ONTLOGIES AND INFORMATION SYSTEMS:
A LITERATURE SURVEY

VAN NGUYEN

COMMAND, CONTROL, COMMUNICATIONS AND INTELLIGENCE DIVISION
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

ABSTRACT

An ontology captures in a computer-processable language the important concepts in a particular domain and the relationships between these concepts. Ontologies are becoming increasingly pervasive in various fields of computer and information science. They are indispensable components of many complex information systems, especially systems in which communication among heterogeneous components is critical. I use the following definition of ontology, which captures the essence of the most widely adopted definitions in the field: an ontology is a specific, formal representation of a shared conceptualisation of a domain. The IO Branch of DSTO’s C3ID Division is interested in the possibility of using one or more ontologies to describe computer networks and support automated reasoning about their properties (particularly security properties). This report provides a basic overview of research and development related to ontologies and their use in information systems. The primary goal of the report is to help readers to discover topics of interest and to conduct further investigation of the literature. To this end, besides information about ontologies in general, the report also includes some specific comments about the use of ontologies to model and reason about computer networks and their security.
Ontologies and Information Systems: A Literature Survey

Executive Summary

This report surveys literature relevant to the Information Operations (IO) Branch’s interest in using ontologies to model and reason about computer networks and their security.

The report first clarifies the definition of *ontology* and identifies important types of ontologies. For an ontology to be used, shared and executed, it needs to be presented in some form. In this aspect, the report presents a review of ontology specification languages, from traditional ontology languages to ontology languages designed specifically for the Semantic Web. If the IO Branch decides to develop an ontology, it may need to adhere to some methodology. For this reason the report presents some of the ontology development methodologies that have been proposed, with a special focus on the Onto-Agent methodology, a methodology which is specifically tailored to multi-agent systems. The development of an ontology often calls for the integration of existing ontologies. To this end, the report discusses many facets of ontology integration as well as methods and tools for ontology matching. With respect to heterogeneous ontologies in an open environment, ontology integration can be facilitated by making use of top-level ontologies which define very general concepts that apply across all domains. For this reason, the report includes a description of the most significant projects in top-level ontologies, namely SUMO, Upper Cyc and DOLCE. Regarding the practical use of ontologies, the report discusses storage of ontologies as well as automated reasoning on ontologies, both in external databases and in main memory. Many ontologies are very large, and this can place a heavy burden on ontology development and maintenance, as well as on storage and automated reasoning. Therefore there has been a significant amount of research directed toward the modularisation of ontologies. Part of the literature survey discusses main lines of research in the area of ontology modularisation, including ontology partitioning, ontology module extraction and composition of modular ontologies. As integral parts of an information system, an ontology is expected to evolve in step with the constantly changing application environment and therefore must be maintained over time. Therefore another part of the report discusses the management and evolution of ontologies. The final sections of the report are devoted to a discussion of work specifically related to modelling and reasoning about computer networks.
Van Nguyen
Command, Control, Communications and Intelligence Division

Van Nguyen joined DSTO in 2010 as a research scientist. She is currently working on cyber security. Her current research focuses on the representation of knowledge using formal ontologies. Van obtained her PhD in 2010 from the University of South Australia, with a thesis on the implementation of membrane computing models on reconfigurable hardware.
Contents

1 Introduction 1

2 Introduction to Ontology 3

3 Types of ontologies 4

4 Ontology specification languages 6
   4.1 Traditional ontology languages 7
   4.2 Web ontology languages 7

5 Ontology development methodologies 11
   5.1 The Onto-Agent methodology 11
   5.2 FIPA Ontology Service Specification 14

6 Ontology integration 16
   6.1 Terminology definition 16
   6.2 Major approaches to ontology integration 16

7 Ontology matching 19
   7.1 Types of matching 19
   7.2 Dynamic ontology matching 20
   7.3 Ontology matching algorithms and tools 20
   7.4 Ontology alignment management 21

8 Upper ontologies 24
   8.1 SUMO (Suggested Upper Merged Ontology) 24
   8.2 Upper Cyc 24
   8.3 DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) 27

9 Ontology storage and reasoning 29
   9.1 Ontology storage 30
   9.2 Reasoning on large-scale ontologies 31
   9.3 Ontology reasoning tools 33
## 10 Ontology modularisation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1 Ontology partitioning</td>
<td>34</td>
</tr>
<tr>
<td>10.1.1 Logic-based partitioning</td>
<td>35</td>
</tr>
<tr>
<td>10.1.2 Structure-based partitioning</td>
<td>35</td>
</tr>
<tr>
<td>10.2 Ontology module extraction</td>
<td>36</td>
</tr>
<tr>
<td>10.3 Ontology integration/composition</td>
<td>37</td>
</tr>
<tr>
<td>10.3.1 Distributed description logics (DDLs)</td>
<td>37</td>
</tr>
<tr>
<td>10.3.2 (\varepsilon)-Connections</td>
<td>38</td>
</tr>
<tr>
<td>10.3.3 Semantic importing</td>
<td>38</td>
</tr>
<tr>
<td>10.3.4 Package-based description logics (P-DLs)</td>
<td>38</td>
</tr>
<tr>
<td>10.3.5 Distributed Reasoning Architecture for a Galaxy of Ontologies (DRAGO)</td>
<td>38</td>
</tr>
</tbody>
</table>

## 11 Ontology management

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1 Review of ontology development tools</td>
<td>40</td>
</tr>
<tr>
<td>11.2 Lifecycle of Networked Ontologies (NeOn)</td>
<td>41</td>
</tr>
</tbody>
</table>

## 12 Ontology evolution

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1 Review of techniques for ontology evolution</td>
<td>43</td>
</tr>
<tr>
<td>12.2 Software and tools</td>
<td>44</td>
</tr>
</tbody>
</table>

## 13 Ontologies of information security

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1 Herzog and colleagues’ ontology of security</td>
<td>48</td>
</tr>
<tr>
<td>13.2 Fenz and Ekelhart’s ontology of security</td>
<td>48</td>
</tr>
</tbody>
</table>

## 14 Applications of ontologies to network management

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1 Common Information Model (CIM)</td>
<td>52</td>
</tr>
<tr>
<td>14.2 Ontology-based network management</td>
<td>54</td>
</tr>
</tbody>
</table>

## 15 Application of ontologies to network modelling

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1 Communications Network Modelling Ontology</td>
<td>58</td>
</tr>
<tr>
<td>15.2 An ontology of distributed computing systems</td>
<td>61</td>
</tr>
</tbody>
</table>

## 16 Summary and final remarks

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>References</td>
<td>66</td>
</tr>
</tbody>
</table>
1 Introduction

Ontology, in its original meaning, is a branch of philosophy (specifically, metaphysics) concerned with the nature of existence. It includes the identification and study of the categories of things that exist in the universe. In the last decade, increases in the size and complexity of knowledge bases, computing systems and especially the Internet have necessitated the availability of a mechanism that facilitates communication among heterogeneous components. This has paved the way for the application of ontologies in many disciplines of computer and information science including artificial intelligence and database theory. In this setting, an ontology captures in a computer-processable language the important concepts in a particular domain (such as commerce, engineering or the legal system) and the relationships between these concepts. If otherwise heterogeneous components in an information system all subscribe to the same ontology for a domain, then it is much easier for these components to communicate and interoperate to realise the functionality of a particular application relevant to that domain. In summary, in information systems, ontologies are used mainly for knowledge representation, knowledge sharing, information retrieval, and knowledge management. Most recently, ontologies have been adopted as a central part of the Semantic Web.

The IO Branch of DSTO’s C3ID Division is interested in the possibility of using one or more ontologies to describe computer networks and support automated reasoning about their properties (particularly security properties). This report provides a basic overview of research and development related to ontologies and their use in information systems. The primary goal of the report is to help readers to discover topics of interest and to conduct further investigation of the literature. To this end, besides information about ontologies in general, the report also includes some specific comments about the use of ontologies to model and reason about computer networks and their security.

The contents of the report are as follows.

Sections 2 and 3 clarify the definition of ontology, and identifies important types of ontologies.

Section 4 discusses ontology specification languages, the languages used to represent ontologies in computer-processable form. It examines the important relationship between the expressiveness of a language and its computational efficiency. It also lists the most important languages in use today.

Section 5 motivates the need for ontology development methodologies, and discusses some of the methodologies that have been proposed. This section also presents a specification for the design and development of ontology-based multi-agent systems.

The development of an ontology often calls for the integration of existing ontologies. Achieving this integration requires techniques for the identification of semantic matches between the ontologies. Section 6 discusses ontology integration, while Section 7 discusses techniques and tools for ontology matching.

Section 8 discusses some of the currently available upper ontologies. Such ontologies define very general concepts that apply across all domains (e.g., object, process and event).

In this report, the term Ontology is used to denote the research field, and ontology/ontologies to denote the concrete deliverables of this field.
By extending an upper ontology, a domain ontology is able to inherit the often very rich and important semantic content of the upper ontology. Furthermore, it is easier to integrate domain ontologies that extend a common upper ontology.

Section 9 discusses methods for the storage of ontologies as well as automated reasoning on ontologies, both in external databases and in main memory. It lists the most important automated reasoners currently available.

Many ontologies are very large, and this can place a heavy burden on both storage and automated reasoning. Therefore there has been a significant amount of research directed toward the modularisation of ontologies. Section 10 discusses this research.

As integral parts of an information system, ontologies must be maintained through time. Due to the complexity of the tasks involved, tool support is essential. Sections 11 and 12 discuss the management and evolution of ontologies, and list tools available to manage changes that are made to an ontology through its period of deployment.

Sections 13, 14 and 15 discuss work that is directly relevant to the IO Branch’s interest in using ontologies for modelling and reasoning about computer networks and their security. In particular, it identifies existing ontologies and other related artifacts that could be reused, or adapted for use, by the IO Branch to achieve its goals.

Finally, Section 16 summarises the findings of the report.
2 Introduction to Ontology

Originating in the discipline of philosophy, and increasingly pervading various fields of computer and information science, the concept of an ontology is defined differently in different contexts. For example, Guarino [127] has compiled the following list of definitions:

- An ontology is an informal conceptual system.
- An ontology is a formal semantic account.
- An ontology is a specification of a conceptualisation.
- An ontology is a representation of a conceptual system via a logical theory.
- An ontology is the vocabulary used by a logical theory.
- An ontology is a (meta-level) specification of a logical theory.

In this report, I use the following definition of ontology, which captures the essence of the most widely adopted definitions in the field: an ontology is a specific, formal representation of a shared conceptualisation of a domain [312]. It is specific in that it clearly specifies concepts, relations, instances and axioms relevant to the domain. It is formal in that it is machine readable and interpretable. It is shared in that its content is consented to by the members of a community. It is a conceptualisation in the sense that it is an abstract model of a domain. When applied in computer systems, ontologies are mainly used to support communication (where the communicating agents are humans or computational systems), to support computational inference, and to support the reuse and organisation of knowledge.

Though essentially different, ontologies are closely related to knowledge bases and database schemas. An ontology can be distinguished from a knowledge base in the fact that it is a conceptual structure of a domain while a knowledge base is a particular state of domain. An ontology also separates itself from a database schema in that an ontology is sharable and reusable while a database schema tends to be specific to the domain and is context-dependent; therefore is unlikely to be shareable and reusable. However, as the published work in Ontology to be discussed in this report reveals, the borderlines between ontologies and knowledge bases, database schemas as well as other conceptual models of systems have gradually faded and there are scenarios where all these concepts are studied in a unified way under the umbrella of ‘Ontology’. For instance, an OWL DL ontology is essentially equal to a description logic knowledge base which contains both a TBox (elements of which constitute an ontology), and an ABox (which comprises instances of the ontology).
3 Types of ontologies

A variety of different types of ontologies are considered in the literature. These types may be characterised according to their granularity, formality, generality and computational capability [133].

In terms of granularity, an ontology can be defined as either coarse-grained or fine-grained [133]. Coarse-grained ontologies facilitate the conceptualisation of a domain at the macro-level, and are typically represented in a language of minimal expressivity. Fine-grained ontologies, on the other hand, allow the conceptualisation of a domain at the micro-level, and tend to be represented in a language of significant expressivity.

In terms of formality, ontologies may be classified as being highly informal, semi-informal, semi-formal or rigorously formal [326]. At one end of the formality spectrum, highly informal ontologies are expressed in natural language. At the other end of the spectrum, rigorously formal ontologies are defined in a language with a formal semantics and with desirable computational properties such as soundness and completeness.

In terms of generality, ontologies may be classified as being top-level ontologies, mid-level ontologies, task ontologies, domain ontologies and application ontologies.

- Top-level ontologies (also called upper ontologies or foundational ontologies) are high-level, domain-independent ontologies.
- Mid-level ontologies (also called utility ontologies) serve as a bridge between top-level ontologies and domain ontologies; they serve a purpose analogous to that of software libraries in the object-oriented programming paradigm.
- Domain ontologies specify concepts and inter-concept relations particular to a domain of interest.
- Task ontologies are ontologies developed for specific tasks.
- Application ontologies are ontologies used in specific applications. They typically utilise both domain and task ontologies.

In terms of computational capability, ontologies may be classified as being heavy-weight or light-weight. Light-weight ontologies lack axioms and other constraints, and so are very difficult to reason on. In contrast, heavy-weight ontologies comprise all the necessary elements (such as a rich axiomatisation) for it to be feasible to make inferences about the knowledge they contain.

Ontologies can also be classified according to their expressiveness. For example, ontologies may be controlled vocabularies, glossaries, thesauri, informal is-a hierarchies, formal is-a hierarchies, formal instances relations ontologies, frames ontologies, value restriction ontologies and general logical constraints ontologies [133]. An ontology can also be either a reference ontology (i.e., an ontology used by a system for reference purposes) or a shared ontology (i.e., an ontology that supports the functionalities of a system). A system is considered ontology-driven if ontologies play a central role in the system architecture and
drive various aspects of the system. Otherwise, the system is viewed as being ontology-aware (i.e., the system is aware of the existence of one or more ontologies, and uses the ontologies throughout its execution).

How an ontology should be represented depends on its particular type. I provide an overview of ontology specification languages in the next section.
4 Ontology specification languages

For ontologies to be understood, shared and executed, they need to be represented in some way. For human understanding, they can be expressed in high-level languages such as conceptual graphs [50], semantic networks [291], UML or even natural languages. However, for ontologies to be processable by computer, they must be represented in a computer-readable language (such as OWL and F-logic).

Ontology specification languages can be distinguished according to their level of expressiveness. For example, languages based on higher-order logics are more expressive than languages based on first-order logics, which, in turn, are more expressive than languages based on description logics. A specification language with a higher level of expressiveness allows a more complete representation of knowledge and more sophisticated reasoning. However, it also increases the effort required to specify the ontology as well as the computational costs of performing reasoning on the ontology. In addition, it is usually more difficult for users to understand a highly expressive ontology, since this requires expertise in logic.

Though there is a variety of formalisms for knowledge representation (e.g., vocabularies, narrower/broader relations, formal taxonomies and logics), most existing knowledge representation systems belong to one of two paradigms: description logic-based systems and frame-based systems. These systems are distinguished according to the languages employed to specify the ontologies. In the first paradigm, the ontology language is based on variants of description logics. As subsets of first-order logics, description logics (DLs) constitute a successful family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a formal, structured and well-understood way. Description logic-based systems (DL systems) provide users with highly optimised reasoning procedures, and have acceptable response times for small databases. Though DLs have a sound, complete and decidable inference procedure while retaining reasonable expressive power, reasoning in DL systems is often intractable. In recent research, less expressive subsets of description logics (see Section 10) have been explored in an effort to deliver ontology-based systems with tractable query answering. Nevertheless, ontology languages in this paradigm continue to command the most attention from researchers. The adoption of OWL (a DL-based language) as the ontology specification language for the Semantic Web is perhaps the most notable success of this particular language paradigm. In the second paradigm, ontologies are represented in classical frame-based languages. The primitive modelling entities in frame-based languages are frames and slots. A frame represents a concept, and its slots represent attributes associated with the concept. The slot values can be altered according to the particular situation at hand; a combination of frame instances constitutes a knowledge base. Intuitive syntax (which aids readability and ease of understanding) and inheritance are among the salient features of frame-based languages. Frame-based languages are particularly suitable when the goal of the ontology is imprecisely defined and thus when the generality of the model is more important than its ability to support reasoning for the accomplishment of a specific task [338].

Existing ontology specification languages fall into the following categories: traditional ontology languages and web ontology languages. In the following, I briefly introduce major
ontology specification languages, starting with Ontolingua, one of the most traditional languages, and ending with OWL, the most popular web ontology language.

4.1 Traditional ontology languages

- **Ontolingua** is the language used by the Ontolingua Server [96]. Ontolingua is implemented as a Frame Ontology, which is built on top of the knowledge interchange format KIF (see below).

- **CycL** is a formal language that was first developed in the Cyc Project [192]; it is based on first-order predicate calculus.

- **Open Knowledge Base Connectivity (OKBC) Protocol** [43] is a protocol for accessing knowledge in knowledge representation systems.

- **Operational Conceptual Modelling Language (OCML)** [238] is a frame-based language which allows for operationalisation of functions, relations, rules, classes and instances.

- **Frame Logic (F-Logic)** [166] combines features from both frame-based languages and first-order predicate calculus. It has a sound and complete resolution-based proof theory.

- **LOOM** [210] is based on a description logic. It facilitates knowledge representation and reasoning.

4.2 Web ontology languages

Web ontology languages include OIL, DAML+OIL, XOL, SHOE and OWL. To facilitate interoperability in the web environment, these languages are based on the web standards XML and RDF.

- **Extended Markup Language (XML)** [30] is a markup language which aims to separate web content from web presentation. Although XML is extensively used as a web standard for representing information, its lack of a semantics is often mentioned as one of its major drawbacks.

- **Resource Description Framework (RDF)** [187] is a W3C standard used to describe web resources. Each RDF statement is called a *triple* which consists of *subject*, *predicate* and *object*. An RDF triple can be visualised as a directed graph where the subject and object are modelled as nodes, and the predicate is modelled as a link which is directed from the subject to the object. RDF aims to facilitate the exchange of machine-understandable information on the web.

- **RDF Schema (RDFS)** [21] is a layer built on top of the basic RDF models. RDFS serves as a set of ontological modelling primitives which allows developers to define vocabularies for RDF data.
• Ontology Inference Layer (OIL) [142] was developed in the On-To-Knowledge project. It is based on description logics, frame-based languages and web standards (e.g., XML, RDF and RDFS). It is designed for both describing and exchanging ontologies.

• DAML+OIL [144] is the result of an effort to combine DARPA Agent Markup Language (DAML) and OIL. DAML+OIL is more efficient than OIL in that it includes more features from description logics. However, many frame-based features were removed from DAML+OIL, which makes it more difficult to use DAML+OIL with frame-based tools.

• XML-based Ontology Exchange Language (XOL) [159] was developed as a format for the exchange of ontology definitions.

• Simple HTML Ontology Extension (SHOE) [213] is an extension to HTML that allows the incorporation of machine-readable semantic knowledge into HTML pages.

• Web Ontology Language (OWL) [57] is a standard for representing ontologies on the Semantic Web. It was developed in 2001 by the Web-Ontology (WebOnt) Working Group [343], and became a W3C recommendation in 2004. The design of OWL is based on DAML+OIL and is therefore heavily influenced by description logics, the frame-based paradigm and RDF (see Figure 1). OWL aims to give developers more power to express semantics, and to allow automated reasoners to carry out logical inferences and derive knowledge. As it is not possible to fully achieve both of these objectives (because of the inherent trade-off between the expressive-
ness and computational power of a language), OWL exists in three dialects known as OWL-Lite, OWL-DL and OWL FULL.

- **OWL Full** is the most expressive OWL dialect. Its expressiveness is similar to that of first-order logic.
- **OWL DL** is less expressive than OWL Full, but has a decidable inference procedure.
- **OWL Lite** was designed for easy implementation. It has the most limited expressivity of the OWL dialects (i.e., OWL Lite provides support for classification hierarchies and simple constraints).

Figure 2 illustrates the fundamental roles of XML, RDF, RDFS and OWL in the Semantic Web architecture.

![Figure 2: Various layers of the Semantic Web architecture (source: http://www.obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html).](http://www.obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html)

Different systems may adopt different knowledge representation formalisms, and therefore may encode ontologies using different languages. In an effort to standardise existing work on knowledge representation, languages have been proposed to serve as interchange formats for knowledge representation paradigms. Examples of such languages are Knowledge Interchange Format (KIF) [175] and Common Logic [49]. KIF facilitates the exchange of knowledge among systems by allowing the representation of arbitrary sentences in first-order logic [175]. Common Logic standardises the syntax and semantics of logic-based languages [49].
As mentioned earlier, when selecting an ontology specification language, it is important to be aware of the trade-off between the expressivity and the computational capability of the language. The more expressive the language, the more difficult it is to build a reasoning machine to infer knowledge contained in the ontology. For example, since first-order logic is highly expressive, it can express semantically rich ontologies. However, the expressiveness of first-order logic comes at the cost of an undecidable inference procedure (and thus an inability to provide answers to all possible queries). Conversely, description logics, as decidable fragments of first-order logic, are more efficient than first-order logic to reason with, but are less expressive than first-order logic.

Recently, in response to the growing interest in modular ontologies, other ontology specification languages have been proposed, such as distributed description logics, packet-based description logics and C-OWL. These languages, which have not yet entered the mainstream, will be discussed in Section 10.
5 Ontology development methodologies

Methodologies for the development of ontologies date back to the time of the development of the Cyc ontology, when Cyc developers published their experiences in developing Cyc [192]. Later on, experiences in developing the Enterprise Ontology [327] and the TOVE (TOronto Virtual Enterprise) [125] ontology were also reported. This laid the foundation for the proposal of the first development guidelines soon after [325, 327]. After this first proposal, a series of ontology development methodologies were presented, including Kactus [288], METHONTOLOGY [98, 117], Sensus [316], On-To-Knowledge [304] and CO4 [84]. A comparison of these methodologies can be found in [97].

Responding to the need for development methodologies that facilitate knowledge sharability, reusability and scalability, and that support collaborative and distributed construction of ontologies, the DOGMA and DILIGENT methodologies have been proposed. DOGMA (Developing Ontology-Guided Mediations of Agents) [152, 153] aims to address the sharability, reusability and scalability of ontologies by separating the specification of ontology concepts from the specification of their axioms.

More specifically, according to the DOGMA methodology, an ontology is decomposed into two layers: the ontology base and the ontology commitment. The ontology base formally defines concepts and relationships between concepts in a domain, while the commitment layer allows software agents to define the commitments made by the ontology from their point of view. Each commitment in this layer contains a set of constraints and description rules relevant to a particular subset of the ontology base as well as a set of mappings between ontological elements and application elements. In this way, the concepts and attributes that are common across applications can be kept at the ontology base, which can then be shared and further specialised by an agent (or an application) in the domain through modifications to its commitment layer. The DILIGENT (Distributed, Loosely-controlled and evolving Engineering of Ontologies) [274] methodology proposes a collaborative and distributed approach to the construction of shared ontologies. In brief, this approach requires stakeholders with different viewpoints about the ontology to be constructed to first build an initial ontology. This initial ontology is then made available to users. The users are allowed to adapt the ontology to suit their local environment, but the original shared ontology may not be changed. Once the control board has revised the shared ontology accordingly, users can locally update their own ontologies, and so on.

Although quite a few ontology development methodologies have been proposed, no methodology has emerged as a standard methodology. Rather, groups in the ontology development community either adopt one of the available methodologies or develop their own methodologies.

5.1 The Onto-Agent methodology

A recently published ontology development methodology that is tailored to multi-agent systems is the so-called Onto-Agent methodology [133]. Onto-Agent is a comprehensive methodology for the construction of ontology-based multi-agent systems which unifies the different approaches of existing ontology and multi-agent system design methodologies.
In the Onto-Agent methodology, developers first need to adhere to an ontology methodology which guides the construction of ontologies for the system. The ontology methodology consists of five major phases. The first phase of the methodology (generalisation and conceptualisation of the domain) involves determining the main concepts and inter-concept relationships relevant to the domain, and establishing a framework for the ontology. In carrying out these tasks, developers need to take into account the following important aspects of ontologies: ontology communities (associated communities such as users, developers and agents need to agree and commit to the designed ontology), the purpose of the ontology (i.e., determining whether the ontology is used for communication, information retrieval or problem solving, etc.), the domain of the ontology, the application of the ontology (although an ontology is usually designed for a specific application, it should be kept as application-independent as possible so that the ontology can be shared/reused by different applications). The second phase of the methodology calls for the alignment and merging of ontologies. This phase specifies the steps required to (i) identify and select a suitable ontology merging and alignment tool and suitable ontologies to be reused, (ii) import the selected ontologies into a new ontology development environment and subsequently modify the ontologies to suit specific requirements, and (iii) to identify inter-ontology correspondences before aligning and merging them. Once a high-level design of the ontology has been completed and agreed on, it needs to be formalised so that it can be processed by computers. As already mentioned, it is important to minimise the dependence of the designed ontology on any one application. For this reason, the Onto-Agent methodology makes use of the DOGMA approach which separates the design of the so-called ontology base (which defines the ontology conceptualisation) from the design of the ontology commitments (which formalise domain knowledge) (see Figure 3). The third phase (formal specification of conceptualisation) involves defining the ontology concepts and their relationships in the ontology base. In the fourth phase (formal specification of ontology commitments), developers define rules and axioms for the commitment layer. These rules and axioms give specific interpretations to items in the ontology base that suits the specific requirements of agents and the application. The fifth and final phase is the evaluation of the designed ontology (ontology evaluation).

Once the ontology for the system has been created, the agent methodology guides the developers through the implementation of a multi-agent system. The multi-agent system development methodology consists of five stages. In the first stage, designers define the roles for agents based on their elementary behaviours. To identify the roles for agents, the designers first need to characterise the problem solving process, including the sharing of tasks and results among agents. Based on this characterisation, agent functions and roles are determined. Examples of different types of agents are interface agents (agents that assist a user in the querying process), manager agents (agents that receive requests from interface agents and send requests to information agents), information agents (agents that search for and retrieve information as well as send the requested information to smart agents), and smart agents (agents that analyse and assemble the received information). In the second stage, the designers determine how ontologies should be used in the process of adding intelligence to agents. For example, ontologies may be used to facilitate the decomposition of the overall problem, to assist in the process of retrieval, analysis, manipulation and presentation of information, and to enable communication among the cooperative agents. In the third stage, the organisation and collaboration of agents in
the system is defined. This should be done in such a way that the system’s functions are executed in the most efficient way possible. In some scenarios, a system functions best with simple organisation agents; in other scenarios, a complex system structure may be required. Also, the agents in the system are assigned their roles. It is possible for an agent to have more than one role (e.g., an agent can be of both the manager and smart agent types), and vice versa (e.g., there may be multiple agents of the information type). In the fourth stage, the designers construct individual agents. First, the designers need to devise a list of different agent components (e.g., human interface, agent interface, communication, cooperation, procedure, task, domain, and environment knowledge components). Then individual agents are constructed by implementing the content for an arbitrary combination of agent components. In the fifth and final stage, different aspects of the security of the system (e.g., authentication, availability, confidentiality, non-repudiation and integrity) are implemented. To achieve this, it is necessary to identify important factors related to security requirements such as the most critical agents, security-relevant actions and environmental factors, and parts of the system most susceptible to attack. For example, agents that are exposed to the outside world are more critical with respect to security than agents not exposed in this way.

![Figure 3](image)

**Figure 3:** According to the Onto-Agent methodology, agents define their task knowledge by sharing and specialising the ontology base [133].

Making use of the merits of existing methodologies as well as the experiences gained in developing ontology-based multi-agent systems, the Onto-Agent methodology is expected to advance the state-of-the-art in development for ontology-based multi-agent systems.
Apart from methodologies, such as the Onto-Agent presented above, individuals with an interest in developing a multi-agent system are also assisted by specifications (or standards) that guide the design of different aspects of the system. One specification that is directly relevant to the application of ontologies in an agent-based system is described in the next section.

5.2 FIPA Ontology Service Specification

Established in 1996 as an international non-profit organisation, the Foundation for Intelligent Physical Agents (FIPA)\(^2\) consists of companies and universities which collaboratively work toward producing specifications for agent-based technologies. Many of the specifications proposed by FIPA have been promoted into standards and are ready for commercial deployment. The set of FIPA specifications includes specifications for the communication, transportation and management of agents, for an abstract architecture devoted to agent systems, and for application areas in which FIPA agents can be deployed. In 2005, FIPA officially became part of the group of IEEE standards committees, and has continued devising specifications and standards for agent technologies in the wider context of software development.

FIPA specifications for multi-agent systems make use of a service-oriented model. Underpinning this model is a stack of multiple ‘sub-layer application protocols’ \(^{282}\), as illustrated in Figure 4 (see \(^{282}\) for a description of these layers). As depicted in Figure 4, ontologies form one layer of the stack, and play a vital role in enabling semantic communication among FIPA agents. Unlike many traditional systems, which implicitly encode shared ontologies as procedures in each of the agents involved in a communication, multi-agent systems intended to be deployed in an open and dynamic environment mandate that shared ontologies be declared externally. FIPA has devised a specification, called the FIPA Ontology Service specification, for ontologies that are intended to provide services to a community of agents. Proposed in 2001, the specification has moved into its experimental phase. Although the specification has not yet become a standard, it can serve as a good reference source for organisations wishing to implement ontology services in multi-agent systems in an open environment.

According to the FIPA Ontology Service specification, an agent can be designed around various ontologies, each of which conforms to the OKBC model (see Section 4) and belongs to one of two types: FIPA ontologies or domain ontologies. FIPA (communication) ontologies (e.g., the FIPA-Meta-Ontology and the FIPA-Onto1-Service-Ontology) define speech acts and protocols \(^{105}\), while domain ontologies enable agents to communicate knowledge about the domain in an effort to provide users with application-specific services. It is required that all the ontology services in the system be provided by dedicated agents, called Ontology Agents (OAs). OAs are capable of assisting other agents in ontology-related activities such as discovering and maintaining public ontologies (stored in ontology servers), providing relationships and translating expressions between different ontologies (e.g., ontology matching), and identifying a shared ontology for communication.

\(^2\)http://www.fipa.org/
between two agents. To become an active part of the system, together with other agents, an OA registers its existence and services with the Directory Facilitator (an agent that provides yellow pages services to other agents); in addition, it registers the list of ontologies it maintains as well as its translation capabilities. In this way, agents can query the Directory Facilitator for the specific OA that manages a specific ontology.

According to Briola and colleagues, major efforts to implement FIPA-compliant multi-agent systems include systems implemented on COMTEC, .NET and Jade platforms; a system where an OA is implemented as a web service; and a system where ontology services are provided by a set of dedicated agents working in a collaborative manner. The multi-agent systems described in strictly comply to the FIPA specification in that they support OKBC ontologies, whereas work presented in attempts to meet the Semantic Web standards by targeting OWL ontologies. A majority of the existing work does not implement ontology matching (as one of the services specified by the FIPA specification) because of its high complexity. The two systems that are capable of performing ontology matching are presented in . In , Li and colleagues implement ontology matching via a set of dedicated agents, including an ontology agent, a mapping agent, a similarity agent, a query agent, an integration agent and a checking agent. However, the implementation of ontology services in this work deviates significantly from the FIPA specifications. The work described in, on the other hand, adheres to the specification, and provides ontology matching for OWL ontologies using both classical matching methods and a newly proposed matching method which makes use of SUMO-OWL, an OWL-based upper ontology.

---

Figure 4: An illustration of the Agent Communication Language (ACL) protocol ‘stack’ with respects to the OSI and TCP/IP stacks [282].
6 Ontology integration

Now that ontology development techniques have reached a certain level of maturity, a large number of ontologies have been developed. The existence of many ontologies necessitates techniques for their integration. From a theoretical point of view, the use of a single ontology for communication and knowledge representation seems to be inadequate in open and distributed environments such as the Semantic Web. From a practical point of view, the development of a universal ontology is infeasible, at least for the foreseeable future. A seemingly more suitable approach in this environment is to make efforts to integrate the pre-existing ontologies. These ontologies may overlap (i.e., describe the same (part of a) domain), may be disjoint but treat related domains, or may be expressed in different languages. Consequently, if an application wants to draw on the semantic content of the various ontologies, it needs to integrate them in some way. In this section and the next section, I discuss issues and solutions related to the various facets of ontology integration, with a particular focus on ontology matching.

6.1 Terminology definition

It is perhaps ironic that there is some terminological confusion within the ontology research community, especially with regard to ontology integration. To help mitigate this confusion for the reader, I will now attempt to clearly define what I mean by ontology integration, ontology mapping, ontology matching, ontology merging and ontology alignments in this report.

Ontology integration is an abstract concept which refers to the simultaneous use of multiple ontologies in a particular system or application context. Ontology mapping and merging are specific tasks performed to achieve ontology integration. In particular, ontology mapping refers to concrete attempts to relate the semantics of one ontology with the semantics of another ontology, whereas ontology merging involves creating a new ontology from two or more source ontologies. Ontology matching is a technique underlying all of the above tasks, which aims to find correspondences between the elements of distinct ontologies. Thus ontology matching refers to processes or techniques used to perform ontology mapping/merging for the purpose of ontology integration. Finally, concrete outcomes of ontology matching are called ontology alignments.

6.2 Major approaches to ontology integration

Ontology integration can be achieved in three main ways: by merging ontologies, by mapping local ontologies to a global ontology, and by integrating local ontologies by means of semantic bridges that define mappings between the ontologies. I now discuss the main features of these approaches.

Ontology merging is suitable for use in traditional systems which are small or moderate in size and are fairly static, and where scalability is not a core requirement. In today’s complex, large and dynamic systems, ontology merging can still be applied on groups of
relevant ontologies in the system. In this approach, ontologies for different information sources and systems are merged into a new ontology which includes the source ontologies. This new ontology can then be used to support the activities of various applications, such as extracting, querying and reasoning about information. According to Choi and colleagues [46], current tools that support ontology merging include SMART, PROMPT/Anchor-PROMPT, OntoMorph, HICAL, CMS, FCA-Merge and Chimaera.

In ontology mapping, specific ontologies can be derived from a global (or ‘reference’) ontology. Ontology mapping in this case becomes much easier since concepts in different ontologies that need to be mapped are derived from the same ontology. There currently exist initiatives to develop top-level ontologies which aim to define concepts that are generic to as many domains as possible. For example, the SUMO upper ontology aims to provide concepts that encompass all of the types of entities that exist in the universe. Some believe that the adoption of a single top-level ontology by all ontology developers would enable the controlled semantic integration of ontologies and consequently make possible some of the grander goals of the Semantic Web. There are ongoing research projects both in the development of top-level ontologies (see Section 8) and in the application of these top-level ontologies in large-scale semantic integration. Some of the achievements and difficulties faced in this line of work are reported in [276, 328]. In a complementary method, different local ontologies can be combined into an integrated global ontology. The global integrated ontology provides a unified view to users, who may query the local ontologies via the integrated global ontology [46]. Tools that facilitate this kind of mapping include MOMIS and OIS framework [46].

In the third approach, the local ontologies are likely to be left unchanged, but are linked by semantic bridges (e.g., bridge axioms in first-order logic) that define the mappings between the ontologies. Querying and answering are carried out by the local servers in a cooperative manner. This approach is the most suitable for growing and highly dynamic systems (e.g., distributed agents and the Semantic Web). According to Choi and colleagues [46], tools that support this type of integration include CTXMatch, GLUE, MAFRA, LOM, QOMm ONION, OKMS, OMEN and P2P ontology mapping. Work that is strongly related to this line of research, but which is focused on web ontology technologies, is discussed in Section 10.3.

According to Noy [249], ontology integration is typically carried out in three stages. First, ontology matching is performed to discover the correspondences that exist between concepts in the ontologies to be matched; this can be carried out by the techniques described in the next section. Second, the ontology alignments derived from ontology matching are represented either as instances in an ontology (cf. the Semantic Bridge Ontology of the MAFRA framework), as bridging axioms in first-order logic which represent the transformation required (cf. OntoMerge), or as views that describe mappings from a global ontology to local ontologies (cf. the OIS framework). Finally, the ontology alignments that result are used for various integration tasks, including data transformation, query answering and web service composition.

---

All of the ontology mapping methods presented in this section are infeasible to carry out without prior knowledge of semantic relationships between concepts in different ontologies. This necessitates the task of ontology matching. Since ontology matching has been a very active research area, I will devote the next section to a presentation of an overview of developments in this area.
7 Ontology matching

As already mentioned, ontology matching is a fundamental technique for ontology mapping and integration. It plays a particularly significant role in systems whose agents communicate using ontologies. Ontology matching is considered to be a very challenging task. Not only are ontologies different at the language level (the same term may have different meanings in different ontologies, constructs supported in one ontology specification language may not be supported in another ontology specification language, etc.), but also at the ontological level (different ontologies may be incompatible in terms of granularity, formality, commitment, etc.) [249]. Especially in the context of the Semantic Web, where there is a large number of heterogeneous ontologies, of continuously increasing size, which need to be matched dynamically by software agents, achieving efficient automated or semi-automated ontology matching is considered a formidable task. Nevertheless, given its importance, ontology matching has attracted significant research attention (witness the recently published book on ontology matching [85] and the repository devoted to the topic of ontology matching[6] which contains more than 250 publications, 71 of which published in 2009). The Ontology Alignment Evaluation Initiative (OAEI)[7] conducts yearly reviews to assess, compare and improve on proposed ontology matching systems. For a comprehensive overview of different aspects of ontology matching, the reader is advised to consult the following survey articles: [18, 156, 161, 185, 278, 296, 298, 338]. Many of the mentioned ontology matching techniques inherit features of techniques utilised in more classical contexts, such as schema integration, data warehousing and data integration. For technical details about the algorithms and strategies underlying various ontology matching techniques, please refer to [87, 88, 89, 90].

7.1 Types of matching

Unless the ontologies to be mapped are derived from the same upper ontology, which allows one to refer to the upper ontology when generating the mapping between the ontologies, most of the proposed ontology matching approaches are either heuristic-based (e.g., Hovy [145], PROMPT/AnchorPROMPT [252] and ONION [234]) or based on machine learning (e.g., GLUE [70], IF-Map [155] and FCA-Merge [313]) [249]. Euzenat and Shvaiko [297, 86] have presented a classification of available matching techniques in terms of granularity and input interpretation. According to Euzenat and Shvaiko, available matching techniques may be classified as lexical matchers, structural matchers or semantic matchers. Lexical matchers operate on the terminological level of an ontology; they incorporate string-, language-, and constraint-based matching techniques. Structural matchers consist of graph-based and taxonomy-based techniques. Semantic matchers, commonly referred to as model-based matchers, consider semantic relations between concepts to find matches. See [297] and [86] for a detailed review of all these techniques.
7.2 Dynamic ontology matching

The unique characteristics of the Semantic Web have called for the development of systems that operate in an open and dynamic manner. This has resulted in an emerging line of applications where autonomous entities in the system communicate and process knowledge on the fly (e.g., multi-agent systems, peer-to-peer systems and web services). Provided that the communication is facilitated by ontologies, these systems, in contrast to traditional applications, require run-time ontology matching functionality. A recent trend in the ontology matching field involves making progress toward this dynamic aspect of ontology matching. For example, Shvaiko and colleagues [300] have proposed a dynamic ontology matching method for peer-to-peer systems. This method enables semantic interoperability within the scope of interaction between peers and thus achieves ‘some level of semantic interoperability by matching terms dynamically’ [300]. Shvaiko and colleagues have also identified plausible directions for dynamic ontology matching, including approximate querying and partial matching (which involve trading some of the quality of the results for greater efficiency) and interactive ontology matching (in which multiple agents negotiate ways of dealing with mismatches). They have also proposed specific approaches for dynamic ontology matching, including ‘continuous “design-time” matching’ (in which mappings are updated when the application is idle), community-driven ontology matching (in which the workload of performing ontology matching is distributed dynamically among the agents), and multi-ontology matching (as opposed to pair-wise matching) [300].

7.3 Ontology matching algorithms and tools

While full automation of ontology mapping is considered impractical, semi-automatic techniques that assist ontology mapping using ontology matching are available [126]. In fact, there exists a plethora of ontology matching systems and prototypes. In a recently published book about ontology matching [91], Euzenat and Shvaiko have classified fifty of these matching systems into four groups: (i) systems that focus on schema-level information (schema-based systems), (ii) systems that concentrate on instance-level information (instance-based systems), (iii) systems which exploit both schema-level and instance-level information (schema-and instance-based systems), and (iv) meta-matching systems (i.e., systems which use and combine other matching systems).

Schema-based systems include DELTA [48], Hovy [145], TransScm [231], DIKE [266], SKAT [233], and ONION [234], Artemis [38], H-Match [40], Tess [193], PROMPT/Anchor-PROMPT [252], OntoBuilder [236], Cupid [218], COMA/COMA+ [68], Similarity flooding [228], XClust [189], ToMAS [332, 333, 334], MapOnto [5, 6, 7], OntoMerge [74], CtxMatch/CtxMatch2 [28, 29], S-Match [116], HCONE [178, 179], MoA [167], ASCO [12], BayesOWL and BN mapping [269], OMEN [232] and the DCM framework.

Instance-based systems include T-tree [83], CAIMAN OntologyMatching-Lac, FCA-merge [182], LSD [69], GLUE [70], IMAP [65], Automatch [21], SBI&NB [149, 148], Dumas [23], the system proposed by Wang and colleagues [341], and sPLMap [246, 247].

Schema- and instance-based systems include SEMINT [197, 198], Clio [229, 230, 244, 132], IF-Map [155], NOM and QOM [75], oMap [308], Xu and Embley [81], Wise-Integrator
Meta-matching systems include APFEL [76] and eTuner [286].

There also exist frameworks that provide a set of operations for the manipulation of the alignments which are usually output by ontology matching tools. For example, MOMIS and IOS framework facilitate mapping between a global ontology and local ontologies, MAFRA [219] provides distributed mapping functionality, and FOAM [78] is a general tool for similarity-based ontology matching.

Shvaiko and colleagues [300] cite the following research tools as attempting to deal with dynamic applications: the DCM framework, NOM, QOM, OntoBuilder, ORS, PowerMap and S-Match.

Mapping tools are often developed as extensions of ontology development tools. For example, PROMPT is a plugin for Protégé, and CHIMAERA is an interactive ontology merging tool that is based on the Ontolingua ontology editor.

7.4 Ontology alignment management

I have discussed numerous attempts to devise algorithms and develop tools for ontology matching. Two further topics merit discussion: the application of ontology matching tools in practice, and the management of their output: ontology alignments. The rest of this section is devoted to a brief discussion of these topics, a large part of which is based on the relevant discussion by Euzenat and colleagues in [94].

As mentioned previously, many semantic applications, especially in an open environment like the Semantic Web, involve the use of multiple ontologies. Hence, it is natural to view ontology matching as a vital process in, and the outcomes of this process (ontology alignments) as indispensable components of, the development and functioning of these applications. Specific applications that necessitate the use of ontology alignments include: ontology evolution (where alignments are used to record changes between two versions of an ontology), schema/data integration (where alignments are used to facilitate the integration of the schema and contents of different databases), P2P information sharing (where alignments are used to capture relations between ontologies used by different peers), web service composition (where alignments are used to compose a web service by combining service interfaces described by different ontologies), multi-agent communication (where alignments are used to facilitate communication among agents using different ontologies), and query answering (where alignments are used to translate user queries). Many of these applications require ontology alignments to be used at design-time (e.g., ontology evolution, schema/data integration and ontology merging), while others demand the run-time deployment of ontology alignments (e.g., P2P information sharing, query answering, multi-agent communication and web service composition).

Design-time ontology alignment activities include the retrieval of stored alignments in order to integrate different ontologies and the creation of alignments between ontologies, which is done using semi-automated matching tools (e.g., Protégé/PROMPT) or manually using alignment editors (e.g., Protégé/PROMPT or VisON[8]. When alignments are

8http://www.iro.umontreal.ca/owlola/visualization.html
generated, they are either stored locally or in external servers for possible later reuse. The former is suitable for a ‘closed’ system, whereas the latter is more appropriate for applications in an open context. In particular, the latter is more desirable in situations where the alignments are between commonly used ontologies.

Run-time ontology alignment activities include the retrieval and manipulation of stored ontology alignments at run-time (e.g., an agent may wish to retrieve certain ontology alignments and have them aggregated or trimmed under a threshold). Since the techniques for dynamic ontology matching that have been proposed so far are not yet ready for practical use, it is often the case that applications which use ontology alignments at run-time can achieve adequate efficiency only if they make use of existing ontology alignments provided by an alignment server.

As an independent software component, an alignment server offers services for both design-time and run-time use of ontology alignments. It includes a library of matching methods and an alignment store. The alignments maintained in the store are expected to be carefully evaluated and certified. Also, applications that are interested in using the stored alignments should be able to discover, access and share the services provided by the alignment server. Applications that need to obtain ontology alignments at design-time can connect to the alignment server in a loosely coupled manner, whereas applications that require the use of ontology alignments at run-time must directly invoke the appropriate services offered by the alignment server in order to obtain/manipulate the required alignments. An example of such a system is the Alignment Server, coupled with the Alignment API. The Alignment Server, which can be accessed and used by clients through the Alignment API, offers ontology matching services as well as alignment manipulation, storage and sharing services.

No matter whether the alignments generated by an ontology matching process are stored locally or externally (e.g., in an alignment server), tools are needed to support the management of the stored ontology alignments. In cases where the alignments are stored locally, this support should be provided in the development environment. If an external alignment store is used, support for ontology alignment management is the responsibility of the alignment server. An infrastructure that supports ontology alignment management may, for example, provide the following services: the matching of two ontologies, the storage of an alignment, the retrieval of an alignment, the retrieval of alignment metadata, the suppression of an alignment, the discovery of stored alignments, the editing of an alignment, the trimming of alignments, the generation of code, the translation of a message, and the discovery of a similar ontology.

Although currently there are no tools available that are capable of managing the whole ontology alignment process, there do exist tools that provide partial support. These tools are listed below.

- **MAFRA** offers the ability to create, manipulate, store and process alignments (in the form of ‘semantic bridges’).
- **Protégé** offers support for ontology matching at design-time through the use of the PROMPT suite. The alignments can be stored and shared through Protégé server mode.

---

9http://alignapi.gforge.inria.fr
• **FOAM** serves as a framework in which matching algorithms can be integrated. FOAM is available as a Protégé plug-in, and is integrated into the KAON2 ontology management environment.

• **COMA++** provides an environment for the integration and composition of matching algorithms (similar to FOAM). It supports the evaluation, editing, storage and processing of ontology alignments.

• **Web Service Modelling Toolkit (WSMT)**\(^{10}\) is a stand-alone system that serves as a design-time alignment creator and editor.

• **NeOn** is a proposed toolkit for ontology management which provides run-time and design-time ontology alignment support. See Section 11 for more information about NeOn.

\(^{10}\)http://wsmt.sourceforge.net
8 Upper ontologies

One way to facilitate interoperability and integration of ontologies is to make use of upper ontologies. It is widely believed that mapping/linking domain ontologies is easier to accomplish if the ontologies are derived from a common upper ontology. An upper ontology can be used either in a top-down fashion (where an upper ontology is used as the foundation for the design of a domain ontology) or in a bottom-up fashion (where a new or existing domain ontology is mapped to the upper ontology). Upper ontologies differ from each other with respect to the expressivity of the knowledge representation language that encodes its content, and with respect to the ontological choices, assumptions and commitments they make. There are several ongoing research projects aimed at the development of standard upper ontologies. Three of the most prominent upper ontologies are called SUMO, Upper Cyc and DOLCE.

8.1 SUMO (Suggested Upper Merged Ontology)

SUMO is currently managed by the IEEE SUO (Standard Upper Ontology) working group. SUMO was originally formed through the merging of several already existing top-level ontologies. SUMO aims to facilitate data interoperability, communication, and information searching and inference. It is written in Standard Upper Ontology Knowledge Interchange Format (SUO-KIF), a variation and simplification of KIF (see Section 4). One advantage of SUMO is that, not only is its content maturing with time, but it has also been extended with many domain ontologies (e.g., ontologies for communications, distributed computing, engineering components, transportation and viruses) and a complete set of links to the lexical database WordNet. SUMO defines high-level concepts such as Object, Process, Quantity and Relation, as well as axioms in first-order logic that describe properties of these concepts and relations among them. SUMO and its related domain ontologies (there are currently 21 of these) comprise one of the largest public formal ontology resources in existence today. It has found application in linguistics, knowledge representation and reasoning. Figure 5 illustrates a subset of the SUMO top-level categories.

8.2 Upper Cyc

Upper Cyc is the upper level of the Cyc ontology. The largest and oldest ontology, Cyc is proprietary and primarily aims to support AI applications. Upper Cyc is implemented in CycL, and is a part of the Cyc Knowledge Base. The Cyc Knowledge Base (KB) is a formalised representation of facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday life. The KB consists of terms and assertions which relate those terms. These assertions include both simple ground assertions and rules. The Cyc KB is divided into thousands of ‘microtheories’, each of which being focused on a...
Figure 5: A subset of the SUMO top-level categories ([244], cited in [292]).
Figure 6: A subset of the Upper Cyc top-level categories ([54], cited in [292]).
particular domain of knowledge, a particular level of detail, a particular interval of time, etc. It has found application in natural language processing, network risk assessment (cf. CycSecure\textsuperscript{14} and terrorism management. Figure\textsuperscript{6} illustrates a subset of the Upper Cyc top-level categories.

8.3 DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering)

DOLCE\textsuperscript{15} was developed as part of the WonderWeb project. It was the first module in a library of WonderWeb foundational ontologies. DOLCE is a single ontology, and serves as a reference module for the library \textsuperscript{292}. DOLCE, which is currently available in KIF, is intended to capture the ontological categories underlying natural language and human commonsense. According to DOLCE, different entities can be co-located in the same region of space-time. DOLCE is an ontology of instances, rather than an ontology of universals (universals are entities that have instances). DOLCE-Lite\textsuperscript{16} encodes the basic DOLCE ontology into OWL-DL, and adds eight pluggable modules — including collections, social objects, plans, spatial relations and temporal relations — to it. DOLCE has been used for multilingual information retrieval, web-based systems and services and e-learning. Figure\textsuperscript{7} illustrates a subset of the DOLCE top-level categories.

An evaluation and comparison of SUMO, Upper Cyc and DOLCE is given in \textsuperscript{292}.

Since SUMO and DOLCE have their own advantages and disadvantages, there is an effort to integrate DOLCE and SUMO into what is called SmartSUMO \textsuperscript{258}. Other available upper ontologies include Basic Formal Ontology (BFO) \textsuperscript{10}, General Ontology Language \textsuperscript{112}, Sowa’s top-level ontology \textsuperscript{302}, Penman Upper Model \textsuperscript{17}, Object-Centered High Level Reference Ontology (OCHRE) \textsuperscript{287}, the Bunge-Wand-Weber (BWW) ontology \textsuperscript{339}, and, most recently, Yet Another Top-level Ontology (YATO) \textsuperscript{235}.

There are also efforts to develop upper ontologies that help realise some of the goals of the Semantic Web. For example, upper ontologies have been developed to address the specific problem of situation awareness (see \textsuperscript{19} for a survey of these ontologies).

Although it is perhaps too early to judge whether a successful standard upper ontology can be developed, it is nevertheless frequently recommended that ontology developers use a top-level ontology to guide the development of domain ontologies. One major reason for this is that, by building a domain ontology as an extension of a top-level ontology, one can inherit all of the relevant semantic content of the top-level ontology with minimal effort.

\textsuperscript{14}http://www.cyc.com/applications/cysecure
\textsuperscript{15}http://www.loa-cnr.it/DOLCE.html
\textsuperscript{16}http://wiki.loa-cnr.it/index.php/LoaWiki:Ontologies#Modules_of_the_DOLite.2B_Library
Figure 7: A subset of the DOLCE top-level categories ([226], cited in [293]).
9 Ontology storage and reasoning

I have discussed various aspects of ontologies, as well as the development and integration of ontologies. The question now is probably how ontologies are used in practice. For ontologies to be used in practice, they need to be stored in repositories and, if required, to be reasoned on (by a reasoner). This section gives an overview of existing ontology repositories and reasoners. In particular, it discusses current research and development of ontology repositories and reasoners from the point of view of scalability.

Once ontologies have been developed, they need to be stored so that agents and other system components which are interested in using the ontologies can access and reason on them to obtain desired knowledge. Since traditional ontologies were typically quite small, the simple storage and reasoning mechanisms provided by some traditional ontology development tools (e.g., simple text files) were adequate. However, with the increasing adoption of ontologies for communication and knowledge management, together with the steady growth in the size of ontologies, comes a need for more efficient and scalable mechanisms for ontology storage and reasoning. Since description logics are very popular languages for specifying ontologies, most existing practical work related to scalable reasoning is devoted to description logics (more specifically to OWL). Description logics (DLs) are decidable fragments of first-order logic. A description logics (DL) knowledge base typically consists of two components, a TBox and an ABox. The TBox, which corresponds to the DL ontology, describes the terminology (e.g., concept definitions), while the ABox contains assertions about individuals (i.e., knowledge about the individuals of the domain). As such, reasoning in DL systems includes TBox reasoning (i.e., reasoning with concepts) and ABox reasoning (i.e., reasoning with individuals). The main inference procedures with TBoxes are concept subsumption and concept satisfiability. As for ABoxes, the main reasoning tasks are ABox consistency and instance checking. Since the ABox of a DL knowledge base contains factual data, it is often much more dynamic and larger in size in comparison to the TBox. Therefore, while existing DL reasoners are able to cope with TBox reasoning, ABox reasoning for real-world ontologies is often beyond the capability of existing reasoners. This has inspired research on scalable reasoning aimed at providing techniques, algorithms and systems for optimised ABox reasoning.

Scalable reasoning has been investigated along different dimensions. These dimensions target different aspects of ontologies as part of an overall quest for more efficient reasoning on large-scale ontologies. The first dimension directs its focus on the reasoning capability of ontology specification languages. Since there is a trade-off between the expressiveness and computational capability of ontology specification languages (e.g., OWL DL is more expressive than RDF(S), but reasoning on OWL DL is too complex in many large-scale contexts [138]), efforts in this dimension aim at determining fragments of existing ontology specification languages that have the potentials to speed up the reasoning process while being expressive enough to capture what needs to be represented. Thus an appropriate balance between expressiveness and computational cost can be achieved. More specifically, since inference in OWL-DL is intractable in the worst case, researchers have attempted to identify fragments of description logics that can serve as less expressive but more computationally efficient ontology specification languages for ontology storage and reasoning. For example, Calvanese and colleagues [35] have proposed the DL Lite family of languages for tractable reasoning and effective query answering over a very large ABox.
Other description logics with polynomial data complexity include Horn-SHIQ [147, 180] and description logic programs [123].

Based on the premise that ontology repositories can benefit from the application of well-established techniques used to promote the scalability and performance of traditional databases, research work in the second dimension combines ontology repositories and databases. Although one of the goals of this work is to devise a scalable reasoning method for large-scale ontologies, most developments in the second dimension have made use of centralised ontology repositories, which are not very suitable in a distributed environment. Work in the third dimension responds to the problem of distribution by proposing an efficient means of utilising large-scale ontologies in an open and distributed environment according to which an ontology is decomposed into or formed from smaller modules.

The rest of this section mainly discusses research work in the second dimension of ontology storage and reasoning. Aspects of the third dimension are briefly mentioned at the end of the section, and are further elaborated in Section 10. In the following discussion, I will start with an overview of major paradigms for ontology storage.

9.1 Ontology storage

There are two major types of ontology stores: native stores and database-based stores.

Native stores are built on top of the file system, while database-based stores use one or more (relational or object-relational) databases as backend stores. Examples of native stores include OWLIM [169, 262], HStart [44] and AllegroGraph [3]. Native stores are themselves divided into two categories: triple file-based stores (cf., OWLIM and AllegroGraph) and hierarchy stores (cf., HStar). Time efficiency in loading and updating ontologies is the key advantage of using native stores.

In database-based stores, the ontology repositories are built on top of the storage and retrieval mechanisms of databases. In this way, not only do database-based stores provide seamless access to both the ontology repository and the database, but also can profitably make use of the available techniques of relational databases relating to transaction processing, query optimisation, access control, logging and recovery. In comparison with native stores, although database-based stores obviously increase the ontology loading and updating time, they can significantly reduce the query response time. Most current research on ontology storage and reasoning concentrates on database-based stores. Currently, there are three major types of database-based stores: generic RDF stores (e.g., Jena [346] and Oracle RDF [240]), improved triple stores (e.g., Minerva [351] and Sesame/MySQL [33]), and binary-table-based stores (e.g., DLDB-OWL [270] and Sesame/PostgreSQL [33]). Figure 8 illustrates major types of ontology stores.

Since, from a representational perspective, ontologies are essentially directed labelled graphs, they are typically stored in triple tables, where each triple stores elements that represent a subject, a property and an object. Repositories that store ontologies in this fashion are called generic RDF stores. Examples of generic RDF stores are Jena and Oracle RDF. Improved triple stores, as their name suggests, aim to improve on the efficiency of generic RDF stores. Examples of improved triple stores are Minerva and Sesame/MySQL.

---

17 A large part of the discussion in this section is based on [141].
Improved triple stores maintain different types of triples (e.g., typeOf and subclassOf triples) in separate tables; this essentially breaks up self-joins in a big triple table into joins among small-sized tables. Another technique to improve data access for queries is to decrease the traversal space. This technique is adopted in binary-table-based stores such as DLDB-OWL and PostgreSQL. In this approach, a table is created for each class in the ontology to be stored; this enables direct access to the classes and properties that are relevant to the queries. However, a side effect of this approach is that the database schemas of binary-table-based stores are sensitive to the structure of the ontology, and therefore need to be altered if the ontology changes. Binary-table-based stores are therefore not suitable for large-scale ontologies, since a very large number of classes may need to be created for the concepts of the ontology.

9.2 Reasoning on large-scale ontologies

Many database-based stores achieve scalable reasoning by eliminating ontology inferencing during query answering. The rationale behind this approach is the trading of space for time: inferencing is done when an ontology (e.g., an OWL document) is imported into a database, and the inferred results are then materialised in the database together with the original ontology assertions. Since inferencing is already done at loading time, the query response time is reduced, because no inferencing is performed when a query is made. One typical example of such a system is Minerva. Minerva aims to provide practical reasoning capability and high query performance for realistic applications. Minerva uses database-based stores for ontology storage, and uses a combination of DL reasoners and rule inference for knowledge inference (see Figure 9). More specifically, it uses a DL reasoner (which can be the built-in reasoner or an external reasoner such as Pellet) for TBox inference and rule logic for ABox reasoning. This combination aims to improve on the reasoning limitations of each type of reasoners: rule logic is unable to infer subsumption relations on the TBox, whereas standard DL reasoners face difficulties when reasoning on a large number of instances in the ABox. Implicit knowledge is derived in advance when the ontology is loaded/parsed for both the TBox and the ABox (using the DL reasoner and rule logic); it is then materialised in the back-end database. In such a scenario, knowledge querying does not involve reasoning; rather, it involves only the retrieval of...
Figure 9: An illustration of major modules in the implementation of the improved triple store Minerva [147].

the inferred knowledge from the database. This significantly reduces the query response time for database-based stores. Unlike in-memory reasoning approaches, this technique eliminates the ‘out of memory’ problem that may arise when large-scale ontologies are processed. However, since in this approach each query involves accessing the hard disk (a very expensive IO operation), the performance is in general low compared to approaches with in-memory reasoning.

As already mentioned, materialising inferred results in the database for possible retrieval later has been proven to improve the efficiency of query processing. Nevertheless, maintaining, updating and deleting the materialised results can be a non-trivial problem.

Fokoue and colleagues [108] have proposed an alternative technique in which the ABox is stored in a traditional relational database, and inferred results are not materialised in the database at loading time. This approach is founded on the well-known fact that querying over DL ontologies (ABoxes) is reducible to a consistency checking problem. Starting from an observation that an ABox often contains many redundant assertions from the perspective of consistency checking, Fokoue and colleagues have presented an algorithm that aggregates relevant individuals and assertions into a summary ABox. Consistency checking performed on an individual/assertion in a summary ABox is equivalent to consistency checking carried out on a set of corresponding individuals/assertions in the original ABox. As shown in the reported empirical evaluation, this approach achieves a significant reduction in the space and time requirements of consistency checking (e.g., a consistency check on an ABox with 1 106 858 individuals and 6 494 950 role assertions only requires a consistency check on 4 045 individuals and 2 942 role assertions). Motivated by mature optimisation techniques in disjunctive Datalog programs, another approach toward scalable reasoning on an ABox is to convert a DL knowledge base into a disjunctive Datalog program, and use a deductive database to reason over the ABox [147].

The aforementioned work is oriented toward the creation of optimised techniques, algorithms and systems for scalable reasoning for large-scale ontologies stored in secondary
storage. Guo and Heflin [130] address the problem of scalable reasoning from a different perspective. They take advantage of highly optimised in-memory reasoners such as Pellet and Racer by decomposing a large and expressive ABox into smaller modules that can be separately input into these reasoners. The composition is done in such a way that the answer to a conjunctive query over the original ABox is the union of the answers of the queries over the smaller modules. This method is strongly related to ontology modularisation (see Section 10). More information about Guo and Heflin’s work can be found in [130].

9.3 Ontology reasoning tools

There exist many highly optimised reasoners, some of which are discussed below. Most of these reasoners are based on either tableau-based or resolution-based reasoning algorithms and make use of main memory for storage (KAON2 and Ontobroker are exceptions).

Cereb19ra is a commercial reasoner based on C++. It provides a tableau-based decision procedure for general TBox and ABox reasoning, and supports OWL API.

FACT++20 is a free open-source reasoner based on C++ for the SHOIQ description logic (a variation of OWL-DL). It provides a tableau-based decision procedure for general TBox and partial ABox reasoning, and supports Lisp-API and DIG-API.21

KAON222 is a free (for non-commercial use) Java-based reasoner for SHIQ description logic. It provides a resolution-based decision procedure for general TBox and ABox reasoning. KAON2 can operate either using main memory or using a deductive database. It provides its own Java-based interface, and supports DIG-API.

Ontobroker [59] is a commercial Java-based deductive, object-oriented database engine and querying interface for F-Logic ontologies. It can operate either using main memory or using a relational database. It also supports KAON2 API.

Pellet23 is a free open-source Java-based reasoner for SROIQ description logic. It provides a tableau-based decision procedure for general TBox and ABox reasoning, and supports OWL-API, DIG-API, and the Jena interface.

RacerPro24 is a commercial Lisp-based reasoner for SHIQ description logic. It provides a tableau-based decision procedure for general TBox and ABox reasoning and supports the OWL-API and the DIG-API.

---

18Tableau- and resolution-based methods prove a theorem by showing that the negation of the statement to be proved is inconsistent.
19http://www.cerebra.com/
20http://owl.cs.manchester.ac.uk/fact++/
21A DIG is an interface that provides uniform access to Description Logic reasoners. For more information, see http://dig.sourceforge.net/.
22http://kaon2.semanticweb.org/
23http://pellet.owldl.com/
24http://www.racer-systems.com/
10 Ontology modularisation

With the success of the World Wide Web, users have been increasingly exposed to an overdose of information and knowledge. In this context, providing users with the most relevant information in the most efficient way helps them to increase their productivity. However, this is a major research challenge, since most existing ontologies specify knowledge across various domains in a monolithic fashion. Furthermore, existing ontologies are continuously growing in size. Not only is this a hindrance to the management and maintenance of the ontologies, processing large ontologies exceeds the capability of state-of-the-art reasoners (which treat ontologies as monolithic entities). Furthermore, the proliferation of existing ontologies and the expected future rapid development of new ontologies necessitates efficient mechanisms for the integration and composition of ontologies. All of these practical concerns have given rise to an active branch of research in applied ontology focused on the development of modular ontologies.

Ontology modularisation was in fact first considered by the developers of Cyc. Modularisation in Cyc is manifested in the fact that the Cyc ontology is divided into so-called microtheories. However, Cyc microtheories, as well as most other types of modules in existing ontologies, are considered as operative mainly at the syntactic rather than at the more important semantic level. For example, an OWL ontology can be specified in different modules, which can then be integrated using the `owl:imports` construct. However, the import function implemented by this construct is essentially just a copy-and-paste function: it merges all of the ontologies participating in the import into a unique reasoning space. Thus what is processed by the reasoner is similar to a singular, monolithic ontology. In addition, `owl:imports` allows only the import of an ontology as a whole even though in practice it is often the case that only a certain part of the imported ontology is of interest to the importing ontology. What is needed is semantic modularisation; it is this fact which motivates investigations into ontology modularisation.

Ontology modularisation is primarily aimed at providing users of ontologies with the knowledge they require with as narrow a scope as possible, offering a query response to users at an acceptable rate, and making the design, development, management and maintenance of ontologies easier.

Currently, most research on ontology modularisation is devoted to the web ontology language OWL. In particular, there are two main lines of research: modularisation (decomposition) of an existing ontology, and integration (composition) of existing ontologies. The former line of research is further divided into two sub-areas, namely ontology partitioning (partitioning of an existing ontology into modules) and ontology module extraction (extracting a part of a given ontology). Researchers in the latter line of research also work in two main areas: the integration of existing ontologies, and the proposal of formalisms for the development of modular ontologies.

10.1 Ontology partitioning

Ontology partitioning concerns the decomposition of a given ontology into smaller modules (see Figure 10). Ontology partitioning can be achieved in either a logic-based or structure-based manner.
10.1.1 Logic-based partitioning

In [53], Grau and colleagues aim to extract a module from an ontology by partitioning the ontology into locality-based modules. Each module contains axioms related to the concepts in the module. This approach was the first serious attempt to partition an ontology based on logical criteria and is still a cornerstone for theoretical work in this area [310]. The algorithm for the proposed approach has been implemented (using Manchester’s OWL API) and evaluated. It has been found that the algorithm is able to produce very small modules for simple ontologies (e.g., SUMO), but much larger modules for more complicated and densely interconnected ontologies (e.g., DOLCE).

10.1.2 Structure-based partitioning

In contrast to the work of Grau and colleagues, the PATO system proposed by Stuckenschmidt and Klein [309] takes a structure-based approach to ontology partitioning. Stuckenschmidt and Klein view the class hierarchy of an ontology as a directed acyclic weighted graph with edges representing the relations between the classes. The graphical representation of the ontology can then be partitioned in such a way that the resulting modules are more strongly internally connected than externally connected. This partitioning method has been applied in a knowledge selection scenario where a semantic web browser plugin called Magpie automatically selects and combines available online ontologies in order to identify instances of concepts. In this scenario, partitioning is used to extract relevant modules from the selected ontologies. The downside of the structure-based approach, in comparison to the logic-based approach, is that the logical properties of the resulting modules (and hence the consistency and coherence of their semantic content) cannot be guaranteed. An upside of the approach is that it can be applied to a diversity of ontology models including simple taxonomies. The PATO system can also be employed in
different applications thanks to its capacity to adapt to the criteria that determine the modularisation.

A similar partitioning idea has also been adopted by MacCartney and colleagues [215]. Appealing to previous theoretical work on partition-based reasoning, MacCartney and colleagues have proposed a graph-based partitioning algorithm and applied it in a first-order logic theorem prover called SNARK. Experimental results indicate that a significant performance improvement can be achieved using their technique.

### 10.2 Ontology module extraction

Researchers working on module extraction by traversal also view an ontology as a graph. However having a different goal from those devoted to ontology partitioning, approaches in this category [22, 290, 257] do not break an ontology into smaller modules. Instead, they traverse the ontology, and return a part of the ontology that is relevant to a given class (concept) (see Figure 11). This is motivated by the fact that it is often the case that users only need to reuse a small part of a large reference ontology in their work. It is understood that the extraction of part of a reference ontology should be done according to the specific needs of the users rather than rely on the initial author’s decomposition (i.e., some existing standard ontology decomposition). A typical scenario is that when developing an application, a developer reuses knowledge from multiple existing ontologies. Consequently, the developers need an ability to extract parts of ontologies that are relevant and self-contained for use in their applications. In regards to reasoning, query processing can be substantially improved by querying ontology modules instead of querying complete ontologies. Roughly speaking, these approaches usually start at a node representing a class (a concept), and then follow the edges associated with the node to obtain all the subsequent nodes to extract [290].

Noy and Musen have implemented an ontology extraction method as a plugin to the Protége ontology development environment (via the PROMPT suite). This approach focuses on defining a specific view of an ontology via the so-called traversal view. In this approach, the user can specify the concepts of interest, the relationships to traverse to find other concepts, and the depth of the traversal [257]. Bhatt and Wouters [22] have presented an approach to distributed sub-ontology extraction in their Materialised Ontology View Extractor (MOVE) system. MOVE offers an architecture for parallel processing which can achieve optimum performance using around five separate processors. A similar approach presented in [290] targets large ontologies (with over 1000 classes and dense connectivity). Seidenberg’s approach [290] traverses an OWL ontology by following its typical modelling constructs, and produces highly relevant segments. The approach also aims to improve on several aspects of previously proposed methods for ontology extraction (e.g., the ontology segments can be transformed on the fly during the extraction process to respond to specific requirements of an application).

Apart from ontology partitioning and extraction (by traversal), there are several approaches (cf., SparQL [289], KAONs [336] and RVL [221]) that adopt well-establish query techniques in the database field for the purpose of obtaining segments from an ontology. These approaches are referred to as query-based methods [290]. Query-based methods retrieve a segment from an ontology using queries defined in an SQL-like syntax. However,
the query-based methods are suitable only for users who are interested in obtaining very small and non-permanent segments of an ontology for a single-use purpose.

10.3 Ontology integration/composition

Ontology integration/composition is concerned with methods of representing and reasoning about a set of ontology modules or a set of existing ontologies. To be able to deal with vast amounts of information that cannot be captured in a singular monolithic ontology, researchers investigate formalisms for the specification of a virtual ontology built out of the content in distributed ontologies representing local (context-specific) knowledge of the domain. To this end, families of modular ontology languages have been proposed, including distributed description logics, package-based description logics, $\varepsilon$-Connections, and semantic importing. In another trend, the notion of conservative extension plays a key role [113, 119, 118, 120]. In this approach, ontology modules which share the same global interpretation domain are combined in such a way that the integrated ontology is a conservative extension of the component ontologies. Following is a brief discussion of modular ontology specification languages.

10.3.1 Distributed description logics (DDLs)

Extending description logics (DLs), and inspired by distributed first-order logics (DFOLs), distributed description logics (DDLs) [25] provide a mechanism for combining distributed knowledge bases in a loosely coupled manner. More specifically, with DDLs, a modular ontology is formally represented as a set of ontology modules which are pairwise inter-related by bridge rules. Bridge rules allow an ontology to access and import knowledge contained in other modules. The DDL formalism has been incorporated into ConTeXt Markup Language (CTXML) [25], and has been added as an extension (called C-OWL
of the OWL ontology specification language. A distributed tableaux algorithm for reasoning in DDLs for ontologies with restricted expressivity has also been proposed; it is implemented in the distributed reasoner DRAGO (see below).

10.3.2 $\varepsilon$-Connections

Driven by the fact that the early versions of DDLs and C-OWL have several limitations (e.g., C-OWL provides no reasoning support, and DDLs are not expressive enough in some scenarios and not intuitive enough in the way they represent inter-ontology subsumption links), another formalism called $\varepsilon$-Connections [121, 122] for representing and reasoning with distributed ontology modules has been proposed. The $\varepsilon$-Connections framework is a formalism for combining OWL ontologies, which facilitates developers in developing web ontologies in a modular way, and provides an alternative to the owl:imports construct. It can be applied in two main ways. First, $\varepsilon$-Connections can be employed as a framework for the combination and integration of ontologies. Second, $\varepsilon$-Connections can be used to decompose a large heterogeneous ontology or knowledge base into a set of interconnected smaller homogeneous ontologies. To ensure localised semantics, $\varepsilon$-Connections assumes that the domains of the smaller ontologies are disjoint. Both the SWOOP ontology editor and the Pellet reasoner incorporate support for $\varepsilon$-Connections.

10.3.3 Semantic importing

Another approach that aims to eliminate obvious problems associated with the syntactic importing mechanism of the OWL language is called semantic importing. Semantic importing [268, 13] facilitates partial ontology reuse by proposing a new primitive to support the semantic importing of ontologies. It allows the semantic importing of classes, properties and individuals from source ontologies. It also provides TBox reasoning support in scenarios where an ontology semantically imports vocabularies from another ontology.

10.3.4 Package-based description logics (P-DLs)

In a P-DL ontology [15], the whole ontology is composed of a set of packages, each of which has its own local interpretation domain. As such, P-DLs support contextual reuse of knowledge from multiple partially overlapping ontology modules by allowing contextualised interpretation (i.e., interpretation from the perspective of a specific package [15]). The development of a distributed algorithm for reasoning in P-DL is currently under consideration.

10.3.5 Distributed Reasoning Architecture for a Galaxy of Ontologies (DRAGO)

DRAGO was proposed by Serafini and Tamilin in [293] as a distributed reasoning system. The rationale of DRAGO is to address the problem of reasoning on multiple ontologies
interrelated via semantic mappings. In the DRAGO system\textsuperscript{25} these ontologies are distributed in a network of DRAGO Reasoning Peers (DRPs) in a peer-to-peer manner (see Figure 12). Each DRP hosts a standard DL reasoner (e.g., FaCT, RACER or Pellet). At the centre of the DRAGO system is a distributed reasoning algorithm whose output is a global reasoning outcome that is the result of combining local reasoning outcomes obtained for multiple distributed ontologies according to semantic mappings between the ontologies. The local DL reasoners hosted in DRPs are modified so that they can work in collaboration with the distributed reasoning algorithm. Unlike standard DL reasoners (such as Pellet), which take a global reasoning approach, the distributed reasoner DRAGO allows the loading of complex modular ontologies. This is required, because in distributed reasoning, the modules of a modular ontology are loaded in parallel into dedicated reasoners executing on different computers, instead of being loaded together as a whole into a single reasoner.

\textsuperscript{25}The Drago system is available for download at http://drago.fbk.eu
11 Ontology management

For most of the aspects of ontologies mentioned so far in this report, software tools (both commercial and free/open-source) have been developed to assist developers and users in fulfilling their tasks. Several of these tools have moved into a mature stage. Nonetheless, many of the existing tools should be regarded as addressing certain niches of the overall task of developing and maintaining an ontology. With ontologies being increasingly applied in many critically important applications, what is needed is a platform that serves as an integrated support environment for the whole ontology development life cycle. Several ontology development tools have been developed in response to this requirement. One of the most versatile ontology development environments currently available is Protégé. Other tools that are also heading in this direction include OntoStudio, TopBraid Composer and IODT.

11.1 Review of ontology development tools

There exist a large number of tools that support the development of ontologies. The earliest developed tools still in use were created more than a decade ago. These tools include Ontolingua, WebODT, OntoEdit, WebOnt, OILEx, DUET, OntoSaurus and Protégé (see [51] for a description and comparison of these tools). In response to the remarkable growth and evolution of the field, existing tools have evolved and new tools have been developed to support a wider variety of tasks and/or to provide a flexible and extensible architecture so that the functionality of the tools can be augmented as required in a simple and efficient way.

I will now briefly discuss some of the state-of-the-art tools.

Protégé is probably the most popular ontology development tool. Protégé is a free, Java-based open source ontology editor. Protégé offers two approaches for the modelling of ontologies: a traditional frame-based approach (via Protégé-Frame) and a modelling approach using OWL (via Protégé-OWL). Protégé ontologies can be stored in a variety of different formats, including RDF/RDFS, OWL and XML Schema formats. Protégé facilitates rapid prototype and application development, and has a very flexible architecture via a plug-and-play environment. For example, researchers have developed a variety of plug-ins (e.g., the PROMPT/Anchor-PROMPT plug-in for ontology merging, plug-ins for versioning support and plug-ins for collaborative ontology development). The Protégé-OWL API can be used to generate Jena RDF models, and reasoning can be performed by means of an API which employs an external DIG-compliant reasoner, such as...
as RACER, FaCT++, Pellet or KAON2. Recently, a lightweight OWL ontology editor for the web (Web-Protégé) has been proposed\textsuperscript{34}.

Since, Protégé was traditionally a frame-based ontology development tool, the existing Protégé-OWL API was built on top of a frame-based persistence API. This resulted in several significant disadvantages. Currently, a new generation of Protégé-OWL with a native OWL API is under development to address these problems.

**OntoStudio** (previously known as OntoEdit)\textsuperscript{35} is a commercial product that is used in industry. OntoEdit is focused on F-Logic, and supports reasoning via the F-Logic inference machine OntoBroker. OntoStudio offers both graphical and textual rule editors, as well as debugging features. It also provides a plug-and-play framework via the use of the Eclipse platform. A number of plug-ins (e.g., query plug-ins and visualisation plug-ins) are available.

**TopBraid Composer**\textsuperscript{36} is a comprehensive editor for RDF(S) and OWL ontologies. Based on the Eclipse platform, it offers a plug-in architecture. It has Pellet as its built-in reasoner, and supports a variety of inference engines.

**Integrated Ontology Development Toolkit (IODT)**\textsuperscript{37} is an ontology-driven development toolkit developed by IBM. IODT includes an Ontology Definition Metamodel (EODM) and Minerva as its OWL ontology repository. The Ontology Definition Metamodel is implemented in the Eclipse Modelling Framework (EMF). Minerva is a high-performance RDBMS-based ontology storage, inference, and query system (see Section 9).

**SWOOP**\textsuperscript{157} is an open-source hypermedia-based ontology development tool. As a native OWL tool, SWOOP is designed around the features of OWL. It provides an environment with a look-and-feel similar to that of a web browser. Reasoning can be performed using an attached reasoner (such as Pellet).

**Altova SemanticWorks**\textsuperscript{38} is a pure OWL editor. It is commercially offered by Altova. One of the strengths of Altova SemanticWorks is its rich graphical interface. Unfortunately, however, it does not support direct interactions with reasoners, and thus serves as a pure OWL editor rather than as a bona fide development tool.

### 11.2 Lifecycle of Networked Ontologies (NeOn)

It should be noted that all of the existing tools are primarily designed to support the development and management of single ontologies by single users. Although Protégé has a plug-in for version management, and both Protégé and TopBraid Composer provide a multi-user mode, the support for these functionalities is still limited. As such, more support for the development and management of ontologies in a distributed and collaborative

\textsuperscript{34} A demo is available at http://hmir-protege-dev1.stanford.edu/webprotege/.

\textsuperscript{35} http://www.ontoprise.de/en/home/products/ontostudio/

\textsuperscript{36} http://www.topquadrant.com/products/TB_Composer.html

\textsuperscript{37} http://www.alphaworks.ibm.com/tech/semanticstk

\textsuperscript{38} http://www.altova.com/semanticworks.html
environment is required. The required infrastructure should provide support for the main types of ontology specification languages (e.g., F-Logic for rules and frame-based representation, and OWL for DL-based ontologies); should be able to integrate and manage a constantly evolving network of ontologies; should provide support for the entire ontology development lifecycle (including, for example, development, deployment and maintenance); and should provide collaboration support [342]. Waterfeld and colleagues [342] have proposed NeOn, a reference architecture for ontology management tools that aims to address the aforementioned requirements. The architecture of NeOn consists of three layers with increasing levels of abstraction: Infrastructure Services, Engineering Components and GUI Components. The bottom layer, Infrastructure Services, provides core services required by most semantic applications, such as specification, storage, reasoning, querying, versioning and security services. The middle layer, Engineering Components, provides engineering functionalities for both tightly coupled and loosely coupled components, such as ontology mapping, translation and collaboration support. The top layer, GUI Components, provides user front-end components, including editors with text-based, graph-based and form-based interfaces. NeOn is built in the Eclipse framework, and so is extensible and highly modular. NeOn is currently under development. More information about NeOn can be found in [342, 242, 243].
12 Ontology evolution

12.1 Review of techniques for ontology evolution

As ontologies play an integral role in semantic applications, they are expected to evolve in step with the constantly changing application environment. In such a dynamic scenario, ontology change management is increasingly recognised as a critical task. Unfortunately, ontology change management is difficult to implement, especially in open and dynamic environments such as the Semantic Web. Ontology change management is a multifaceted task. More specifically, it encompasses ontology evolution, ontology versioning, ontology merging (see Section 7) and ontology integration (see Section 6). Here, ontology evolution refers to the process of modifying an ontology in response to change in domain knowledge, and ontology versioning refers to the process of modifying an ontology while preserving the original version. Ontology evolution includes (i) ontology population, where new instances for existing concepts need to be added, and (ii) ontology enrichment, where new concepts are added to the ontology, or the existing concepts and inter-concept relations are modified. Since ontology enrichment is more challenging than ontology population, most research on ontology evolution focuses on ontology enrichment.

As mentioned earlier, the ontology development community is still in need of tools that are able to fully support the entire ontology development life cycle, especially ontology evolution. Except for a few tools (e.g., Protégé, which has a plug-in for ontology versioning), most of the tools supporting ontology evolution exist in the form of system prototypes. Moreover, most of the proposed solutions for ontology evolution in distributed and collaborative settings have not yet been implemented as working tools. This section provides a flavour of the field of ontology evolution.

Taking advantage of the similarities between database schema and ontologies, researchers have successfully adapted established techniques from the field of data schema evolution to the problem of managing the evolution of ontologies. This work is described in detail in [56, 200, 137, 172, 306, 305, 220]. In addition, a variety of methods for the tracking of changes to ontologies have been proposed. For example, PromptDiff [251] tracks structural changes in ontologies; the method proposed by Stojanovic [305] traces changes using a log; KAON [220] keeps track of elementary changes by means of an evolution log ontology; and OntoAnalyser [283] is able to trace certain kinds of complex ontological changes. Regarding the sources of ontology changes, researchers have looked at ontology evolution from user-driven and discovery-driven perspectives. In user-driven evolution, ontologies are changed to adapt to modified business requirements [306]. In discovery-driven evolution, ontology evolution is triggered by the discovery of new concepts through the analysis of external knowledge sources [39]. Stojanovic [305] distinguishes three types of discovery-driven evolution: usage-driven, data-driven and structure-driven change discovery. Stojanovic [306] has also proposed structure-driven, process-driven, instance-driven and frequency-driven strategies for the modification of an ontology during the process of ontology evolution; these strategies take into account the structure of the ontology, the process of changes, a given state that needs to be achieved, and the most/least recently used evolution strategy, respectively.
Researchers have also proposed frameworks that integrate different aspects of and methods for ontology evolution. Klein and Noy [173] have proposed an evolution framework for distributed ontologies. Stojanovic [305] has presented a framework for evolving ontologies mainly in response to internal sources of change. Noy and colleagues [250] have described a framework for ontology evolution in collaborative environments, which is supported by various Protégé plugins. Finally, Khattak and colleagues [163, 164, 165] have developed an integrated framework for ontology evolution management.

Several tools support ontology versioning as part of their ontology change management functionality. For example, PromptDiff [251] can discover differences between two versions of a particular ontology. Klein and colleagues [171] have proposed OntoView as a web-based change management system for ontologies which provides a transparent interface to different versions of ontologies. The same authors [172] later proposed a state-based approach to ontology versioning [56], which allows users to compare versions of an ontology and to specify the relations between these versions. In contrast, Maedche and colleagues [220] have proposed a change-based approach which tracks and records all the performed changes, based on which change detections, integrations, and conflict managements are performed [56].

In the context of the Semantic Web, where only very minimal assumptions can be made about the participating systems/agents and their interaction protocols, ontology change management needs to be performed in a dynamic manner. This means that semi-automated approaches that require some degree of human intervention are inadequate in this context. Proposed as a response to this problem, Evolva [348, 349] is intended to be a fully automated tool that utilises background knowledge to support evolution. More specifically, Evolva aims to detect and validate new knowledge added to the base ontology; to discover the relations between the added knowledge and the existing knowledge by using sources of background knowledge (e.g., the lexical database WordNet); to check for problems related to inconsistency and duplication of knowledge; to perform the changes on the ontologies; and to propagate the changes made to dependent ontologies.

Inspired by various developments in ontology evolution, De Leenheer and Mens [56] have proposed a unified change process model for the evolution of single ontologies and guidelines for ontology evolution in a distributed and collaborative environment. These proposals exploit well-studied methods from software and system engineering. Also, ontology evolution has been incorporated into DILIGENT, a methodology for collaborative development of ontologies [275].

Ontology evolution is also supported to a certain extent in various other tools, including Text2Onto [47], Dynamo [261], OntoLearn [331] and DINO [248]. However, these tools are better regarded as ontology learning tools useful for the initial construction of ontologies from textual data than as tools supporting ontology evolution.

### 12.2 Software and tools

There exist software and tools that provide some support for ontology evolution. Among these tools are Protégé (see Section 11), KAON, WSMO Studio and DOGMA Studio [56]. A brief overview of KAON, WSMO Studio and DOGMA Studio is presented below.
KAON\textsuperscript{39} is a framework that targets e-commerce and B2B applications using Semantic Web ontologies. KAON’s main design goal is robustness and scalability. Based on Java EE, it has been adapted to the Model-View-Controller architecture proposed by Sun. KAON proposes a framework aimed at providing a comprehensive management infrastructure for ontologies and metadata. KAON provides support for change presentation, configurable evolution strategies and dependent ontology evolution \[56\].

WSMO studio\textsuperscript{40} is a modelling environment for the Web Service Modelling Ontology (WSML). It includes the Ontology Management Suite and a WSML ontology versioning tool. The WSML ontology versioning tool includes an ontology versioning API, allows various evolution strategies, supports version identification, and provides version change log functionality and partial version mapping \[56\].

DOGMA studio\textsuperscript{41} is the tool suite developed for the DOGMA collaborative ontology engineering approach. DOGMA Studio consists of two main components: a Workbench and a Server. The DOGMA Server is an advanced Java EE application running on the JBoss application server. The DOGMA WorkBench is based on Eclipse, and thus provides a plug-in architecture for the incorporation of different ontology engineering activities as required. DOGMA Studio also supports ontology evolution via the plug-in DOGMA-MESS \[42\].

\textsuperscript{39}http://kaon.semanticweb.org
\textsuperscript{40}www.wsmostudio.org/
\textsuperscript{41}http://starlab.vub.ac.be/website/dogmastudio
\textsuperscript{42}http://www.dogma-mess.org
13 Ontologies of information security

There have been several attempts to develop ontologies related to aspects of information security (e.g., the work described in [71, 186, 67, 111, 114, 239, 131, 8, 80, 45, 79, 303, 337, 222, 20, 102, 99]). I present a brief summary of these attempts below.

- Dobson and Sawyer [71] have proposed an OWL-based ontology from the perspective of dependability requirements engineering, which focuses on attributes such as availability, reliability, safety, integrity, maintainability and confidentiality.

- Amaral and colleagues [67] have presented initial work toward an ontology for the domain of information security, which has investigated the extraction of knowledge from security documents such as information security standards policies.

- Fenz and Weippl [114] have proposed a ontology of security that is intended to be used in IT applications for small- and medium-sized enterprises.

- Geneiatakis and Lambrinoudakis [111] have developed an ontology that describes security flaws in the session initiation protocol (SIP).

- Giorgini, Mouratidis, Zannone and Manson [114, 239] have integrated security and trust considerations into Tropos, an agent-oriented software development methodology.[43]

- The work reported in [131, 8, 80, 45] is devoted to the modelling of vulnerabilities and risk analysis, assessment and control in information systems.

- Ekelhart and colleagues [79] have proposed an ontological mapping of the ISO/IEC 27001 standard[44], which can be used with an existing ontology of security to increase the degree of automation of the certification process.

- Squicciarini and colleagues [303] and Vorobiev and Bekmamedova [337] have presented an ontology-based approach to information security and trust. More specifically, they have proposed ontologies for trust-based collaboration of application components [337] and for trust negotiation systems [303].

- Martimiano and Moreira [222] have proposed an ontology that facilitates the correlation and management of security incidents.

- Beji and Kadhi [20] have proposed an ontology of security for mobile applications.

- Fenz, Tjoa and Hudec [102, 99] have studied the use of ontologies of security for threat probability determination and security risk management.

- Cyc’s ontologies for faults and vulnerabilities [295] are similar in nature to the ontology for network intrusion detection proposed by Undercoffer and colleagues [323, 324]. These ontologies are proprietary and thus not publicly available.

---

I now list some other relevant work, which is primarily concerned with ontologies of security in the context of the Semantic Web.

- Kagal and Finin [154] have proposed an ontology of speech acts (i.e., ‘actions that are implied when an agent makes an utterance’) which aims to enhance interoperability by decoupling conversation policies from the agent communication language.

- Maamar, Narendra and Sattanathan [214] have presented an ontology-based approach for specifying and securing web services.

- McGibney, Schmidt and Patel [227] have developed a standard ontology for intrusion detection.

- The work reported in [106, 107, 110, 61, 181] proposes ontologies for: different forms of access control (e.g., network access control (NAC) policies for firewalls and proxies) [106, 107], resource access within an organisation [110], authentication and data integrity [61], and role-based access control [181].

- The work reported in [63, 62, 168] proposes ontologies of security for the annotation of information resources and web services using DAML [63] and OWL [62, 168]. The NRL security ontology developed in [168], which is publicly available, consists of several sub-ontologies addressing such sub-domains as service security, agent security, credentials and security assurance.

- Bao, Slutzki and Honavar [14] have investigated how to secure the sharing of ontologies between autonomous entities.

Many of the existing ontologies of security are designed according to pre-existing taxonomies which provide a rich source of concepts and terms related to information security. For instance, Herzog and colleagues [140] compiles a collection of existing taxonomies for different aspects of information security such as: intrusion detection [10, 37, 58], countermeasures [150, 340], threats [4, 41, 60, 199, 344] and security technology [333]. There exist ontologies/taxonomies devoted to the modelling of network attacks [323, 134, 177, 352, 301, 324, 176, 1].

Most existing security ontologies (such as those mentioned above) focus on only one or a few aspects of information security. Moreover, several of them are works in progress, and very few of them are published online. Thus there does not exist an ontology that covers the whole domain of information security, that supports efficient machine reasoning, that is well-received by different communities, and that is publicly available. To the best of my knowledge, only two research projects have worked toward developing such an ontology. The first project was carried out by Herzog and colleagues [140], and the second work by Fenz and Ekelhart [100].

\[45\text{http://chacs.nrl.navy.mil/projects/4SEA/ontology.html}\]
13.1 Herzog and colleagues’ ontology of security

Herzog and colleagues [140] have proposed an OWL-based ontology of information security, which is available for downloading and importing at [http://www.ida.liu.se/~iislab/projects/secont/](http://www.ida.liu.se/~iislab/projects/secont/). They endeavoured to deliver an extensible ontology for the information security domain that includes both general concepts and specific vocabulary of the domain, and supports machine reasoning and collaborative development. Based on the classical risk analysis carried out by Whitman and Mattord [345], the proposed ontology is built around the following top-level concepts: assets, threats, vulnerabilities and countermeasures. These general concepts together with their relations form the core ontology which presents an overview of the information security domain in a context-independent and application-neutral manner. In order to be practically useful, the core ontology is populated with domain-specific and technical vocabulary which elaborate the core concepts and implement the core relations. Figures 13 and 14 illustrate the refinement of the core concepts countermeasure and threat by specific concepts and relations. The ontology is also designed to be reusable and extensible. The ontology can be processed via inferencing (to infer subsumption relationships among concepts) and via querying (to provide additional processing of the results of inferencing). For example, to perform an inference to categorise concepts according to certain criteria, one needs to define a ‘view’ concept and then utilise a reasoner (e.g., Pellet, RACER or FaCT) to derive subclasses of the ‘view’ concept [140]. If one then wishes to perform some post-processing on the results (e.g., arrange concepts in the result in alphabetical order), this can be done easily using the sorting function of the standard query language SPARQL. Herzog and colleagues developed their ontology using the Prot´eg´e-OWL tool, the SWOOP editor and the Pellet reasoner. SPARQL queries are used to query OWL files, and Jena APIs are used to support some programming tasks. At the time of publication, the ontology contained 88 threat classes, 79 asset classes, 133 countermeasure classes, and 34 relations between these classes. To enable collaborative development and to ensure the ontology can be accepted by various communities, Herzog and colleagues designed their ontology according to codified design principles [124] and best practices [46].

13.2 Fenz and Ekelhart’s ontology of security

In the same vein, Fenz and Ekelhart have proposed an ontology[47] that has similar goal but attempts to cover a broader spectrum: their ontology models a larger part of the information security domain, including non-core concepts such as the infrastructure of organisations [100]. Also, in an endeavour to deliver an ontology with desirable qualities such as clarity, coherence, extendibility and minimum encoding bias, Fenz and Ekelhart have evaluated and selected the most suitable information security standards and best-practice guidelines for the design and development of their ontology. For example, security concepts and relations are derived from the German IT Grundschutz Manual [34], the ISO 27001 standard [151], the French EBIOS standard [55], the NIST computer security handbook [245], the NIST information security risk management guide [307] and Peltier’s threat classification [272].

Figure 13: A classification of countermeasures \cite{140}.
The top-level concepts in the ontology are categorised into three subontologies: namely, *security*, *enterprise* and *location* ontologies. The security ontology includes the concepts *attribute*, *threat*, *vulnerability*, *control* and *rating*; the enterprise ontology includes the concepts *asset*, *person* and *organisation*; and the location ontology simply describes a list of locations. The information in the ontology can be processed via inferencing (using a reasoner) and querying (using SPARQL for simple queries and the Protége OWL API for more expressive queries).

Like Herzog’s group, Fenz and Ekelhart also developed their ontology using the Protégé-OWL tool and Pellet reasoner. Their ontology contains about 500 concepts and 600 formal restrictions, specified in graphical, textual and description logic forms to ensure a minimal encoding bias.

---

Figure 14: A classification of threats [140].

In more recent work, Fenz and Hudec have investigated the use of the domain knowledge included in ontologies in the generation and maintenance of Bayesian networks. In particular, Fenz and Hudec have demonstrated how to use their security ontology to construct a Bayesian network for threat probability determination [103]. Bayesian networks, in brief, are graphical models that encode the probabilistic relationships among the influence factors of certain events and allow reasoning on the probabilities of these factors. For information about similar work predating Fenz and Hudec’s work, the reader is referred to [64, 284, 350].

According to Fenz and Hudec, the process of generating a Bayesian network involves challenging tasks such as determining the relevant influence factors and their relationships, and calculating the conditional probability tables for each node in the network [103]. In their proposed approach, the generation and maintenance of a Bayesian network is achieved by employing a domain ontology that facilitates the (semi-)automated completion of the aforementioned tasks. For example, the concepts and relations in the ontology are used to generate nodes and links in the Bayesian network, while the axioms in the ontology are used to create scales and weights, and the ontological knowledge base is used to support the calculation of conditional probability values [103]. Although it has been demonstrated that the use of ontologies in the generation of Bayesian networks is a promising approach, Fenz and Hudec claim the following two important limitations of the approach as presently formulated: (i) functions for the calculation of conditional probability values have to be provided by a source external to the ontology (since they are not part of the ontology), and (ii) human input is still required if the knowledge modelled by the ontology does not exactly match the domain of interest.
14 Applications of ontologies to network management

Managing heterogeneous network resources in a distributed environment has long been a challenge for network and system administrators. Already, there have been numerous studies on the topic of integrated network management, which have given rise to the development of various standards for information and network management (e.g., Simple Network Management Protocol (SNMP), Guideline for Definition of Managed Objects (GDMO), Desktop Management Interface (DMI) and Web-Based Enterprise Management (WBEM)). Having different information and network management systems adhering to the same standard promotes compatibility and interoperability. However, because standards such as SNMP, GDMO, DMI and WBEM tend to focus on different aspects of information and network management, a large-scale network management system may include network resources that are modelled according to different standards. Managing such a system can be problematic due to the inherent incompatibility of the standards in terms of the information models they implement. Proposed as a solution to this incompatibility problem, the Common Information Model has emerged as a consensus standard that is independent of the managed environments and their underlying implementations (e.g., the platforms, programming languages and network protocols utilised).

14.1 Common Information Model (CIM)

Developed by the Distributed Management Task Force (DMTF), CIM\(^49\) is an object-oriented information model that describes management information, and enables information sharing and interoperability among various network and system elements in a distributed system. The CIM consists of the CIM specification and the CIM schema. The CIM specification defines basic concepts, the language to describe CIM constructs, and techniques for mapping from other management/information models (e.g., SNMP). The CIM schema describes the actual information model. It is graphically described in the Unified Modeling Language (UML), and formally defined in a managed object file (MOF). The CIM schema consists of three distinct layers: the Core schema, the Common schema and the Extension schema. At the highest level of abstraction, the Core schema is an information model that captures concepts common to all areas of management. The Common schema is an information model that captures concepts common to particular management areas but independent of a particular technology or implementation. Currently, the Common schema focus on four specific areas: systems, devices, networks and applications. The Extension schema includes technology-specific information models that extend the Common schema. Figure\(^5\) illustrates the CIM network model.

As an evolving standard that has been increasingly adopted by organisations, CIM provides a complete solution to the problem of achieving interoperability among disparate information and network management systems. As things stand now, different systems may adhere to different standards. If CIM becomes a universal standard, the integration of

\(^{49}\)http://www.dmtf.org/standards/cim/
Figure 15: The CIM network model (source: [http://www.dmtf.org/standards/cim/cim_schema_v2250/CIM_Network.pdf](http://www.dmtf.org/standards/cim/cim_schema_v2250/CIM_Network.pdf))
CIM-compliant systems with legacy systems modelled according to various other standards will be necessary. To this end, techniques have been developed for the translation of information written in other languages (in other standards) into MOF/CIM. However, without a mechanism that defines semantic relationships between the concepts in the CIM and the other standards, such integration can be achieved only at a syntactic level [203].

Issues such as those just described have stimulated a research program that is aimed at bringing well-defined semantics (through the medium of ontologies) into the information and network management realm. The use of ontologies in this context has two significant implications: (i) it provides semantics to the content of the information model, and (ii) it provides a means of adding formal axioms and constraints to the information model, thereby enabling the description of the behaviour of the network and reasoning on the information model [204].

14.2 Ontology-based network management

As described in [203, 201, 206, 210, 207, 205, 208, 211, 202, 200, 109], López de Vergara and colleagues have made an important contribution to this line of research. They address issues related to managing network resources in different management domains. More specifically, they aim to enable semantic interoperability among information management standards such as SNMP, GDMO MIBs and DMTF’s CIM.

The overall approach for ontology-based network management proposed by López de Vergara and colleagues consists of three phases (see Figure 16), each of which are described below.

In the first phase, network management languages (e.g., GDMO, MOF/CIM and Structure of Management Information (SMI)) that describe network resources in different management information models are analysed from a semantic perspective and mapped to the most popular web ontology language OWL [203, 201, 206]. To assist developers in performing this task, a plug-in has already been developed for Protégé that allows SMI MIB files and CIM MOF files to be imported into the editor in a standardised model [204] which can then be exported to OWL or other ontology languages supported by Protégé.

Once the management information has been specified in the same language (i.e., OWL), in the second phase, the various management information specifications are combined into a (new) common model that integrates the existing management information using a merge-and-map (M&M) technique [207, 205]. The mapping outcomes from the original definitions to the ontology specification derived from applying the M&M technique are then stored in so-called gateways, as illustrated in Figure 17. In this way, semantic interoperability can be achieved between the management information models via the use of the common model and rules in the gateways that implement mappings between the common model and each of the original management information models.

In the third phase, formal axioms and constraints are added to the common model. This effectively describes behaviours of the system. López de Vergara and colleagues envision that the techniques they propose for the third phase can be used to constrain the network parameters, thereby predetermining some network behaviours and allowing
Figure 16: Ontology-based network and system management according to López de Vergara and colleagues’ approach [201].

the automation of many network management processes (e.g., service monitoring, security management, quality of service management, and fault management) [201]. Since CIM and information models implemented in other standards are similar to lightweight ontologies in that they define a hierarchy of concepts and their basic relations without specifying axioms and constraints, definitions of behaviours in such models are usually expressed in natural language. By integrating management information definitions into a management ontology (corresponding to the OWL-based common model created in the second phase), these behaviours can be formally defined as part of the ontology using the same ontology language (or a variant of the language). This possibility has been investigated by López de Vergara, Guerrero, Fuentes and colleagues in subsequent work (see [128, 129, 210, 208, 109]), in which the behaviours superimposed on the concepts are implemented in Semantic Web Rule Language (SWRL) [133]. As an extension to OWL, SWRL is designed as a standard rule language for the Semantic Web. It provides the ability to write conditional rules expressed in terms of OWL concepts. For instance, [128] describes in SWRL different aspects of the behavior of a management system, including implicit managed object constraints (which define the behaviour of the modelled objects), explicit manager behavior (which defines how the manager should behave in response to obtaining and analysing information from agents), and network management policies (which specify the dynamic behaviour and configuration of the managed resources). [129] presents an ontology-based approach that promotes interoperability between high-level policies and low-level policies in a framework for automated management. To hide the underlying complexity and allow administrators to manage their systems at a suitably high
level of abstraction, it is useful to define policies at different network management levels \[129\]. In this way, connecting high-level ontologies (which define behaviour policies) to low-level ontologies (which define network behaviours) by means of relationships between classes allows transparent and seamless data communication between the different levels and thus enables efficient policy execution at run-time. Instead of defining the behaviour of the system via management policies, an alternative approach presented in \[208\] \[109\] attempts to describe the system behaviour by making use of Web Service Management Interfaces. Here, the upper ontology of Semantic Web services (OWL-S), the Web Service Modeling Ontology (WSMO), and the Semantic Web Services Ontology (SWSO) are used to implement the web services and semantically describe how the managed resources are to be managed. One of the advantages of this approach is that the managers and the managed resources can exchange management information as ontology instances (i.e., OWL instances). Provided that the manager can interpret the ontology definitions, this approach eliminates the hassle of translating between management information definitions.

López de Vergara and colleagues have also prototyped their design in a number of projects, including ontologies for autonomic systems for home gateways and services \[211\] \[212\], ontologies for network security and policy management \[202\], and ontologies for network monitoring \[200\]. See \[211\] \[212\] \[202\] \[200\] for a detailed description of this work and \[201\] for a discussion of the advantages and drawbacks of these prototypes.

Also devoted to addressing the problem of mapping management information, other researchers \[329\] \[162\] \[294\] have investigated the problem from different perspectives. For instance, Van der Meer and colleagues \[329\] presents an approach to the integration of management policies, with a focus on the mobility of policies in a pervasive computing context; Keeney and colleagues \[162\] proposes an approach to the merging of management information from different sources at run-time for the purpose of delivering network knowledge for decision making; and Serrano and colleagues \[294\] develops an ontology-based, context-aware information model for the management of pervasive services.
Other relevant research includes efforts to formalise CIM models. To name a few, there exist proposals to describe CIM models in description logics [184], in OKBC [188], in RDF/OWL [277], and in Object Constraint Language (OCL) [209]. Since the specifications of CIM models are described in UML, the OMG group assists in the task of formalising the CIM models by working on the mapping between UML and RDF/OWL.

The work described in [174] [191] is focused on the problem of automating network management. [174] proposes an ontology-based automatic web service composition approach intended to support the automation of network management, while [191] presents an ontology-based knowledge representation approach for self-governing systems. However, this work is still in its early stages.
15 Application of ontologies to network modelling

Most of the work described in the previous section is concerned with either (i) developing simple ontologies for specific aspects of information and network management, or (ii) constructing an ontology by formalising an existing standard information model (e.g., CIM) or by translating and mapping/merging management information definitions. What appears to be lacking here is research work on the applications of ontologies to network modelling. An extensive search of the literature has revealed that there has been little work published in this area. It should however be noted that this literature survey is obviously restricted to a discussion of ontologies and ontology-related artifacts in the public domain. It is likely that relevant research and development has been carried out in the private domain. For instance, the Shapes Vector network security system [82] developed by DSTO’s C3ID Division incorporates an ontology that models different aspects of sensitive computer networks. As this ontology is not publicly available, it is not possible to include a discussion of this ontology in this report. Nevertheless, there are two research efforts worthy of discussion. These research efforts are described below.

15.1 Communications Network Modelling Ontology

As part of their overall project to develop an integrated network design and simulation tool, Rahman and colleagues [279] have proposed an ontology for network modelling called Communications Network Modelling Ontology (CNMO). The development of CNMO was motivated by two major factors: (i) the confusion of terminologies used in a stream of publications in the field of network design, analysis and simulation, and (ii) the incompatibility of the network models on which supporting tools in the field are built [279].

Expressed in first-order logic, CNMO consists of five component ontologies, which were developed separately and then unified. The five component ontologies are the Communication Network Node ontology, the Communication Network Link ontology, the TransportEntity ontology, the TECConnection ontology, the TrafficSource ontology, the ModellingFiles ontology and the NetOperation ontology. The Communication Network Node

Figure 18: Node ontology (adapted from [279]).
ontology models a communication node with different attributes as shown in Figure 18. Node visualisation and node location are also described in the subclasses of the communication node. The Communication Network Link ontology models a communication link that transfers data between the communication nodes together with its attributes and a subclass for link visualisation (see Figure 19). The TransportEntity ontology, depicted in Figure 20, is devoted to describing transport protocols (transport entities), while the TEConnection ontology is dedicated to describing a transport connection between a pair of transport entities. The TrafficSource ontology (Figure 21) defines an application (e.g., FTP or Telnet) that is using a transport connection, whereas the NetOperation ontology specifies information that is either produced during, or found after, the network operation. Finally, the ModellingFiles ontology presents the names of the files associated to the network modelling process. The overall structure of CNMO is displayed in Figure 22. An integrated tool called NeDaSE (Network Design and Simulation Environment) has been implemented based on CNMO [280]. It enables the transformation of network models between different network simulation tools.

The vocabulary and design of CNMO draws on the network models represented by a wide range of existing tools dedicated to network modelling, simulation, generation and discovery.
Figure 21: TrafficSource ontology (adapted from [279]).

Figure 22: Communications Network Modelling Ontology (adapted from [279]).
Despite its seeming simplicity, CNMO efficiently meets its specified goal. Its content could be complemented with ontologies containing relevant concepts from the domain of information systems or distributed computing systems to achieve a wider scope [279].

15.2 An ontology of distributed computing systems

Another ontology that is related to network modelling is the ontology of distributed computing systems [244] that has been developed as a domain ontology extending SUMO (see Section 8.1). This is a comprehensive ontology that includes high-level concepts such as computer network, hardware system and software system, as well as low-level concepts such as packet, processor, memory and computer process. The complete ontology is available at [315]. Although it does not directly address network modelling, the ontology could serve as a solid foundation, or at least a very useful reference source, for the design of an ontology that more directly addresses network modelling.
16 Summary and final remarks

Ontology has been a fruitful research area in computer and information science for the last two decades. The field has attracted prominent research attention, and has been continually expanding. Not only is this evident in the vast number of publications devoted to ontologies and their applications, it is also indicated by the fact that semantic technologies have been listed among the top ten disruptive technologies for 2008-2012 ([29]). The current overall picture of research and development in the field can be characterised as involving (i) much work on the development of ideas and prototypes (mostly in academia), (ii) attempts at devising specifications and standards, and (iii) many projects aimed at realising mature ideas and techniques as practical systems. I will now elaborate on certain parts of this overall picture.

When designing an ontology, one of the most important things one needs to consider is the language used to express the ontology. Ontology specification languages have a long history of development, so there are many proposals for languages that have their roots in formal logics or frame languages. It is clear that the currently most popular ontology specification language is the standard Semantic Web ontology language OWL (OWL-DL in particular). According to a survey conducted by Cardoso [36], OWL has been adopted by 75.9% of researchers and practitioners, followed by RDF(S) (64.9%) and description logics (17.0%). Current research on ontology specification languages is mainly concerned with enhancing computational efficiency while maintaining expressivity.

Regarding ontology development practices, currently none of the proposed development methodologies has the status of a standard. Indeed, the majority of ontology development projects use ad hoc methodologies. According to Cardoso’s survey, 60.0% of projects do not adopt any pre-defined methodology, 13.9% of projects use On-To-Knowledge, and 7.4% use METHONTOLOGY [36]. More recently proposed methodologies such as DOGMA and especially ONTO-AGENT seem to possess features that promise to promote the flexibility, scalability and efficiency of ontologies. However, being relatively new, they are yet to gain popularity or receive adequate practical evaluation by practitioners. However, it seems that an investigation into the use of these methodologies, especially the Onto-Agent methodology, for a given problem would be worthwhile. Apart from ontology development methodologies, there also exist specifications and standards (e.g., those provided by FIPA) that provide a general framework and architecture for ontology-based multi-agent systems.

The process of developing an ontology can be facilitated by a number of tools. Many of these tools concentrate on specific tasks of ontology development, whereas others are more versatile in addressing a wider range of tasks. The most prominent tool is Protégé: according to Cardoso’s survey [36], 68.2% of practitioners use Protégé as an ontology development environment. The next most popular tools, far behind Protégé, are SWOOP (13.6%) and OntoEdit (12.2%). The significant gap between the level of adoption of Protégé and that of OntoEdit is partially due to the fact that Protégé is freely available while OntoEdit (which has recently been renamed OntoStudio) is a commercial product. Furthermore, Protégé is open-source and supports plug-ins. This allows researchers and practitioners worldwide to augment the functionality of Protégé by developing plug-ins for

---

50 Please note that Cardoso’s survey focused exclusively on ontologies for the Semantic Web, and therefore provides only a partial (albeit significant) perspective on general aspects of Ontology.
specific ontology development tasks. The growth of some ontologies to sizes beyond the capability of in-memory reasoners has triggered efforts to develop methods of implementing secondary-storage support for reasoning. The methods proposed so far include (i) storing both the original and inferred knowledge of an ontological knowledge base in a database to eliminate the need for run-time inference and thus reduce query response times; (ii) deriving a summary of the knowledge base and reasoning on the summary instead of on the original knowledge base; and (iii) making use of well-known deductive database techniques for reasoning by translating an ontology into a Datalog program. Although these methods eliminate the need for run-time inference, their performance is still low in comparison to in-memory reasoners due to expensive hard disk accesses. Hence, on the one hand, in-memory reasoners seem to be more adequate from a pure performance point of view, but on the other hand, employing an in-memory inference engine for practical applications can give rise to out-of-memory problems for large-scale ontologies (e.g., it has been found that the LUBM (Leigh University Ontology benchmark) ontology\footnote{http://swat.cse.lehigh.edu/projects/lubm/index.htm} which contains 319714 instances, fails to be loaded by the Racer DL reasoner (due either to an out-of-memory error or an out-of-time error) when running on a platform with the Linux kernel 2.6.20-1, a 1.8GHz processor and 1GB of memory\footnote{http://swat.cse.lehigh.edu/projects/lubm/index.htm}). According to Cardoso’s survey\footnote{http://swat.cse.lehigh.edu/projects/lubm/index.htm}, more than half of the survey respondents (53.6\%) use Jena, a framework that supports both in-memory and secondary-storage reasoning, as their ontology reasoner. Among those reasoners devoted to in-memory reasoning, the most widely used are RACER (28.2\%), Pellet (23.4\%) and FaCT++ (12.4\%).

The out-of-memory problem potentially faced by in-memory reasoners has motivated attempts to modularise ontologies so that a reasoner needs to process only a portion of an ontology. With an aim to ease the design, construction and maintenance of large-scale ontologies, as well as to enhance the scalability and reusability of such ontologies, this line of research has produced many ideas and prototypes related to partitioning and subsequently recombining ontologies, including modular ontology specification languages and appropriate reasoning algorithms and frameworks. As this research area is relatively new, it is probably too early to expect current state-of-the-art work on modular ontologies to be employed in real-world large-scale applications. Having said this, there is already some support for the modularisation of ontologies in some mainstream tools (e.g., the SWOOP editor and Pellet reasoner support the modular ontology language \(\mathcal{E}\)-Connections), and new distributed reasoning frameworks have also been proposed (e.g., DRAGO). Open questions in the field include whether current support for modular ontologies in existing reasoners (e.g., Pellet) still requires a global reasoning space when reasoning is to be performed on a set of ontology modules (if so, this would defeat the major purpose of modular ontologies, namely, scalability); whether modular ontology specification languages supported in a distributed reasoner (i.e., DRAGO) are expressive enough to express ontologies of interest; and how support for modular ontology specification languages should be integrated into other mainstream tools (e.g., Protégé).

Ontology matching, a crucial activity for many ontology-based applications (e.g., multi-agent systems, query answering systems and web service composition), is one of the most active areas of research related to the Semantic Web. With a remarkable number of re-
search efforts having been devoted to it, semi-automated design-time ontology matching is reaching maturity. Fully automated run-time matching has been studied at a theoretical level, but further research is needed to improve the practical usefulness of dynamic matching techniques. Current practice in ontology matching is to perform ontology matching at compile-time, and to apply the resulting ontology alignments at both design-time and run-time. It is anticipated that ontology matching will reach mainstream adoption in three to eight years from now [52].

Good progress has been made on upper ontologies, with SUMO, UpperCyc and DOLCE the most significant achievements in this area. As many upper ontologies are still under development, a precise, comprehensive and conclusive evaluation of upper ontologies as universal ontologies is not available. However, SUMO seems to have attracted a lot of attention. In particular, several SUMO-compliant domain ontologies have been developed that can either be adopted as is or adapted to the particular needs of an application. If one were to seek to capitalise on the currently available upper ontologies (e.g., SUMO), at best one could directly reuse one of the domain ontologies that extends an upper ontology, and at least one could extend an upper ontology in constructing a new domain ontology (thereby inheriting the well-defined generic semantic content of the upper ontology).

Given that the world changes and human knowledge evolves, nearly all ontologies require maintenance. This necessitates the controlled management of ontology evolution. At present, the evolution of single ontologies has been studied well at a theoretical level, ontology evolution strategies for collaborative multiple ontologies exist mostly as proposals, and basic ontology change and versioning is supported by mainstream tools (e.g., Prompt-Diff in the PROMPT plug-in for Protégé). Like ontology matching, ontology evolution is expected to take three to eight years from now to reach mainstream adoption [52].

As for realising the full vision of the Semantic Web, there is still a long way to go. Dynamic ontology matching, dynamic ontology evolution management, automation of ontology integration, and collaborative development of ontologies are just some of the areas that require much more work.

Finally, regarding the modelling of a computer network, the SUMO-compliant ontology of distributed computing (presented in Section 15.2) deserves serious examination by interested parties, since it is the product of careful design by experts. An organisation interested in using ontologies to model a network, such as DSTO, could capitalise on the ontology of distributed computing either by reusing it, extending/adapting it, or using it as a reference to guide the construction of an ontology from scratch. In addition, the CIM model and other existing ontologies for the domain of networking could be used as sources of domain knowledge and terminology. If SUMO domain ontology of distributed computing were to be adopted and managed in the Protégé-OWL development environment, however, there would be the problem of converting KIF to OWL-DL, which would lead to a loss of semantic information. Even so, if the developers of the ontology were able to restrict the expressivity of all querying and reasoning on the ontology in such a way that all querying and reasoning could be handled by a description logic inference engine, this problem could be solved. Naturally, whether the problem could be solved would depend
on the complexity of the queries and reasoning required. In all ontology development activities calling for the integration of distinct and especially disjoint ontologies, DSTO should consider the possibility of using modular ontology specification languages such as the $\mathcal{E}$-Connections framework. On the specific issue of modelling the security properties of a computing network in an ontology, the literature contains several articles that present ideas and techniques that should be considered by an organisation wishing to undertake such modelling.
References


55. DCSSI 2004. EBIOS - Section 2 - Approach.


120. Grau, B. C. and Kutz, O. 2007. Modular ontology languages revisited. In Workshop on Semantic Web for Collaborative Knowledge Acquisition (SWeCKa), co-located with the 20th International Joint Conferences on Artificial Intelligence (IJCAI).


## Abstract

An ontology captures in a computer-processable language the important concepts in a particular domain and the relationships between these concepts. Ontologies are becoming increasingly pervasive in various fields of computer and information science. They are indispensable components of many complex information systems, especially systems in which communication among heterogeneous components is critical. I use the following definition of ontology, which captures the essence of the most widely adopted definitions in the field: an ontology is a specific, formal representation of a shared conceptualisation of a domain. The IO Branch of DSTO’s C3ID Division is interested in the possibility of using one or more ontologies to describe computer networks and support automated reasoning about their properties (particularly security properties). This report provides a basic overview of research and development related to ontologies and their use in information systems. The primary goal of the report is to help readers to discover topics of interest and to conduct further investigation of the literature. To this end, besides information about ontologies in general, the report also includes some specific comments about the use of ontologies to model and reason about computer networks and their security.

---

### DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA

| 2. TITLE | Ontologies and Information Systems: A Literature Survey |
| 3. SECURITY CLASSIFICATION | Document (U) |
| | Title (U) |
| | Abstract (U) |
| 4. AUTHOR | Van Nguyen |
| 5. CORPORATE AUTHOR | Defence Science and Technology Organisation |
| | PO Box 1500 |
| | Edinburgh, South Australia 5111, Australia |
| 6a. DSTO NUMBER | DSTO–TN–1002 |
| 6b. AR NUMBER | 014-980 |
| 6c. TYPE OF REPORT | Technical Note |
| 7. DOCUMENT DATE | June, 2011 |
| 8. FILE NUMBER | 2010/1172477/1 |
| 9. TASK NUMBER | N/A |
| 10. TASK SPONSOR | N/A |
| 11. No. OF PAGES | 92 |
| 12. No. OF REFS | 352 |
| 14. RELEASE AUTHORITY | Chief, Command, Control, Communications and Intelligence Division |
| 15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT | Approved for Public Release |
| 16. DELIBERATE ANNOUNCEMENT | No Limitations |
| 17. CITATION IN OTHER DOCUMENTS | No Limitations |
| 18. DSTO RESEARCH LIBRARY THESAURUS | Ontology, Information System, Security, Computer Networks, Reasoning |
| 19. ABSTRACT | An ontology captures in a computer-processable language the important concepts in a particular domain and the relationships between these concepts. Ontologies are becoming increasingly pervasive in various fields of computer and information science. They are indispensable components of many complex information systems, especially systems in which communication among heterogeneous components is critical. I use the following definition of ontology, which captures the essence of the most widely adopted definitions in the field: an ontology is a specific, formal representation of a shared conceptualisation of a domain. The IO Branch of DSTO’s C3ID Division is interested in the possibility of using one or more ontologies to describe computer networks and support automated reasoning about their properties (particularly security properties). This report provides a basic overview of research and development related to ontologies and their use in information systems. The primary goal of the report is to help readers to discover topics of interest and to conduct further investigation of the literature. To this end, besides information about ontologies in general, the report also includes some specific comments about the use of ontologies to model and reason about computer networks and their security. |