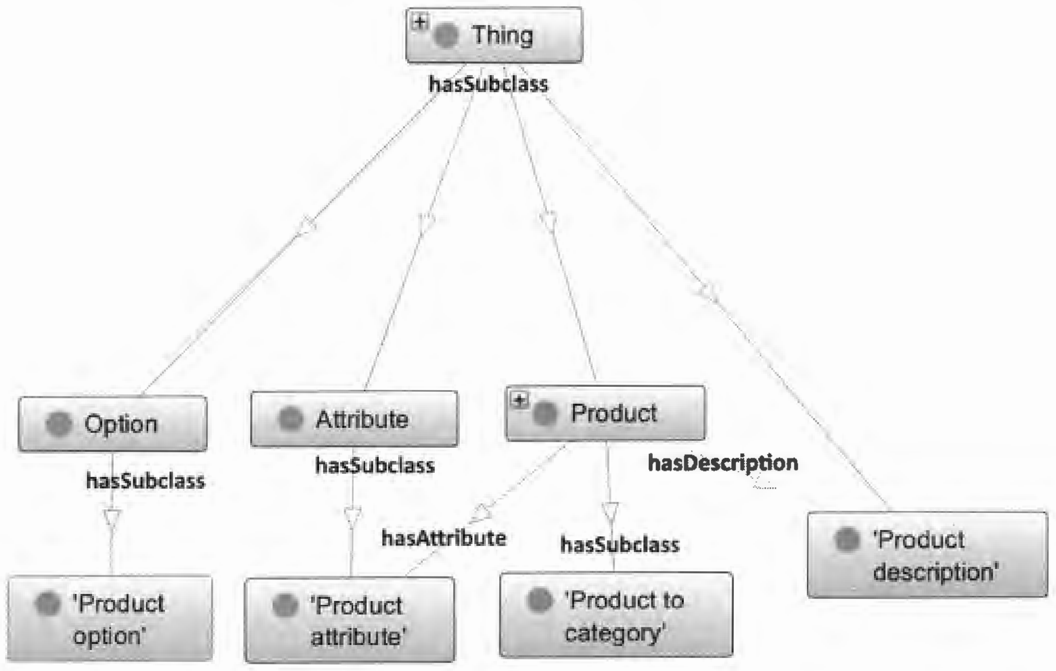


Figure 6.6: The primary reference ontology

As an illustration, we provide a simple example of looking for suitable abstraction names in a thesaurus. The OBA responds to a request from the DE to find candidate subclass names for the given class name *Product*. From WordNet (Miller, 1995), the OBA reports the hyponyms of the word *Product*: *Cargo*, *Stock*, *Refill*, and *Yard goods*. The DE requests more options. For that, the OBA needs a list of terms similar in meaning to *Product*, in order to consider their hyponyms. A starting point for such a list are the "sister terms" (sharing the same hypernym) of *Product*, such as *Consumer goods*, *Fancy goods*, and *Shopping*. Some of the sister terms may have divergent meanings or not suit the knowledge domain, so the choice of relevant terms is left to the DE. When instructed to use *Consumer goods*, the OBA returns a set of hyponyms which include words like *Clothing*, *Consumer durables*, *Fashion*, and *Grocery*. From these suggestions, the DE instructs the OBA to introduce *Clothing*, *Fashion product*, and *Grocery* as subclasses for *Product*.

While the example illustrates only the use of linguistic external expertise in looking for new abstractions, ideas can also come from external sources related to the knowledge domain of the reference ontology. The advancement of the Semantic Web is expected to significantly expand such possibilities, and the cooperation of the DE and the SWCA can lead to their effective use. The use of external ontologies is an explicit recommendation of the Ontology Development 101 methodology (Noy et al., 2001).

6.1.5 Adding Subclasses and Superclasses

In order to enrich the reference ontology through generalization or specialization, the OBA communicates with the other agents and the DE. The process can be initiated by a request from the DE or by the OBA as a step in the ontology development methodology. The objective may be either to create a reference subclass, which is a class more specific than the primary classes; or to create a reference superclass, which is a class more general than the primary classes. Once the reference superclasses and subclasses are added to the reference ontology that can serve as basis for further generalizations and specializations.

Creating a Reference Subclass: The OBA interacts with the DE and the other agents to introduce a new subclass in the reference ontology. The process involves at most the following three steps: identifying a class name for the class, adding the subclass and its relation with the parent, and enriching the subclass through new properties or property restrictions.

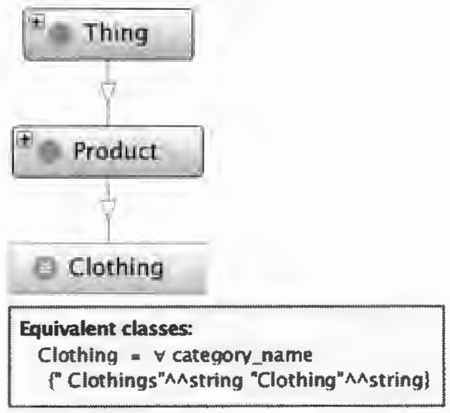
Identifying Class Name for Reference Subclass. The OBA interacts with the DE and communicates with the other agents to identify the most appropriate name for the class, as discussed in the Subsection 6.1.4. It verifies that the new name does not conflict with existing names in the class hierarchy.

Adding the Relation between a Class and Subclass. The OBA adds the subclass to the reference ontology and creates its subclass relation with the parent class. It also generates an RDB-to-RDF mapping using the mapping rule for new subclass, described in Subsection 5.2.1; this mapping is used during translation of semantic queries that refer to the new subclass. If some of the existing subclasses of the parent are candidates for becoming subclasses of the newly introduced subclass, the OBA and DE may consider reclassification; in our example, this happens with the subclass *Consumer goods* discussed below.

Enriching the Reference Subclass. The OBA interacts with the DE in order to enrich the new subclass with one or more of the following options: new data property, object property, or property restriction. For the selected option, the OBA requests additional information from the DE, and then creates the entity in the reference ontology. If the DE decides to skip the process, the OBA denotes the task as pending in order to revisit it later during its idle time.

In the example for the *Clothing* subclass, the DE instructs the OBA to enrich the subclass by adding a property restriction on *category_name* property with values Clothing or Clothings. The OBA also modifies the RDB-to-RDF mapping for the *Clothing* class in order to include value restriction in the SQL query. Fig. 6.7 illustrates a fragment of the reference ontology with the *Clothing* subclass, the property restriction, and the RDB-to-RDF mapping.

Fig. 6.8 illustrates the development of a class hierarchy with new subclasses and



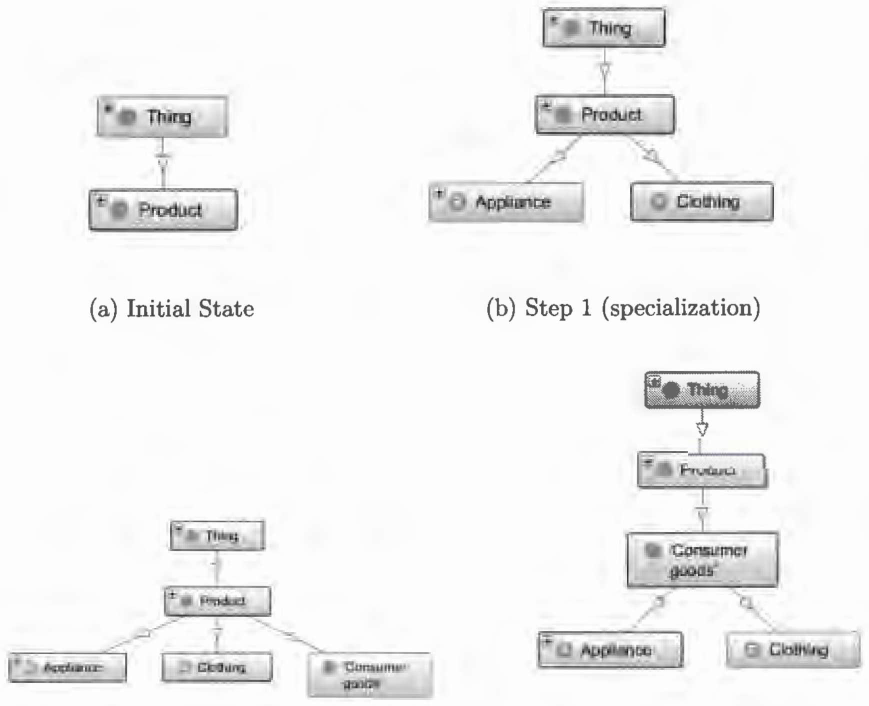
Equivalent classes:
Clothing = ∀ category_name
{" Clothings"^^string "Clothing"^^string}

(a) Clothing Subclass

```
mappingId  MAP@Clothing
target
     <http://www.unbc.ca/.../ref/
     Product/product_id={product_id}>
        a ref:Clothing .
source
     SELECT *
       FROM oc_product
         WHERE category_name
         IN ('Clothing', 'Clothings')
```

(b) Mapping for Clothing

Figure 6.7: Adding the *Clothing* subclass to the reference ontology



(a) Initial State

(b) Step 1 (specialization)

(c) Step 2 (specialization)

(d) Step 3 (reclassification)

Figure 6.8: Adding a new subclass: specialization and reclassification

reclassification. In the first step, *Appliance* and *Clothing* are created as subclasses for the *Product* class. In the second step, *Consumer goods* is introduced as the third subclass of *Product*, and in the third step it becomes the parent of the other two subclasses of *Product* through reclassification. During the third step, the OBA modifies the RDB-to-RDF mappings associated with the *Appliance* and *Clothing* classes in order to reflect the new class hierarchy.

Creating a Reference Superclass: The OBA interacts with the DE and the other agents to introduce new superclass in the reference ontology. The reference superclass can be introduced by the DE directly, or in response to suggestions from the agents based on analyzing knowledge sources such as RDB schema or external ontologies. The process to add reference superclass involves at most the following three steps: identifying a class name for the superclass, adding the superclass and its relation with the subclasses, and creating common properties for the superclass.

Identifying Class Name for Reference Superclass. The process to identify the primary class name for reference superclass is similar to the process of identifying class name for the reference subclass. A main difference in how the agents query external knowledge sources. For example, the agent now looks for hypernyms instead of hyponyms of a class name while querying thesauri.

Adding the Relation between a Class and the Superclass. The OBA adds the superclass to the reference ontology and creates a subclass relation for each of its subclasses. The hierarchy is reviewed for possible reclassification.

Creating Properties for the Reference Superclass. In this step, the OBA identifies and suggests common properties of the subclasses which can be moved to the superclass. The DE verifies the suggestions and interacts with the OBA to identify, and possibly modify, primary names for the common properties. The OBA creates common properties in the reference ontology and removes them from the subclasses. The properties of the primary classes are not removed during the process because they are directly mapped to the RDB tables. In that case, the OBA creates sub-property relations between the common properties of the superclass and the primary class.

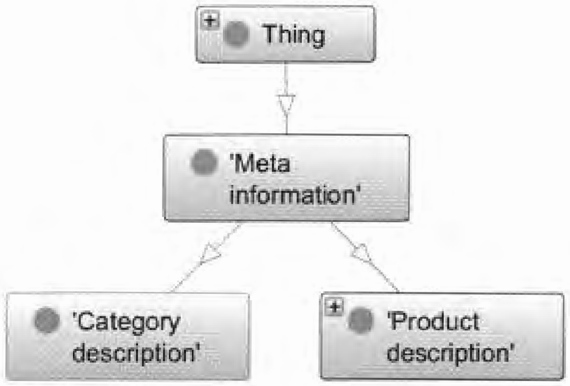


Figure 6.9: A part of the reference ontology after creating reference superclass

An example is shown in Fig. 6.9. The OLA identifies that the classes *Product description* and *Category description* share properties such as *name*, *description*, *meta description*, and *meta keyword*. The DE suggests that the two classes can have a superclass called *Meta Information*, and the OBA creates it as their parent. The DE reviews the common properties, and requests that they be moved to *Meta Information*. The OBA creates the properties in the reference ontology, and introduces sub-property relations because both of the subclasses are primary classes.

The complete reference ontology developed in this section is shown in Fig. 6.10.

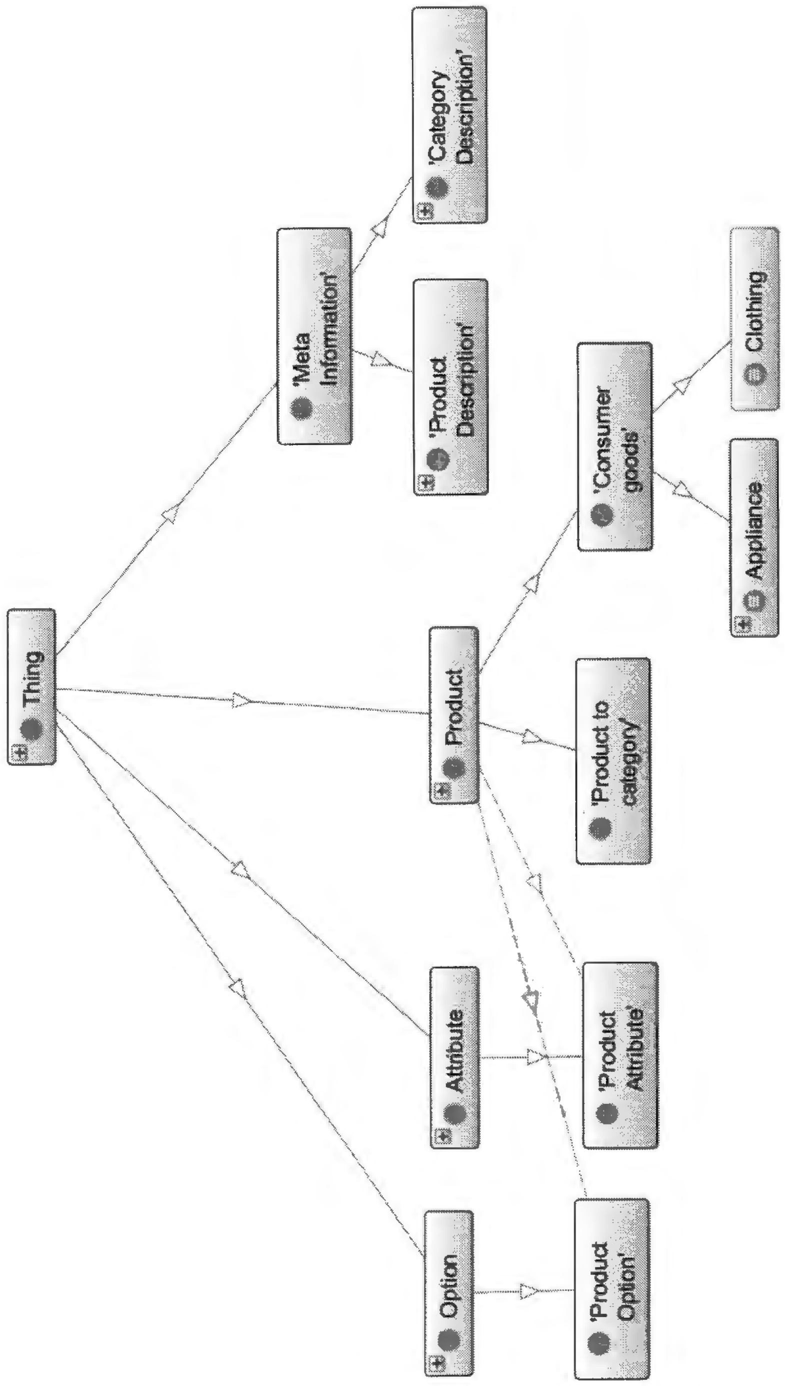


Figure 6.10: The complete reference ontology

6.2 Handling RDB Schema Changes

When the legacy RDB schema is modified, OBMAS captures the change event and automatically compares it with the previous version of the RDB schema in order to analyze and identify the following types of changes: addition or removal of tables, views, columns, or foreign key relations; and changes in table name, view name, column name, or column datatype. The DBA supervises the process in order to correctly review and categorize the identified changes.

The OBA assesses the impact of the schema changes on the reference ontology by identifying the class hierarchies which can be affected, and shares the results with the DBA who interacts with the OBA in order to incorporate the changes through reclassification or other modifications. During the interaction, the OBA carries out the following tasks: bootstrapping a new version of the base ontology; copying the previous version of the reference ontology in a new version; and interacting with the DBA to reclassify the affected class hierarchies as well as rectifying the associated RDB-to-RDF mappings. This is an area which requires further research.

6.3 Semantic Querying of Legacy RDB

The user of OBMAS submits a semantic query (written in SPARQL) using terms of the knowledge domain (e-commerce) in order to access data from the RDB. OBMAS uses Ontop to perform RDB-RDF translation which involves translating SPARQL queries to SQL queries and SQL results to RDF triples using the mappings generated by the mapping rules (proposed in Section 5.2). In this section, I explain the process of semantic query translation with an example query and present a set of semantic queries which were used as test queries for the semantic data access. For each semantic query, I list the corresponding mapping rules and mappings.

Fig. 6.11 illustrates the semantic query translation for the reference subclass. It shows: a SPARQL query, its equivalent SQL query, and the results as RDF triples. The RDB-to-RDF mappings used during the translation are shown in Fig. 6.12. The first mapping helps create the base query for *Clothing* class with a filter condition on the *category_name* from *oc_product* table. The second mapping is then invoked to add *Name* data property to the result set through a join of tables *oc_product* and *oc_product_description*.

(i) Semantic Query in SPARQL syntax (prepared by human partner)

```
SELECT ?name ?price
WHERE {
  ?query a ref:Clothing.
  ?query ref:Product#Name ?name.
  ?query ref:Product#Price ?price.
}
```

(ii) SQL Query (converted by RDB-RDF translator)

```
SELECT QVIEW2.'name', QVIEW1.'price'
FROM  ''oc_product'' QVIEW1, ''oc_product_description'' QVIEW2
WHERE  (('Clothing' = QVIEW1.'category_name')
          OR ('Clothings' = QVIEW1.'category_name'))
        AND (QVIEW1.'product_id' = QVIEW2.'product_id')
```

(iii) Query Results in RDF format (presented by Query Interface)

```
"Slim Fit Shirt In Stretch Cotton Poplin"@en, "100.0000"^^xsd:decimal,
"510 Skinny Jeans"@en, "199.9900"^^xsd:decimal,
```

Figure 6.11: Executing semantic query for a reference subclass

The set of test semantic queries, and the mappings required for their execution are as follows. The first semantic query finds a list of instances for a primary class called *Product* with data properties *Name* and *Price*; here, a mapping generated by the basic mapping rule (described in Section 2.2.4) is used by the translator to create

(i) Mapping from *New Subclass* rule (created by OBA)

```
mappingId  MAP@Clothing

target     <http://www.unbc.ca/.../ref/Product/product_id={product_id}>
             rdf:type <http://www.unbc.ca/.../ref/Clothing> .

source     SELECT *
           FROM  oc_product
           WHERE category_name IN ('Clothing', 'Clothings')
```

(ii) Mapping from *New Data Property* rule (created by OBA)

```
mappingId  MAP@Product#Name

target     <http://www.unbc.ca/.../ref/Product/product_id={product_id}>
           <http://www.unbc.ca/.../ref/Product#Name>
           {name}@en .

source     SELECT oc_product.product_id, oc_product_description.name
           FROM  oc_product, oc_product_description
           WHERE oc_product.product_id
                   = oc_product_description.product_id
```

Figure 6.12: Mappings for semantic querying of a reference subclass

an SQL query in order to load the data from the underlying RDB table.

The second query finds a list of instances for a reference subclass called *Clothing* with data properties *Name*, *Price*, and *Description.* Here, the following mapping rules are used: *New Subclass* (5.2.1) for *Clothing* subclass to *Product*, and *New Data Property* (5.2.4) for *Name* data property. For execution, the translator generates an SQL query containing a condition to filter records with category name as *Clothing* or *Clothings.*

The third query finds a list of instances for a reference superclass called *Meta Information* with data properties *Meta description* and *Meta keyword.* Here, the translator queries the RDB to load instances for the primary classes which are subclasses of *Meta information*, and then the ontology reasoner infers the instances of

Meta information from its subclass instances.

The fourth query finds an aggregated value of the *Product reward* data property for orders grouped by each store. Here, the mappings corresponding to the referred classes are loaded in addition to the mapping for *Product reward* data property generated by the *New data property* mapping rule 5.2.4. Then the translator generates an SQL query, which loads *Product reward* from *Order product* through a join with *Order* and sums up the values for each store.

Chapter 7

Conclusions and Future Work

This thesis demonstrates the feasibility of a software system in which intelligent software agents assist the human user in developing an ontology. The initial version of the ontology is extracted from a relational database (RDB) using the Ontop RDB-RDF translator, and then enriched with higher-level entities such as new subclasses, superclasses, and data properties to support semantic queries as a part of ontology-based data access (OBDA) to the underlying RDB. We propose the architectural design of Ontology-Building Multiagent System (OBMAS) in which an agent team assists a human partner in ontology-building tasks. We implement a prototype of the key agent role, the Ontology Builder Agent (OBA), and demonstrate its functions through concrete ontology development scenarios. The agent is implemented in AgentSpeak on the Jason platform and assists in building an ontology in OWL 2. The role of the human partner in OBMAS is to provide domain expertise, evaluate ontology design options, and make decisions. The agents gather and present information, explore and propose solutions, and manipulate knowledge representation structures.

The agents can construct an ontology because they are endowed with *meta-ontology*, i.e., an ontology that represents the knowledge of ontology-building domain. In OBMAS, the agents can communicate and reason with their *model meta-ontology*, and develop the ontology by using and enforcing a methodology represented in their *planning meta-ontology* as agent plans. The model meta-ontology, i.e., the concrete metamodel, is instantiated using the proposed method of concretization from the reference ontology and the OWL 2 Metamodel; the conversion method then translates it into agent beliefs that represent an agent's ontological knowledge for reasoning and communication in the ontology-building process. The design of model meta-ontology is complete, and it needs to be supported by implementation in the prototype. The *planning meta-ontology* is designed and implemented as a hierarchical model of ontologies for representing agent plans at different levels of abstraction. It relies in part on concepts and techniques developed in Hierarchical Task Network (HTN) planning systems.

The planning meta-ontology of OBMAS guides the agents through the steps of a popular methodology known as Ontology Development 101; it is instantiated for specific agents according to the skill profiles associated with their roles. The current OBA prototype focuses on enrichment of ontology by adding new entities. It needs to be extended to include the remaining aspects of the methodology in support for larger-scale systems. Our meta-ontology designs contribute to the conceptual foundation of the AAOB approach as well as to practical development of AAOB systems. We expect that they will facilitate further research in the domain of automated ontology development.

The OBMAS agents directly provide advanced support for semantic query translation using a set of mapping rules that correspond to specific operations in the construction of higher-level entities in the reference ontology (the term for domain ontology in SQAS). In the prototype, the OBA automatically constructs mappings using the mapping rules while adding new entities such as subclasses, property restrictions, and data properties in the reference ontology. The generated mappings allow immediate use of the new entities in semantic queries, as we have shown through the examples of queries executed using the prototype. The rules enable a user to delegate the technical work of mapping creation to the agents. The remaining rules will be included in a larger-scale implementation.

In order to build the agent prototype, the tools and languages were selected using the following criteria: a set of usability aspects such as the support for working and reasoning with an ontology, semantic query translation, and building an agent team for ontology development; and the factors such as the OBDA domain standards, compatibility with other tools, and active development for new features. The matching options based on the criteria were analyzed and examined with a test setup before selecting the final group which include Ontop, Jason with AgentSpeak, OWL 2 API, OWL 2 QL, and SPARQL. Since we had satisfactory results with the tool selections during the prototype implementation, they will also be used in a larger-scale system development.

The feasibility study was successful, leading to the OBMAS design and then the prototype implementation. The OBA facilitates the reference ontology development from an RDB by supporting technical ontology-oriented tasks such as bootstrapping a *base ontology* (in the terminology of SQAS) from the RDB, automatically generating mappings for new entities, and creating those entities in the reference ontology. It also assists the user by executing semantics-oriented tasks such as finding an appropriate class name from WordNet, and recommending to enrich a new subclass and helping in the enrichment process. Those OBA features reduce the technical work for a user to develop ontology as compared to ontology editors such as Protégé; instead, the user can focus on making decisions. The resulting enriched reference ontology demonstrates the agent's ability to assist the user in developing a correct ontology.

OBMAS contains five types of agents with the following individual skill profiles: building the reference ontology, ontology learning from the underlying RDB, ontology lookup on the Semantic Web, alignment with external ontology, and ontology validation. The agent roles were identified through an analysis of the OBMAS requirements and other OBDA projects. In a larger-scale system, the OBA may be able to help in building a complete reference ontology using the knowledge discovery services of the other agents.

With respect to possible future research, the following directions can be considered:

- The Semantic Web Crawler Agent (SWCA), Ontology Learning Agent (OLA), Ontology Alignment Helper Agent (OAHA), and Ontology Validator Agent (OVA) were not included in the scope of the present prototype. They can assist the OBA in looking for new abstractions as well as complement its skill profile. These agents should be designed and implemented in order to support their use cases for building a better domain ontology. The OBMAS use cases for the agents are not complete; advanced features can be identified and developed to better equip the agents in looking for higher-level abstractions for the reference ontology.

- The planning meta-ontologies can be instantiated to include the remaining aspects of the generic ontology development methodology. That will help us validate the planning meta-ontology design for full support of such methods.

- A SPARQL-to-SPARQL query rewriting engine can be designed and built, which can translate a semantic SPARQL query to a base SPARQL query through a reduction from the higher-level abstractions to the primary classes in the reference ontology. It may provide a better and scalable solution compared to the mapping rules.

- In order to demonstrate the agent reasoning and communication using ontological concepts in the model meta-ontology, converters should be developed to create a concrete metamodel, and then translate it to the agent beliefs.

- The schema monitor component can be designed and implemented in order to show the agent assistance in evolving the reference ontology with changes in the underlying RDB schema.

Bibliography

Agent factory, 2014. URL `http://www.agentfactory.com/`. (visited on 2014-08-10).

Kenneth Baclawski and Todd Schneider. The open ontology repository initiative: Requirements and research challenges. In *Proceedings of workshop on collaborative construction, management and linking of structured knowledge at the ISWC*, page 18, 2009.

Jie Bao and Vasant Honavar. Collaborative ontology building with Wiki@nt - a multi-agent based ontology building environment. 2004.

Tim Berners-Lee. Semantic Web Road Map. W3C Design Issues Architectural and Philosophical Points, 1998. URL `http://www.w3.org/DesignIssues/Semantic.html`. (visited on 2014-05-21).

Tim Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. 285(5):28–37, 2001.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.

Saartje Brockmans, Peter Hasse, and Boris Motik. MOF -Based Metamodel - OWL, 2008. URL `http://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel`. (visited on 2015-03-13).

Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Unifying class-based representation formalisms. *J. Artif. Intell. Res.(JAIR)*, 11:199–240, 1999.

Ji Woong Choi and Myung Ho Kim. Generating OWL Ontology from Relational Database. In *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, pages 53–59, June 2012. doi: 10.1109/MUSIC.2012.17.

Richard Cyganiak, Chris Bizer, Jörg Garbers, Oliver Maresch, and Christian Becker. The D2RQ Mapping Language — The D2RQ Platform, 2012. URL `http://d2rq.org/d2rq-language`. (visited on 2014-08-31).

Mathieu d'Aquin, Laurian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta. Watson: A gateway for next generation semantic web applications. 2007.

Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language, 2012. URL `http://www.w3.org/TR/r2rml`. (visited on 2015-02-23).

Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

Orri Erling and Ivan Mikhailov. Mapping Relational Data to RDF with Virtuoso's RDF Views, 2007. URL `http://virtuoso.openlinksw.com/whitepapers/relationalrdfviewsmapping.html`. (visited on 2015-09-25).

Kutluhan Erol, James Hendler, and Dana S Nau. Htn planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128, 1994.

Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 128–135. IEEE, 1998.

Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM, 1994.

Artur Freitas, Daniela Schmidt, Alison Panisson, Felipe Meneguzzi, Renata Vieira, and Rafael H Bordini. Semantic representations of agent plans and planning problem domains. In *Engineering Multi-Agent Systems*, pages 351–366. Springer, 2014.

Dragan Gašević, Dragan Djurić, and Vladan Devedžić. *Model driven architecture and ontology development*. Springer Science & Business Media, 2006.

Lushan Han, Abhay Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. UMBC EBIQUITY-CORE: Semantic textual similarity systems. *Atlanta, Georgia, USA*, 44, 2013.

Pascal Hitzler, Markus KrÃtzsch, Bijan Parsia, Peter Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition), 2012. URL `http://www.w3.org/TR/owl2-primer/`. (visited on 2015-02-23).

Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

Matthew Horridge and Peter Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (Second Edition), 2012. URL `https://www.w3.org/TR/owl2-manchester-syntax/`. (visited on 2016-01-23).

Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. *University of Manchester*, 2004.

TopQuadrant Inc. TopBraid Composer - Getting Started Guide Version 3.0, 2011. URL `http://www.topquadrant.com/resources/products/docs/TBC-Getting-Started-Guide.pdf`. (visited on 2015-09-09).

William Kent. A simple guide to five normal forms in relational database theory. *Communications of the ACM*, 26(2):120–125, 1983.

Evgeny Kharlamov, Ernesto Jiménez-Ruiz, Dmitriy Zheleznyakov, Dimitris Bilidas, Martin Giese, Peter Haase, Ian Horrocks, Herald Kllapi, Manolis Koubarakis, Özgür Özçep, et al. Optique: Towards OBDA systems for industry. In *The Semantic Web: ESWC 2013 Satellite Events*, pages 125–140. Springer, 2013.

Thomas Klapiscak and Rafael H Bordini. JASDL: A practical programming approach combining agent and semantic web technologies. In *Declarative Agent Languages and Technologies VI*, pages 91–110. Springer, 2009.

Frank Manola, Eric Miller, and Brian McBride. RDF 1.1 Primer, 2014. URL `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/`. (visited on 2014-05-21).

George A Miller. WordNet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Alvaro F Moreira, Renata Vieira, Rafael H Bordini, and Jomi F Hübner. Agent-oriented programming with underlying ontological reasoning. In *Declarative Agent Languages and Technologies III*, pages 155–170. Springer, 2006.

Boris Motik. On the properties of metamodeling in owl. *Journal of Logic and Computation*, 17(4):617–637, 2007.

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles (Second Edition), 2012. URL `http://www.w3.org/TR/owl2-profiles/`. (visited on 2014-08-10).

Denish Mumbaiwala. OBMAS project report - in preparation, 2016.

Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

Celina Olszak and Ewa Ziemba. Approach to building and implementing business intelligence systems. 2:135–148, 2007.

Opencart. OpenCart - Open Source Shopping Cart Solution, 2014. URL `http://www.opencart.com/`. (visited on 2014-08-16).

Kevin Ottens, Nathalie Hernandez, Marie-Pierre Gleizes, and Nathalie Aussenac-Gilles. A Multi-Agent System for Dynamic Ontologies. *Journal of Logic and Computation*, 19(5):831–858, 2009. doi: 10.1093/logcom/exn050. URL `http://logcom.oxfordjournals.org/content/19/5/831.abstract`.

Desanka Polajnar, Mohammad Zubayer, and Jernej Polajnar. A multiagent architecture for semantic access to legacy relational databases. In *Systems Conference (SysCon), 2012 IEEE International*, pages 1–8, March 2012. doi: 10.1109/SysCon.2012.6189521.

Desanka Polajnar, Jernej Polajnar, and Mohammad Zubayer. Autonomous evolution of access to information in institutional decision-support systems using agent and semantic web technologies. In Fatos Xhafa and Nik Bessis, editors, *Inter-cooperative Collective Intelligence: Techniques and Applications*, volume 495 of *Studies in Computational Intelligence*, pages 141–166. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-35015-3. doi: 10.1007/978-3-642-35016-0_6. URL `http://dx.doi.org/10.1007/978-3-642-35016-0_6`.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, 2008. URL `http://www.w3.org/TR/rdf-sparql-query/`. (visited on 2014-03-17).

C. Ramathilagam and M. L. Valarmathi. Article: A framework for OWL DL based Ontology Construction from Relational Database using Mapping and Semantic Rules. *International Journal of Computer Applications*, 76(17):31–37, August 2013. Published by Foundation of Computer Science, New York, USA.

Anand S Rao. AgentSpeak (L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*, pages 42–55. Springer, 1996.

Mariano Rodríguez-Muro and Diego Calvanese. Quest, an OWL 2 QL reasoner for ontology-based data access. In *OWLED*, 2012.

Mariano Rodrıguez-Muro and Diego Calvanese. High performance query answering over dl-lite ontologies. In *Proc. of the 13th Int. Conf. KR*, 2012.

Mariano Rodríguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *The Semantic Web–ISWC 2013*, pages 558–573. Springer, 2013.

Yoav Shoham. Agent-oriented programming. *Artificial intelligence*, 60(1):51–92, 1993.

OpenLink Software. Virtuoso R2RML Support, 2014. URL `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtR2RML`. (visited on 2014-08-31).

I. Sommerville. *Software Engineering.* Pearson/Addison Wesley, 2015.

Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. Bringing relational databases into the semantic web: A survey. *Semant. web*, 3(2):169–209, April 2012. ISSN 1570-0844. doi: 10.3233/SW-2011-0055. URL `http://dx.doi.org/10.3233/SW-2011-0055`.

Quang Trinh, K. Barker, and R. Alhajj. RDB2ONT: A Tool for Generating OWL Ontologies From Relational Database Systems. In *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 170–170, Feb 2006. doi: 10.1109/AICT-ICIW.2006.159.

W3C. Semantic Web - W3C, 2016. URL `https://www.w3.org/standards/semanticweb/`. (visited on 2016-01-29).

Michael Wooldridge. *An Introduction to Multiagent Systems.* Wiley, 2nd edition edition, 2009.

Guohui Xiao, SkomLaebri, Timi Bagosi, Martin Rezk, and Benjamin Cogrel. ObdalibIssues ontop/ontop wiki Github, 2015. URL `https://github.com/ontop/ontop/wiki/ObdalibIssues`. (visited on 2015-09-23).

Glossary

Agent-Assisted Ontology Building An approach which proposes the application of intelligent software agents in assisting human experts in order to build domain ontology.

Ontology Alignment Helper Agent An agent in the system which assists a user to create alignments for interoperability between the reference ontology and an external ontology.

Ontology Builder Agent The chief agent in the system interacting with a user to build and evolve the reference ontology.

Ontology Learner Agent An agent in the system which proactively learns the classes and relations from data and RDB schema.

Ontology Validator Agent An agent in the system which identifies logical conflicts and modelling errors in the prototype system, and interacts with a use to resolve them.

Relational Database A database which employs a relational model to represent the data structure.

Resource Description Framework A class of specifications used to describe design and structure of the information on the web.

Semantic Query Access System An innovative distributed agent-oriented architecture enabling semantic querying over legacy RDB with the help of the intelligent software agents.

Semantic Web Crawler Agent An agent in the system which explore the Semantic Web to find external ontologies for the knowledge domain and gathers existing reusable knowledge from them.

SPARQL Protocol and RDF Query Language A language to describe structured queries for data access from the RDF data store.

Web Ontology Language An common knowledge representation language to describe ontologies.

Abbreviations

AAOB Agent-Assisted Ontology Building.

OAHA Ontology Alignment Helper Agent.

OBA Ontology Builder Agent.

OLA Ontology Learner Agent.

OVA Ontology Validator Agent.

OWL Web Ontology Language.

RDB Relational Database.

RDF Resource Description Framework.

SPARQL SPARQL Protocol and RDF Query Language.

SQAS Semantic Query Access System.

SWCA Semantic Web Crawler Agent.