Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

INFORMATICA

Ciclo XXV

**Settore Concorsuale di afferenza:** 01/B1

**Settore Scientifico disciplinare:** INF/01

TITOLO TESI

# An architecture for scaling ontology networks

**Presentata da:** **Alessandro Adamou**

**Coordinatore Dottorato** **Relatore**

**Prof. Maurizio Gabbrielli** **Prof. Paolo Ciancarini**

**Esame finale anno 2013**

# Abstract

Constructing ontology networks typically occurs at design time at the hands of knowledge engineers who assemble their components statically. There are, however, use cases that require ontologies to be assembled dynamically as a network that is processed at runtime. Running Description Logics reasoning on an ontology network that combines an ABox obtained from multiple sources with its corresponding TBox, in turn combining multiple schemas, is one such use case. It occurs, for example, in extensible service frameworks which share a common knowledge base. Each application in the framework needs to process concurrent service calls from multiple clients. Each call, in turn, provides its own semantic payload for processing and would require an ontology network to be assembled ad hoc, otherwise a large chunk of the whole knowledge base would have to be processed on every call. These concurrent ontology networks generated out of shared knowledge resources, without altering the stored ontologies and without tampering with one another, are what we call "virtual [ontology] networks". Keeping track of which virtual networks use an ontology in a certain configuration is what we call "multiplexing".

Issues may arise from the connectivity mechanism of ontology networks. In many cases, simply appending each ontology as a subtree to a root node will not work, because many ontology managers visit networks in a way that can cause object property assertions to be erroneously interpreted as annotations and ignored by reasoners. Moreover, repository managers need to keep the knowledge base from growing uncontrollably, especially when it is being fed large volatile data for use by only one service call or a few.

Our claim is that ontology engineering alone does not suffice for tackling these problems altogether, and that they should be handled by the software

that serves these ontology networks. However, when the underlying framework is being observed, resource usage must be considered: the memory footprint of an ontology network that maximizes connectivity for the correct interpretation of axioms should be comparable to the sum of those of each ontology taken standalone. Also, multiple virtual networks that reuse the same ontologies should optimize their cumulative memory footprint, and where they cannot, this should occur for very limited periods of time.

We hypothesized that spreading the components required by an ontology network across a 3-tier structure can reduce the amount of erroneously interpreted axioms, under certain circumstances concerning the distribution of raw statements across the components. To that end, we assumed OWL 2 to be the core language handled by semantic applications in the framework at hand, due to the greater availability of reasoners and rule engines for this language. We also assumed OWL axiom interpretation to occur in the worst case scenario of pre-order visit, which we have verified to happen with widespread OWL management software such as the OWL API and Protégé. Because tackling ontology modularization is beyond the scope of this work, we limited to contexts where it is always possible to determine the minimal subset of the shared knowledge base that each application needs in order to obtain expected reasoning results. We expected that an aggressive ownership and persistence policy applied to the data payload (outer tier), and a conservative policy applied to the shared knowledge base portions (inner tiers), could reduce the memory occupation of virtual networks by at least one third of the memory occupied by each ontology network if it were fully in its own memory space.

To measure the effectiveness and space-efficiency of our solution, a Java implementation was produced as the "Stanbol ONM" (Ontology Network Manager) able to serve 3-tier virtual networks with REST services, therefore completely transparent to OWL applications. We compared the Java Virtual Machine footprint of ontology network setups in this proposed 3-tier configuration, against that of the same ontology networks when each runs in its own Virtual Machine (i.e. the framework's caching capabilities). We

measured the memory overhead of multiple ontology collector artifacts that reference the same ontology (i.e. the framework overhead). Also, measures applied to OWL axiom interpretation of virtual networks on a case-by-case basis verified that a 3-tier structure can accommodate reasonably complex ontology networks better than flat-tree import schemes can.

After a brief and informal introduction to the research challenges, Chapter 2, as well as related work sections for other chapters, encompasses the state of the art in the fields of research related to our work. Chapter 3 illustrates the research problems we were faced with both initially and during the course of the work, along with the goals that arose from them. The model for 3-tier ontology multiplexing and the theoretical framework behind it are discussed in Chapter 4, while Chapter 5 describes the step where said model can be bound to a software architecture. The aforementioned implementation has a dedicated description in Chapter 6, while Chapter 7 describes both the qualitative evaluation on top of a graph-based theoretical framework, and the quantitative evaluation of the memory efficiency of the implementation. The final chapter concludes this work with a discussion on lessons learnt and the open endpoints for future work to be taken up from there.

*If I have one dollar and you have one dollar and we trade them,*
*we will have one dollar each.*
*But if I have one idea and you have another idea and we share them,*
*we will have two ideas each.*

(Sr. Don Rafael del Pino y Moreno)

# Acknowledgements

Having grown on metal albums, renowned for their overly elaborated thank-you lists, I'm going to have a hard time staying within the mark.

My mentor Paolo Ciancarini for trusting this outsider whom he had never met until he first showed up as a candidate for a Ph.D. student position.

Aldo Gangemi and Valentina Presutti, without whom I'd be working as a humble system integrator now, if anything. Fellow worker bees in NeOn and IKS/Apache Stanbol all across Europe. We nailed 'em big time.

*Magistri* who mentored me sometime over the years: Lora M. Aroyo, Sean Bechhofer, Gerardo Canfora, Nicolò Cesa-Bianchi, Giancarlo Bongiovanni, Irina Coman, Ugo Dal Lago, Paolo Ferragina, Asunción Gómez-Pérez, Mark Greaves, Francesco Lo Presti, Dunja Mladenic, Enrico Motta, Harald Sack, Davide Sangiorgi, François Scharffe, m.c. Schraefel, Guus Schreiber.

Those who got to chew from the same sandwich: Gioele Barabucci, Eva Blomqvist, Elisa Chiabrando, Concetto Elvio Bonafede, Andrea Burattin, Enrico Daga, Francesco Draicchio, Alfio Gliozzo, Casandra Holotescu, Alberto Musetti, Andrea Nuzzolese, Paolo Parisen Toldin (also for the great collaboration in the Bertinoro courses), Silvio Peroni, Lino Possamai, Elena Sarti, Castrense Savojardo, Balthasar Schopman, Roland Stühmer, Anna Tordai (for bearing with the stone guest that I am), Sara Zuppiroli.

I would like to make a special mention to Óscar Corcho and Boris Villazón-Terrazas for their invaluable guidelines on how to setup my research context.

Gabriele and Andrea, for sharing with strength most that I endured. Michael, for shelter and bearing with me when I had nowhere to go in Amsterdam.

The others in the Vrije Universiteit for instantly making me one of the team: Hans, Victor, Dan, Davide, Kathrin, Marieke, Riste, Paul, Christophe, Willem, Frank, Michiel, Szymon, Rinke, Spyros, Elly, Jacco, Louren, Jan.

OpenWetWare for the LaTeXtemplate[1] that I just slightly modified for the layout of this dissertation. And the LaTeX guys and gals too, no less.

Leto-Sensei and fellow Shotokan Karate Do followers - Osu!

Stuff I (re-)discovered during the Ph.D. work: Asphyx, Diocletian, Jex Thoth, Japanische Kampfhörspiele, The Obsessed, Sayyadina, Hybernoid, Optimum Wound Profile, Coffins, Primate, Perdition Temple, Totenmond, Khlyst, Inner Thought, Scalplock, The Hounds of Hasselvander, Afgrund, Resistant Culture, Watchmaker, Greymachine. What keeps the mind fresh.

The whole staff behind "Strange Days", a movie I totally fell in love with on the very day my doctoral studies began.

People from 1980's comedy who personally gave me a good laugh and a blessing for my career: Maurizio Fabbri (tutti li donno!), Matteo Molinari (lo squalo bianco) and my all-time idol Giorgio Bracardi (chettefreca?).

Friends who were all the time out there to remind me what I am, over and above a computer scientist and whatnot. This isn't over. Grind on – UH!

Dad watching from afar. Mom. Family. Anyone who has given me a chance.

[1] OpenWetWare LaTeXtemplate for Ph.D. thesis, `http://openwetware.org/wiki/LaTeX_template_for_PhD_thesis`

# Contents

# List of Figures

# List of Tables

# Glossary

**ABox** The assertional component of a knowledge model.

**API** Application Programming Interface, the specification of a protocol for software components to communicate with each other.

**CMS** Content management system, a software system that allows publishing, editing and otherwise managing the workflow of content on a website or in a collaborative environment.

**DAML+OIL** An early ontology representation language.

**JSON** JavaScript Object Notation, a data interchange format whose syntax is derived from that of arrays and objects in the JavaScript programming language.

**JSON-LD** JSON syntax for Linked Data and RDF.

**MIME** Multipurpose Internet Mail Extensions, a standard for extending the format email to support non-text message bodies, also adopted in the HTTP protocol.

**N-Triples** A serialization format for RDF derived from N3 and Turtle, targeting ease of parsing by software systems.

**N3** Notation 3, a compact RDF serialization format.

**OWA** Open world assumption, a principle of formal logic that holds in OWL.

**OWL** Web Ontology Language, a formalism for representing ontologies in description logics. Can be applied on top of RDF or represented in standalone formats.

**OWL/XML** A syntax for serializing OWL ontologies to XML documents directly.

**RDF** Resource Description Framework, a standard for representing logical statements using a subject-predicate-object pattern.

**RDF/JSON** A serialization format for RDF graphs in JSON notation.

**RDF/XML** The normative serialization format of RDF graphs to an XML language.

**RDFS** RDF Schema, a knowledge representation language for constructing RDF vocabularies.

**REST** REpresentational State Transfer, an architectural paradigm for distributed systems, prominently adopted for current-generation Web Services.

**RIF** Rule Interchange Format, a proposed standard for rule representation in the Semantic Web.

**SPARQL** Recursive acronym for "SPARQL Protocol and RDF Query Language", a query language for databases that can retrieve and manipulate data stored in RDF.

**SWRL** Semantic Web Rule Language, a formalism for expressing rules compatible with OWL.

**TBox** The terminological component of a knowledge model.

**Turtle** Terse RDF Triple Language, a human-readable RDF serialization format built on top of N3.

**UML** Unified Modeling Language, a standard modeling technique in software engineering.

# 1

# Introduction

It is an established fact that the World Wide Web is evolving in a multitude of directions: some involving the quantity and variety of data; others dealing with the diversity and specialization of intelligent agents, no longer limited to humans, which are able to consume and generate these data. The target state commonly known as the Web 3.0, or the Web of Data [Hen10], involves features such as personalization [HF11, LK11] and the Semantic Web [SBLH06]. The latter, in turn, heavily relies on manipulating formally represented knowledge, what on a higher degree is known as *ontologies*. The distributed nature of the Semantic Web hints at several use cases, some of which will be exemplified across this thesis, where the reuse and combination of heterogeneous ontologies into ontology networks can be beneficial.

Methodologies for ontology modeling have acknowledged this matter and are veering towards authoring ontology networks, rather than monolithic models. However, ontology network assembly mostly remains a task for knowledge engineers to accomplish at authoring time, incognizant of the possible future uses of the same ontologies into larger networks. A dynamic assembly process can itself be tricky and have non-trivial implications, as ontology networks inappropriately assembled can lead to data clutter and loss of expressivity. We argue that methods should exists for mildly knowledgeable agents to construct ontology networks dynamically, and that these methods should accommodate software applications in incorporating them into their workflows.

## 1.1 Background

In the transition period from the 2000s to the 2010s, we as researchers, engineers, service providers and technology consumers, have witnessed numerous transformations of the computing world and its relationship with automated, intelligent data processing. More specifically, the Web industry and communities have acknowledged their interest in data processing, reconciliation and harmonization practices [BCMP12] coming from the artificial intelligence (AI) field. Initiatives like Google's *Knowledge Graph* [Goo] and *OpenRefine* [Opec], as well as the collective endorsement of *Schema.org* [Sch] by the major Web search providers, are living proof that the Web industry is ultimately acknowledging the importance of the Semantic Web, with its many alternative names.

On the verge of the much anticipated Big Data outbreak [LJ12], hardliner statisticians have pointed out the importance of experimenting on well-chosen samples before bringing techniques and applications out to the actual data bulks[1], as well as to the creeping dangers of not doing so. While we are not concerned with corroborating or disproving this stance in the present work, we can distill it into something that Semantic Web specialists can interpret as a piece of constructive criticism.

Sampled or not, Big Data need to scale [FMPA12]. Processing them with effective results and in an efficient manner requires ways to (logically) break them apart, or *segment* them, so that: *(i)* each segment can be fed to a component of a distributed computing system; *(ii)* minimum to no significant output is lost, as if one were centrally processing the entire dataset; *(iii)* the intermediate output of segment processing can be monitored to provide reasonably frequent feedback at a minimum, but possibly also to support iterative fine-tuning of phases in a process. None of these notions is new to computer scientists and engineers in the era of distributed computing and predictive data mining. However, in this very conjuncture where the data bulk reached far beyond the critical mass, we may wonder whether the *quality* of data has reached a degree of manageability that is aligned not only with their quantity, but also with processing techniques. Should the aforementioned quality standard not be reached yet, we would then have to resort to manipulation techniques that allow us to reconcile data with a manageable equivalent.

[1] An online article by statistician Meta S. Brown, titled *The Big Data Blasphemy* [Bro], was widely regarded as an omen to the ultimate failure of greedy data processing techniques applied to Big Data.

### 1.1.1 From Linked Data to combining ontologies

As Big Data have a general scalability requirement, so do linked data [HB11a]. Linked data (LD) are, in brief, structured data published on the Web that conform to a small set of principles, as in Berners-Lee's seminal article [BL06], whereby every datum is identifiable in the same way as Web pages are, and provides some interlinking mechanism that conforms to certain standards (RDF and SPARQL being some)[1].

Over the turn of the past decade, Linked Data practices have seen a gradual and steady uptake from content providers, although the vast majority of data published this way comes from a restricted set of domains such as biology, medicine and scientific publications. Other domain data providers, such as news and media publishing, have begun to follow their example [Pel12], however, the race to linked data publishing alone is not exempt from drawbacks and caveats.

In order to be published in Linked Data form, content providers need to conform to some model that expresses the categories of, and relations between, the things described in the contents they intend to publish. At a bare minimum, a *vocabulary* that gives names to these categories and relations is necessary. If a restriction on the exclusive usage of terms from a vocabulary is enforced, then the vocabulary is *controlled*. However, if one wants these terms to represent something more meaningful than barebone character sequences, some formal standard should be adopted for giving them a structure for non-human agents to do something about. Ways for doing so include assembling terms into a tree-like hierarchy called *taxonomy*, or relating them with one another by using standard relations that also indicate similarity, complementarity etc., thereby constructing a *thesaurus* (cf. Section 2.2).

Taxonomies, thesauri, (controlled) vocabularies and other knowledge representation paradigms such as *topic maps* [CGP+03] use different levels of expressivity to achieve what is essentially a common goal, i.e. to provide a *model* for structured data to conform to. An all-encompassing concept that tries to summarize those mentioned above is that of *ontology*, or Web ontology if kept within the realm of the Web. In fact, the concept

[1] In fact, the term "Linked Data" was originally coined to denote the publishing method and set of principles in question, but its versatile semiotics allow for its usage to denote the data that conform to them. To provide some form of disambiguation, we shall henceforth refer to the principles as "Linked Data" in capitals, and to the data themselves as lowercase "linked data". The intended meaning of the LD acronym is to be drawn from context from time to time.

of ontology is broad enough to describe both the model and the published data that conform to it, even in combined form. When we request information on an entity in linked data by submitting its identifier, the structured information we receive in return can be called an ontology, no matter if it contains only statements that describe that entity directly, or also other entities that belong to its category.

There are a number of methods for consuming linked data on the Web [SM11], depending on which ones each provider decides to support. For minimum Linked Data compliance, it has to be possible to access data piecemeal, by resolving entity identifiers one by one. However, data servers should also be able to respond to structured queries that can encompass multiple entities. Some providers go as far as to make their whole dataset available as a single resource [DBP]. This dataset can be either generated on-the-fly upon request (live dump) or updated from its backend at scheduled intervals (periodic dump). Size aside, the delivered results can be said to constitute an ontology, whichever the method used. It is the case of entity-by-entity queries, whole dataset dumps, and the results of certain queries that construct new knowledge graphs.

Note, however, that the reason behind querying linked data is to process them, which implies that an agent on the client's side will have to interpret them, and this agent is not so likely to be human. In such a scenario, interpreting a large dataset dump could be not only computationally unfeasible, but also an overkill if the objective is to extract information on only a subset of those data. In addition, a dataset is not necessarily self-descriptive: in fact, it should reference and reuse as much knowledge already defined as it makes sense to do, for the sake of the "linked" property. In practice, realizing this requirement at the instance level means stating in a dataset, for example, that "*Veronica Ciccone* as described in dataset X is the same as *Madonna* as described in dataset Y, and everything stated about either individual in either dataset also holds for the other individual in the other dataset", all the while maintaining, by using the same formalism, that *Madonna* is a separate entity from *Saint Mary*, for whom "Madonna" is one possible appellation.

It is not infrequent for the model of a dataset to be described in an ontology separate from the data descriptions. Although it is a plausible practice of ontology engineering, it is also an issue to be taken into account, in that if one wishes to infer additional knowledge, it should be possible to have access to both the model and the data and blend them together. More so, the terminology used in the model could either be totally

developed in-house, or reuse members of other shared and/or widespread vocabularies, in which case it can be beneficial to include them as well in the knowledge base to be processed. We conducted ourselves a study on heterogeneous Linked Data resources in the multimedia domain, and detected datasets whose underlying model was not made public, and yet it was still possible to extract an approximate yet nontrivial model, by combining recurring patterns of vocabulary usage [PAA+11, PAG+11].

To add some realistically even greater complexity, the knowledge published by linked data providers and ontology engineers does not necessarily make up all that is required for performing certain knowledge-dependent tasks. For instance, if an application in a multi-user system were to recommend its users open-air events that match their preferences, then the knowledge base should include material that is typically not published as Linked Data. This could include, among others, a semantic representation of the target user and her preferences (typically a piece of privileged information within the system) and a representation of weather forecasts for the region of interest (e.g. a weather reporting service feed that needs to be somehow turned into an ontology).

### 1.1.2 The role of ontology networks

The bottom line of this rather hurried and informal introduction is that we cannot expect to work with a single source of knowledge for performing knowledge-intensive operations. Structured knowledge is typically layered and distributed across the Web and beyond. Different portions of knowledge are authored in different contexts, unaware of what other *a posteriori* formal knowledge they will be combined with.

Along with the need to combine distributed Linked Data come *ontology networks*, collections of ontologies somehow linked with one another. Although the majority of the work carried out on ontology networks is rather recent [SFGPMG12b], mentions of related terminology date up to a decade earlier [Var02]. So far, ontology networks were treated mainly on the ontology engineering side, in that most phases of their lifecycle, including design, reengineering, modularization, repair and argumentation, are focused on developing ontologies according to a set of best practices. In other words, when ontology management methodologies mention the usage of ontology networks, they mainly refer to how formal knowledge should be organized at *design time*, and in that context they encourage techniques such as reuse and modularity, which help shaping knowledge in a network-like structure.

From such an angle, managing the networked architecture is a process that ends once the ontology is deployed, with a new one starting the moment that ontology is reused as part of another network; a network which is intended to follow a similar methodology and persist on the Web upon deployment. To rephrase this more synthetically, ontology network management is treated as a *static* process. This is natural for methodologies of ontology engineering, to be summarized in the next chapter, and not to be intended as a shortcoming on their part. However, the need for reusing and combining knowledge manifests itself in dynamic forms as well. For use cases such as semantic mashups and those described in the previous section, chunks of heterogeneous knowledge may have to be combined, processed, and possibly even disposed thereafter. Well-structured linked data reuse vocabularies from other sources and reference entities in other datasets whenever possible, but they do so without necessarily referencing the *sources* where those terms are defined, and whose definitions could have an impact on the outcome of procedures applied to linked data. Creating the missing formal interlinks then becomes a task for the agents that perform these procedures, which are different, in general, from those that deployed those data. This process is akin to the creation of ontology networks, although it is required to be performed *dynamically*.

Likewise, semantic agents such as reasoners typically treat a knowledge base as a single artifact, therefore a set of seemingly disconnected ontologies needs to be presented to them as if it were a single ontology [HPPR11]. This is a limitation that recent approaches are trying to overcome by extending the ontology language models to allow distributed reasoning [Mut12, VSS+08]. However, traditional reasoning techniques, whose implementations are now tried-and-true, should still be accommodated where this is possible. Once again, assembling these ontologies into networks dynamically can help solve these issues.

#### 1.1.2.1 Use case: multi-user content directories

One use case that will be a driving example across this dissertation comes from the discipline of content management, which will be further contaxtualized in Chapter 3. For now, let us consider a content management of system (CMS) to be a document repository of scholarly publications, which multiple users can have access to, and customize by supplying their own documents. The document repository contains a shared

directory, in turn containing documents which multiple users have access to, as well as a private directory for each user.

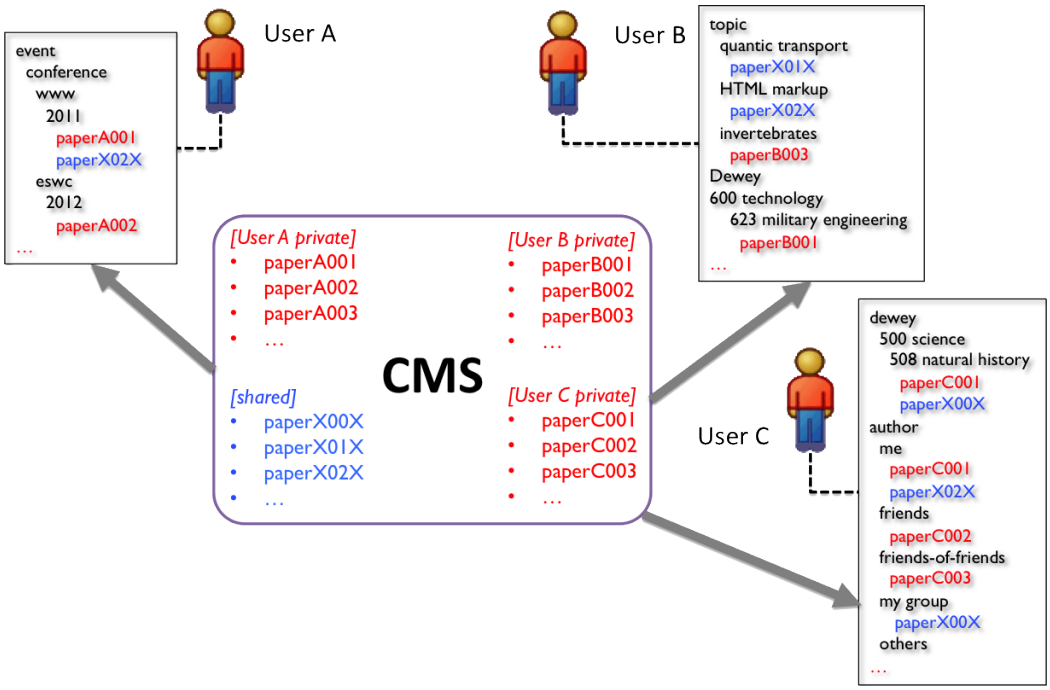


**Figure 1.1: Overview of a multi-user content directories use case** - The boxes next to users A, B and C show the view each user has on the documents she has access too, according to criteria specific to each user. One way to implement this classification scheme is by executing inference rules or reasoning on ontology networks built for each user.

The directory each user sees is initially flat, i.e. the publications are not organized in a sub-directory structure as in file systems. In this use case, each user requires her accessible papers to be organized according to a logical structure that reflects her preferences and profile, including the possibility to retrieve one document under multiple subdirectories. For example, one user (A) might want to retrieve documents in a directory structure that involves events associated to them (e.g. the conference where a paper was presented, such as `conferences/eswc/2012`); another user (B) might want to see them classified by topic, where the topics are categories in Wikipedia [Wik] combined with Dewey decimal classes [Mit09]; a third user (C) might require only the Dewey classification as well as a second classification by the place their authors have in her social (e.g. `friends`, `friends-of-friends`, `out-of-network`) or corporate/academic networks (e.g. `mine`, `my-group`, `collaborators`, `others`); and so on. A

schematic representation of this use case is exemplified in Figure 1.1.

This process should be automated as much as possible. One of the ways to do so is by classifying the content directory of each user by running classification tasks on ontologies aggregated by combining data from various sources. These include metadata extracted from the documents themselves and integrated with metadata from the Web, as well as Linked Data such (the Dewey Linked Data representation being one [Dew]) and the ontologies that describe the vocabularies used. If too many different vocabularies are used by the sparse Linked Data sources, additional ontologies that align them to a restricted set of terminologies used by the application may have to be factored in.

Note that this scenario is akin to using symbolic links in a file system. In such a scenario, one has to choose or create a directory where a file is initially placed, and only then is it possible to make the same file appear in other locations by creating symbolic links that point to the original file, possibly even with different names that make more sense in the directory at hand.

### 1.1.3 Issues

There can be stringent practical limitations to dynamic ontology network assembly. Write access rights, for one, are enforced very effectively on Web resources, and can restrict the possibility to modify existing ontologies. Modification can be a fundamental requirement, since ontology networks can be constructed by injecting links into the ontologies themselves. However, even assuming there were no access restrictions, modifying the source of an ontology for creating a custom network could create race conditions with other agents that intend to reuse the same ontology into another network, or even use it as a singleton. Thus, creating ontology networks through an intervention in their sources becomes an unacceptable practice.

Maintaining local copies of ontologies is then a fallback strategy that can be followed, but it exposes agents responsible for assembling networks to other risks. For instance, if these local copies were to be republished on the Web as parts of a network for public consumption, this would give rise to redundancies that risk violating guidelines and best practices for reconciling ontologies in the Web. That is, unless certain formal precautions are taken, such as logically renaming local copies, before republishing them.

Drawbacks can be even more subtle. Generating an ontology network through a naïve or brute-force approach, such as creating a single ontology that imports every

other ontology in the prospective network, has a potential for causing loss of expressivity in the resulting combined ontology, since the choice of a network structure can influence the underlying logical profile. This is a creeping issue that can occur without throwing error conditions, unless these are expected; it was proven in the early stage of our work and formalized by negating our null hypothesis (cf. Section 3.5.1).

The motivations for the research work presented herein take into account not only the issues described above, but also the great likelihood that, in this scenario, ontology network assembly is treated on the side of knowledge consumers. Consumers (as are human agents, e.g. users or application developers) are less likely to be ontology specialists than the knowledge engineers who authored those datasets, vocabularies, and more generally ontologies, in the first place. One of our aims is to provide some guidance as to how ontology networks should be assembled even when lacking such knowledge.

## 1.2 Structure of the dissertation

The remainder of this dissertation is structured as follows:

**Chapter 2 - Ontologies and their management: state of the art.** This chapter outlines the seminal work, bodies of standards, similar problems and alternative solution proposals around our area of research, including a quick and by no means complete introduction to the Semantic Web. Mentions of related work are also featured in other chapters.

**Chapter 3 - Goal Overview.** The choice to tackle the issues arising in ontology network management was partly made *ab initio*, partly specialized and corroborated serendipitously as work proceeded. This chapter is a summary of the **motivations** that drove the whole work, the research problems addressed, a sketch of the proposed solution and the assumptions and restrictions adopted along the course.

**Chapter 4 - A model for ontology network construction.** This chapter describes the **theoretical and logical framework** that makes up the notions, both existing and new, of ontology networks as are needed for understanding the present work.

**Chapter 5 - Architectural framework.** The model presented is made of theoretical and logical components. However, since we are addressing ontology management issues on the *application* side as well, a demonstration on how these components can be brought into at least one meaningful software model is necessary. This chapter describes a possible **software architecture binding** for modular application frameworks and RESTful services.

**Chapter 6 - Implementing the model: the Stanbol ONM.** This thesis work is backed by **technological support** for the hybrid architecture presented earlier. A reference implementation was developed as part of a software project fully endorsed by the Apache Software Foundation. This chapter provides an insight on the engineering details of the Apache Stanbol Ontology Network Manager (ONM), as well as a quick access guide and technological stack.

**Chapter 7 - Evaluation and discussion.** The aforementioned reference implementation was used as the testbed to evaluate our proposal. Various aspects of the evaluation were taken into account, both qualitative with respect to ontology features in description logics, and quantitative with regard to memory efficiency. This chapter describes all these aspects of the **evaluation** and discusses its results. A theoretical framework grounded on graph theory is also outlined for a better comprehension of the qualitative evaluation process.

**Chapter 8 - Conclusions.** A summary of the overall research activity and possible lines of **future work** to be followed from the current state concludes the dissertation.

# 2

# Ontologies and their management: state of the art

The aim of this chapter is to provide an overview of the research context of our work, and guide the reader throughout this landscape so that the drivers of, and motives behind, the arising requirements and implemented solutions, are justified and understood. Rather than limiting this survey to what formal ontologies and controlled vocabularies are being used in certain domains, which would be merely an exercise in researching industrial trends, we intend to make use-cases emerge, which can in turn raise nontrivial requirements on the knowledge management front. To this end, it is paramount to give an overview of the Semantic Web philosophy, the guidelines and efforts of Linked Data, and their relationships with ontologies. These relationships are not obvious and not uniformly perceived across scientific and technological communities. The understanding of the threads that link these domains and philosophies is deeply influenced by the cultural background of individual scholars, whether they are logicians, newly-formed Web scientists or knowledge engineers; whether they were aware of, or contributing to, the history of this field, and if so, on which side. Given the relative youth of the "ontologies for the Web" discipline, debates involving these different *formae mentis* are not uncommon, nor is the tendency to call fundamentally equivalent concepts by different names. As with many worldly things, the truth probably lies in the middle, therefore it is our aim here to unify these notions in a way as harmonically comprehensive as possible.

## 2.1 On Linked Data and the Semantic Web

In 2001, Berners-Lee, Hendler and Lassila published an article [BLHL01] that anticipated an ongoing and foreseen transformation of the Web as it was known then. According to this vision, information that was almost exclusively open to consumption by human agents would slowly be structured in ways that allow computing agents to interpret and process it using techniques that mimicked human reasoning. This vision was labeled **Semantic Web**, meaning in the authors' own words:

> *a web of data that can be processed directly and indirectly by machines.*

Clearly, this vision implies lending the Web to machine-processing techniques that target human needs. Web laypersons would benefit from this extended Web by being able to retrieve, share and combine information more easily than on the traditional Web, unaware that this greater ease is guaranteed by the ability to *unify the data* behind the information presented to them.

For a simple example, let us consider a customer who is searching for a record album to purchase. The album is found to be sold by a digital music store as MP3 and by an online retail store on CD and vinyl, and also three second-hand copies are sold on an online auction service. In a traditional Web, these would be six distinct objects, not to mention the number of available copies on the retail store and the potentially limitless digital downloads. These object would not be formally related with each other, except by a few character strings, e.g. in their titles and tracklists, which could still contain alternate spellings, different titles by country and mistakes. The customer would have to monitor them as distinct items, and in order to retrieve more instances of that record album she would have to issue a text search which may or may not be resistant to inexact spelling. In a Semantic Web, the user would be given a unique identifier of that album that is valid for the whole Web, and use it in order to be notified of any digital downloads, retail availability, second-hand copies, auctions, or special editions in any version and country. Besides this consumer-centered example, the endless application areas and strategies of the Semantic Web also involve the unification of knowledge for life sciences and healthcare, but also coordinating distributed service-oriented architectures (SOA) for processes in eGovernment as well as molecular biology. For an overview of the active and potential application areas and

scenarios of the Semantic Web and related technologies, we refer to existing surveys and in-use conference proceedings in literature [CHL07, FBD+02, BT05, HHK+10].

In this form, the Semantic Web is not tied to any specific standards or technologies, although its conception is necessarily rooted in the pre-existing, hypertext-based Web and plans to leverage and extend its technologies and protocols. As the Internet protocol suite, commonly known as the TCP/IP stack, is one possible and widespread implementation of the OSI model of open systems (ISO/IEC 7498-1) [Com00], so was an abstract architecture devised for the elaborate Semantic Web layered system – what is known today as the *Semantic Web stack*, or *Semantic Web layer cake*[1]. It is essentially an architectural paradigm aimed at covering every aspect of this extension of the Web, assuming a working connectivity stack to be in place. The stack covers aspects such as a shared lexicon, interchange and query mechanisms, knowledge structuring and processing, up to the aspects of security, provenance and presentation.

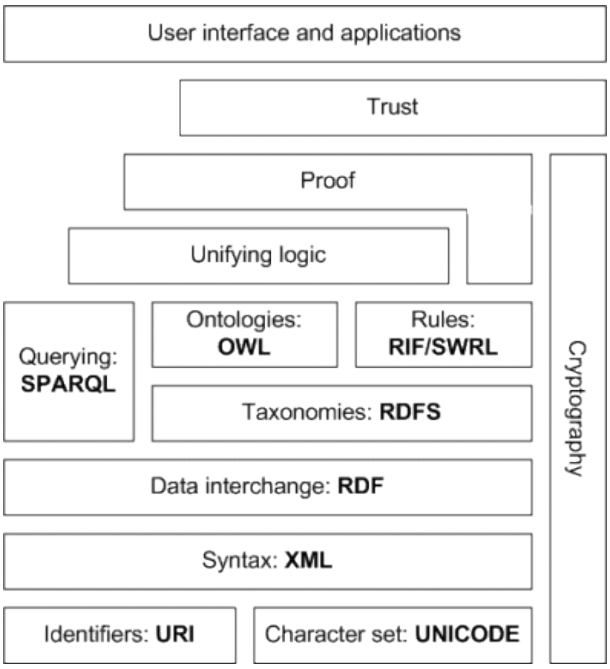


**Figure 2.1: Status of the Semantic Web stack implementation as of 2012** - Retrieved from [Sem].

[1]First presented in a keynote at XML2000, slide available at: `http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html`

Not unlike the relationship between the OSI model and the TCP/IP stack, one aim is to pair a technological stack with the abstract Semantic Web model and construct a body of standards around it. In response, the World Wide Web Consortium (W3C) is spearheading a full-scale initiative to create this body of standards iteratively [W3Cb]. Figure 2.1 displays the state of play of a reference implementation of the Semantic Web stack at the time of this writing. The W3C has issued technological recommendations that cover mainly syntactical and interchange standards, its cornerstones being:

- a schema of uniform identifiers for all the things that can be represented and referenced, i.e. the **Uniform Resource identifier (URI)** [BLFM05];
- a data interchange format based on a simple linguistic paradigm (**RDF**) [MM04];
- a language for querying data in the agreed interchange format (**SPARQL**) [PS08];
- a vocabulary for the above interchange format that allows a simple organization form for knowledge (**RDFS**) [BG04];
- languages for representing complex knowledge (**OWL**) [Gro09] and inference rules for execution (**SWRL**) [HPSB$^+$04] and interchange (**RIF**) [Kif08].

Many of the above formalisms incorporate the XML markup language as the base syntax for structuring them[1], plus a body of knowledge representation formats and query languages.

As it turns out, the higher-order layers of the stack covering user interaction, presentation, application (one example being the support for compressed RDF datastreams), trust and provenance are going largely uncovered, and so is the vertical, cross-layer security and privacy component. This has raised concerns over the short-term feasibility of a secure [Pal11] and interactive [HDS06] Semantic Web. However, efforts within and outside the W3C are being undertaken for establishing *de facto* standards with the potential for becoming recommendations and completing the Semantic Web stack reference implementation. These endeavors will be discussed in the course of this chapter, along with many of the aforementioned technologies and standards.

[1] In response to the asserted redundancy of XML notation, standardization work is underway for setting JSON as an alternate exchange syntax for serializing RDF, thus leading to proposals such as RDF/JSON and JSON-LD [Woo].

It is noteworthy from Figure 2.1, that this architectural stack includes a set of high-level knowledge representation structures, and that this set of structures allowed for *ontologies* in all of its revisions since 2001, alongside rule languages and as an extension of taxonomies. Ontologies were always somehow legitimated as an integrating and essential way to construct formal structures that could serve as a logical backbone to all the references published by peers on the Web. This notion on ontologies and their evolving trend towards networked, interconnected structures has encouraged us to further study and support this field in the present work.

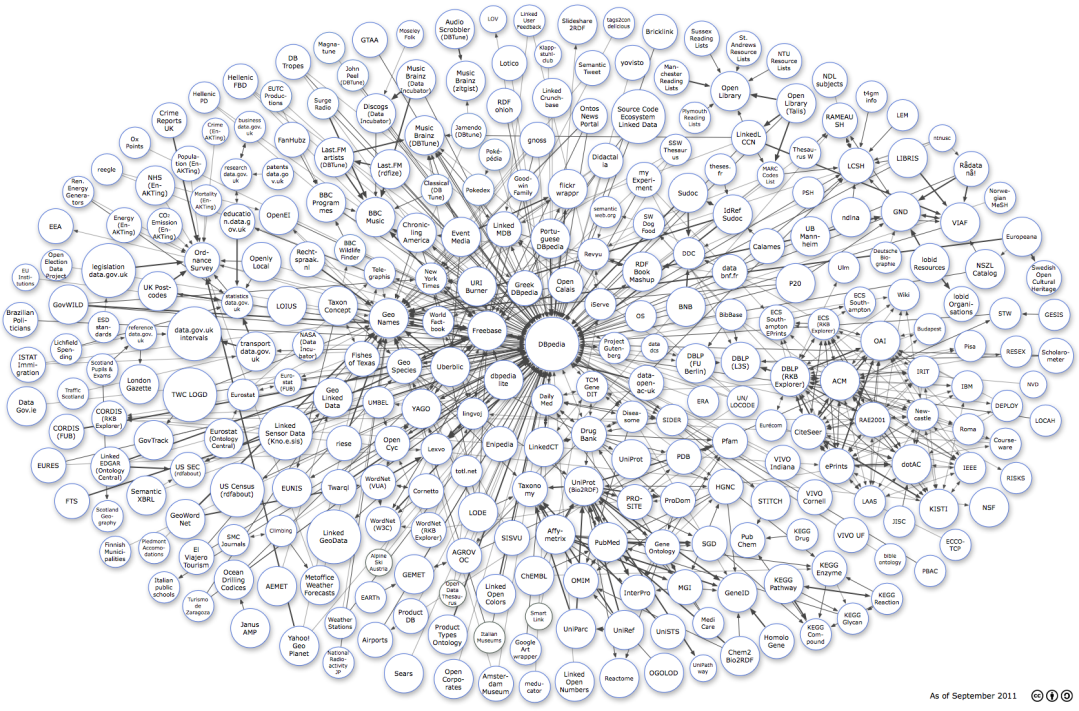


**Figure 2.2: Linked Data Cloud as of September 2011** - The latest graphical snapshot taken by the Linking Open Data community. Retrieved from [Lin], issued under a Creative Commons License.

The Semantic Web stack is but a portion of the Semantic Web vision conceived by Berners-Lee et al.: it describes the basis of any possible protocol suite that conforms to its technological specifications, but does not cover the principles by which *data* should be generated, formats aside. To that end, a method for publishing data accordingly was outlined and called **Linked Data** (LD). The principles behind this method are as simple as using HTTP URIs for identifying things, responding to standard lookups

(e.g. SPARQL or URI dereferencing) with standard formats (e.g. RDF or a specific XML schema for query results) and curating cross-links between things [BL06]. In order to encourage the adoption of these principles and maintain a harmonic data publishing process across the Web, the *Linking Open Data* (LOD) community project brings guidelines and support to Linked Data publishing and performs analysis and reporting on the state of affairs of the so-called "Linked Data cloud" [Lin].

The most recent report from the LOD group is summarized as the picture in Figure 2.2, where each node represents a dataset published by the LD principles, and a directed arc between two nodes indicates that a reasonable amount of entities in one dataset is described with relations that link them to entities in the other dataset. The purpose of reproducing the LOD cloud in so small a scale in this dissertation is clearly not to illustrate who the players and participants in the LOD cloud are, but to provide a visual overview of how distributed and interlinked it is, as well as the apparent linking trend to a limited range of datasets. Related figures report 295 datasets and over 31 billion statements, or *triples*, over 42% of which coming from the eGovernment domain and 29% from the geographic and life science domains, totaling over 53 million cumulative outgoing links [BJC11].

## 2.2 Knowledge representation methods and technologies

The following is a rapid survey on techniques, formalisms, standards and applications targeting the representation of knowledge (or *knowledge representation*, KR) in a software-processable manner. Some of these will serve either as an inspiration for the KR technologies to be researched and implemented, or as mechanisms that have been extensively used in the past for encoding knowledge as in our proposed solution. A tremendous amount of knowledge published as Linked Data over the years conforms to vocabularies and specifications antecedent to the second revision of OWL, a state-of-the-art KR language. Sometimes, this choice is dictated by computational constraints and lower expressivity requirements even for up-to-date linked data providers. However, the Linked Data principles imply that backwards compatibility should be ensured across representation languages, so long as an agreed-upon interchange format is used. To our concern, this means that we should address the need for reusing third-party knowledge and being compliant with the standards adopted for representing it. Yet,

when generating ontology networks with our presented method, we should take state-of-the-art technologies and standards into consideration as a preferred choice for KR.

Many of the KR standards described here are built upon, or compatible with, a data model that has seen widespread adoption and usage in Web resources and applications. The **Resource Description Framework (RDF)** [MM04] is a family of World Wide Web Consortium (W3C) specifications for representing information on the Web [Res]. Among these specifications, the RDF data model exploits a recurring linguistic paradigm in Western languages, by representing all statements as subject-predicate-object expressions, hence their name *triples*. In an RDF triple, subject and predicate are resources identified by URIs, while the object can either identify a resource or a literal value. Since the object of one triple, when a resource, can be the subject of one or more triples, the resulting triple set forms a *graph*, whose nodes and arcs represent resources depending on the roles of the latter in each triple.

### 2.2.1 Taxonomies

Many widespread and well-received knowledge representation schemes are hierarchical, in that knowledge items are arranged in a tree structure according to a selected relation or set of relations. In **taxonomies**, the relation in question is derived from the types assigned to items, i.e. their *categorization*, thereby forming a *subsumption hierarchy* [Mal88]. Taxonomies have been featured in a variety of application domains over the years, including the linguistic domain and the problem of biological classification, one prominent example being the Linnaean taxonomy from 1735 and its derived studies [Ere00]. In the medical domain, the classification of diseases, or *nosology*, arranges diseases based on their causes, pathogeneses, or symptoms [Sni03], all of which imply the creation and assignment of categories and therefore a taxonomy. One core resource is medical classification is the International Classification of Diseases (ICD) from the World Health Organization [Int]. Other hierarchies other than taxonomies can adopt relations other than subsumption, with *meronomies* established by part-whole relations as their cornerstones.

Representation formalisms for taxonomies and hierarchical structures were generally a responsibility of the knowledge domains where they generated. Little was achieved in terms of standardization, beyond visual representation primitives and hierarchical database models [BHN80], until the problem of representing more complex structures

eventually accommodated taxonomies as well and led to the birth of the **RDF Schema (RDFS)** [BG04]. Taxonomies cannot be represented using the RDF vocabulary alone, since the latter offers very limited semantic structure beyond the possibility to assign a type to a resource, leaving it up to knowledge management processes to define terminologies and subsumptions. RDFS provides a base mechanism for making these terminologies shareable, by providing specifications and additional constructs for expressing classes and subsumption relationships, and by extension taxonomies [McB04].

### 2.2.2 Thesauri

Still predominantly a linguistic framework with a focus on term relationships, the *thesaurus* is a knowledge structure that attracted great interest in a possible standardization. A thesaurus is, in its broadest sense, a grouping of terms by their similarities in meaning. At a bare minimum, thesauri admit a hierarchical relation as in taxonomies, namely the *hypernym/hyponym* pair for specifying broader and narrower terms. The semantics of these relations can be associated to those of weak subsumption, which admits relations other than type subsumption, such as "myrtle green" (a shade of the green color) being narrower than "green", in turn being narrower than "color". At this stage, there is no distinction between instance-based and class-based subsumption. In addition to taxonomies, thesauri can express term equivalency (*synonym*), similarity (*near-synonym*) and opposition (*antonym*). Typically, a further associative relation exists for terms that are related, yet in none of the ways described above. This "related term" indicator is no further specialized, thus its usage should be controlled [RT04].

The study of thesauri was of particular interest to us mainly due to the heavy standardization work and subsequent bulks of computer-processable structured data that derived from them. Aside from ontology languages and schemas to be described later in this section, a state-of-the-art knowledge representation method to date is a language for representing thesauri. The **Simple Knowledge Organization System (SKOS)**, a W3C Recommendation since August 2009 [MB09], is a purely RDF(S)-based vocabulary that implements the above relationships and constitutes a leading approach for bridging knowledge organization and the Semantic Web [MMWB05]. SKOS encompasses several formal languages for representing knowledge organization systems, thus allowing to model taxonomies, thesauri [vAMMS06] and controlled vocabularies in

general [Can06]. The SKOS core is concept-centric, in that its primitives are abstract concepts that are represented by terms.

Like the very notion of thesaurus, SKOS is a general-purpose knowledge representation schema that is widely used in domains such as eGovernment, scholarly publications and digital libraries [MBS12]. Other contemporary controlled vocabularies rely on custom languages for thesauri: some being RDF-based, such as the *AGROVOC* language for agricultural resources [SLL+04] and the representation schemes for the *WordNet* linguistic resource [vAGS06]; others being built upon custom XML dialects such as the Medical Subject Headings (*MeSH*) project [MTG09, NJH01].

### 2.2.3 Frames

Originally introduced by Minsky, linguistic frames, or knowledge frames, or simply **frames** [Min81], are another contribution to studies in artificial intelligence to which Web ontologies owe greatly. The author's original definition follows:

> *A frame is a data-structure for representing a stereotyped situation.*

In knowledge representation theory, the derivation of frames from taxonomies is made explicit: there, a frame is a structure that aggregates facts about objects or their types into a taxonomic hierarchy [RN10]. Frames, however, aggregate multiple relations at once, which are held in *slots*. The slot values for a single frame determine the set of all and only its valid predicates, or *facets*, hence the general assumptions of a *closed world* being modeled in each frame. We will return to this notion shortly. Another noteworthy feature of frames is that they allow for procedural attachments, which hold procedures for setting constraints to slot values, e.g. "age" being a positive number, or even computations, e.g. on how to calculate "age" given the value of "date of birth". So defined, frames effectively served as seminal knowledge structures for the object-oriented programming paradigm [Rat93].

To formally represent frames for consumption by computer agents has been one of the key challenges prior to the paradigm shift towards ontologies. *Frame languages* tend to incorporate specific frame logics. *KL-ONE* described frames as "concepts" and slots as "roles", and employed structured inheritance networks to define well-formed concepts on the basis of their multiple inheritance [BS85]. It should also be observed that other frame languages such as *F-logic* [KLW95] and the *Ontology Inference Layer (OIL)*

[FvHH+01] laid the foundations of modern ontology languages and were among the first to be concretely employed for, and conceived within, the Semantic Web, granted that their semantics differ from the description logic most ontology languages are based on. F-logic, for one, retains the closed world assumption typical of frames [WNR+06] and, unlike common description logic fragments, is generally undecidable.

Consumption and usage of frames as expressed in frame languages is rare. However, their employment in other forms such as Linked Data is not uncommon. One of the largest publishing resources for frames to date is *FrameNet* [FBS02] [Fra], which stores frames using custom XML schemas and exposes them using lexical units as bindings between terms and their meanings.

### 2.2.4 Ontologies

Despite the historical significance of *ontology* as the branch of metaphysics that studies categories of beings, the adoption of constructs in artificial intelligence called *ontologies* is recent compared to those discussed in the previous sections. More precisely, the consolidation of ontologies as theories of a modeled world open for consumption by machines coincides with the birth of **Web ontologies**, i.e. ontologies represented in formal languages that can be rendered using standards of wide acceptance on the Web.

In 1993, Gruber gave an initial definition of ontology that was widely accepted in the computer science domain [Gru93a]:

> *An ontology is an explicit specification of a conceptualization.*

This definition was further elaborated upon by the author, who wrote in 2009 [Gru09]:

> *...an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.*

In literature, an ontology is also described as an artifact whose structure matches both its domain and intended task, and where the design is motivated by some specific purpose, e.g, the capability to solve some reasoning task or modeling problem [GP09].

Web ontologies have since become widespread standard artifacts for representing and reasoning upon knowledge. The formal concepts from these sets of knowledge are defined either within a given domain or across multiple domains, along with the logical relationships between and constraints upon these concepts. Ontologies combine many types of semantic structures, both hierarchical (e.g. taxonomies, meronomies) and non-hierarchical (e.g. the equality and opposition relationships in thesauri), and can do so on any amount. Ontologies also allow for a significant, albeit conceptually lax, decomposition into terminological and assertional components. A terminological component, or *TBox*, is a statement that describes the mechanisms in a part of the world in terms of controlled vocabularies. An assertional component, or *ABox*, is a statement that describes the population of that part of the world in compliancy with the controlled vocabularies defined by TBoxes; in other words, a *fact*. An alternative terminology has TBoxes containing *intensional* knowledge and ABoxes containing *extensional* knowledge [Gru93b]. While there are no strict rules set as to what levels of representation should be handled by ABoxes rather than by TBoxes, this being in fact an open issue in knowledge representation, this distinction has some essential implications on the practice of modeling systems. The ability to discern assertional components, such as factual and domain knowledge on the objects and event occurrences in the real world, allows us to keep controlled vocabularies general, reusable and independent, in principle, of the model population. This is far less evident when dealing with conceptual models such as those of relational databases, where the underlying semantics are implicitly expressed in relational schemas. Note that the classical TBox/ABox dichotomy is not the only decomposition of knowledge accepted in computer science: in fact, in the information extraction field ontological knowledge can be modeled to distinguish referential domain entities, conceptual hierarchies, entity relationships and the domain population [NN06].

The state of the art in Web ontology authoring is the **Web Ontology Language (OWL)** [Gro09]. Its definition relies on a stack of representational schemas that cover multiple layers of Berners-Lee's Semantic Web layer cake (cf. Section 2.1). OWL provides the framework for bridging RDFS and ontologies, but its foundations are to be traced back to frame languages and beyond. In fact, it superseded the earlier ontology language **DAML+OIL** [MFHS02], which combined features of the OIL language described earlier while discussing frames, with the **DARPA Agent Markup Lan-**

**guage (DAML)**, a United States defense program for the creation of machine-readable representations for the Web [HM00].

OWL is actually a *family* of languages that allow knowledge engineers to model domains under the principles of Description Logics (DL). These are knowledge representation languages that model concepts, roles, individuals and their relationships through an *axiomatic* theory. Languages in the OWL family are founded on model-theoretic formal semantics and define fragments of the First-Order Logic (FOL) with well-defined computational properties. In the first version of the languages, these fragments are called *OWL-Lite*, *OWL-DL* and *OWL-Full* and consist of specifications of which constructs of the OWL language should be used for coding ontologies and under which restrictions. OWL fragments provide each a trade-off of expressivity and decidability [MvH04], with OWL-DL being the most widely used, as it is the most expressive fragment that guarantees the decidability of OWL reasoning procedures.

A revision of the language for accommodating the feature requests that arose from the heavy adoption of OWL in semantic applications led to an updated W3C recommendation for the updated OWL 2 in late 2009 [Gro09]. In OWL 2, the distinction between existing OWL fragments fell in favor of three new language profiles, all decidable but with different computational properties. These profiles are called *OWL 2 EL*, *OWL 2 QL* and *OWL 2 RL*, and named after their intended usage in knowledge management procedures for largely expressive ontologies, query answering procedures and rule systems, respectively [MGH$^+$09]. Because they address computational complexity issues rather than calculability, these profiles are also called *tractable fragments*, since they pose several restrictions on OWL language constructs and axioms in order to address certain scalability requirements deriving from interoperability with rule languages and relational databases. Other innovations contributed by the OWL 2 specification that are interesting for our work include the introduction of reflexive, irreflexive, and asymmetric object properties, i.e. properties holding between individuals; a clearer distinction between ontology locations, set in statements following an import-by-location scheme, and their logical names; a versioning system with an impact on physical and logical referencing; use of the Internationalized Resource Identifier (IRI) [DS05] as a generalization of the URI for referencing entities and ontologies; and support for *punning*, a meta-modeling capability that allows the same term to reference, under certain

restrictions, more entities of different types, e.g. classes, individuals and properties [GWPS09].

### 2.2.5 Serialization of ontologies

This section briefly reviews the most common existing formats that can be used to serialize, or *marshall*, an ontology into a document, e.g. a text file or data stream. For each format, the Multipurpose Internet Mail Extensions) (MIME) type for content negotiation using the HTTP protocol is also indicated.

- **RDF/XML** is the normative syntax for serializing RDF for machine readability [MM04]. It is a standard method for writing RDF graphs to an XML format and can fully accommodate the OWL language [Bec04] (`application/rdf+xml`).

- **Notation 3 (N3)** is a compact text-based RDF serialization format [BLC11]. It allows triples, which share either the subject or the subject+predicate pair, to be collapsed. N3 has a few additional capabilities on top of RDF, such as the ability to express rules [BLCPS05] (`text/rdf+n3`).

- **Terse RDF Triple Language (Turtle)** is a variant of N3 that support prefixes and qualified names [BBL11] (`text/turtle`, or `application/x-turtle` prior to type registration as a standard).

- **N-Triples** is a subset of both Turtle and Notation 3 designed for ease of parsing by software systems, thus lacking the shortcuts provided by qualified names and grouping [GBM04] (`text/rdf+nt`).

- **JSON-LD** (JSON for Linked Data) is a Linked Data interchange language for JavaScript and Web Service environments, as well as unstructured databases [JSO]. There is a specific syntax for serializing RDF graphs in JSON-LD, thereby embedding OWL as well [SL12] (`application/json`).

- **RDF/JSON** is a proposed, non-normative syntax for serializing RDF graphs using the JavaScript object notation [RDF] (`application/rdf+json` (not registered) or `application/json` if JSON-LD is not being used alongside).

- **OWL functional syntax** uses the actual grammar that formally defines the OWL language [MPSP+09]. It is the text serialization of the OWL abstract syntax, which expresses ontologies in terms of their axioms. As such, it has full support for the OWL 2 language features (`text/owl-functional`).

- The **Manchester OWL syntax** is a derivative of the OWL functional syntax that is more concise and built for human readability, especially by practitioners with a lesser background in description logics. It supports nearly the full OWL 2 specification, with a few exceptions such as annotations on undeclared entities and general concept inclusions [HDG+06] (`text/owl-manchester`).

- **OWL/XML** is the recommended syntax for serializing OWL 2 ontologies in an XML format without transforming them to RDF graphs first. As such, it is not directly interoperable with RDF/XML [MPPS+12] (`application/owl+xml`).

Note that only OWL/XML and the OWL functional and Manchester OWL syntaxes are *native* OWL formats, in that they store axioms and entities in terms of what constructs they are in the OWL languages. Every other format is used to encode RDF graphs, which have to be interpreted in OWL according to RDF mapping guidelines of the W3C recommendation [PSMG+09]. As the next chapter will clarify, however, these guidelines do not cover the whole spectrum of possible side-effects.

In this thesis, we shall make use of the OWL functional syntax as a litmus test for evaluating the interpretation of non-native ontological resources as OWL ontologies. This syntax will be used for representing the interpretations, be they actual, optimal or expected, of non-natively expressed ontologies.

### 2.2.6 Querying and processing ontologies

The core language for querying RDF data, and by extension LD sets and ontologies serialized in an RDF format, is called **SPARQL**. It is based on the expression of triple patterns for retrieving matches in the target RDF graphs. The initial W3C recommendation of 2008 [PS08] implements verbs for retrieval only, either as arbitrary results sets (SELECT verb), or as RDF graphs in their turn (CONSTRUCT and DESCRIBE verbs). This specification had no support for create, update and delete operations and

supports OWL only as one possible RDF vocabulary. Recent proposed recommendations for an updated SPARQL 1.1 specification are targeting extended features [SPA] such as graph update [GPP12], concatenated triple patterns [Sea10], federated querying [PBAS+12] and, most importantly for the current context, support for querying entailed triples according to OWL entailment regimes [GOH+12], among others.

The underlying formal semantics of OWL is also being applied to **rules and rule systems** in the Semantic Web, where a *rule* is an implication axiom that defines conditional facts. The **Semantic Web Rule Language (SWRL)** [HPSB+04] is one such language that combines OWL DL and Lite with Datalog [HG85]. SWRL retains the expressivity of OWL DL, albeit at a price paid in decidability, and is supported by a variety of applications of OWL and RDF too [MB06]. Other rule language, such as the **Stanbol Rule Language** [Apai], a derivative of the Semion rule system [NGCP10], aim at versatility and selectively compute the applicability of clauses depending on their target application being OWL (SWRL) or RDF (SPARQL). Versatility is also addressed on the interoperability department, with proposed standards such as the **Rule Interchange Format (RIF)**, which proposes dialects for porting basic logics and production rules across different rule systems [Kif08].

Because of its strong foundation on formal logics, OWL lends itself to semantic processing by **DL reasoners**, i.e. software engines with the ability to infer the logical consequences from a set of formal axioms belonging to the realm of Description Logics. Among the most widespread ontology reasoner implementations we cite *Pellet* [SPG+07], *HermiT* [DRS09], *FaCT++* [FaC] and the commercial *RacerPro* [Rac]. In addition, the interoperability protocol **OWLlink** provides a unified interface for accessing reasoning services [LLNW11]. An OWLlink implementation exists as an extension of the *OWL API* library [OWLb] and is interoperable with most of the reasoners mentioned above.

Implementations of DL reasoning sport a great variety in terms of the Description Logics they can support, as well as the classes of consequences they are able to infer. Some reasoners are limited to detecting the specific DL flavor of an ontology and checking whether it is consistent, in that the truth or falsehood of every fact in the ontology can be stated. Others can derive inferred taxonomies and arbitrary predicates that hold implicitly for general TBoxes (subsumption, satisfiability, and classification) and ABoxes (retrieval, conjunctive query answering).

## 2.3 Networked ontologies

In recent literature [SFGPMG12a], an **ontology network**, or network of ontologies, was defined to be

> *a collection of ontologies related together via a variety of relationships, such as alignment, modularization, version and dependency.*

The ontologies in such a collection are then called **networked ontologies**. The above definition is wilfully ample in scope, as it is not established a priori, but a partial consequence of existing practices in the management of ontologies and linked data, which were moving towards consolidation faster than an all-encompassing formalization could be attempted on them.

Let us consider one of the core principles of Linked Data, which advocates referencing of shared entities across multiple data sources, each of them improving upon the description of an entity monotonically, i.e. by adding statements that further describe it and consolidate its model. Then if we treat every such portion of linked data as an ontology, we can intuitively argue that these linked data chunks altogether form an ontology network, where the connectivity mechanism is provided by reuse. Although reuse is not specifically addressed by the above definition, it can be implied by other relationships such as alignment and dependency, where if an entity is depended upon or must be aligned with, then the goal is to reuse it.

This section provides a summary of known research efforts targeting the treatment of ontologies as networked artifacts. Aspects both formal and practical/methodological will be illustrated in order to set a comprehensive context for our work.

### 2.3.1 Formal specifications

We have recorded some attempts, both prior and contemporary to our work, at providing specifications of the relationships in multiple ontologies, which can be used to establish connectivity to one another. Those which surfaced first are actually the result of a bottom-up analysis work on domain ontologies, and as such have a bias towards the similarities that occur at a conceptual level [KA05]. Some such properties, albeit not formally defined at this stage, descend from the perspective on ontologies as collections of agreements over domain terminologies, and identify relations such as sharing the

same conceptualizations (e.g. by equivalence of applied logical theories), resemblance, simplification and composition of ontologies.

A further step towards a rigorous definition of these relationships came in the form of an ontology itself, i.e. the **Descriptive Ontology of Ontology Relations (DOOR)** [AdM09]. This is a modeling approach to the problem of formal ontology relations, as it uses the primitives and rules of ontological languages to define a hierarchy of possible relations that may occur within an ontology network, namely binary relations occurring between two ontologies in a collection of ontologies, or ontology space. Note that the notion of ontology space is different from those given in other contexts, such as the Cupboard ontology repository (cf. Section 2.4.1) and even our own ontology network model, where the matching concept is that of *knowledge base*. We will return to this aspect in due course, as Chapter 4 is introduced. DOOR distinguishes relationships holding at the lexicographic, syntactic, structural, semantic and temporal levels. On the top level of the hierarchy, there are inclusion relations, agreement and disagreement (which are further specialized using compatibility factors which can cause incoherence or inconsistency), ABox/TBox connections, alignments, similarities and versioning (which, in turn, incorporates aspects of ontology evolution on the conceptual front).

Diaz et al. argued that the relationships described in DOOR require further specification, claiming that the logical constructs in the original ontology aimed at classifying relations in a hierarchical structure, rather than applying them directly. Therefore, they proceeded to provide DL-based definition of some relations extracted from DOOR, namely *isTheSchemaFor*, *isAConservativeExtensionOf* and *mappingSimilarTo*, as well as extending them with a purely syntactic relation called *usesSymbolsOf* [DMR11].

The work presented in this thesis partly takes inspiration from the bottom-up work that led to DOOR, and is contemporary to the attempts made at formalizing its approach. However, our theoretical goal is not to provide a framework for describing existing relations, rather, a general-purpose framework that can be used to *accommodate* relations resulting from bottom-up analysis, and then some; so, if further connectivity factors emerge from future ontology use, our proposal can serve as a base toolkit for rendering them formally. Another reason for us to come up with a basic theoretical framework is that the aforementioned approaches do not take into account the discrepancy between the ontologies as high-level logical artifacts and their raw sources which have to be interpreted as such, and only deal with the former level. This is expectable,

as these approaches were the result of an analysis work over the way ontologies are engineered. However, if the content of these sources alone, in terms of raw statements that map to logical axioms, no longer matches these axioms deterministically, as we will later verify, then any formal specification falls short of practicability. For instance, if an ontology that is supposed to extend an object property of another ontology turns out having its extensions classified as annotation properties, then a relation such as *isAConservativeExtensionOf* can no longer hold between these.

### 2.3.2 Distributed ontology management

Many research directions somehow related to ours are often found as part of the discipline that goes by umbrella term "distributed ontology management". The various meanings of this term can be grouped into two categories. One, more closely related to ontology networks even in the dynamic context targeted by our work, is the class of operations that can be performed on top of a set of ontologies scattered around a distributed system like the Web, and which can, in some cases, sport logical connectivity as in ontology networks. These forms of distributed ontology management address several popular challenges such as ontology evolution, versioning and modularization. The other category concerns performing distributed computation on ontologies that could possibly even be large and standalone. As such, it addresses computational issues including federated querying and scalability in reasoning processes.

One of the seminal approaches to tackling several problems in ontologies *as distributed Web artifacts* dates back to the early 2000's [MMS+03] and was a precursor in terms of providing a modeling framework that could encompass both standalone and distributed ontologies, where a distributed ontology is implicitly defined to be formed of some ontologies stemming from others via reuse and domain-specific adaptation. In such cases, aspects of versioning ontologies and managing change propagations [Kle04] come into play and were explicitly addressed with respect to the formalisms and technologies available then.

Understandably, distributed ontology versioning is also a concern of engineering disciplines, and appears in proposed implementations of ontology management systems. The aptly-named DOMS (Distributed Ontology Management System) was a proposal that came prior to the normative recommendation of version IRIs in the OWL 2 specification. DOMS was a comprehensive ontology management service provider backed

by a relational database system (RDBMS) for storing ontologies. Version control was managed internally as part of the embedded schema in the backend RDBMS, therefore it was mainly an internal use case that would typically not be reflected on the publishing process of ontologies in DOMS, as it was mostly meaningful to DOMS users only. In this and other cases [NM04, SHL10, PNC12], distributed ontology versioning was tackled mainly from the point of view of detecting, tracing and managing changes in an ontology [Yil06], rather than from that of *transparency*, i.e. representing different versions in a meaningful way for ontologies published on the Web.

One interesting notion in distributed ontology management, which we considered in the definition of our theoretical model, is that of dependency [KSKM07]. Dependency is defined in terms of specific relationships holding between descriptions of one or more concepts in multiple ontologies. Some dependency types are defined from class hierarchy, such as *super-sub* relations, while others are defined from cross-referencing in class restrictions and so on. While dependency detection and management is crucial, in our view, for understanding the subtleties of ontology management in distributed environments, the experiments we have taken hinted that it should also be escalated to dependency relations holding between ontologies and the resources that represent them.

We briefly review some advancements in terms of computational aspects on distributed management. Lee et al. [LPP+10] propose a model and method for decomposing ontology queries in a distributed way and optimizing the results from the various decompositions. This method heavily relies on mapping techniques and assumes the triple, in terms of binary relation, as the atomic component of any ontology. As such, queries are treated as triple patterns, a notion that maps very reasonably with low-level languages like SPARQL but would require further abstraction work to high-level representation languages like OWL. Most of other related work in this department concerns the distribution of reasoning or query execution *agents* across a network built around the peer-to-peer model. Edutella [NWQ+02] was one such ad-hoc network, which however was broadcasting queries across all peers, until the system evolved with the introduction of routing strategies based on peer clustering techniques in 2003 [NWS+03]. Other systems concentrated on distributing reasoning process rather than optimizing queries: it is the case of the DRAGO framework, which still relies on a peer-to-peer architecture, where every peer holds not only a set of ontologies assigned to it, but

also a set of mappings from its local ontologies onto those managed by a selection of other peers. DRAGO was focused on TBox reasoning and inferring subsumption relationships [ST05].

Once again we could find little mention, in these approaches, of the problem of assessing the impact of the interpretation of distributed sources as ontologies, since many such approaches assumed ontological axioms (TBox, ABox and non-logical) to be already computed at the time reasoning or querying takes place. One exception is the KAONp2p system, which, although mainly concentrated on distributed querying and reasoning, introduced a first hint at a "virtual ontology" system that would mimic a standalone ontology on top of a distributed set [HW07].

### 2.3.3 Methodologies

Taxonomies, thesauri and ontologies in general follow a life-cycle of which development is merely a part. Ontologies need to be both built at one time and maintained afterwards, therefore the choice of methods and best practices for engineering them becomes a concern. It should also be noted that ontology development is a practice that involves domain experts as much as seasoned knowledge engineers, therefore methodologies and best practices become crucial if it is the users who are expected to author knowledge and/or semantic content based thereupon.

Early ontology development methodologies, such as **On-To-Knowledge** [DFv02, SS02] and **METHONTOLOGY** [FGPJ97], did not focus on a possible distributed nature of the ontologies under development, being mainly concerned with ontology generation from semi-structured industry documentation for the former, and the application of software engineering activities to ontology development for the latter [CFLGPLC03, PSM08]. Subsequent development methodologies, however, embodied a degree of awareness of ontology networks and related conceptual and structural features, hence their mentions in this overview. **DILIGENT** [VPST05] expanded upon methodologies like the above, by including a five-step process for managing the evolution of ontologies rather than their initial design, thus recognizing knowledge as a tangible yet moving target. The **NeOn Methodology** [SFGP09] is arguably the best-known methodology to openly address ontology networks as its target – NeOn standing for "Networked Ontologies" [NeOa]. This methodology introduced and formalized the collaborative

aspects of ontology development and reuse, as well as the dynamic evolution of ontology networks in distributed environments. At the different stages of the development process, contextual information may be introduced by domain experts and ontology practitioners. Great focus is given on practices such as reuse, reengineering [VT12] and modularization [dSSS09], which became the keystones of ontology network management [SFGPMG12b].

One peculiar case is that of *pattern-based methodologies*. These are methodologies that make extensive use of so-called *Ontology Design Patterns* (ODPs) [GP09], or ready-to-use building blocks for solving atomic modeling problems. Most methodologies of this type cover primarily the logical level, in that they provide support for ontology learning, enrichment and similar tasks [NRB09, Blo09], while putting little to no focus on solving concrete modeling problems. However, it should be noted that their ultimate products are most necessarily ontology networks connected through reuse at a bare minimum, but possibly also through versioning and dependency management. Examples of pattern-based methodologies include the **Ontology Pre-Processor Language (OPPL)** [IRS09] and methods for applying it as a means for logical ODP reuse; the high-level pattern language proposed by Noppens and Liebig [LvHN05]; and **eXtreme Design (XD)** [PDGB09, BPDG10], which adopts the use of competency questions [GF94] as a reference source for requirement analysis. XD focuses on producing modular networked ontologies that extensively reuse ODPs, and applies best practices of software engineering and a test-driven development approach for ontologies.

Finally, we note that many of the methodologies mentioned earlier come with a degree of tool support that integrates with software systems for authoring ontologies. Recent development platforms such as **Protégé 4** [Pro], the **NeOn Toolkit** [NeOb], and the commercial **TopBraid Composer** [Top], provide support for methodologies such as XD and the NeOn Methodology and, to various extents, acknowledge the networked nature of the ontologies under development. Most of these tools rely on a plugin-based architecture and support basic functionalities such as graphical editing of ontological elements and axioms, but also more advanced features such as visualization, reasoning, and query support.

The reason behind this quick overview is that ontology networks are typically objects whose creation and assembly is mainly a design process and a responsibility of

teams that combine domain experts and knowledge engineers who follow a certain design paradigm. As such, ontology design occurs in a controlled environment where, for example, there is next to no ambiguity in treating knowledge objects as classes, individuals or properties of a certain type, or on what are the types of axioms handled. Serializing the resulting ontology networks into a set of resources that are not natively ontological (i.e. whose raw statements are not coded as high-level axioms), such as RDF, is an operation performed often automatically and under the assumption that no resolution ambiguity will occur when the resources are deserialized back into ontologies. However, outside these controlled environments such as at runtime assembly, this assumption is heavily put under question, as Chapter 3 is set to demonstrate.

## 2.4 Ontology repositories

The present work is strongly tied to pre-existing and ongoing advancements in the field of ontology repositories. In its most general form, an **ontology repository** [VTMH12] is a location where the content of ontologies is stored. On top of this vague notion, the goal of an ontology repository also defines its more specific features. Ontology repositories can exist for simply preserving ontology data, and as such will almost exclusively sport storage capabilities. Other repositories are aimed at serving ontologies to Web applications, thereby providing service interfaces for accessing them, negotiating their formats for applications, rendering them for Web browsers and indexing them for search engines. Others are focused on the interaction between ontologies and their users or developers, and therefore may provide advanced functionalities for collaborative editing, peer review and version control. Finally, some repositories can be used for storing knowledge on specific domains, and therefore can restrict some of their functionalities, such as indexing, search facets and key concept detection, to these domains.

As will become clearer further in this dissertation, our interest in ontology repositories for the sake of this work is not so much on specific ontology repositories available on the Web, rather, on the software systems used for setting them up and deploying them. These, in turn, can consist of redistributable packages that in effect make up the skeleton of multiple ontology repositories. Throughout this dissertation, these packages will be interchangeably labeled *ontology repository systems*, *ontology repository software* or *ontology servers*.

Dedicating a section to state-of-the-art ontology repository software has purposes not only comparative, but also complementary. Some evaluation results (cf. Chapter 7) will imply that our approach fares better than, or equal to, other existing software architectures. However, the novelty of the approach introduced herein is not necessarily incompatible with these architectures. On occasion, it can be regarded as a companion architecture that can either extend the functionalities of another repository system, or serve as a backend for it, or even be part of its reference implementation.

### 2.4.1 Cupboard

**Cupboard** [dEDL09, dL09] is an online ontology repository with dedicated functionality packages targeted at human-ontology interaction. Being a multi-agent environment that can be used to host multiple configurations for ontology sets, it is also self-described as an ontology repository hosting system. Each Cupboard user can setup a space to upload her own set of ontologies, independent from those of other users, and make them selectively available to the community and the Web. Uploaded ontologies are indexed for retrievability by services such as the *Watson* API [dM11], which is also available within Cupboard, and can be peer-reviewed and rated publicly. Argumentation in Cupboard implements the *TS-ORS* open rating system [Ld10]. Ontology alignment services are also incorporated within Cupboard via the *Alignment API* [Ali], and allow users to produce alignments between the ontologies loaded into their personal spaces.

In slight anticipation of the description of our virtual ontology network infrastructure, to be described in the remainder of this dissertation, Cupboard does feature a notion of *ontology space* too. However, even though this definition lies in a similar ambit as ours, they are not in contrast for the most part. An ontology space in Cupboard is a private resource associated to a user account, where all management operations (incl. alignment, indexing and search) are confined. Ontologies loaded in multiple Cupboard spaces can be completely redundant and map to distinct graphs, and the notion of sharing ontology networks applies within a single ontology space, rather than across multiple spaces. In the architecture we devise, an ontology space is a component of one or more virtual ontology networks, with privileged write access depending not on user account preferences, but on the state of its associated ontology networks at one time. Multiple spaces can expose a shared graph as multiple Web resources for the

same ontology, and the policies defined for spaces define whether a stored ontology can outlive the spaces that manage it or not.

Plans exist for making the Cupboard repository software available as a redistributable open source package [dEDL09]. At the time of writing, a release is still expected.

### 2.4.2 The TONES repository

Part of the **Thinking ONtologiES (TONES)** project, [TONa], the TONES repository [TONb] is mainly targeted at application developers who need to reference common and diverse ontologies for testing purposes. At the time of writing, the repository hosts 219 ontologies containing up to one million logical axioms. The repository can be browsed by ontology DL expressivity, size in number of axioms, signature size and conformance to the OWL 2 EL profile. A RESTful service allows each ontology to be exported in either RDF/XML or OWL/XML, although no recombination services are available other than exporting the whole repository as a single merged ontology. The repository is not publicly write-accessible and the framework it is built upon is not redistributed. However, it has been disclosed that the TONES repository uses the OWL API [OWLa] [HB11b] for ontology access, export and rendering.

### 2.4.3 ontologydesignpatterns.org

**ontologydesignpatterns.org (ODP.org)** [ont] is an ontology repository specifically targeting the lifecycle management of ontology design patterns [BJ08]. It is mainly intended to store ontology design patterns, and encourage reuse thereof, for the benefit of ontology engineering tasks [PDGB09]. In addition, it serves as a backend for several tasks in the XD methodology (cf. Section 2.3.3). However, ODP.org also stores ontology networks that incorporate design patterns, such as the *C-ODO Light* meta-model for modeling the design aspects of ontology network management [APH+12], as well as a limited range of singleton (non-networked) ontologies. The repository is characterized by a feature-rich Web interface targeting user interaction with ontologies, encouraging formal specifications, strict reviewing processes, trust and provenance indications and argumentation. The storage mechanism of ODP.org is filesystem-based and ontologies are served verbatim in the same format as they are stored, mainly the XML serialization of RDF. Therefore, the ontology networks hosted by it are static and must be imported

as they are, making it impossible to add runtime dependencies to an ontology without undergoing the strict review process that applies to final changes to the ontology. It is also worth mentioning that ODP.org implements its own registry mechanism for logically organizing ontologies into libraries, and tool support exists as part of the *eXtreme Design* toolkit, for resolving references contained in libraries, browsing registries and interpreting their metadata vocabulary [PBDG12].

### 2.4.4 Oyster

The **Oyster** ontology sharing system is a rather peculiar entry in this section, in that it is a software system for maintaining not ontology repositories, but ontology registries [PHGP06, Pal09]. An ontology registry of this form stores and maintains metadata on ontologies, rather than their contents. Oyster uses a metadata vocabulary called OMV [APH+12] for managing such metadata in a collaborative, multi-user environment. It is mentioned in this section for two reasons: one, if we abstract on the distributed information sources referenced by a registry, we can consider the combination of a registry management system and its sources as an ontology repository; two, because of vocabulary support provided by OMV for these features, Oyster does provide a host of change capture and version management features for ontologies, which are typical of repository systems. An alternative angle on registry systems is to view them as client agents for ontology repositories. On the change capturing side, Oyster can be set to monitor a referenced ontology and store modifications as atomic axioms in an internal metalevel ontology compliant to the OMV. On the versioning side, the Osyter specification assign precise semantics to the versions assigned to revised ontologies, although it is not concerned with the syntactic binding of version publishing. This aspect was later addressed in the OWL 2 specification, which we adopted as a base mechanism for identifying an ontology stored in our system, and distinguishing it from its disposable variants created by our virtualization method. Reusing parts of the OMV for representing the metadata of our system is one possible evolution direction for recording change management and propagation from dependencies. However, we anticipate that the policy taken by the OMV and other approaches, which assume a total ordering for ontology versions, is not compatible with our proposed solution (cf. Sections 4.1.3.1 and 4.3) and restricts the potential of the OWL 2 version specification which we chose to exploit.

### 2.4.5 The OBO Foundry

The **Open Biological and Biomedical Ontologies (OBO)** is an open community that fosters the application of the scientific method to the discipline of ontology engineering. This community spearheads a collaborative experiment of ontology authoring in the biomedical and biological domains called the **OBO Foundry** [CMBR11]. Its goal is to create a suite of orthogonal interoperable reference ontologies in these domains. The OBO has established a set of basic principles for authoring, reusing and versioning ontologies published through the Foundry, and provides some degree of software support. Although CVS, a first-generation general-purpose versioning system [CVS], is encouraged for versioning ontologies, this initiative is notable for providing a toolkit for preparing an OBO ontology for release. This toolkit, called **Oort (OBO ontology release tool)**, provides facilities like pre-release classification using OWL reasoners and just-in-time compilation which is akin to the interpretation of non-natively OWL ontologies. Although the OBO community does not explore the realm of ontology networks in depth, it does show awareness of the need for organizing multiple ontologies in certain ways, for instance by distinguishing bridge ontologies (containing alignments) from a single main ontology and the remaining reference ontologies. The Oort features ontology merging mechanisms based on **MIREOT (Minimum information to reference an external ontology term)**, which replaces direct OWL imports with a term-by-term selective mechanism [CGL+11]. MIREOT is essentially a way to prove fine-grained imports, however its experiments did not prove to solve, or even consider, the problem of erroneous OWL interpretations deriving from imperfect connectivity between ontologies.

### 2.4.6 The Open Ontology Repository

The **Open Ontology Repository (OOR)** initiative [Opea] is a community effort to promote the global reuse and sharing of ontologies on levels both methodological and technological [BS09]. OOR contributes with a modular, open source repository software with multiple deployments and a set of best practices for sharing. Ongoing work in OOR concerns registry management functions, support for multiple knowledge representation languages, alignment features, recommenders based on text analytics, curation policies and user interfaces for managing them. Great concern is also present for ontology

version control, provenance and change management. Moreover, the OOR community has shown awareness of the practical problems arising from inconsistent imports in OWL ontology networks, which are among the non-logical semantic structures that our proposed solution manipulates. Although the reference implementation of our solution was ultimately deployed within a separate project incubated in the Apache Software Foundation [Apag], we are closely following the evolution of the OOR initiative and participating in the argumentation occurring therein. While by no means a high-priority nor short-term goal, contributing our theoretical framework and part of its implementation to the OOR remains a possibility.

### 2.4.7 COLORE

Finally, we cite a very recent notable contribution provided by the **COLORE** ontology repository [COL], an extended instance of an OOR prototype. Like ODP.org, COLORE also uses ontology design patterns, but it also relies upon them as a backend for a general-purpose, cross-domain repository. It is based on techniques such as relative interpretation to characterize the models of ontologies based on a set of stored core ontologies [GK12]. Although the concept is never mentioned in the context of COLORE, this technique has strong similarities with ontology network management and sports an application of a logical framework to them. Therefore, a possible integration or reformulation of the underlying logics in the context of our work is possible and under consideration.

## 2.5 Existing ontologies and knowledge vocabularies

As the nature of the Semantic Web and ontologies is assumed to be an *open world*, many knowledge engineering methodologies advocate the reuse of knowledge and discourage the re-authoring of basic concepts and shared domain entities from scratch. In order to achieve this goal and support semantic interoperability, several Semantic Web projects, communities, and representatives are devising vocabularies of shared knowledge, which describe very general concepts that can be regarded as the same across all domains. Other ontologies focus on selected abstract domains, such as planning or state machines, in order to serve as references for certain widespread fields of practice and research. This way, processes such as the annotation and retrieval of knowledge will require minimum

effort in the alignment of concepts. The following is a quick and by no means complete survey on some well-known or scientifically relevant ontologies and ontology networks used for modeling domains, e.g. as core vocabularies for Linked Data or the semantic annotation of content.

We have already cited **SKOS** (cf. Section 2.2.2) as a leading example for modeling controlled vocabularies. Another noteworthy resource of shared foundational knowledge is the **WonderWeb** upper ontology library [OSV05], whose core module, the **Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)** [Des], uses first-order logic to capture the categories of beings from a cognitive perspective, i.e. through the cognitive artifacts of human perception, natural language and common sense [GGM+02]. The nature of DOLCE ascribes it as a *foundational ontology*.

The **Description and Situation (DnS)** ontology [GM03] aims at the efficient and expressive representation of contextualized knowledge: it allows knowledge engineers to re-interpret types and relations asserted in so-called 'ground ontologies' from a context-sensitive angle. DnS is reused as a pattern in specific domain ontologies including the **Computational Ontology of Mind (COM)** [FO04] and the **Core Ontology for MultiMedia (COMM)** [ATS+07]. DnS is aligned with DOLCE, which is used in its top level, so that a light version of DnS+DOLCE exists. **DUL** is a reworked version of DnS+DOLCE that retains most of its expressivity while significantly improving upon the original on the computational side [Gana].

One of the greater challenges in knowledge representation is the modeling of *context* [Guh91, McC93], a versatile term by its own right, as it can concern the linguistic, environmental, or social domain, among many others. The domain of ubiquitous computing [BBRC09] regards context as a collection of attribute values of a physical environment. Recently, several proposed context models for household automation, application development and reasoning, have also surfaced in ontological form [KC06, OGA+06, WZGP04]. **Constructive Description and Situations (c.DnS)** [Gan08] is a fork of DnS that does not depend on DOLCE. C.DnS can be used jointly with any ontology, be it foundational or domain-oriented. It represents different notions of context, such as situational, descriptive, informational, social and formal. The expressive power of c.DnS lies at the level where the semiotic activity of cognitive systems occurs, i.e. where agents encode expressions that have a meaning in order to denote

or construct entities in the world. c.DnS enables the representation of complex configurations of contexts that could occur simultaneously, but need to be distinguished in order to identify the most relevant contextual entities in a certain situation, e.g. a social network (a social entity) together with a tag cloud (an information entity) and a model (a formal entity).

Static and dynamic models of evolving and contextualized knowledge are a major focus of research on ontological grounding, which identifies some key aspects to be taken into account in order to create a bridge between the modeling layer and real-world facts and event occurrences. One such approach involves the meta-level manipulation of ontologies: the **C-OWL** language is an extension to OWL that introduces a discrepancy between *local semantics* and *global semantics* [BGH+03]. C-OWL adds bridge rules for the creation of context mappings, thus generating embedded ontologies that are contextualized, in that they do not share their data with the outside world. Other approaches tackle the more pragmatic aspects of contexts by providing knowledge engineers ways to model actual contexts and the phenomena that alter their states. It is the case of the **Time Indexed Participation** ontology design pattern [Gand] and **Linked Open Descriptions of Events (LODE)** [STH], two lightweight ontologies consisting of a limited amount of semantic terms for describing events, their spatial and temporal coordinates, and the agents and objects involved therein.

Other ontological approaches have dealt with the modeling issues deriving from planning tasks and workflows, as is the case of the work presented by Rajpathak and Motta in [RM04] as well as a DnS-based plan ontology that comes in versions with light [Ganc] and full [Ganb] axiomatization. The world of software engineering has also contributed to this field with formal methods for representing state-driven behavioral models, as in Dolog's **Ontology for State Machines** [Dol].

On the side of linguistic and semiotic approaches, the **Linguistic meta-model (LMM)** addresses the representation of thesauri and controlled vocabularies, yet does so on the basis of lexical and semiotic entities and multilingualism [PGG08]. LMM is an ontology network that extends DUL and is aligned with SKOS, the meta-model of OWL and the WordNet data-model. A pure structure-oriented approach is given by a model called **Linguistic Information Repository (LIR)** [PMPdCGP06, APH+12], also deployed as an OWL ontology. The LIR uses an internal OWL meta-model for directly associating lexical entries to ontology terms, thus allowing straightforward

localization of the terminological layer, and organizes linguistic information within and across languages.

In the horizontal domain or ontology engineering, several meta-models have been proposed and are currently in use. The **Ontology Metadata Vocabulary (OMV)**, a standalone ontology that provides a specification of reusability features for enhancing ontologies for human- and machine-processing. The **Collaborative Ontology Design Ontology (C-ODO) Lite**, an ontology network with light axiomatization and heavy reuse of ontology desigs patterns, aimed at modeling and exploiting the design features of the entire lifecycle of ontology networks [APH+12].

Alongside general-purpose semantic frameworks as the ones mentioned above, the collective effort to elicit a set of shared vocabularies for data-related knowledge has brought increasingly popular results. **Friend of a Friend (FOAF)** [GCB07] is an agent-centric vocabulary meant for describing persons, the interlinking between them, their occupations, related concepts and activities, with a special focus on the social (network) identity of an individual on the Web. It includes specific properties for instant-messaging identifiers, homepages and online accounts, while it does not represent their historical information, other than discerning past and present. Other popular Web domains, such as e-commerce and online communities, have given birth to vocabularies like **GoodRelations** [Hep08], which is used as a Web content annotation vocabulary for describing products, pricing, producer and payment methods; **Smart Product Description Object (SPDO)**, a DUL-based core model for semantically describing physical consumer products in communicative environments [MJ09]; and **Semantically-Interlinked Online Communities (SIOC)** [BDHB06], a vocabulary for describing the structure and contents of communities. As an example of semantic interoperability achieved through shared vocabularies, SIOC allows for reuse of the FOAF vocabulary for describing individual community members.

Cross-domain controlled vocabularies also exist, which by all means can be treated as ontologies, although their specification does not allow them to qualify as foundational ontologies, but rather as annotation vocabularies. In this respect we cite **Schema.org** [Sch], a cross-domain controlled vocabulary endorsed by major stakeholders of the Web search market. It is recommended for usage with the annotation of Web pages for better content ranking, retrieval and recommendation. As a response to the concerns that had risen over the migration from existing vocabularies to a shared one, Schema.org

was made extensible and interoperable with other vocabularies, through a review and approval process that determines whether search engines should support these extensions.

On top of the above, most common RDF vocabularies that deal with authored material are open to reuse of terms deriving from the **Dublin Core** metadata element set, a standard vocabulary for digital libraries [SBW02]. The RDF specification of the Dublin Core set [NPJN08] allows the straightforward definition of relationships such as creator, subject, format or language for any authored object.

Formal models for the representation of policies, such as copyright and access rights, have also proved to be an essential subject matter of investigation. The wide availability of vocabularies for describing content (whatever content is agreed to be within each context) also calls for models that define how this content should be allowed to be treated and by whom, and models that share the same basic formal semantics as these vocabularies. One such attempt to define a *copyright ontology*, aimed at the development of a semantic Digital Right Management (DRM) system, was developed by Gil and García in 2006 [GG06] and 2008 [GG09]. The ontology itself is monolithic, yet it models parts of the copyright domain in separate phases, such as creation, economic and moral rights, actions (e.g. redistribution, public performance, transformation etc.), licenses, roles, obligations and event patterns (e.g. peer-to-peer exchange). The proposal makes extensive use of cardinality restrictions and other DL features for modeling the complex domain of copyright. Another policy representation model is **Rei** [Kag02], implemented in OWL-Lite, which allows specification of policies as constraints applied to a resource, as well as conflict resolution and 'what-if' policy analysis. Another policy representation approach was defined as the **KAoS Policy Ontologies**, originally developed in DAML, then migrated to OWL [UBJ+03]. An example of a non-OWL approach is **Ponder**, a declarative object-oriented language that defines policies as system rules declared between sets of subjects and targets [DDLS01]. Ponder allows for the definition of obligation and authorization policies, as well as composite policies such as roles and relationships. A comparison of KAoS, Rei and Ponder was made available as of 2006 [TBJ+03].

Significant advancements were also spotted with respect to formal models for interaction and presentation. What is possibly the most mature RDF-based and application-independent proposal to date is **Fresnel**, an RDF vocabulary for application-independent

data presentation. Fresnel was designed an a response to the limitations of prior presentations models, which were argued to be excessively tied to the assumed data-model of the domain where the data belong, or the framework in which they are managed [PBKL06]. Fresnel is founded on the definition of *lenses* and *formats*. Lenses define which properties of an RDF resource, or group of related resources, should be displayed and in what order. Formats determine how resources and properties are rendered, and provide a way to relate to existing styling languages such as CSS. Implementations of Fresnel-based rendering engines are present in applications such as the faceted RDF browsers *Longwell* [Lon] and *Piggy-Bank* [Pig], and the *IsaViz* graph modeler [Isa], which is able to render specific properties as indicated by Fresnel formats.

# 3

# Goal overview

The world of ontology-based knowledge management has been evolving across multiple directions. It has been creating cross-discipline hybrids, specializing for domain-based adoption and developing new branches, all the while consolidating some of its standards, methods and technologies as the foundations of a discipline that can be taught as it gets steadily improved upon [GJ10]. In a context so vast and varied, it is inevitable for any study on ontologies to adopt a specific course of actions, based on which assumptions and restrictions are established, yet still with minimum loss of generality. As part of the aforementioned category, this work is no exception: while it stemmed from the requirements and aims set within the domain of ontologies in content management, service frameworks and data repositories, it stays as true as possible to *de facto* standards, World Wide Web Consortium recommendations [W3Ca] and the doctrine built thereupon. This chapter rolls out the motives, purposes and directions taken for the present work, in the traditional form of problem statements, assumptions and restrictions [AMH08].

## 3.1 Motivation

As stated earlier in the introductory chapter, there is a technological and conceptual gap between the publication of semantically structured data on the Web and their manipulation in the form of ontologies. This gap can be summarized as a *vexata quaestio*, i.e. a scientific and ideological problem over which a debate exists between

schools of thought in the research community: should semantic applications be aware that the data they process represent ontologies?

The implications of how this matter is approached are crucial. Application developers in the various fields where linked data are explored, consumed or created, are faced with wondering what the benefit of bringing ontologies into their workflows would be. This means ignoring that, when applications handle linked data and the vocabularies they are built with, they are actually manipulating ontologies, albeit at a lower level in terms of logical profiles.

There is a clash of doctrines at the basis of this debate [DW03]. The pure Semantic Web doctrine as in Berners-Lee's vision states that the ultimate goal of the Semantic Web is to make heterogeneous information available for agents to consume autonomously in order to satisfy high-level user goals. This implies that the Web should have its own harmonization mechanisms for manipulating structured data before presenting them to client agents. It would then be guaranteed that an agent that obtains information from various sources will know that it is reliable, accurate and consistent. This is indeed one goal of the currently un-implemented layers of the Semantic Web stack such as Trust, Proof, and the Unifying Logic according to which harmonization should occur. This doctrine justifies the existence of ontologies and the OWL language in the Semantic Web, as well as the need for reasoning processes taking place in the back-end of semantic data providers. It is then implied that, when a client obtains a chunk of data using a query language like SPARQL, these data would already contain the inferences and entailments that follow from treating the original resources as ontologies.

Another doctrine, which we may dub as the "pure Linked Data perspective", states that the Semantic Web is merely a way of publishing meaningful yet disaggregated information. Such information is made available for for consumption by agents, and it is up to them to decide which information to consider trustworthy and accurate, which should be considered for consistency issues and which should not. This leaves the choice of employing formal ontologies, rule systems and reasoners up to the responsibility of application developers. According to this perspective, the very need for unifying logics or even ontologies as components of the Stack is put into question, as clients should need to support no more than RDF, SPARQL and the technologies that form the basis for Linked Data.

In the introductory chapter, a use case was sketched in the realm of content management systems (CMS). The community around these systems is relatively new to semantic technologies, yet it had begun to show interest in Linked Data [HB11a] as a resource to be consumed in order to integrate or enhance pieces of content in a multitude of ways [MK12]. While eager to improve their knowledgeability, community members are rarely knowledge management experts, and viewed the Semantic Web from a perspective rather technological than scientific. Bringing awareness of the ontology language layers of the Semantic Web into this community is itself a research challenge, which can be conducted in accordance with either of the doctrines described earlier.

On the other hand, the great quantities of data that can potentially be stored or consumed by a CMS translate into equally great quantities of semantic data, and by extension ontologies, if the CMS is backed by a Semantic-Web-aware framework [MK12]. Thus a scalability problem ensues [Var02], as combining all the semantic data that are fed or stored into a single, huge ontology network can be unthinkable at times, as it may lead to an unbearable computational overhead for the responsiveness of a system that should be as close to real-time as possible.

A set of requirements for an ontology-enabled CMS [Pre12] pointed out the importance of cutting down a knowledge base by (de-)activating portions thereof, depending on the computational task being performed. This requirement analysis was part of the preliminary work for this thesis [ABG+10] and has led us to take up from it and tackle one problem emerging from these requirements. This problem is described next.

## 3.2 Final aim

Although it can specialize in ways that will be discussed further in the chapter, a generic formulation of our ultimate scientific target is as follows:

> **To exploit good practices of software solutions in order to manage concurrent ontology networks built dynamically, by maximizing reuse but with reduced usage of computational resources.**

Very informally, this statement is akin to assuming that building "good" ontologies is not enough for guaranteeing that their intense and concurrent usage does not place

an unreasonably high burden on the software that makes these ontologies available; or, to state something less obvious, that it is much less than enough. A great deal of research work has gone into laying out definitions of what a "good" ontology is (cf. 2.3.3 for details), and by extension, practices for managing their lifecycles. Most recommendations head towards authoring ontologies within manageable logical profiles (as do the tractable fragments, or "profiles", of the OWL 2 language [MGH+09]) and advocate reuse and connectivity techniques [SF10]. However, there comes a point where applications need to come to terms with the combined usage of ontologies in ways that their respective authors could hardly have foreseen.

Tackling the above problem is well above the level of a mere exercise in programming. Several factors are to be taken into consideration, such as the shape of an ontology network; its relationship with standard ontology languages; the persistence of network components based on their provenance, and so on. The serendipitous emergence of these factors gave rise to additional problems worth of study, some of which we accepted to tackle over the course of this research.

### 3.2.1 Host framework features

The aspect of computational resource usage is usually treated on the side of reasoners. Not as much publicity on this aspect is given to the side of ontology repositories and servers, having historically delegated its treatment to database specialists concerned with building efficient triple stores or quad stores [RDE+07]. However, graph persistence is but one low-level angle to regard resource usage from. Between that and the application level there can be additional stages in the ontology management pipeline to be considered, depending on the desired versatility. If the only use case of an ontology repository is to serve each ontology from a remote file as-is to one client at a time, there is virtually no resource usage issue in intermediate layers. If a more elaborate set of requirements and functionalities is at hand, though, its complexity could bottleneck the communication between repository and client(s) if not paid heed to.

The setting of our work falls in this latter category. The context of this work is based around the software model of a modular, extensible framework for Semantic Web applications, which will henceforth be called **host**, having the following characteristics:

- **shared knowledge base**: it features a single knowledge base shared across all applications in the framework;
- **Web services**: it provides a Web interface for external applications, i.e. based on the HTTP protocol, to expose its functionalities and those of its hosted applications;
- **concurrency support**: it needs to serve multiple requests for potentially different tasks and services at the same time;
- **locality**: it needs to process the payload of one or more requests in a privileged space, without altering other privileged spaces in unpredictable ways.

The challenge we were faced with was to introduce ontology management in such a software framework at a less than trivial level, that is, by making sure the requirements listed above held for ontology management as well, once appropriately ported to fit the domain.

### 3.2.2 Challenges in ontology management

In the early stage of this research, a preliminary requirement study was conducted, concerning what features a modular software framework should require from an ontology network when it needs to manage or utilize one [ABG+10]. The domain was content management, so the model of the framework in question was a platform for backing CMSs with semantic technology support. As a consequence, some requirements encompassed several horizontal domains of content management: an ontology network within this context was required to include ontologies that modeled domains such as trust and provenance for content items, users, linguistic and lexical resources, organizations, communities and workflows.

If we escalate the issue to general problems in ontology network management on a modular software platform, i.e. the host, then the distinction between domain ontologies loses its relevance. However, several orthogonal requirements can still be singled out. For full detail on ontological requirements for CMS applications, we refer to related publications [Pre12] and technical reports [ABG+10]. Requirements that concern the present study on a general host are listed below. According to them, an *ontology*

*network management system*, or *ontology network provider*, is a software system able to:

1. enable or disable an ontology (*node*, or *vertex*) within an ontology network upon request;
2. add ontologies to an ontology network;
3. remove ontologies from an ontology network;
4. bind one or more ontologies to a hosted software component;
5. unbind one or more ontologies from a hosted software component;
6. retrieve, parse and translate across standards and formats, ontologies *and other models* defined externally to the host system;
7. modify and extend ontologies in an ontology network;

In addition, a particular requirement encompassed the software binging of ontology networks, that is:

1. the software components registered with the host that can be bound to networked ontologies must share a common interface.

There is a degree of fuzziness in some of these requirements, due to their nature as software requirements aimed at addressing scalability. For instance, the meaning of "enabling" or "disabling" an ontology is unspecified in ontology theory. So should the differences between enabling (resp. disabling) and adding (resp. removing) and ontology be cleared out in the context of ontology networks. On the practical side, if enabling or disabling ontologies in a network were to require an intervention in their axioms, there would be a risk of shortfalls due to the lack of access rights on some ontologies. That is, unless they are somehow copied over to the host and served by it. In that case, there would be a risk of creating a mismatch between the actual location of an ontology and the one expected by its logical identifier. In addition, a specification of "binding" an ontology to, or "unbinding" one from, a software component would be necessary at least from a software-architectural perspective, in terms of what procedural

impact it has on a software component, but also what structural impact it has on ontology networks.

One objective of this thesis is to shed some light on these underspecified notions and their practical utility. Before formulating the issues described above as research questions, though, let us review the framework characteristics in terms of ontology management. An application able to serve ontologies from, and manage ontologies submitted by, client agents should:

- be able to extract ontological artifacts from the storage facility (or facilities) associated with the host, where applicable *(shared KB)*;
- serialize ontological artifacts to multiple formats and perform content negotiation to select the appropriate format on a per-request basis; allow create-read-update-delete (CRUD) operations [Mar83] via HTTP methods *(Web services)*;
- serve and handle multiple ontologies and multiple occurrences of the same ontology through simultaneous client requests *(concurrency support)*;
- for each client request, provide a network of ontologies that combines stored data and schemas with data supplied by the request. Changes triggered by, or affecting the supplied data should not be reflected into the ontology networks of other client requests *(locality)*.

As a consequence of the *Web services* requirement, since ontological artifacts are to be serialized to data streams in standard formats, we have to take account how client applications typically consume these data streams and "restore" their status as ontologies. In other words, this means we have to pay heed to what procedures these applications follow for *interpreting* these resources as ontologies.

### 3.2.3 Research questions

Let us now formulate the above challenges in terms of research questions:

**RQ1.** In a modular host framework with a shared knowledge base, is there a way for each application to combine a portion of the knowledge base with ontologies from outside the knowledge base, without affecting the ontologies managed by another application? Can ontology network management be an effective way of doing so?

**RQ2.** If the answer to the last question in RQ1 is yes, how can we define theoretically the following operations: (*a*) enabling and disabling ontologies in an ontology network, and (*b*) add/remove operations of ontologies in an ontology network, if there even is a need for (*a*) and (*b*) to differ?

**RQ3.** How structurally complex should the ontology networks built by the software system be, in order to reduce the likelihood of an erroneous interpretation of their contents?

**RQ4.** How can we bind a method for the dynamic construction of ontology networks to a procedural pattern for ontology-aware applications, so that they can set up ontology networks that respect the principles of RQ2 and RQ3?

**RQ5.** How can we maximize or encourage reuse of ontologies across ontology networks, with no disruptive impact on the memory footprint of these networks?

RQ1 is a direct consequence of having *locality* and a *shared knowledge base* as features of the host framework (cf. Section 3.2.1) and, at the same time, a requirement for managing ontologies retrieved externally, either verbatim or by reengineering non-ontological resources [VT12]. RQ2 openly addresses the requirements concerning enable, disable, add and remove operations on ontologies in an ontology network. RQ3 is subject to verifying whether the structure of an ontology network affects its interpretation in terms of axioms, which will be addressed shortly. RQ4 concerns the application of the theory behind the previous research questions, and treats the problem of describing how a software system registered with the host, or otherwise using its functionalities, should behave in order to exploit ontology networks. Last, RQ5 covers efficiency aspects, their critical nature being a consequence of the fact that the host framework supports *concurrency*. A measure of a non-disruptive memory footprint is that, if an ontology belongs to multiple ontology networks and there are $n$ requests for it in a short time, then the cumulative memory footprint of the requested resources should be less than $n$ times the memory taken by a single occurrence of the ontology.

The combination of requirements 1-3 for ontology network providers and research question RQ5 can be summarized by stating that ontology networks managed dynamically need to *scale*, hence the title of this dissertation.

## 3.3 Intermediate objectives

In order to answer the research questions above, and tackle the final aim as of Section 3.2, a number of intermediate targets were scheduled for accomplishment. These set objectives can be either methodological/theoretical i.e. aimed at delivering a combination of theory and best practices, or technological i.e. focused on delivering tangible products in terms of software and data.

#### Methodological/theoretical objectives

**O1.** To outline a theoretical framework that accommodates notions of "ontology network" and "interpreting as an ontology" and others related.

**O2.** Based upon the theoretical framework in O1, to define a method for constructing ontology networks dynamically. This will be based on the concept of "virtual ontology network", to be introduced alongside the theoretical framework itself.

**O3.** To provide an architectural specification of an ontology network provider that implements the elements required for executing the method in O2.

#### Technological objectives

**O4.** To provide the specification of a RESTful Web Service API for manipulating ontology networks, and the networked ontologies therein.

**O5.** To implement the ontology network construction method, as well as the part of the software-architectural framework in O3 that concerns the ontology network provider.

## 3.4 Assumptions

The work presented herein is based on the following assumptions, which we deemed reasonable with regard to the current state of play in Semantic Web science and applications:

**A1.** Any RDF-based resource on the Web can be interpreted as a set of axioms in the OWL language, even where the OWL vocabulary is not explicitly used. Therefore,

when discussing ontologies we shall also intend these resources, most including those retrieved from Linked Data.

**A2.** Both ontologies and other resources from which knowledge can be extracted as ontologies, are freely available with no restrictions to read-access or reuse, *republishing* rights included.

**A3.** An ontology is published on a Web resource by a certain peer, e.g. its author or otherwise rights holder (limited as per A2), and no one but that peer can physically modify the content of that resource.

**A4.** Software tools exist, both as end-user application and as service clients, that are able to consume networks of OWL ontologies built on import declarations, as well as interpret RDF resources in OWL.

**A5.** For each member of a set of axioms, it is always possible to determine whether it is part of *volatile* knowledge, which should not outlive the duration of a task, or *persistent* knowledge, which is worth storing and maintaining over time.

**A6.** Ontologies on the Web are not published optimally OWL-wise, i.e. there are redundant ontologies with the same axioms published as different Web resources with no adequate version indication, or with no OWL identifier.

**A7.** Rule engines and DL reasoners are freely available and support the OWL and OWL 2 languages.

**A8.** We are not given the details on how the software tools from A4 proceed in converting a resource into an ontology, nor can we assume them to always be free and open-source software (FOSS) [Fre].

**A9.** Practitioners and software developers who wish to *consume* ontologies do not necessarily know how to combine them into ontology networks.

Some of these assumptions, such as A1 and A4, are actually provable by example. Applications and program libraries that can be used to verify these assumptions were illustrated as part of Chapter 2. This software-related aspect is crucial to the completeness of our research setting, because the theoretical work done on ontology

management, as well as the documental contributions provided by the World Wide Web Consortium, do not fully address the problem space delimited by the aspects of networking and relatedness. Therefore, part of the burden of addressing these problems lies upon application developers, who are faced with computational resource management issues and the need to trade them off with the normative specifications above. Assumptions A4 (i.e. the existence of software) and A8 (i.e. the black-box nature of its ontology interpretation functions) altogether form this complementary space, which is both a solution provider and a problem source, as verified further on in Section 3.5.1.

## 3.5 Hypotheses

The objectives identified earlier, under the assumptions made, allow us to formulate a series of hypotheses, to be proven or disproven as our research work proceeds.

**H0.** (*null hypothesis*) The layout of ontology networks, i.e. the ways networked ontologies are connected with one another, has no effect on their interpretation in OWL.

**H1.** If each application sets up dedicated ontology artifacts that reference the (persistent and volatile) knowledge required, then each artifact determines a standalone ontology network that lives independently on any other ontology network.

**H2.** Spreading the components required by an ontology network across a 3-tier ontological structure will reduce the amount of axioms that cannot possibly be interpreted correctly compared to using a flat import scheme.

**H3.** An aggressive ownership and persistence policy applied to the data payload, and a conservative policy applied to the shared knowledge base portions, can reduce the memory occupation of concurrent ontology networks by at least one third of the memory occupied by each ontology network if it were fully in its own memory space.

### 3.5.1 Null hypothesis negation

That the null hypothesis H0 be untrue under the assumptions made, it can be verified through a simple counterexample to begin with. In the following, the original ontologies have been serialized in Turtle syntax.

Given the following prefix assignment:

```
@prefix : <http://www.ontologydesignpatterns.org/ont/test/ontonet#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix connected: <http://www.ontologydesignpatterns.org/ont/test/
  ontonet/imports-connected/> .
@prefix disconnected: <http://www.ontologydesignpatterns.org/ont/test/
  ontonet/imports-disconnected/> .
```

let ontology network $O_{disconn}$ have the following root node (i.e. the ontology that is encountered first when visiting the whole network):

```
disconnected:root.owl rdf:type owl:Ontology ;
  owl:imports disconnected:abox.owl , disconnected:tbox.owl .
```

The root ontology therefore directly imports both a TBox ontology and an ABox ontology. The imported ABox ontology is encoded as follows:

```
disconnected:abox.owl rdf:type owl:Ontology .

:Alex a :Person ;
      :knows :Begona .

:Begona a :Person .
```

The imported TBox ontology is encoded as follows:

```
disconnected:tbox.owl rdf:type owl:Ontology .

:knows a owl:ObjectProperty .
```

When `root.owl` was loaded using the OWL API 3 [OWLa] (e.g. by using Protégé 4.2) [Pro], the following OWL interpretation was returned (in OWL functional syntax)[1]:

[1]This ontology network example is available online at the intended URL `http://www.ontologydesignpatterns.org/ont/test/ontonet/imports-disconnected/root.owl`

```
Declaration(ObjectProperty(:knows))
ClassAssertion(:Person :Alex)
ClassAssertion(:Person :Begona)
AnnotationAssertion(:knows :Alex :Begona)
```

Note the presence of the axiom `AnnotationAssertion(:knows :Alex :Begona)`. The fact that it was interpreted as an annotation assertion in OWL implies the impossibility to reason on it, as non-logical axioms are not computed in DL. In terms of the leading example introduced in Section 1.1.2.1, this means that, even if we were able to place the papers authored by `Begona` in the `friends` directory of `Alex`, we would still be unable to place those from people in `Begona`'s network in the `friends-of-friends` directory, unless we fell back to low-level querying such as SPARQL.

Now let ontology network $O_{conn}$ have the following root node:

```
connected:root.owl rdf:type owl:Ontology ;
  owl:imports connected:abox.owl .
```

The root now only imports an ABox ontology, which is exactly the same as before, except that it imports the TBox ontology:

```
connected:abox.owl rdf:type owl:Ontology ;
  owl:imports connected:tbox.owl .

:Alex a :Person ;
      :knows :Begona .

:Begona a :Person .
```

The imported TBox ontology is also the same as before, except for its name:

```
connected:tbox.owl rdf:type owl:Ontology .

:knows a owl:ObjectProperty .
```

When `root.owl` is loaded using the same OWL API version, the following OWL interpretation is given[1]:

```
Declaration(ObjectProperty(:knows))
ClassAssertion(:Person :Alex)
ClassAssertion(:Person :Begona)
ObjectPropertyAssertion(:knows :Alex :Begona)
```

That is, what was interpreted as an annotation assertion in the network layout of $O_{discconn}$, is interpreted as an object property assertion in $O_{conn}$. The latter is more desirable, since it respect the explicit declaration of `:knows` as an object property made in each TBox ontology.

This counterexample proves that simply appending each ontology as subtrees to a root node will not work, because there are common ontology management applications which visit ontology networks (in this case built on OWL import declarations) in a way that can cause object property assertions to be interpreted as annotations, and to be potentially ignored by reasoners.

Further tests, here omitted for simplicity, have also invalidated the null hypothesis, and hinted that the manipulation of ontology network structures could be one way to counter the problem of erroneous OWL interpretation.

## 3.6 Restrictions

In order to draw reasonable conclusions from it, the contributions as per this dissertation were delivered under a set of restrictions. These restrictions limit the scope of our work, but also set potential future objectives established by relaxing some of them. They are:

**R1.** The time-efficiency of managing ontology networks using our method was not considered, so long as ontologies whose order of magnitude is up to $10^3$ axioms can be provided in realtime on standard desktop hardware.

[1]This ontology network example is available online at the intended URL `http://www.ontologydesignpatterns.org/ont/test/ontonet/imports-connected/root.owl`

**R2.** We will refer to software engines able to parse ontology documents and interpret them in OWL by visiting the import graphs in pre-order from a single start node, and performing interpretations greedily and on the first visit only.

**R3.** It is never the case that *all* the applications accessing the shared knowledge base need to construct ontology networks that are *completely* disjoint from one another at once.

**R4.** Multitenancy [CCW06] and security features, such as user account management, account-based access restrictions and encrypted data transfer, were not considered as locality factors for ontology networks.

**R5.** Ontology modularization [TDL+11, dSSS09, OY12, SCG12] was not contemplated at this stage of the work. We originally aimed at preserving the state of physical mappings of an ontology references to a set of axioms, with modularization support scheduled for future work.

**R6.** Following R5, we will limit to contexts where it is always possible to determine the minimal subset of the shared knowledge base that each application needs in order to obtain expected results in terms of reasoning and other procedures that process ontologies.

## 3.7 Summary of the contributions

The result of this research work is articulated into several products and byproducts. These include a base theoretical framework, logical and architectural specifications, open-source software, and even hints at a novel analytic approach to ontology network analysis. We hope these can serve as the drivers for future developments in ontology networking and aid the blending of so called "ontology-oriented" and "Linked-Data-oriented" mindsets.

In summary, this thesis is the result of a 3-year work that delivered the following:

**C1.** a dissertation on the state of play of ontology management that tries to harmonize the standpoints of Web ontology science and Linked Data technology;

**C2.** an overview of use cases, and classes thereof, where the need for processing dynamically generated ontology networks is felt;

**C3.** the formulation of a theoretical framework for describing ontology networks; the artifacts, concepts and relations that connect them with the logical theory behind traditional ontologies;

**C4.** a method, called *3-tier multiplexing*, for combining ontologies of heterogeneous provenance and logical profiles into ontology networks that do not exist in their original form;

**C5.** the specification of a software architecture that maps the artifacts of the 3-tier multiplexing method with the artifacts of an extensible service framework for serving ontologies;

**C6.** a Web Service interface specification for Web clients to interact with dynamically generated ontology networks;

**C7.** *Stanbol ONM*, an implementation of the above specifications, as part of an open source project maintained by the Apache Software Foundation;

**C8.** an evaluation method for ontology network interpretation grounded on graph theory;

**C9.** an insight as to how the above method, and by extension its implementation in Apache, could be automated or improved, and the possible directions to that end.

It is of particular note that contributions C3 through C7 are intended to fulfill the intermediate objectives O1 through O5 respectively, in the given order (cf Section 3.3).

## 3.8 Public problem statement

Although my first public statement on tackling ontology network architectures as a doctoral thesis work was made at the ESWC12 conference [Ada12], my active involvement with this line of research began in 2009, one year prior to enrolling as a Ph.D. student. It is worth mentioning that a similar problem statement was made on the same occasion [Roh12], which focuses on the sub-problem of formally specifying ontology network. A future integration of the work described in Section 4.1 with the one mentioned above, once its ultimate results are disclosed, is being kept in thorough consideration.

# 4

# A model for ontology network construction

Ontology networks are a relatively recent concept [PVSFGP10, GPSF09], therefore their theoretical background is still expanding and has not fully dealt with the notions, methods and caveats associated with the practical use of ontology networks. In Section 2.3.1, an overview on the few existing approaches in that direction was presented. It was noted then, that those approaches were mainly a product of nontrivial analysis work performed on existing distributed ontologies, with a degree of variety in defining the relationships that establish connectivity between ontologies . Additionally, being these approaches either conceptual, as in DOOR [AdM09], or purely DL-based, as in the extension of DOOR relations by Díaz et al. [DMR11], they assume to be operating with artifacts that are natively ontological, i.e. whose axiomatization was deterministic. In the reality of the Semantic Web, however, we have to come to terms with the facts that semantic information sources are heterogeneous; that their data need to be combined and interpreted before they can be treated as an ontology network; and that the transition between these two states is non straightforward.

Having outlined our research problem, its related challenges and technological landscape, we shall now proceed to describe the architectural perspective of our proposed solution to the dynamic assembly of ontology networks. Two models are required for exhaustively describing our approach, i.e. the *ontology network model* and the *architectural framework*. The former describes the connected structure that is enforced upon a given set of ontologies that need to be assembled together. The latter draws a cor-

respondence between components of the software framework at hand and elements of the ontology network model, effectively depending on the former. These two models denote each an architecture by its own right and are described in the given order in this chapter and the next.

The theoretical framework described in this chapter does not oppose to those described in Section 2.3.1 in general, nor is it aimed at extending them. Rather, one of its aims is to *accommodate* formal specifications from bottom-up approaches such as those described earlier and then some. Then, the specific relations from these other approaches could be described in terms of this more general framework. We will provide an insight as to how multiple ontologies can be connected; the meaning of "interpreting" a resource as an ontology, and why this distinction is to be made; the meaning of "giving a name" to an ontology and what its potential implications are. Further in the chapter, we apply this theoretical framework to a method, compounded with algorithmic support, for improving the expressivity of ontology networks once their resources are interpreted as such. No specific focus exists on one given underlying logic, but working in a decidable logic, such as OWL-DL or tractable fragments of OWL 2, is a reasonable assumption.

On a historical note, this model, and by extension this chapter describing it as well as the whole doctoral work, are a consequence of, and justified by, the hardships encountered – some unexpectedly – while developing an application whose design did not involve any interpretation of the ontology network model. Had it been possible to deploy the application straight ahead, it would not be the subject of this work presently. In a process closer to agile software development methodologies than to a plain bottom-up approach, the application under development (the description of which is deferred to Chapter 6) ended up incorporating the solutions to ontology modeling problems encountered along the way.

This chapter describes the ontology network model as follows. Sections 4.1.1 and 4.1.2 are devoted to porting non-novel concepts, such as ontologies and ontology networks, into our model. To that end, these concepts will be given definitions compatible with the theoretical framework being established, and backed up by additional definitions of preliminary concepts. Section 4.1.3 defines the newer concepts and artifacts that we introduce in the model, and in section 4.2 these concepts are *instantiated*, that is to say, a method is described for combining them so that ontology networks can be

generated dynamically. The remainder of this chapter will be devoted to describing how the ontology network model can relate to, and exploit the features of, OWL and ontology languages in general. Section 4.2.5 describes how ontology networks so generated can be rendered in the OWL 2 language. Section 4.3 explains how the ambiguity between logical names and physical references of ontologies can be tamed. Finally in Section 4.4, we explore related work in this specific area that was not exhaustively covered by the chapter on the general state of the art for the sake of uniform narration. We then single out its relationships and potential integration points with our work.

## 4.1 The ontology network model

Cognitive, logical and information studies on ontologies, as well as their technological applications, apparently deliver discordant definitions of ontologies from different viewpoints. Consequently some practitioners, such as Semantic Web technology connoisseurs, think of them as something different from Linked Data, or "more complex" than RDF graphs. Discrepancies range from believing that an ontology "must be made in OWL or RDFS and cannot be made in RDF", to considering OWL as "yet another vocabulary for RDF" to be used alongside vocabularies like Dublin Core or SKOS. In truth, an interpretation of Linked Data, RDF graphs and ontologies as the same thing would not be that far-fetched. There is, however, a degree of plausibility in this discord, due to the sparseness of application support which tends to favor one viewpoint at the expense of the others.

Earlier in this dissertation (cf. Section 2.2.4), we cited conceptual and linguistic definitions of ontologies, and mentioned some knowledge representation languages, such as OWL, that are apt to encode the description logics that ontologies are founded on. We will henceforth restrict to considering an ontology to be any artifact *already expressed* in such description-logical languages, namely OWL itself. With that cleared, we can then formally distinguish ontologies from other resources, such as the files, RDF graphs and Web service endpoints that encode their "raw" forms. Also, the *name* of an ontology can be distinguished from the filename, URL or database key of such a resource. These and other distinctions will be formally laid out in this section as our model of ontology networks is gradually uncovered.

### 4.1.1 Fundamentals

So far we have given or assumed informal, intuitive and officially accepted definitions of what an ontology is (cf. Section 2.2). This reflects the actual mutability of this notion and its interpretation in diverse contexts. In Linked Data parlance, for instance, there is a tendency to refer to a set of instance data as "dataset" and the underlying formal model as "ontology", DBPedia itself being a well-known example of such a distinction [BLK+09]. However, the term "ontology" applies to either of the above, as a well as combinations of them. In knowledge extraction disciplines, the umbrella term "ontology" can encompass any piece of structured information such as an XML tree or a relational database, or even unstructured information such as natural language text, so long as a formal knowledge model can be extracted from them. Here, we apply a simple formal restriction that encompasses many notions of ontology, provided that a logical representation of its atoms is possible.

**Definition (ontology).** An *ontology* $o$ is a triplet $o = (N, T, A)$, where $N$ is a set of *non-logical axioms*, $T$ is a set of *terminological* axioms and $A$ is a set of *assertional* axioms. All axioms in $T$ and $A$ can be represented in description logics.

We are therefore focusing on the relationship between description logics and ontology representation and maintaining the classic *TBox/ABox* dichotomy [Gru93b], here represented by the sets $T$ and $A$, respectively. Non-logical axioms are all those axioms that do not influence the model of the domain referred to by the ontology, and from which no other axioms can be inferred. They are, therefore, either annotations or statements that identify the ontology itself as a concept, plus any further annotations on that concept. No restrictions are set on the representation formalism for axioms, so long as it is able to map them to some description logics. Both the DL syntax and the functional syntax of OWL 2 [MPSP+09] will be regarded as reference formalisms for axioms.

**Definition (knowledge base).** Given $\mathcal{O}$ the set of all ontologies, a *knowledge base* $O$ is any subset of $\mathcal{O}$, i.e. $O \in \wp(\mathcal{O})$. Then $\mathcal{O}$ is also called the *universal knowledge base*.

#### 4.1.1.1 Ontology sources

It is a common mistake for ontology laypersons to consider ontologies to be, among other things, XML documents that follow a particular schema (XSD). In early 2008, two years before this research work began, I was submitted a set of semantically annotated Web Service manifests for review. These were XSD and WSDL 1.1 documents [CCMW01] supposedly annotated in accordance with the SAWSDL standard [FL07]. Although this standard recommends that XML elements in Web Service manifests reference entities described in some semantic model using full URIs or qualified names, the manifests at hand were referencing them using XPath predicates. XPath is a query language used for navigating XML trees [CD99], which meant it could not be used to retrieve knowledge about these entities from ontologies in non-XML formats (e.g. Turtle or Manchester OWL syntax) or with a different RDF/XML serialization than the one expected. This unorthodox annotation method was a consequence of identifying an ontology with the document it was serialized into. It is a limiting and misleading notion, but it called on us to try to keep this ambiguity under consideration.

The next definition can then be useful for disambiguating a knowledge model and its serialized form.

**Definition (ontology source).** An *ontology source* $s_o$ for an ontology $o$ is an object so that a resolution function $f$ exists, that can be applied to $s_o$ in order to produce $o$. It is then said that $f(s_o) = o$, or $s_o$ *resolves to* $o$, or $s_o$ *is a source for* $o$.

By interpreting the above definition practically, an ontology source can be of any commonly used type of artifact known for storing an ontology, i.e. a file, the output of a Web Service, or a named graph in a triple store. Other sources such as a relational database, an RSS feed or a text document, imply more complex resolution functions that involve data reengineering and natural language processing, yet can still be regarded as legal ontology sources. As for the resolution function $f$, we will not go into detail as to what such functions are, because in Section 3.4 we made the reasonable assumption that, for any object we deem interpretable as an ontology, there is the machinery for doing so. Thus the resolution function exists and that object is a valid ontology source. Intuitively, resolution functions can combine syntactic parsers for representation languages, rule execution and so on.

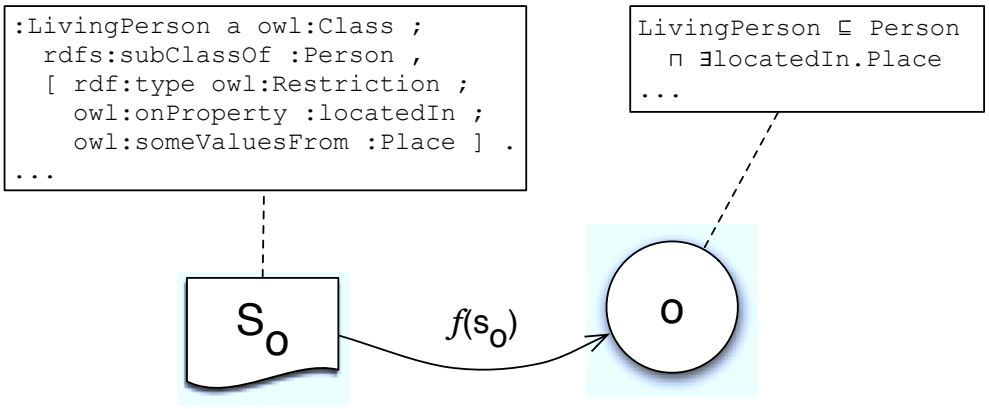


**Figure 4.1: Resolution of an ontology source to an ontology** - To the right, an ontology $o$ whose axiom set $T$ says that all living persons are persons and are located in some place. To the left, its source $s_o$ is an object that encodes one possible way to state this in Turtle syntax for RDF. A resolution function $f$, which will involve parsing Turtle syntax and interpreting its statements, can produce $o$ from $s_o$.

Note that the above definition does not rule out ontologies written in a native ontology language or in DL syntax, as such native code will have to be stored in some resource which will be the ontology source. To that end, an example is given in Figure 4.1.

**Example 1.** To the left of Figure 4.1 is an object $s_o$, which encodes a list of statements, namely triples of the RDF graph in the Turtle language. According to the triples displayed, `:LivingPerson`[1] is a class that is a subclass of `:Person` and also a subclass of the anonymous class of all the things `:locatedIn` some `:Place`. This is expressed with the aid of RDF resources using the OWL (`Class`, `Restriction`, etc.), RDF (`type`) and RDFS (`subClassOf`) vocabularies for serializing ontologies as triple collections. There is a function $f$ that, when applied to $s_o$, delivers an object $o$ that is an ontology. Its axioms state that every `LivingPerson` is a `Person` and something that is `locatedIn` a `Place` (or, more formally, "*so that a* `locatedIn`*-successor exists in* `Place`"). Therefore $s_o$ resolves to $o$ via $f$ and is a source for $o$. Note that $s_o$ is not the only possible ontology source for $o$: aside from possibly being in other representation languages, even the Turtle syntax can allow some degree of liberty. For example, if the

[1]In Turtle, the colon denotes a blank prefix for the entity that follows, i.e. its IRI belongs to a default namespace, which is here unspecified for the sake of simplicity.

specification of `:LivingPerson` as an `owl:Class` were missing, the result would still be a valid source for $o$. Likewise, we used native DL syntax for $o$ in order to make its nature as an abstract syntax clear, since this syntax is not very likely to be consumed by programs as it is. We could as well have used OWL directly and represented the ontology using the OWL functional syntax, like this:

```
SubClassOf(:LivingPerson :Person)
SubClassOf(:LivingPerson ObjectSomeValuesFrom(:locatedIn :Place))
...
```

For clarity, we shall henceforth assume that a necessary condition for an object to be an ontology source is that it be a *collection of statements*, no matter what language they are represented in, whether RDF triples or even sentences in natural language. These statements are then called the **raw statements** of the ontology source. This distinction is important, as the heart of the problem at hand is how raw statements become axioms in an ontology.

**Example 2.** Going back to our leading example introduced in Section 1.1.2.1, let us assume CMS user Alice is the one who wants to have her research paper directory classified according to the status of their authors in hheris social network. Suppose the semantic data and models Alice has for representing the social network are spread across three ontology sources $s_{o_1}$, $s_{o_2}$ and $s_{o_3}$ as follows (in Turtle syntax and with no OWL import statements):

```
:inNetworkOf a owl:SymmetricProperty .
```

```
:inExtendedNetworkOf a owl:TransitiveProperty ;
        rdfs:subPropertyOf :inNetworkOf .
```

```
:Alice a :Person ;
        :inExtendedNetworkOf :Bob .
:Clara a :Person ;
        :inExtendedNetworkOf :Bob .
```

The occurrences of properties `:inNetworkOf` and `:inExtendedNetworkOf`, both asserted and inferred, would be those used for populating the directories called `friends` and `friends-of-friends`, assuming the exclusion of the former from the latter to avoid repetitions.

If $s_{o_0}$ is the source of an ontology that imports $s_{o_1}$, $s_{o_2}$ and $s_{o_3}$, then a non-optimal yet common $f$ is a function that, once applied to $s_{o_0}$ resolves the above into the following $o_1$, $o_2$ and $o_3$, respectively (in OWL functional syntax):

```
Declaration(ObjectProperty(:inNetworkOf))
SymmetricObjectProperty(:inNetworkOf)
```

```
Declaration(ObjectProperty(:inExtendedNetworkOf))
TransitiveObjectProperty(:inExtendedNetworkOf)
SubObjectPropertyOf(:inExtendedNetworkOf :inNetworkOf)
```

```
ClassAssertion(:Person :Alice)
ClassAssertion(:Person :Clara)
AnnotationAssertion(:inExtendedNetworkOf :Alice :Bob)
AnnotationAssertion(:inExtendedNetworkOf :Clara :Bob)
```

Note that the usage of `:inExtendedNetworkOf` in annotation assertions, rather than object property assertions, is the anomaly that would prevent us from inferring that Clara is also in Alice's extended network, and therefore that Clara's papers should appear in Alice's `friends-of-friends` directory. An optimal $f$ would deliver the following $o_3$:

```
ClassAssertion(:Person :Alice)
ClassAssertion(:Person :Clara)
ObjectPropertyAssertion(:inExtendedNetworkOf :Alice :Bob)
ObjectPropertyAssertion(:inExtendedNetworkOf :Clara :Bob)
```

We have split the axioms for $o_1$, $o_2$ and $o_3$ into three frames in order to preserve their relationships with their corresponding sources. Note, however, that the axioms in OWL functional syntax represent the final state of the applied resolution function, and these axioms are carried over verbatim to any importing ontology. In the light of this

consideration, we will sometimes show the interpreted ontology network as enclosed within a single frame.

#### 4.1.1.2 Context-dependent OWL axioms and expressions

A resolution function, if looked on a finer granularity scale than ontologies, delivers a set of axioms (in the ontology) from an initial collection of raw statements (in the ontology source). There are a number of considerations to be made at this point:

**Bijectivity.** In general, there is no guarantee of a one-to-one mapping holding between raw statements in an ontology source and axioms in its ontology. For example, in the canonical RDF representation of OWL [PSMG+09] it can take up to four triples in order to be able to express a class. One quite complex case is the one of qualified cardinality class expressions, e.g. for the class of those having exactly two persons as parents, we could have the following RDF triples (in Turtle syntax):

```
_:x rdf:type owl:Restriction ;
  owl:qualifiedCardinality 2 ;
  owl:onProperty :hasParent ;
  owl:onClass :Person .
```

and obtain the following axiom from them (in OWL functional syntax):

```
ObjectExactCardinality( 2 :hasParent :Person )
```

Likewise, the presence of an axiom in an ontology could be drawn from context and not match any particular raw statement. For example, the declaration of `hasParent` as an object property:

```
Declaration( ObjectProperty :hasParent )
```

could be drawn from the same four RDF triples as above (where the property is used in conjunction with the class `Person`, which makes it an object property), or by an explicit triple declaring it, such as :

```
:hasParent rdf:type owl:ObjectProperty .
```

The latter is actually the normative form for parsing object property declarations [PSMG+09], however, several widespread OWL libraries and applications, such as the OWL API and Protégé 4, generate entity declarations automatically when serializing an ontology whose declarations were not made explicit in the source.

**Ordering.** We stated axioms in an ontology and raw statements in its source to be *collections*, simply because the ordering of raw statements might or might not be influential to the axioms delivered by the resolution function. This is a general consequence of the black box assumption made in Section 3.4.

**Ambiguity.** Unambiguous axioms in an ontology can result from interpreting ambiguous raw statements in its source. For instance, property declarations in OWL 2 explicitly state what kind of property is the one being declared. If all that an ontology source has to say about two properties `hasFather` and `hasParent` is in the triple:

```
:hasFather rdfs:subPropertyOf :hasParent .
```

then there is no explicit clue as to whether these properties are object, data or annotation properties. Therefore, the interpretation of these properties is strategy-dependent: a conservative strategy, such as the one we have observed in our preliminary experiments (null hypothesis negation included), will materialize annotation property declarations in the absence of further statements corroborating their nature.

The considerations made so far are an essential part of our research problem. Since there is no guarantee that a certain triple will lead to a certain axiom being produced by a certain resolution function, and having observed that it is indeed not the case for certain types of axioms, we surmise that the resolution of ontology sources into ontologies is, in general, a *context-dependent*, process. As far as RDF, by far the most common representation formalism for OWL, is concerned, phenomena of context-dependency occur beyond the normative conditions established by the W3C recommendation for RDF serialization [PSMG+09]. However, when an ontology is processed by applications such as DL reasoners, there are resolutions of raw statements into axioms which are more desirable than others. In principle, if a property *can* be interpreted as an object property (OP) in a certain context, then it *should*. Otherwise, if it is interpreted as

| AP-order | DP-order | OP-order |
|---|---|---|
| Declaration( AnnotationProperty ) | Declaration( DataProperty ) | Declaration( ObjectProperty ) |
| – | DataMinCardinality | ObjectMinCardinality |
| – | DataMaxCardinality | ObjectMaxCardinality |
| – | DataExactCardinality | ObjectExactCardinality |
| AnnotationAssertion | DataPropertyAssertion | ObjectPropertyAssertion |
| – | NegativeDataPropertyAssertion | NegativeObjectPropertyAssertion |

**Table 4.1: Context-dependent axioms and expressions with ambiguous RDF representations** - Declarations are considered ambiguous when there is *no* corresponding RDF triple. All cardinality expressions are intended to be non-qualified, since qualified cardinalities are unambiguously expressed in their RDF representation.

an annotation property (AP) or a data property (DP), certain characteristics of that property and its usage are lost and reasoners will not consider them.

For convenience, we have therefore organized some OWL axiom and expression types, or more simply, constructs, into three categories. This distinction is shown in Table 4.1. All the OWL constructs considered share the following characteristics:

1. they comprehend exactly one property;
2. they are explicitly derived from an object property, or a data property, or an annotation property;
3. their determination from a given set of raw statements is generally ambiguous.

The categories have been called *AP-order*, *DP-order* and *OP-order*, depending on whether the one property they use is an annotation property, or a data property, or an object property, respectively. If we consider an ontology as a family of three axiom sets $(N, T, A)$ as in our prior definition, then we have the following: any AP-order axioms are included in $N$; declarations and cardinality expressions, for both data and object properties, are included in $T$; data and object property assertions, both in DP-order and in OP-order, are included in $A$.

Because annotations are generally ignored by DL reasoners, being non-logical axioms, the amount of raw statements interpreted as annotations should be only limited to those intended to be so[1]. Similarly, OP-order constructs should not be mistaken for

[1]Recall from the OWL structural specification [MPSP+09], that annotations in OWL 2 allow IRIs as their values, and yet they still do not participate in reasoning procedures.

their DP-order equivalents, which is also possible if the object or the original raw statement is interpreted as a value in a legal datatype (such as an `xsd:anyURI`) instead of an OWL individual. For this reason, we state that DP-order OWL constructs are *higher-order* constructs than AP-order ones, and that OP-order constructs are higher-order constructs than DP-order and AP-order ones.

### 4.1.2 Ontology networks

In literature, an *ontology network*, or *network of ontologies*, is "*a collection of ontologies related together via a variety of relationships, such as alignment, modularization, version and dependency*" [SFGPMG12a]. Let us now attempt to give a formal equivalent of this intuitive definition. For the following, the term *signature* of an axiom (resp. ontology) will be used to identify the set of all the entities referenced by that axiom (resp. all the axioms in the ontology), i.e. not including literals. This is an informal yet commonly accepted notation [MPSP+09] derived from the definitions of signature in ontologies [BCM99] and in description logics [BHS08].

**Definition (direct dependency).** Let $\mathcal{O}$ be the set of all ontologies, or *universal knowledge base*. Then a *direct dependency* is a relation $d : \mathcal{O} \to \mathcal{O}$ so that $d(o_i, o_j)$ (to be read as "$o_i$ directly depends on $o_j$") is true iff, given $o_i = (N, T, A)$, $\exists a \in N \cup T \cup A$ so that the signature of $a$ includes either $o_j$ or $s_{o_j}$.

The above definition states that an ontology directly depends on another ontology if it references that ontology in one of its axioms, even (and especially) non-logical ones. The meaning of this definition is as simple as it is strong. Since the universe of ontologies $\mathcal{O}$ is assumed to be known in the definition, we can always tell if the concept referenced by an axiom is an ontology, an ontology source, or neither. In other words, there is a *global knowledge* of which identifiers reference ontologies and which do not, but that does not imply *local knowledge* within one ontology of $\mathcal{O}$. The property expression used as the axiom predicate alone does not necessarily imply its subjects or objects being ontologies, although this may be true in some cases, such as for `owl:imports`.

**Example.** Suppose an ontology has the following axioms (in OWL functional syntax):

```
ClassAssertion( owl:Thing :in1 )
AnnotationAssertion(
  rdfs:isDefinedBy :in1 <http://example.com/resource/res1> )
```

These axioms state that `:in1` is a named individual (which follows from it being an `owl:Thing`), and that any further information that defines `:in1` is somehow addressed by `<http://example.com/resource/res1>`, which incidentally is an IRI. If that IRI is the location of an ontology source or the identifier of an ontology, then we can say that a direct dependency relation holds. Otherwise, if `<http://example.com/resource/res1>` resolves to an image or a document in PDF that describes it, and we are not accepting that resource as an ontology, then no direct dependency is implied.

Whereas, if the ontology has an import declaration (e.g. the result of interpreting an `owl:imports` statement found in its source RDF graph) such as:

```
Ontology( <http://www.example.com/resource/ontology/1>
   Import( <http://www.example.com/resource/ontology/2> )
)
```

then this ontology, whose *name* is `<http://www.example.com/resource/ontology/1>`, depends on the ontology whose *source* is identified by the IRI in the `Import` declaration `<http://www.example.com/resource/ontology/2>`. Recall [MPSP+09] that OWL 2 uses an import-by-location scheme, so the object of the `Import` declaration, i.e. the import target, is expected to be a dereferenceable URL that resolves into a resource that can be interpreted as an ontology, otherwise an anomaly occurs.

Other vocabularies can define (potential) direct dependency relationships if so we choose: for example, `page` in FOAF or `source` in Dublin Core may or may not be regarded as predicates for direct dependency relations whenever their objects are ontologies or ontology sources.

Direct dependency is not transitive. For completeness, a recursive definition of its transitive extension is provided.

**Definition (dependency).** A *dependency* is a relation $d^* : \mathcal{O} \to \mathcal{O}$ so that $d^*(o_i, o_j)$ (to be read as "$o_i$ depends on $o_j$") iff:

1. either $d(o_i, o_j)$ or
2. $\exists k, k \neq i, j : d(o_i, o_k)$ and $d^*(o_k, o_j)$.

We shall now define a weaker kind of dependency, called *connectivity*.

**Definition (connectivity).** A *connectivity relation* is a relation $c : \mathcal{O} \to \mathcal{O}$ so that $c(o_i, o_j)$ (to be read as "$o_i$ is connected to $o_j$") iff:

1. either $d(o_i, o_j)$ or
2. $o_i = (N, T, A)$ $\exists a \in N \cup T \cup A$ so that the signature of $a$ includes an *element* of $o_j$, i.e. an entity contained in the signature of $o_j$.

Connectivity is weaker than direct dependency, insofar as in $\mathcal{O}$ there are more connected ontologies than there are ontologies that directly depend on one another, since by condition 1 of the definition follows that the set of the former includes that of the latter. Also note that connectivity is not an equivalence relation, since it is not transitive and condition 1 denotes a non-symmetric (but not antisymmetric) relation. On the contrary, the relation denoted by condition 2 is symmetric, because the usage of one concept in two ontologies implies its presence in both of their signatures, and there is no such notion as "using in ontology A a concept defined in ontology B", since the definition of a concept lies in its usage. We are not ruling out the possibility of an equivalence relation for connectivity, but it is not necessary for the ontology network definition that follows.

**Definition (ontology network).** Given a *knowledge base* $O = \{o_i\}$, an *ontology network* for $O$ is any *maximal* subset $O^N$ of $O$ where:

1. $\forall o_i \in O^N$ $\exists o_j \in O^N$, $i \neq j$ so that:
   (a) either $c(o_i, o_j)$ or $c(o_j, o_i)$ is true;
   (b) if both $c(o_i, o_j)$ and $c(o_j, o_i)$, then $\exists k, i \neq k \neq j$ so that $\{o_i, o_k, o_j\}$ is an ontology network for itself.

This definition states that mutual connectivity between two ontologies in an ontology network is possible, so long as it is not the only existing connectivity relationship involving those ontologies.

The maximality requirement in the above definition means that if any ontology in $O$ that is not connected (i.e. does not fulfill the connectivity conditions) is added to $O^N$, then $O^N$ is not an ontology network anymore. In other words, we are implying that ontology networks cannot have islands. Also note that there is no minimality requirement, implying that a so-defined ontology network tolerates broken links. This is necessary in order to conform to the open world assumption (OWA) [RN10] made in OWL [HPPSR09]. Suppose two ontologies $o_1$ and $o_2$ in $O$ are strictly weakly connected, i.e. $o_1$ is connected to, but not dependent on, $o_2$. Then it means these two ontologies are using some entity $e$ in common. If $o_1$ belongs to an ontology network that does not include $o_2$, this simply means that the axioms in $o_2$ concerning $e$ will not be part of the ontology network, but will not invalidate the ontology network either, otherwise the OWA would be violated.

Detecting ontology networks in a set of ontologies means picking as many mutually connected ontologies as possible from the set, without picking multiple connected components. All this is performed by taking the ontologies *as they are*, with all the logical and non-logical axioms they come with. In other words, they are "real" ontology networks, whose connectivity is established by the agents that created and published the ontologies. We refer to these ontology networks as "real" in order to distinguish them from those created by manipulating some axioms of these ontologies and not necessarily re-publishing them in this form. We will call those "virtual" ontology networks.

**Definition (virtual ontology network).** Given a set of ontologies $O = \{o_i\}$, a *virtual ontology network* for $O$ is a set of ontologies $O^{\mathbf{V}}$so that:

1. $O^{\mathbf{V}}$ is an ontology network for its own ontologies;
2. $\forall o = (N, T, A) \in O$:
   (a) $\exists o^{\mathbf{V}} = (N', T', A') \in O^{\mathbf{V}}$
   (b) for any axiom in $N'$ that references an ontology, that ontology is in $O^{\mathbf{V}}$;
3. $\exists o \in O : o \notin O^{\mathbf{V}}$

A way to read this definition informally could be that "a virtual ontology network can be constructed by taking a set of ontologies and adding or replacing some axioms that connect these ontologies". Per (2), there is an ontology in the virtual network for each ontology in the initial set $O$ (a), and the virtual network references only ontologies within itself (b). Per (3), if the original set of ontologies is already an ontology network, a virtual ontology network cannot *match* it, since they have to differ by at least one ontology. However, a virtual ontology network can add axioms to a real one and preserve its nature as an ontology network.

By the above definition, one can surmise that a virtual ontology network can be constructed on top of a set of ontologies by adding axioms (both logical and non-logical) that reference ontologies in the set, or their entities. This is inexact, as one of our goals is to demonstrate that ontologies in a virtual ontology network can contain higher-order axioms that those in the ontologies the network is built upon. More lax forms of ontology network, albeit admissible, imply modularization and other refactoring operations beyond the scope of this work (cf. R5, section 3.6).

**Corollary.** For any virtual ontology network $O^{\mathbf{V}}$ for $O$, $|O^{\mathbf{V}}| \geq |O|$.

*Proof.* Omitted. Follows from (2a) in the definition of virtual ontology network. □

Having now provided a basic theoretical framework for ontologies and ontology networks, we shall now proceed to illustrate how our contribution is built on top of the described framework, and to synthesize a method for constructing virtual ontology networks given an arbitrary set of ontologies.

### 4.1.3 Artifacts

This section introduces the novel classes of ontological artifacts that will be used as nodes of the ontology network structures to be built using the method we are introducing. There is no question that the core, and probably most interesting, artifacts introduced herein are those that allow ontologies to be part of a network. However, there are additional supporting artifacts and notions employed in combination with them, which can be interesting to note due to their connection with standards such as features of OWL 2.

#### 4.1.3.1 Referencing ontologies

First and foremost, let us introduce the *ontology referencing* mechanism of our method, which is the way a mnemonic identifier can be mapped to an actual ontology. To that end, a definition shall be used, which is a light abstraction of the *naming* mechanism of OWL 2 ontologies [MPSP+09], except that the uniqueness of the resulting references can have a scope more restricted than the whole Web, if required.

**Definition (ontology reference).** A *reference* $\$_o$ for an ontology $o$ is a pair $\$_o = (i, v)$, where $i$ and $v$ are two objects respectively called *ontology identifier* and *version identifier*. Each ontology reference is unique within a knowledge base, that is, given a knowledge base $O = \{o_j, j = 1..n\}$, then for any $k, l \in \{1..n\}$ $\$_{o_k} = \$_{o_l}$ iff $k = l$.

The relationship between this definition and the IRI pair $(ontologyIRI, versionIRI)$ that identifies ontologies in OWL 2 [MPSP+09] is evident. At this stage, however, there is no enforcement over ontology references matching their logical identifiers or even being Web resource identifiers such as URIs or IRIs. This is the reason why ontology and version identifiers are willfully underspecified as being "objects". These restrictions will be introduced when discussing the relationship between ontology identifiers and their references (cf. Section 4.3).

It can be deducted from the definition above, that ontology references and the integers used earlier for indexing members of a knowledge base are interchangeable, i.e. a bijective function $f : O \to \mathbb{N}$ exists for any $O$.

#### 4.1.3.2 Ontology collectors

So far, we have sketched the theoretical framework behind the established domain of Web ontologies, and adapted its notation to the scope of the present work. Let us now proceed to lay out the definitions of the new components for an ontology network architecture as described here. The following will provide a definition of the abstract notion of *ontology collector*, which is not utilized in its primitive form for constructing ontology networks, but has instead to be specialized further for different network components.

An **ontology collector** is the primary artifact that keeps track of references to those ontologies which were not originally created as part of an ontology network, or at

least not a subnetwork of the (explicit) virtual network that is being assembled. The name is to signify that this type of artifact is not a *collection* of, or a *container* for ontologies, assuming that collecting or containing ontologies means referencing their axioms, and not the ontologies as atomic objects.

**Definition (ontology collector).** An *ontology collector* $C$ in a knowledge base $O$ is a pair $C = (I, R)$, where:

1. for no $o \in O$ there is $x$ so that $I||x = \$o$, where '||' is a concatenation operator that applies to members of the domains of $I$ and $\$_o$.

2. $R = \{\$_o\}$ for some $o \in O$.

$I$ is called the *identifier* of the ontology collector, and per (1), it is *never* the prefix of any reference $\$_o$ for any ontology in $O$. $R$ is called the *reference set* of the ontology collector. If $\$_o \in R,\ o \in O$, then $o$ is said to be *managed by* $C$ *through* $\$_o$, or simply *managed by* $C$. Conversely, $C$ *manages* $o$. Because $I$ has to be comparable to ontology references, as it must share their concatenation operator, it will sometimes be expanded as $(i_C, *)$ or $(i_C, nil)$, indicating that any version identifier is either disregarded or explicitly null for ontology collectors.

#### 4.1.3.3 Imaging

An ideal assumption concerning the existence of ontologies on the Web is that each logical identifier, comprehensive of a version indicator, identifies a single ontology, intended as a set of logical axioms. While the Web is far from realizing this desideratum, the assumption becomes far more reasonable when the knowledge base at hand is a controlled environment. As the present work is dealing with an ontology serving framework backed by a controlled knowledge base, even supposing its contents are selectively imported from the Web, we can shift the problem towards ensuring the uniqueness of identifiers for the provided ontologies with respect to the outside world.

The above requirement hints at a possible distinction which can be summarized as "*global uniqueness* versus *local uniqueness*"[1]; the former applying to the entire unbounded Web, the latter being restricted to the knowledge base directly accessed by the

[1] This terminology is lifted from differential equation theory [Den95]

framework. In global uniqueness, any $\$_o$ is a reference for $o$ in the universal knowledge base $\mathcal{O}$, whereas local uniqueness holds within a knowledge base $O$ but not necessarily in $\mathcal{O}$. The uniqueness problem can then be formulated as follows: *given a knowledge base where local uniqueness is guaranteed, expose its ontology to the Web so as not to impair global uniqueness.*

Guaranteeing local uniqueness within $O$ means ensuring that ontology references can be used for unambiguously addressing its ontologies. If $O$ is an ontology network, local uniqueness comes for free, but if it is *not*, then local uniqueness should be guaranteed at least for a virtual ontology network $O^V$. A strategy for doing so is by manipulating the sources of the ontologies in O, thereby obtaining other ontologies. This operation is what we call *imaging*.

**Definition (ontology image).** Given an ontology $o = (N, T, A)$ and an ontology collector $C = ((i_c, *), R)$ so that $\$_o \in R$, then the *image* of $o$ with respect to $C$ (or $C$-*image* of $o$) is an ontology $o^C = (N^C, T^C, A^C)$ so that:

1. $N^c$ always contains exactly one ontology naming axiom and one ontology versioning axiom;
2. $\$_o^C = (i', v')$, where $i_C$ is a prefix for either $i'$ or $v'$ (i.e. $\exists w.(i' = i_C||w \vee v' = i_C||w)$ where '||' is a concatenation operator), is a valid reference for $o$;
3. if $f$ is a resolution function so that $f(s_o) = o$, then also $f(s_{o^C}) = o^C$.

From the standpoint of ontology collectors, the image of an ontology is the outcome of the manipulation that an ontology collector performs on the ontology it manages. The essential meaning of this definition, especially in (3), is that the raw statements in the source of $o$ can be manipulated so that the *same* resolution function, once applied to the modified source, still produces an ontology, and that ontology has *at least* characteristics (1) and (2).

With the definition of ontology image in place, it is now possible to reformulate our objective O2 ("to define a method for constructing ontology networks dynamically", cf. Section 3.3) in terms of the theoretical framework so far described.

In order to fulfill O2, given an arbitrary knowledge base $O$, we want to obtain a virtual ontology network $O^{\mathbf{V}}$ for $O$, so that $\forall o^{\mathbf{V}} \in O^{\mathbf{V}}$:

- $o^{\mathbf{V}}$ is the image of an ontology $o \in O$ with respect to some ontology collector, i.e. there is $C$ such that $o^{\mathbf{V}} = o^C$;
- if $o^{\mathbf{V}} = (N^{\mathbf{V}}, T^{\mathbf{V}}, A^{\mathbf{V}})$ and $o = (N, T, A)$, then every axiom in $T^{\mathbf{V}}$ and $A^{\mathbf{V}}$ is of order greater than, or equal to, every axiom in $T$ and $A$ (see Section 4.1.1.2 for the definition of higher-order axioms).

Note that the above conditions do not imply that a *single* ontology collector should be responsible for imaging the entire knowledge base into a virtual ontology network, but only that every ontology in the virtual network must be managed by at least one collector. Also, recall from the corollary to the definition of virtual ontology network, that there can be more ontologies in a virtual ontology network than in its original set of ontologies. Fixing these two considerations is a *clé de vôute* for understanding the means to achieve the aforementioned objective, which the remainder of this chapter will be devoted to.

Recall from Section 4.1.2 that an ontology directly depends on another if it has at least one axiom of any type, which references the other ontology or its source. The transitive extension of direct dependency is simply called *dependency*. We will call the first ontology *dependent*, the other ontology *[direct] dependency*, and the axiom that references the dependency a *[direct] dependency axiom*.

Prior to introducing specific ontology collectors, we need to assume the following statements.

**Hypothesis A.** All ontology collectors are ontology sources.

This hypothesis states that for every ontology collector $C$ there is a resolution function $f$ that associates $C$ with an ontology. Let that ontology be called the *ontological form of* $C$ (by $f$).

**Hypothesis B.** Under Hypothesis A, the following holds for any ontology collector $C$: if $C$ is managing an ontology $o$, then the ontological form of $C$ has a dependency on $o$.

These hypotheses will not be verified at this stage, however, formulating them was necessary in order to justify the definition that follows. When tackling ontology

collectors from the OWL language standpoint, a resolution function for interpreting an ontology collector itself as an ontology will be defined, thus the hypotheses verified.

#### 4.1.3.4 Ontology space

Let us now specialize the definition of ontology collector into items for immediate use in our strategy. **Ontology spaces** are the first class of ontology collectors so specialized.

**Definition (ontology space).** Given a knowledge base $O$, an *ontology space* $s = (I, R)$ is an ontology collector such that every ontology $o$, where $\$_o \in R$, has an $s$-image $o^s, s^s \notin O$ whose dependency axioms reference either ontologies in $O$ or other ontology spaces.

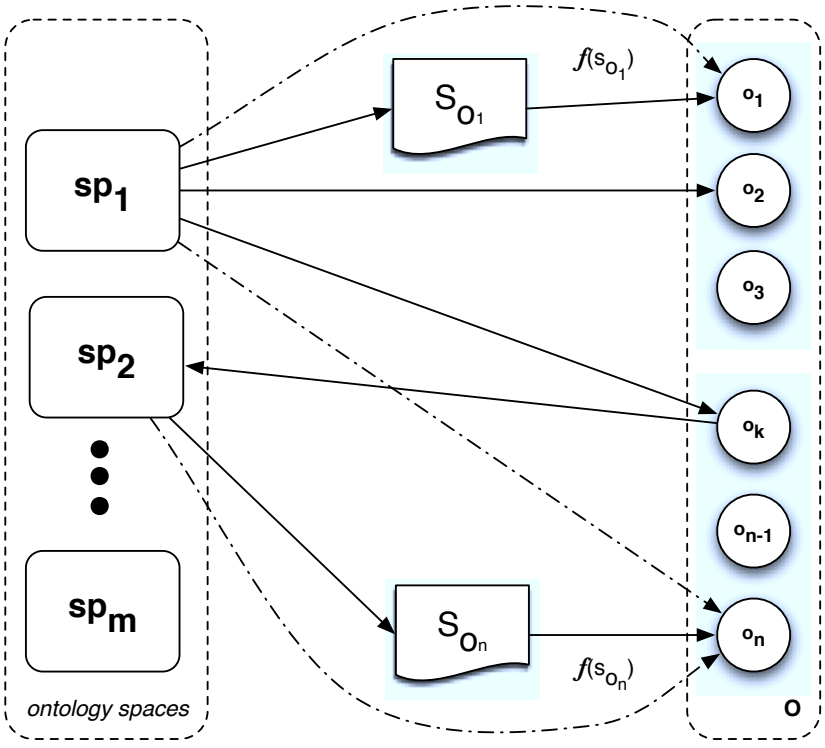


**Figure 4.2: Scenario with multiple referencing mechanisms in ontology spaces** - Ontology space $sp_1$ manages $o_2$ and $o_k$ directly, $o_1$ indirectly via its source $s_{o_1}$, and $o_n$ via ontology space $sp_2$. Ontology spaces and the ontologies they resolve to are collapsed for convenience. Indirect references are shown as dashed and dotted arrows.

Figure 4.2 provides an example that summarizes possible scenarios for an ontology space to reference ontologies. $sp_1$ is an ontology space that depends on $o_1$ because its ontological form (which is omitted in the figure for simplicity) has a dependency on its source $s_{o_1}$. $sp_1$ also manages two ontologies $o_2$ and $o_k$ directly, which means its reference set includes at least one reference for each ontology. It also means that $sp_1$ can create images of these ontologies, and these only. However, $o_k$ also has a dependency on the ontological form of another space $sp_2$, and since this second ontology space is managing another ontology $o_n$, then $sp_1$ has a dependency on $o_n$ too. This dependency is indirect, which means that if $sp_2$ were to sever its link with $o_{n-1}$, the dependency would no longer hold nor need to be satisfied through the existence of $o_{n-1}$. Finally, $o_3$ and $o_{n-1}$ are unmanaged and they are not dependencies of any of the aforementioned ontologies, so they belong to the common knowledge base $O$ but are not part of any ontology network.

The usage of dependency axioms in ontology spaces places a criterion on the selection of ontologies managed by these particular ontology collectors, which allows us to state the following

**Theorem 1.** If an ontology space manages at least one ontology, then its ontological form is part of an ontology network.

*Proof.* Under Hypothesis A, an ontology space $sp$ is an ontology source, therefore it has an ontological form, namely $f(sp)$ for some $f$. By induction on the number of ontologies managed by $sp$:

1 If only one ontology $o$ is being managed by $sp$, then under Hypothesis B we have a connectivity relation $\rho = d^*(f(sp), o)$. Then, by induction on the network structure of $o$:

- If $o$ is a singleton, i.e. it has no connectivity relations other than the dependency of Hypothesis B, then $\{f(sp), o\}$ is an ontology network.
- If $o$ is already part of an ontology network $O$, then $O \cup \{f(sp)\}$ is an ontology network, since $\rho$ connects $f(sp)$ to an element of $O$.

n+1 The induction hypothesis is that $n$ ontologies managed by $sp$ form an ontology network $O_a$ together with $f(sp)$. Let $o_{n+1}$ be the n+1-th ontology managed by $sp$, $o_{n+1} \notin O_a$. Per case 1, $o_{n+1}$ forms an ontology network with $f(sp)$,

let it be $O_b$. From the definitions of connectivity and ontology network follows that the union of two ontology networks is an ontology network iff there is one connectivity relation holding between two ontologies one from each network. Let us take $f(sp) \in O_a, O_b$. By induction hypothesis, $f(sp)$ is connected to some ontology in $O_a$, and since it is also connected to $o_{n+1} \in O_b$, the thesis follows.

□

#### 4.1.3.5 Scope

Let us now define a way to group ontology spaces, as it will turn up to be convenient when our method is described. Since the aim is to preserve the nature of ontology networks, we cannot simply define this grouping in terms of standard set algebra, lest we risk creating islands. We shall therefore use another specialized ontology collector.

**Definition (scope).** A *scope* for a knowledge base $O$, $S = (I, R)$ is an ontology collector such that:

1. $\forall \$_o \in R$, there exists an ontology space $s$ that references $o$;
2. there is one possible selection of ontology spaces as above (called the *spaces of* $S$), so that all of the following hold:
   (a) for any space $s$ of $S$, $s = (I_s, R_s), R_s \subseteq R$;
   (b) for any two ontology spaces $s_j = (I_{s_j}, R_{s_j}), s_k = (I_{s_k}, R_{s_k})$ in the selection, $R_{s_j} \cap R_{s_k} = \emptyset$;
   (c) there is a partial ordering of the spaces of $S$, be it $s_1, s_2, \ldots s_n$, so that if any ontology managed by $s_i$ depends on some space, then that space is one of $\{s_{i-1}, s_{i-2}, \ldots\}$.
3. if $o^S$ is the $S$-image of $o, o \notin O$ and $o^s$ is its $s$-image, $s$ being a space of $S$, then $o^S$ differs $o^s$ by non-logical axioms only;

From the definition of scope as given above, it emerges that scopes are quite peculiar forms of ontology collectors, in that all of their ontology references are *inherited* from other ontology collectors, namely spaces. Per condition 1 of the definition, there are no ontologies exclusively referenced by a scope. Condition 2 further consolidates the

function of scopes as aggregators of other ontology collectors: a scope contains all the references of each of its spaces, which in turn share no references with one another. However, when space reference one another recursively, they follow an ordering in doing so. Finally, per condition 3 are expected to behave so as to not influence the resolution of the sources managed by their spaces.

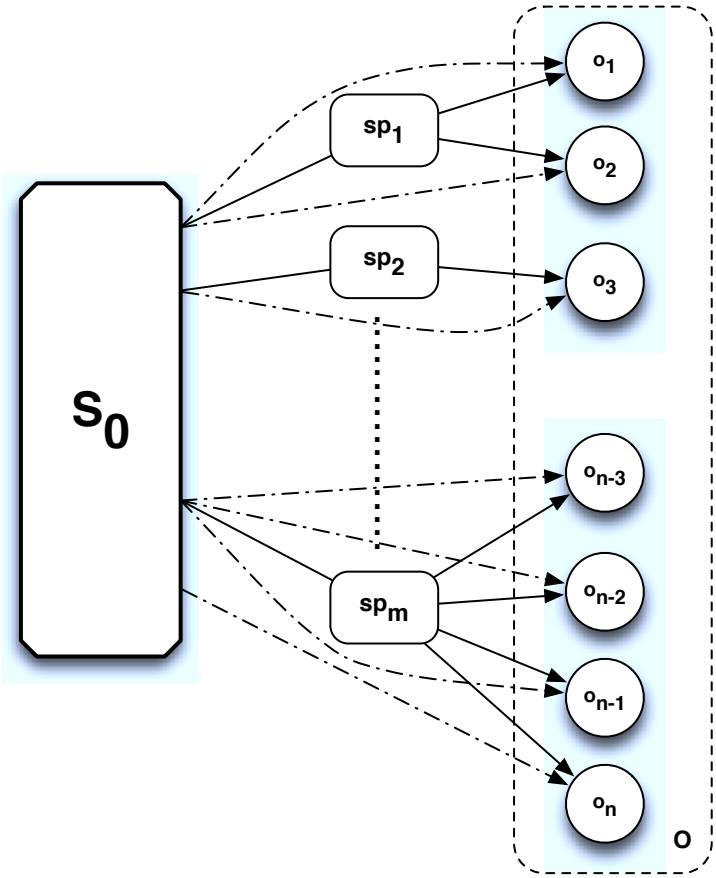


**Figure 4.3: Ontology referencing mechanism for scopes** - Scope $S_0$ references $n$ ontologies in the knowledge base $O$, but does so indirectly via the $m$ ontology spaces that belong to it. Indirect references are shown as dashed and dotted arrows from $S_0$ to its ontologies.

An example of the way ontologies are managed and/or referenced by scopes is depicted in Figure 4.3. $S_0$ is a scope that references a number of ontologies in a knowledge base $O$, and does so entirely by inheriting the references of its $m$ ontology spaces $sp_1 \ldots sp_m$.

**Theorem 2.** Let $\{(I, \emptyset)\}$ be the class of *trivial* ontology collectors. Then every nontrivial scope is part of an ontology network that includes the ontological forms of all its spaces.

*Proof.* Let $S = (I_S, R_S)$ be a nontrivial scope, i.e. $R_S \neq \emptyset$. Then, by definition of scope, all references in $R_S$ are inherited from some ontology space. By induction on the number of spaces of the scope:

- If $S$ has only one space, then let $sp_1$ be the one space owned by $S$. Because $S$ is nontrivial, and all its managed ontologies must be inherited from $sp_1$, then $sp_1$ is managing at least one ontology. Therefore, per Theorem 1 $f(sp_1)$ is part of an ontology network $O'$ for some $f$. Per Hypothesis A, $S$ has an ontological form $f(S)$. Then $O' \cup f(S)$ is an ontology network, because $f(S)$ depends on $f(sp_1)$, which is an element of $O$.

- If $S$ has $n$ spaces and they all form an ontology network $O''$, then let $sp_{n+1}$ be the $n+1$-th scope of $S$. if $sp_{n+1}$ is not managing any ontologies, then $O'' \cup f(sp_{n+1})$ is an ontology network, because $f(S)$ still depends on $f(sp_{n+1})$ by construction. Otherwise, if $sp_{n+1}$ is managing some ontology, then per Theorem 1 $f(sp_{n+1})$ is part of an ontology network $O'''$. Because $S$ inherits the ontology references of $sp_{n+1}$, there is a dependency between $f(S)$ and $f(sp_{n+1})$, therefore $O'' \cup O'''$ is an ontology network.

□

Scopes are mainly defined for practical purposes due to their convenience in grouping ontology spaces where simple sets do not suffice, hence the requirement of not influencing the logical axioms of ontology images. However, their definition can also be regarded as a guideline for distributing ontology management across spaces, especially according to the conditions in (2).

#### 4.1.3.6 Session

The last specialization of ontology collector introduced here is the *session*, which is an ontology collector that influences the images of its managed ontologies using an aggressive policy on managing and referencing ontologies.

**Definition (session).** A *session* is an ontology collector $z = ((i_z, nil), R_z)$, where:

1. $R_z$ only contains references to ontologies or scopes;
2. ontologies managed by $z$ are not managed by any other ontology collector;
3. for every $o \in O$ so that $\$_o \in R_z$, where $O$ is a knowledge base, there exists an image $o^z$ with identifier $(i_o^z, v_o^z)$, where $i_z$ is a prefix for $v_o^z$ iff $o$ is unversioned, i.e. whose references are of type $(i_o, nil)$.

In other words, a session has exclusive management rights to its managed ontologies, and tries to manipulate them so that their images publicly declare their binding to the session, unless they were given a version identifier before being managed by the session.

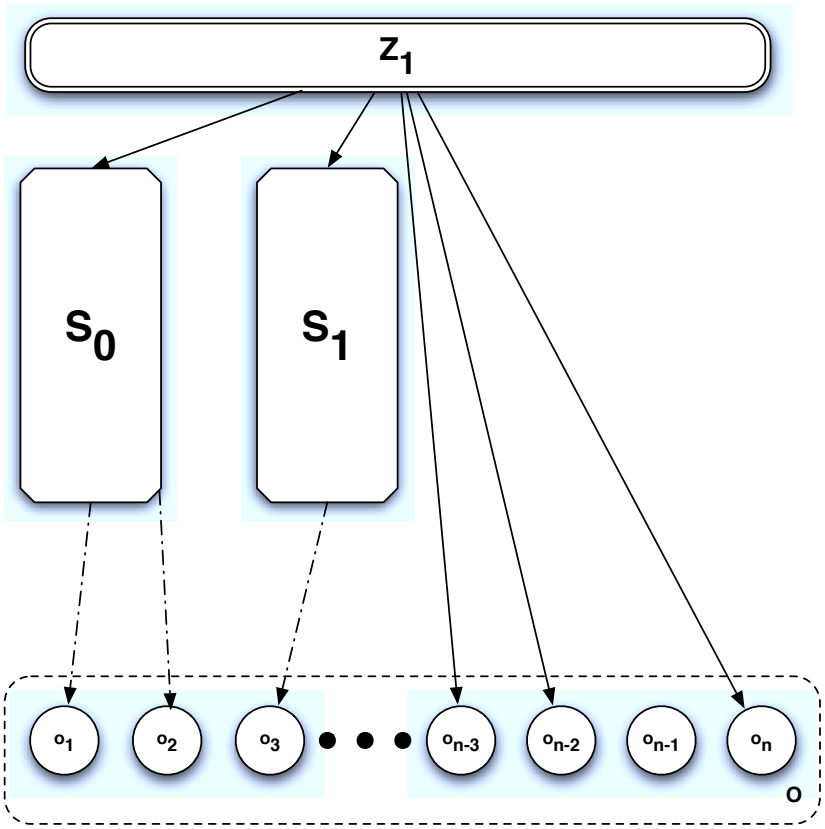


**Figure 4.4: Ontology referencing mechanism for sessions** - Session $z_1$ references two scopes $S_0$ and $S_1$ and manages three ontologies in $O$, $o_{n-3}$, $o_{n-2}$ and $o_n$.

Figure 4.4 depicts an example of a session $z_1 = (I_{z_1}, R_{z_1})$, which references two scopes $S_0$ and $S_1$, meaning that $R$ contains two references $\$_{f(S_0)}$ and $\$_{f(S_1)}$, where

$f(S_0)$ and $f(S_1)$ are the ontological forms of $S_0$ and $S_1$ respectively. In addition, $z_1$ directly references its three managed ontologies $o_{n-3}$, $o_{n-2}$ and $o_n$.

**Theorem 3.** Every nontrivial session is part of an ontology network that includes the ontological forms of all the scopes referenced by it.

*Proof.* For a session $z$ to be nontrivial, it must contain at least one reference to a scope or ontology. Per Theorem 2, every scope either is part of an ontology network or, if trivial, is still the source of an ontology per Hypothesis A. Because a session is an ontology collector, it is also an ontology source, and if $z$ references a scope $S$, then their ontological forms depend on one another (i.e. $d(f(z), f(S))$ is true). Therefore, referenced scopes form an ontology network with $z$, hence the thesis. □

## 4.2 Virtual ontology network assembly in OWL 2

With the fundamental artifacts of virtual ontology networks now in place, it is now possible to describe how these can be combined into structures that can generate ontology networks. We will show how this is possible in OWL 2 using the features and bindings expressed by its specification [GWPS09, MPSP+09].

The structural skeleton of virtual ontology networks is sketched in Figure 4.5. With a venial notation abuse, since we are not modeling a software system in this chapter, UML component notation was used for describing the relationships between network artifacts. Aggregation and composition connectors are used in lieu of the "references" relation described earlier in this chapter. Recall that in UML 2 a *composition* relation holds when the lifetime of a component matches that of its composite, otherwise an *aggregation* relation holds [Amb05]. The same principles holds for the lifetimes of ontology collectors.

In the figure, a *session* is represented as an aggregate of both scopes and ontologies, as it does not cease to exist if it is referencing neither scopes nor ontologies. On the contrary, a *scope* is represented as a composite of one more *ontology spaces* exclusively, as we recall from the definition of scope, that it does not have any ontology references other than those inherited from its spaces, and there are no scopes without at least one space. Each ontology space, in turn, aggregates zero or more ontologies. What is not shown in the figure, but is known from the definition of scope in Section 4.1.3.5, is that

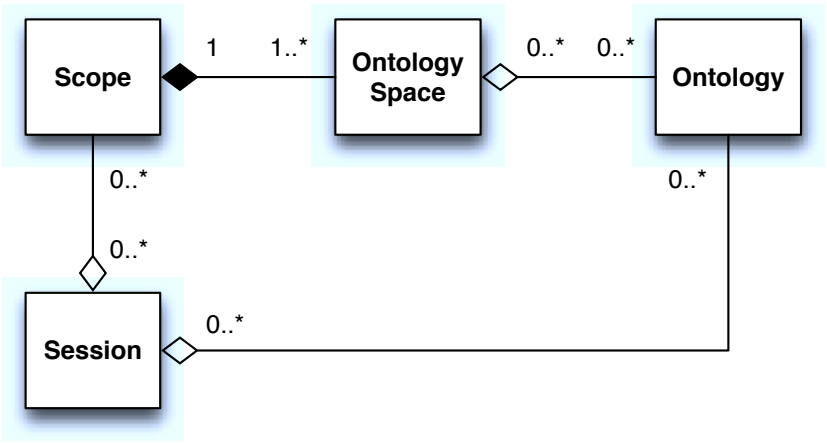


**Figure 4.5: Virtual ontology network composition diagram** - While sessions and ontology spaces are *aggregates* of any number of ontologies, and can in fact exist without referencing any, scopes are *composites* of ontology spaces, and a scope cannot exist without at least one space belonging to it.

the sets of ontologies managed by the spaces belonging to one same scope must have nothing in common with one another. Sessions, on the other hand, are not given this constraints, nor are those scopes that form different ontology networks.

### 4.2.1 Multiplexing

Recall from Section 4.1.3.3, that when an ontology $o, o \in O$ is managed by an ontology collector $C$, then a new ontology exists, called the $C$**-image of** $o$, which does not belong to the original knowledge base $O$. In addition, $o$ can be managed by multiple ontology collectors at the same time, with the only exception of a single ontology space within each scope. This means that, if $m$ are such ontology collectors, then $m$ ontology images will be created for $o$ or, to rephrase, $o$ will have $m$ images. In addition, these images will all be different, if anything because of the axioms that give them their names, per conditions (1) and (2) of the definition of ontology image.

This process of creating many different images from a single ontology is called *multiplexing*. Therefore, in order for an ontology to be multiplexed, it must be managed by at least one ontology collector, and there has to be a procedure that queries the ontology collector for the corresponding image.

An example of how ontologies can be multiplexed by an ontology collector is shown in Figure 4.6. In the figure, $o_1$ and $o_2$ are two ontologies in a knowledge base, i.e.

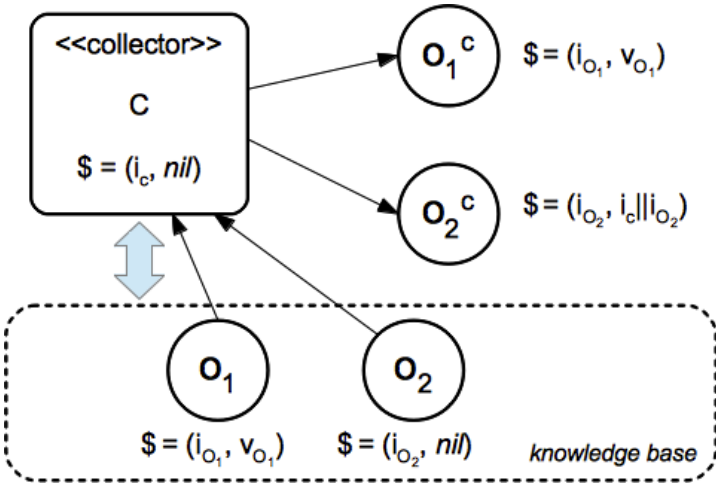


**Figure 4.6: Multiplexing in an ontology collector** - Ontology collector create images for ontologies stored in the knowledge base. Whenever the ontology reference has no version identifier, the collector sets its own identifier as a prefix for the version identifier of the image.

in their plain, unmanaged form. $o_1$ ha a fully qualified reference $(i_{o_1}, v_{o_1})$, while $o_2$ has a reference that lacks a version identifier, $(i_{o_2}, nil)$. $C$ is an abstract ontology collector, whose specific type we need not know at this stage. We do, however, need to know its identifier, which we have expanded as $(i_C, nil)$ for compatibility with ontology references[1]. $C$ manages both $o_1$ and $o_2$, therefore each ontology has a $C$-image, $o_1^C$ and $o_2^C$, respectively. Because the reference of $o_1$ was fully qualified, the collector does not rewrite the version identifier of $o_1^C$, which remains $(i_{o_1}, v_{o_1})$. However, a version identifier can computed for $o_2^C$ as the concatenation of the collector identifier with the ontology identifier, $(i_C||i_{o_2})$[2]. The reference of $o_2^C$ then becomes $(i_{o_2}, i_C||i_{o_2})$.

The method described here uses specific ontology collector types in order to organize ontology images across three layers of complexity, which are called *tiers*. Each tier accommodates one single type of ontology collector only, and mainly communicates with adjacent ones via connectivity and dependency relations.

A schematic representation of the rationale behind this method is given in Figure 4.7. The basic tier, which we have numbered as zero, is the original knowledge base that

[1]Recall that the identifiers of ontology collectors are not versioned.

[2]We have used the vector concatenation operator '||' for generality; however, in cases such as RDF and OWL 2, where the identifiers are represented by URIs or IRIs, the string concatenation operator '.' can be assumed.

constitutes the pool by which ontologies are taken from ontology collectors and thereby imaged. Tiers 1 and 2 accommodate two different classes of ontology space, called *core spaces* and *custom spaces*, respectively, while in tier 3 ontologies are distributed across *sessions*. Two adjacent tiers in the figure, naming those of core and custom spaces, are delimited by thicker strokes and grouped as the *scopes* layer, which hints at a grouping of core and custom ontology space pairs into scopes.

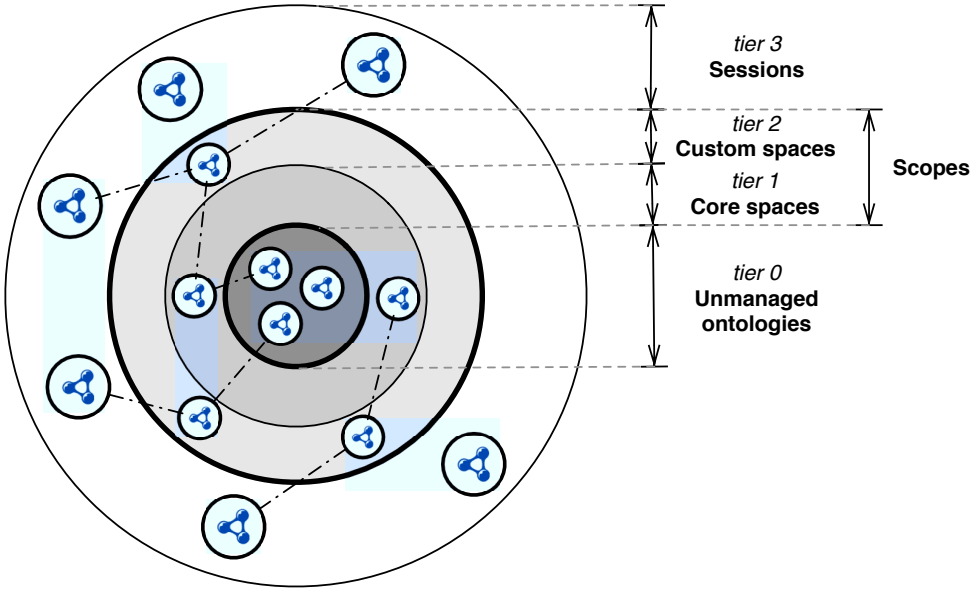


**Figure 4.7: Distribution of ontologies across tiers in virtual ontology networks** - Tier 0 contains the original knowledge base. In cases where an ontology with images in tiers 1 or 2 is natively networked, its image can still reference ontologies in tier 0. If an ontology is natively networked in tier 0, then its connectivity is preserved, thus still allowing for paths longer than 3.

The circles with an ontology icon inside denote ontologies[1]; those in tier zero are the original, unmanaged ontologies, while those in the upper tiers are ontology images created by some collector. Dashed and dotted edges that connect ontologies indicate some connectivity relation between them, therefore every connected graph resulting from combining nodes edges, the latter being at least one, denote *ontology networks*. Also note that, in the figure, some unmanaged ontologies in tier zero belong to ontology networks. This happens whenever an ontology collector is set to manage an ontology

[1]The standard ontology icon was used in lieu of identifiers because the latter are not relevant in the figure.

that is *natively networked*, i.e. designed by its developers to already be part of an ontology network, or in other words, that is part of a *statically assembled* ontology network. For example, if the *Agent-Role* ontology design pattern [Prea] is set to be managed by a core space [Preb], because this pattern inherits from other patterns such as *Object-Role* [Prec] and *Classification* [Preb] and is therefore part of a static ontology network, these other patterns will be in Tier 0 and will be indirectly referenced by the ontology space that manages *Agent-Role*.

Each tier has its own way of setting its management status on submitted ontologies. The following sections provide algorithmic support to each tier built upon the base one.

### 4.2.2 Tier 1: core spaces

This section illustrates how ontology collectors in tier 1, which are ontology spaces, proceed in managing ontologies submitted to them. Algorithm 1 explains the procedure. Ontologies are passed to collectors *by reference*, which is why they take $\$_o$ as an argument instead of $o$.

**Data**: $\$_o$ : an ontology reference; $s = ((i_s, *), R_s)$ : a core space; $O$ : a knowledge base
**Result**: modified states of $s$, $o$ and $O$
**if** *$o$ is already managed by the custom space of the same scope* **then**
  ▷ do nothing
**else**
  **if** $o \notin O$ **then**
    load $o$;
    $O := O \cup \{o\}$; ▷ push $o$ into tier zero
  **end**
  **foreach** $o' : d^*(o, o')$ **do** **if** $o' \notin O$ **then**
    load $o'$; ▷ store _every_ dependency that is not present
    $O := O \cup \{o'\}$;
  **end**
  $R_s := R_s \cup \{\$_o\}$;
**end**

**Algorithm 1:** Algorithm for setting an ontology to be managed by a core space.

The algorithm reads as follows. When an ontology reference is submitted to a core space, since all ontology spaces in tiers 1 and 2 belong to some scope, one has to

check whether the custom space of the same scope is not already managing the same ontology. This also implies that the ontology is already stored. If this is true, then the core space cannot manage the same ontology. Otherwise it must be checked whether the ontology is natively networked: is so, all its dependencies (both direct and indirect) are brought into the knowledge base, i.e. pushed into tier zero. Finally, the reference to the ontology is added to the reference set of the core space.

### 4.2.3 Tier 2: custom spaces

This section illustrates how ontology collectors in tier 2, which are ontology spaces as well, proceed in managing ontologies submitted to them. Algorithm 2 explains the procedure.

**Data**: $\$_o$ : an ontology reference; $s = ((i_s, *), R_s)$ : a custom space; $O$ : a knowledge base
**Result**: modified states of $s$, $o$ and $O$
**if** *$o$ is already managed by the core space of the same scope* **then**
  ▷ do nothing
**else**
  **if** $o \notin O$ **then**
    load $o$;
    $O := O \cup \{o\}$; ▷ push $o$ into tier zero
  **end**
  **foreach** $o' : d^*(o, o')$ **do** **if** $o' \notin O$ **then**
    load $o'$; ▷ store _every_ dependency that is not present
    $O := O \cup \{o'\}$;
    **if** $d(o, o')$ **then**
      ▷ direct dependencies must always be hijacked
      remove $d(o, o')$ from $o$;
      add $d(o, s_{o'})$ to $o$;
    **end**
  **end**
  $R_s := R_s \cup \{\$_o\}$;
**end**

**Algorithm 2:** Algorithm for setting an ontology to be managed by a custom space.

The algorithm is nearly the same as Algorithm 1, with one exception. When an ontology reference is submitted to a custom space, since all ontology spaces in tiers

1 and 2 belong to some scope, one has to check whether the core space of the same scope is not already managing the same ontology. This also implies that the ontology is already stored. If this is true, then the custom space cannot manage the same ontology. Otherwise, the ontology can be managed, but first the algorithm must check whether the ontology is natively networked: is so, all its dependencies (both direct and indirect) are brought into the knowledge base, i.e. pushed into tier zero. Finally, the reference to the ontology is added to the reference set of the custom space.

### 4.2.4 Tier 3: sessions

This section illustrates how ontology collectors in tier 3, which are all sessions, proceed in managing ontologies submitted to them. Algorithm 3 explains the procedure.

**Data**: $\$_o$ : an ontology reference; $z = ((i_z, *), R_z)$ : a session; $O$ : a knowledge base
**Result**: modified states of $z$, $o$ and $O$
**if** $o \notin O$ **then**
  load $o$;
  $O := O \cup \{o\}$; ▷ push $o$ into tier zero
  give $z$ and exclusive write-lock on $o$;
**end**
**foreach** $o' : d^*(o, o')$ **do** **if** $o' \notin O$ **then**
  load $o'$; ▷ store _every_ dependency that is not present
  $O := O \cup \{o'\}$;
  **if** $d(o, o')$ **then**
    ▷ direct dependencies must always be hijacked
    remove $d(o, o')$ from $o$;
    **if** *exists space* $c = (i_c, R_c)$ *that manages* $o'$ **then**
      $R_c := R_c \cup \{\$_S\}$, where $S$ is the scope that owns $c$;
    **else**
      ▷ no such space : set a dependency using the ontology source
      add $d(o, o')$ (via $s_{o'}$) to $o$;
    **end**
  **end**
**end**
$R_z := R_z \cup \{\$_o\}$;
**Algorithm 3:** Algorithm for setting an ontology to be managed by a session.

The algorithm for managing ontologies in sessions is more complex than those for

managing ontologies in spaces, due to the different nature of these ontology collectors and the fact that they are designed to collect volatile ontological data, therefore their policies are more aggressive.

As with core and session spaces, if an ontology that is not part of the knowledge base is added to a session, the algorithm proceeds to store it, i.e. push it into tier zero. In addition, however, it gives the session an exclusive write-lock on the ontology, which means that the session will be the only object entitled to dispose the ontology once it has run its course. This lock is not given if the ontology was already in the knowledge base, because it is assumed that the ontology was stored by another procedure on the assumption that other ontology networks, real or virtual, could require it.

Afterwards, every dependency of the submitted ontology is stored into the knowledge base, as with tiers 1 and 2. However, a further check is performed here as well: if the dependency relation at hand is a *direct* dependency relation, hence the use of $d(o, o')$ instead of $d^*(o, o')$, then the corresponding ontology is pushed into the lower tiers. The rationale is that, if a static ontology network is added to a session, then the other components of the network would not participate in the interpretation of the other network components added to the same session in the form of scopes. If these ontologies are moved to the lower tiers, it will be more likely that an interpretation procedure will visit them *before* it visits the content of the session, which is more likely to be comprised of ABox axioms. Therefore, if the algorithm finds that the direct dependency is already being managed by a *scope* attached to the sessions, i.e. belonging to the same ontology network in tier 1 or 2, then the dependency is removed, as it will be inherited from the scope itself, which in turn inherits it from its space. Otherwise, the dependency is rewritten, so that the submitted ontology depends on the other ontology via its *source* (which is now pointing to the knowledge base, because the ontology was stored beforehand)[1].

#### 4.2.4.1 Example: multi-user content directories use case

We recall here the use case introduced in Section 1.1.2.1, where in a multi-user CMS, different users – three in this simple example – need to classify their research paper directories according to different criteria. Suppose there is an application plugged into

[1]Recall from Section 4.1.2, that a dependency between two ontologies can hold by referencing, in one ontology, either the other ontology or its source.

the system, that is responsible for creating and synchronizing virtual directories. We now describe one possible way this application can exploit multiplexing techniques and lay out the ontologies required for obtaining the desired virtual networks. These ontologies are also shown in Figure 4.8. Once a virtual network is reasoned upon, subdirectories will be created for each user.

Let us take three CMS users A, B and C, each with their own private flat directory, and all sharing a common directory, also originally flat. For the metadata of all of these directories, there is an RDF representation computed by the system itself. We can assume Schema.org as the core vocabulary for representing these data, as it is a widespread standalone vocabulary.

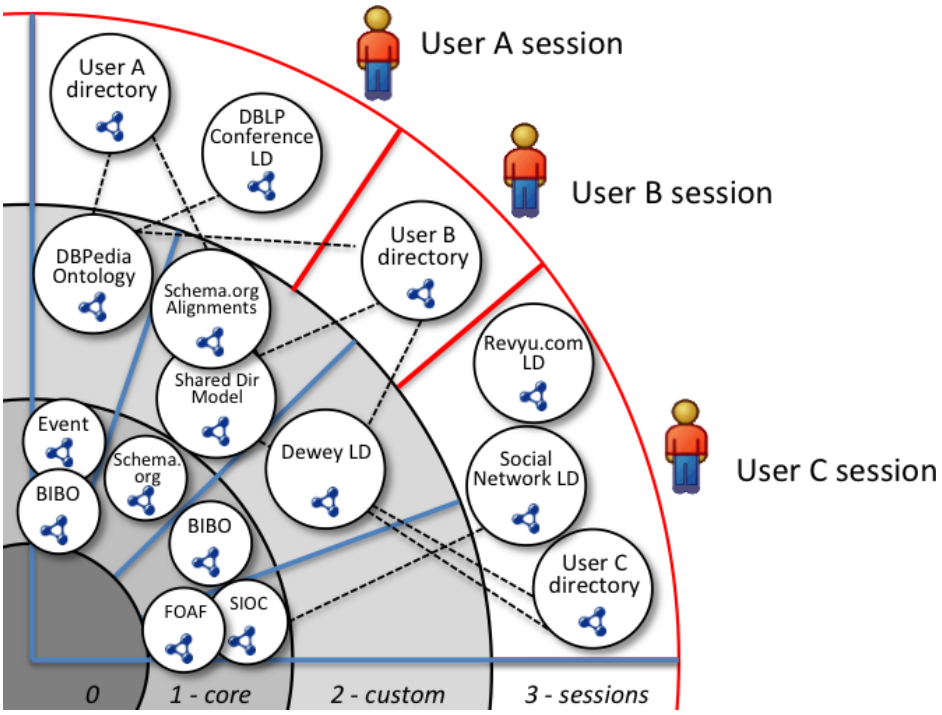


**Figure 4.8: Virtual ontology networks for multi-user content directories** - Core vocabularies and standalone TBox ontologies such as FOAF, Dublin Core and the Event ontology lie within core spaces of tier 1. Persistent ABox and dependent TBox ontologies are placed in custom spaces (tier 2). Volatile ABox ontologies are placed in tier 3, one session per user.

- User A wants to classify the papers she has access to according to the events they were presented at, when conference or workshop papers. The representation of

scientific events is provided by a Linked Data set that is aligned with the DBLP biographical data [DBL] and represented using a particular event ontology [RA07]. Let us assume, however, that the application responsible for creating content directories queries against the Schema.org event representation [Sch]. Therefore, an ontology containing alignments between the Schema.org OWL representation and the Event ontology is placed inside a custom space. This ontology space will belong to a scope whose core space manages both the event ontology and Schema.org. This way, the three ontologies will appear connected in the resulting ontology network that is obtained from user A's session (outer tier). This session, in turn, will be managing the metadata ontology of user A's private directory, as well as the DBLP biographical data for it.

- User B wants to classify her private and shared papers by topic using two schemas, i.e. the Wikipedia categories (available from DBPedia [DBP]) and the Dewey classification system [Mit09]. To that end, the application can keep track of a *topic* scope, whose core space contains standalone core vocabularies, e.g. Schema.org, and whose custom space contains one of the other TBox ontologies that extend or align with the former, such as the DBPedia ontology or the Dewey linked data. To obtain the desired ontology network, this scope (or these scopes, if one is use for each classification system) is appended to a dedicated session containing User C's private directory metadata in RDF, and so is the scope containing the RDF description of the shared directory.

- User C wants her papers to be classified according to the roles of their authors or reviewers in a social network she belongs to. Suppose for simplicity that there are RDF data available, which represent user C's profile and social graph, either because the social network itself exports them as Linked Data, or because a third-party application provides an RDF export for them. These data are represented using common vocabularies such as FOAF, SIOC and Schema.org (cf. Section 2.5). Reviews published on the Web are retrieved by querying the Revyu dataset [HM08]. The resources containing the data in question are managed by a dedicated session for C. Vocabularies for describing social network data, such as FOAF and SIOC, are stored in the core space of a dedicated scope. Since C also

wants to be able to retrieve papers by Dewey classification, the scope containing the Dewey schema as Linked Data can be reused from user B.

### 4.2.5 Exporting to OWL

This section illustrates how components of an ontology network assembled using the 3-tier multiplexing method can be exported as ontological artifacts fully compliant with OWL 2. All the constructs can be expressed as OWL 2 non-logical axioms, therefore the specific OWL Functional Syntax [MPSP+09] will be used. In turn, this syntax can be rendered in common OWL-compliant formats for consumption, including but not limited to RDF/XML, N3, Turtle, JSON-LD, OWL/XML and the Manchester OWL syntax (cf. Section 2.2.5).

#### Preliminaries

The OWL export strategy of our applied method will be described on a case-by-case basis in terms of the artifacts to be exported, whether they are ontology collectors or a specific type or their respective ontology images. The strategies are shown in Tables 4.2 through 4.9. For each artifact we will assume to be starting with a new, blank ontology with no axioms, logical or nonlogical, not even one that names the ontology itself.

In order to keep the tables concise and legible, we will make use of a few shortcuts. These are motivated by the following considerations related to some parts of the model that were defined earlier as abstract, and that can now be contextualized in OWL:

1. All objects defined as *identifiers*, such as $i$ and $v$ in an ontology reference $(i, v)$, are now typed as internationalized resource identifiers (IRIs) [DS05].

2. The concatenation operator '||' is implemented for IRIs using the string concatenation operator '.' as follows: $i_1||i_2 = i_1.i_2$ if $i_1$ ends with a slash character '/', otherwise $i_1||i_2 = i_1.\text{`/'}.i_2$ .

3. There is a function $iri$ that applies to ontology references, so that $I = iri(\$_o)$ is an IRI that is not part of any identifier of any other ontology in the knowledge base. Such a function can be, e.g. $iri((i_o, v_o)) = i_o.\text{`:::'}.v_o$ .

4. When an ontology $o$ is loaded and pushed in tier 0, as in $load(o)$, an ontology source $s_o$ is created for $o$, and there is a mechanism for resolving $\$_o$ to $s_o$.

5. A global variable is set, which will be called *connectivity policy*. Its possible values have been labelled as `LOOSE` and `TIGHT`. Their meanings will emerge from the procedures in the tables, but their rough meaning, informally, specifies whether the connection between two ontology collectors should be established directly between them, or delegated to the ontologies managed by one collector.

Each table shows, in their respective columns: (*i*) the condition that must be verified for a certain axiom to be added to the ontology; (*ii*) if an axiom is to be added as a sub-node of some other node, the latter; (*iii*) the syntactic pattern for the axiom to be added. The tables shown here are intended to provided *minimal* exports with as few axioms as possible, but richer exports e.g. with more ontology annotations are certainly possible.

### Core spaces

Table 4.2 shows how a core space should be exported to an OWL ontology, thus obtaining the ontological form of the space. The new ontology is named after the space itself and not given a version IRI. For any ontology managed by the space, an import statement is added. This import statement references the $c$-image of the managed ontology using its newly created source $s_o$.

| **Core space** $c = ((i_c, *), R)$ | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| Has ID $(i_c, *)$ | | `Ontology(`$i_c$`)` |
| $\$_o = (i_o, v_o) \in R, o \in O$ | `Ontology()` | `Import(`$iri(\$_{o^c})$`)` |

**Table 4.2: OWL 2 export of core spaces.**

### Core space ontology images

Table 4.3 shows how an ontology, in its managed form by a core space, should be exported to OWL, thus obtaining its ontology image with respect to the space. If the ontology name is fully qualified (i.e. it has both an ontology IRI and a version

IRI), then the ontology image has exactly the same name, otherwise a version IRI is added, which results from the concatenation of the identifier of the scope that the core space belongs to with the ontology IRI. Note that identifiers with a version IRI but no ontology IRI are illegal in OWL 2 [GWPS09].

Additionally, whenever the ontology has a dependency on another ontology, the corresponding import statement will reference the dependency via its ontology reference. Recall that ontology references in OWL can be exported to IRIs; that they resolved to ontology sources created after loading the ontology; and that every dependency was loaded into the knowledge base prior to exporting (cf. Section 4.2.2).

Finally, we do not give an indication as to how the set of raw statements $W$ in the ontology source should be interpreted. This is an aspect we intend to evaluate in Chapter 7. For now, we simply state that any raw statement that maps to an ontology naming axiom or an import declaration (i.e. one that could conflict with the ones we are adding here) must be ignored.

| **Image of ontology $o$ wrt. Core space $c = ((i_c, *), R)$** | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| $\$_o = (i_o, v_o)$ | | `Ontology(`$i_o$ $v_o$`)` |
| $\$_o = (i_o, nil)$<br>$c$ belongs to $S$ with ID $(i_S, *)$ | | `Ontology(`$i_o$ $i_S \| \| i_o$`)` |
| $\exists d(o, o'), o' \in O$ | `Ontology()` | `Import(`$iri(\$_{o'})$`)` |
| $s_o$ has raw statement set $W$ | | *interpretation of* $W$ |

**Table 4.3: OWL 2 export of core space ontology images.**

### Custom spaces

Table 4.4 shows how a custom space should be exported to an OWL ontology, thus obtaining the ontological form of the space. The new ontology is named after the space itself and not given a version IRI. For any ontology managed by the space, an import statement is added. This import statement references the managed ontology image using its newly created source $s_o$. In addition, if the global connectivity policy is set to `LOOSE`, an additional import statement will connect the custom space with its corresponding core space.

| **Custom space** $c = ((i_c, *), R)$ | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| Has ID $(i_c, *)$ | | `Ontology(`$i_c$`)` |
| $\$_o = (i_o, v_o) \in R, o \in O$ | `Ontology()` | `Import(`$iri(\$_{o^c})$`)` |
| Owning scope is $S$ with ID $(i_S, *)$. Core space of $S$ is $g$ with ID $(i_g, *)$. Connectivity policy is `LOOSE`. | `Ontology()` | `Import(`$i_S \| \| i_g$`)` |

**Table 4.4: OWL 2 export of custom spaces.**

### Custom space ontology images

Table 4.5 shows how an ontology, in its managed form by a custom space, should be exported to OWL, thus obtaining its ontology image with respect to the space. If the ontology name is fully qualified (i.e. it has both an ontology IRI and a version IRI), then the ontology image has exactly the same name, otherwise a version IRI is added, which results from the concatenation of the identifier of the scope that the space belongs to with the ontology IRI.

If the global connectivity policy is set as `TIGHT`, then an additional import statement is added in order to set a dependency on the *core* space that shares the same scope as the custom space at hand. The core space is referenced by the concatenation of the scope identifier and the core space identifier. Note that approach is dual with respect to the export of custom spaces themselves: if the connectivity policy is not `TIGHT`, then it is `LOOSE` and the import statement is added to the custom space itself instead of its ontologies.

Additionally, whenever the ontology has a dependency on another ontology, the corresponding import statement will reference the dependency via its ontology reference. Recall that every dependency was loaded into the knowledge base prior to exporting (cf. Section 4.2.3).

### Scopes

Table 4.6 shows how a scope should be exported to an OWL ontology, thus obtaining the ontological form of the scope. The export of scopes is fairly simple. The new

| **Image of ontology $o$ wrt. Custom space $c = ((i_c, *), R)$** | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| $\$_o = (i_o, v_o)$ | | `Ontology(`$i_o$ $v_o$`)` |
| $\$_o = (i_o, nil)$<br>$c$ belongs to $S$ with ID $(i_S, *)$ | | `Ontology(`$i_o$ $i_S \vert\vert i_o$`)` |
| $\exists d(o, o'), o' \in O$ | `Ontology()` | `Import(`$iri(\$_{o'})$`)` |
| Owning scope is $S$ with ID $(i_S, *)$.<br>Core space of $S$ is $g$ with ID $(i_g, *)$.<br>Connectivity policy is `TIGHT`. | `Ontology()` | `Import(`$i_S \vert\vert i_g$`)` |
| $s_o$ has raw statement set $W$. | | *interpretation of* $W$ |

**Table 4.5: OWL 2 export of custom space ontology images.**

ontology is named after the scope itself and not given a version IRI. Exactly two import statements are added, one for the core space of the scope, and one for its custom space. All the ontologies managed by the scope are so by inheritance from these two spaces, therefore *no* import statements are added for referencing those ontologies, as they will be inherited from the ontological forms of the two spaces.

| **Scope $S = ((i_S, *), R)$** | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| S has ID $(i_S, *)$ | | `Ontology(`$i_S$`)` |
| Has core space with ID $(i_c, *)$ | `Ontology()` | `Import(`$i_S \vert\vert i_c$`)` |
| Has custom space with ID $(i_g, *)$ | `Ontology()` | `Import(`$i_S \vert\vert i_g$`)` |
| $\$_o = (i_o, v_o) \in R, o \in O$ | | *nothing* |

**Table 4.6: OWL 2 export of scopes.**

### Scope ontology images

Table 4.7 shows how an ontology, in its managed form by a custom scope, should be exported to OWL, thus obtaining its ontology image with respect to the scope. As a scope inherits ontology references from its spaces, so its ontology images are inherited from those with respect to those spaces. Recall from the definition of scope, that two spaces of the same scope cannot share any ontologies, therefore whether an ontology is

being managed via one space of another is deterministic. Also recall from Tables 4.3 and 4.5, that the version IRIs assigned to the ontology images generated by scopes are prefixed by the *scope* identifier since, because of this property of scopes, it would be a useless addition for referencing an ontology.

| **Image of ontology $o$ wrt. Scope $S = ((i_S, *), R)$** | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| $o$ is managed by core space of S | | *See core space image table* |
| $o$ is managed by custom space of S | | *See custom space image table* |

**Table 4.7: OWL 2 export of scope ontology images.**

### Session

Table 4.8 shows how a session should be exported to an OWL ontology, thus obtaining the ontological form of the session. The new ontology is named after the session itself and not given a version IRI. For any ontology managed by the session, an import statement is added. This import statement references the managed ontology using its newly created source $s_o$. In addition, recall that sessions can reference scopes as well (assuming their ontological forms). Then, if the global connectivity policy is set to `LOOSE`, for every referenced scope an additional import statement will connect the session with that scope.

| **Session $z = ((i_z, *), R)$** | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| $z$ has ID $(i_z, *)$ | | `Ontology(`$i_z$`)` |
| $\$_o = (i_o, v_o) \in R, o \in O$ | `Ontology()` | `Import(`$iri(\$_{o^z})$`)` |
| $\$_S = (i_S, *) \in R$, $S$ is a scope. Connectivity policy is `LOOSE`. | `Ontology()` | `Import(`$i_S$`)` |

**Table 4.8: OWL 2 export of sessions.**

### Session ontology images

Table 4.9 shows how an ontology, in its managed form by a session, should be exported to OWL, thus obtaining its ontology image with respect to the session. In this respect, a

session behaves exactly like a custom scope, *mutatis mutandis*, when exporting ontology images. The version IRI of the ontology is set to one prefixed by the session identifier whenever possible, and import statements are added for all dependencies. If the global connectivity policy is `TIGHT`, then the ontology image will also reference the ontological form of *each* scope attached to the session.

| **Image of ontology $o$ wrt. Session $z = ((i_z, *), R)$** | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add** |
| $\$_o = (i_o, v_o)$ | | `Ontology(`$i_o$ $v_o$`)` |
| $\$_o = (i_o, nil)$ | | `Ontology(`$i_o$ $i_z \| \| i_o$`)` |
| $\exists d(o, o'), o' \in O$ | `Ontology()` | `Import(`$iri(\$_{o'})$`)` |
| $\$_S = (i_S, *) \in R$, $S$ is a scope. Connectivity policy is `TIGHT`. | `Ontology()` | `Import(`$i_S$`)` |
| $s_o$ has raw statement set $W$. | | *interpretation of* $W$ |

**Table 4.9: OWL 2 export of session ontology images.**

### Unmanaged ontologies

Finally, some manipulation of ontologies in their original, unmanaged form is required. If an ontology imported from the Web is added to a core space, and this ontology itself was part of an ontology network, then there will be import statements that reference other ontologies on the Web. This would prevent our method from assembling self-contained ontology networks, as at some point they branch outside of the controlled environment where the method is applied.

However, we note from Algorithms 4.2.2, 4.2.3 and 4.2.4, that every time a dependency is encountered, it is resolved into an ontology, and that ontology is loaded into our shared knowledge base. Therefore, all the dependencies could be resolved internally using their new ontology sources: it is only a matter of "hijacking" the import statements in order to reference the same ontologies via their new sources. This is essentially the meaning of the second row in Table 4.10.

Concluding this elucidation, we recollect having made two hypotheses before defining ontology spaces (cf. Section 4.1.3.3), namely that ontology collectors are valid ontology sources (Hypothesis A), and that an ontology management relation implies a

| **Unmanaged ontology** $o$ | | |
|---|---|---|
| **Condition** | **Node** | **Axiom to add/remove** |
| `Import(`$i$`)` exists in $o$.<br>$i$ resolves to $o'$. | `Ontology()` | replace with `Import(`$iri(\$_{o'})$`)`<br>($\$'_o$ references $o'$ via the new $s_{o'}$). |
| $s_o$ has raw statement set $W$. | | *interpretation of* $W$ |

**Table 4.10: OWL 2 export of unmanaged ontologies.**

dependency (Hypothesis B). Given the export rules defined earlier, we can now verify both hypotheses in the context of the OWL language and three tiers.

Hypothesis A is verified by example, having presented a way to export each type of ontology collector allowed into an artifact compliant with the OWL 2 specification [MPSP+09].

To verify Hypothesis B, we note that in Tables 4.2, 4.4 and 4.8 we had defined a way to add OWL import for every image of an ontology that is a dependency of a core space, custom space or session, respectively. Moreover, Table 4.6 shows that scopes add no import statements pointing to managed ontologies, but they inherit imports from the ontological forms of their spaces. Therefore, in the case of scopes the dependencies hold and are all indirect. Since we also provided a way to export ontology images with respect to all these ontology collector types in Tables 4.3, 4.5, 4.7 and 4.9, Hypothesis B is verified as well.

## 4.3 Ontology referencing vs. naming

When assembling ontology networks, be they real or virtual, users and applications need to be able to always locate the content of an ontology with some *reference object*, so that it is possible to specify a *connectivity mechanism* between two or more ontologies. Even when references are available for any ontology involved in the assembly process, the questions whether that reference will uniquely identify a single resource, and whether that resource is able to confirm that it is what it is expected to be, remain both unanswered.

However, according to the ontologies present on the Web, no uniform attention seems to be paid to naming issues and conventions. Homonymy and anonymity are not infrequent for Web ontologies, Linked Data sets and the like. One of the main

causes of the former is that, as an ontology is improved and modified over time, each revision often generates a new OWL file, and different revisions are simultaneously present on the Web and share the same identifier. The latter is often due to the lack of both naming conventions and collective consciousness on the importance of rendering a resource uniquely identifiable[1].

Throughout its history, the OWL ontology language has always provided simple and relatively lax methods for specifying the identifiers and references of an ontology. The OWL 2 specification requires that ontology identifiers be absolute internationalized resource identifiers (IRIs) [DS05]; encourages the use of an IRI pair for identifying and versioning ontologies; and uses an import-by-location scheme for referencing imported ontologies, regardless of the actual resource found when that location is resolved [MPSP+09]. These methods, however, have proven to be a double-edged sword, due to the scarce backwards compatibility with RDFS and other prior ontology languages and representation formalisms. That granted, since our framework is intended to serve ontologies gathered from multiple sources, we still need a way to make sure that each served ontology has a unique identifier, or at least a set of references, because import-by-location schemes require a resolvable one, and that these identifiers and references do not generate redundancy with other occurrences of that ontology on the Web. When the logical IRIs declared in the ontologies differ from the IRIs that were dereferenced to obtain their content (the physical IRI), it would still be preferable to keep track of these discrepancies. Doing so can be useful, for example, in order to avoid reloading an anonymous ontology from the same URL over and over again, simply because it has no logical identifier to memorize.

Native OWL representation formats, such as Manchester syntax, OWL functional syntax and OWL/XML include a specification of these identifiers in the header itself. For instance, in Manchester syntax an ontology header specifies identifiers as follows:

```
Ontology: <[ontologyIRI]><[versionIRI]>
```

In OWL/XML, ontology headers are expressed as XML attributes in this form:

[1]The problem of managing the identities of Web resources has been tackled from multiple angles, yet an extensively ontological approach exists too and has led to a formal model called *Identity of Resources on the Web (IRW)* [HP11].

```
<Ontology
  ontologyIRI="[ontologyIRI]"
  versionIRI="[versionIRI]"
  [...] >
  [...]
</Ontology>
```

Non-native representation formats, such as RDF/XML, Turtle and Notation-3 in general, resort to RDF-compliant workarounds to serialize ontology headers. These are represented using RDF triples under restrictions such as the following:

1. A *single, named* resource of type `owl:Ontology` must be declared in the graph. The name of that resource is the ontology IRI of the ontology represented by the graph.

2. A *single* triple *may* exist in the graph, where the aforementioned resource is the subject of an `owl:versionIRI` predicate. The object of that predicate is the version IRI of the ontology represented by the graph.

### 4.3.1 Public keys

The {`ontologyIRI, versionIRI`} pair is sometimes non-conventionally called *Ontology ID*. Although this [OWLc] Our proposal is to generalize this notion as follows.

**Definition (public key).** A *public key* $k_o$ is a reference $\$_o$ such that $o$ is *not* the ontological form of an ontology collector.

No assumption is made as to whether the IRIs of a public key are absolute or relative and whether they differ of match. In fact, any combination of these characteristics is possible and has its own interpretation, as it will be shown. In addition, the association function from public keys to ontologies is not a bijection, as explained by the following

**Definition (primary key and alias).** Let $K_o$ be the set of public keys of an ontology $o$. If $|K_o| > 1$ then $\exists! k_{io} \in K_o$ where $k_{io}$ is the *primary key* of $O$. Every other public key for $o$ is an *alias* for $k_{io}$.

The choice of which public key to elect as the primary key is arbitrary and depends on the implementation. It can either be the actual OWL ontology ID or some other object.

**Definition (traceable origin).** An ontology has a *traceable origin* if the resource where its content can be fetched from has a public locator. Being able to represent this public locator as an IRI is a necessary condition for an origin to be traceable.

The above definition is informal, but useful in its simplicity. Intuitively, it means that an ontology fetched from a URL with schemes such as `http`, `https` or `ftp` has a traceable origin. Conversely, an ontology fetched from a URL with a scheme for local usage, such as `file` or `dav`, does not have a traceable origin. Likewise, an ontology whose content was provided through a bitstream does not have a traceable origin.

The criteria adopted herein for generating public keys, and for selecting primary keys and aliases, are as follows:

1. If the ontology is non-anonymous (i.e. either named or versioned), its Ontology ID is a public key for that ontology.

2. if the ontology is anonymous and its origin is not traceable, then its primary key will be of type `(generatedIRI, generatedIRI)`, where `generatedIRI` is an arbitrarily computed IRI that can be absolute or relative.

3. if the ontology has a traceable origin, whose IRI form `originIRI` differs from either the ontology IRI or the version IRI if present, then the pair `(originIRI, originIRI)` is a public key for that ontology. This pair will be the primary key of the ontology if it is anonymous, otherwise it will be an alias for its primary key.

The key generation algorithm for case 2, i.e. anonymous ontologies without a traceable origin, is left to arbitration and depends on what information the repository system intends to encode within the public key itself. Multiple metadata can be concatenated together to form a single IRI. Examples include:

- the timestamp denoting when the ontology loading process was started or completed.

- the size of the ontology, calculated either in triples, axioms of any type, or logical axioms only.
- a hash code calculated either on the data stream of the graph, or on the collection of triples present in the graph itself. This computation can be costly, but can be performed alongside the ontology loading process and is of utmost significance for either indexing the ontology or content-based conflict detection.
- the canonical name of the object that was created for storing the ontology, e.g. a Java object or its class.

## 4.4 Relation to other work

Given the great availability of applications for processing OWL ontologies, and the increasing distribution of terminologies from the OWL vocabulary injected in Linked Data and other RDF graphs in the Semantic Web, we opted for assuming OWL as the family of reference languages that every ontology network should be interpreted as. Consequently, we agreed to deal with the procedural limitations of OWL, one being the assumption of *centralized resolution*, meaning that a single system is responsible for performing inferencing on the entire terminology of an ontology network. In the light of this limitation, our approach opted for manipulating the network layout and accommodating ontologies to the single system in question. If the assumption of using OWL is relaxed, then some limitations can be overcome, as demonstrated in Schlicht and Stuckenschmidt [SS09] theorizing a distributed resolution approach applied at reasoning time on an established ontology network. We have detected no counter-indication as to applying distributed approach to a virtual ontology network assembled by a multiplexing method, provided that the assumption of OWL is relaxed, without undermining the expressivity of the resulting ontology network. Within the domain of OWL, distributed approaches have also been defined, such as the $\varepsilon$-connection language solution proposal devised by Grau et al. [GPS06]. Although this approach makes provision for an extension of OWL that implements $\varepsilon$-connections using link properties, whereas ours has a greater concern for preserving standards, that does not make the two approaches incompatible. The proposal by Grau et al. dated as of OWL 1, namely OWL-DL, and as such made claims, such as `owl:imports` failing to deliver logical modularity

and contextualized axioms, which the introduction of punning in OWL 2 is bringing into question once again. This was further corroborated by observing the behavior of OWL 2 applications and libraries such as those we dealt with. We are therefore reasonably confident that an implementation of $\varepsilon$-connections in our multiplexing method would be not only possible, e.g. by manipulating OWL import statements applied to modularized ontology images, but also beneficial for the acceptance of their theoretical underpinnings in the context of industrial adoption.

As stated earlier, multiplexing can be implemented within, or sometimes in contrast to, the solutions adopted in existing ontology repository software as per Section 2.4. We will later show, when evaluating this method qualitatively, that our approach behaves, in the worst case scenario, in the same way as repositories that do not compile or interpret their ontologies server-side, such as TONES and ODP.org, yet in many cases it is able to readily provide contextualized OWL interpretation of dynamically networked ontology sources. In addition, multiplexing could be integrated in Cupboard as a method for users to only need to upload the ontologies that are private to each user, and which could be stored in sessions that are restored every time the user logs onto Cupboard. Instead, shareable ontologies as those belonging in the inner tiers of the virtual infrastructure, could be stored as singletons and mirrored to each Cupboard user space in their interpreted form once appended to each user's session. As for versioning, what emerges from this chapter is that our usage of version indicators has a broader scope than on ontology versioning as in the OMV vocabulary and Cupboard. In multiplexing, not only is the version indicator a component of ontology references, which is compatible with the Oyster approach (cf. Section 4.1.3.1), but also it is used for uniquely identifying ontology images with respect to collectors (cf. Section 4.1.3.3). This latter aspect transcends the totally-ordered approach of Oyster, as ontology image version indicators typically have no ordering, however, they never override version indicators that are already assigned, therefore our method approaches other versioning policies conservatively. Other ongoing efforts, such as the OOR and its notions of ontology mirroring and multiple instancing, are contemporary to our proposed solution, and the directions taken by them will have to be investigated as they proceed [Opeb].

# 5

# Architectural binding

So far we have sketched a theoretical and logical framework for dynamically assembling virtual ontology networks out of a shared knowledge base. In order to build a software platform, or enhance an existing one, for implementing or conforming to such a framework, a few additional steps are necessary. This chapter illustrates not only an architectural paradigm which maps to 3-tier multiplexing and existing software paradigms effectively, but also some support features needed for completing these steps. Because the exposure of Web ontologies is highly relying upon Web protocols, we will outline how software components should conform to a common REST architecture, which can be seen as an abstraction over Web protocols such as HTTP.

## 5.1 Conventions

This section provides a summary of the technological paradigms, naming conventions, and the like, which will be assumed as the starting grid for outlining the software binding with the ontology network model described in Chapter 4.

### 5.1.1 Service and component nomenclature

As a reference for service-based software paradigms, we will adopt the latest OASIS specification for service oriented architectures (**SOA**) [BEL+11]. The specification states a service to be a mechanism for providing access to a number of functionalities, *where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.* We are therefore

restricting to considering as services only those conforming to an interface defined statically, even in cases where the service in question is the only one to comply to its own interface.

Loosely bound to the service specification is the software component specification. We refer to the definitions provided by the **OSGi Alliance** as a base framework for defining software components, their grouping as bundles and activity lifecycle. The version of the OSGi specification we refer to is number 4.2 released in 2009 [All09b], as it was the latest complete specification at the time this research work started covering the software architecture aspect. In addition, the enterprise specification that compounded this release was the first one to introduce a particular interface-based service binding for components, called *declarative services* [All10], which reconnects us to the service paradigm adopted.

Although the OSGi specification provides a complete framework that can be implemented in the Java language, much of its core vocabulary and architecture also hold in platform-independent contexts. Therefore, we will not assume the Java bindings to hold as far as this architectural specification is concerned. These will be brought into context when the actual reference implementation, which was indeed developed in Java, is described in Chapter 6.

In addition, any mentions of services are not to be interpreted as Web Services unless otherwise specified (either explicitly or as RESTful services). Specifically, Web Services will mostly be addressed to when the exposure of ontology networks in standard language is treated, or when the output of services, be they singleton or composite [BC06], that are external to the platform, is deemed a valid ontology source.

### 5.1.2 The CRUD paradigm

Originally introduced by database engineers in the early 1980's [Mar83], **CRUD** is a popular acronym for **Create, Read, Update and Delete**, the four basic operations in persistent storage. One criterion for feature completeness in storage systems is to provide a full implementation of CRUD operations. With the introduction of the REST architectural paradigm [Fie00], it is a generally accepted notion that RESTful Web Services should implement CRUD operations in the HTTP protocol for all of their resources, modulo specific constraints for granting access to them.

In literature, the CRUD paradigm is sometimes extended as CRUDL, with the additional Listing operation with features such as pagination support, and BREAD, which reformulates create and update operations into add and edit, and splits read operations into browsing and reading. However, we shall maintain the standard CRUD paradigm due to its tight binding with REST architectures and RESTful interfaces which we are interested in respecting.

## 5.2 Artifacts

As previously done in Chapter 4 for the logical model of ontology networks, let us now proceed to outline the main actors of a software architecture compatible with this logical model. The artifacts described in this section are standalone items and are *not* direct implementations of those described in the previous chapter. Nevertheless, they are useful for expressing bindings with the latter, to be described later in this chapter (cf. Section 5.3).

### 5.2.1 Components and factory components

A **component** is a *class* of pieces of software that runs on a host application. The actual running piece of software that belongs to that class is a *component instance*. The lifecycle of a component instance is managed within the running time of the host platform. Each component instance is characterized by a manifest, or *configuration*, containing properties established by the developer and values set by either the developer or the host. Within the lifecycle of a component instance, its configuration can be *activated* or *deactivated*, in which cases it creates or disposes of its objects, respectively, depending on their activation and deactivation *functions* and the availability of their dependencies [All09a].

The host or a component can have the need to instantiate a component multiple times using multiple configurations. If it does so by implementing a factory pattern [Gam95] and creating a *component factory* (i.e. a service able to create, activate and deactivate multiple component instances), then the component whose configurations are managed in this way is called a *factory component* [All10].

### 5.2.2 Knowledge base

In the previous chapter (cf. Section 4.1.1), we simply defined a *knowledge base* (KB) to be any set of ontologies, not worrying about the form in which these ontologies actually exist. Whether they were files in a file system, tables in a relational database, graphs in a triple/quad store or dynamic objects built as the result of a SPARQL query, we simply introduced the definition of *ontology source*.

In the *architectural specification*, it is still possible to retain a degree of abstraction, as the knowledge base is here defined to be a *single* artifact that implements a storage mechanism for *any* set of ontologies, i.e. one that from time to time is storing a given set of ontologies $O$, where $O \in \wp(\mathcal{O})$. By notation, $\wp(\mathcal{O})$ is the power set of all ontologies.

The assumption here is that the knowledge base is able to, and solely responsible for, maintaining its own integrity. To that end, it has the following features:

- It MUST handle concurrency and serialization of simultaneous Update and Delete operations, or between said operations and Read operations.
- It SHOULD handle transaction control and commit points for operations on stored ontologies (*optional*).
- It MUST prevent Create operations on an existing named resource.

Whether a knowledge base is physically implemented as:

- a file system, or directory therein;
- one or more databases in a DBMS;
- a triple store or quad store;
- a live query broker for linked data and other Web ontologies;
- an in-memory heap, i.e. a portion of the memory pool.

it can be assumed to guarantee the features it MUST have and, on occasion, the features it SHOULD have (as is the case of feature-rich DBMSs).

### 5.2.3 Bundle

Components are not delivered as a single piece of software each. They are grouped into aggregate objects called *bundles*. A bundle is also responsible for selectively stating the dependencies of the components included, as well as what artifacts are exported for public access and which are kept private.

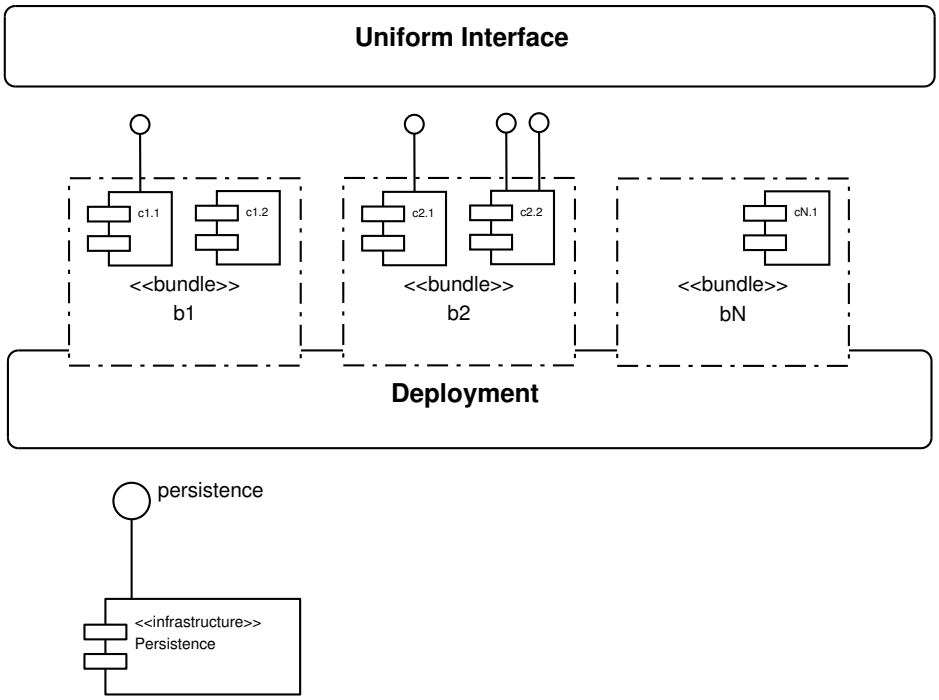


**Figure 5.1: Component architecture overview -**

### 5.2.4 Service

Within the context of this architectural framework, the concept of service is bound to the one of component. A **service** is a component whose functionalities are exposed by implementing some *interface* whose prescription is public. Service interfaces can be published by the component itself, or by another component, or by part of a software module that is not a component (possibly any interface code in a bundle). That same paradigm applies to RESTful Web Services, if the component is the factory that creates the RESTful resource and the interface is the specification of allowed HTTP methods.

## 5.3 Ontology network model bindings

The purpose of this section is twofold: (*i*) to provide an object model for a reference implementation of the ontology network model; and (*ii*) to instruct application developers on a possible way to tune their ontology-driven applications so that they can exploit such an object model. While the former was implemented as described in Chapter 6, the latter was simulated for some of the experiments described in Chapter 7 and is instantiated in some applications that adopt the reference implementation.

### 5.3.1 Object model

In Section 4.1.3, formal definitions of the concepts and constituents of the 3-tier multiplexing model for virtual ontology networks were given. Here, we shall now propose a model that maps those constituents to members of a model of object-oriented programming.

The UML class diagram [Amb05] in Figure 5.2 displays how artifacts of the ontology network model are bound to classes in an object-oriented software model. All the nodes in this UML schema are interfaces, because (*i*) in this API specification we do not need to keep track of specific class attributes, which can all be encapsulated into methods; and (*ii*) the actual implementation, which will be discussed in the next chapter, can have strong dependencies on the backend used for storing and retrieving ontologies, therefore multiple concrete implementations could be necessary (as they did indeed prove to be), which would drive away from an abstract interface specification.

The bindings of ontology network components to this object model are as follows:

- All ontology collectors in general are mapped to the `Ontology Collector` type in the object model. The `Ontology Collector` is in turn a `Named Resource`, which allows it to manipulate its own identifier, i.e. $i_C$ in the expanded ontology collector definition $C = ((i_C, nil), R)$, through encapsulation via `getId()` and `setId()` methods. The `manageOntology(inputSource)`, `unmanageOntology(key)` and `listManagedOntologois(key)` methods are the operations for managing the ontology references in $R$. Ontology export procedures, such as those illustrated in Section 4.2.5 for OWL, are bound to the `getOntology(key,merged)` method for the images of managed ontologies, and the `toOntologicalForm()` method of

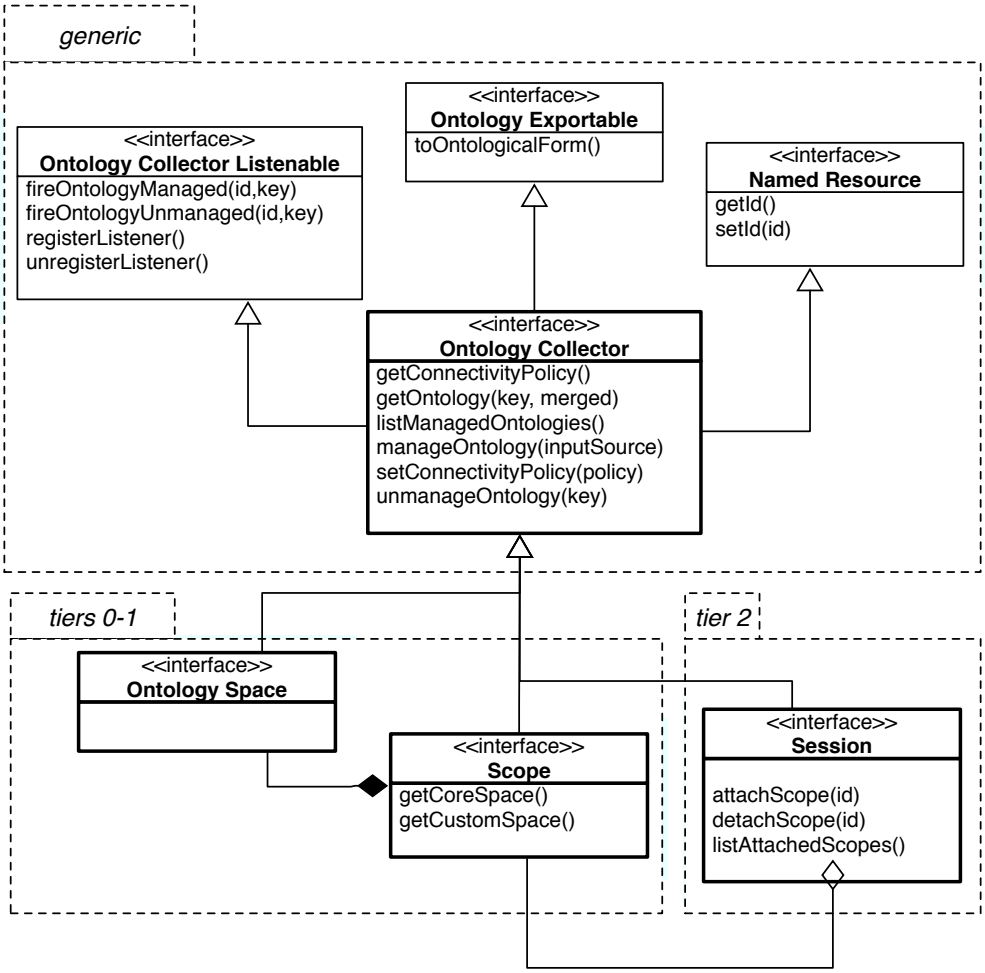


**Figure 5.2: Class diagram of artifacts from the ontology network model** - Interfaces that directly map to artifacts described in Section 4.1.3 are shown as having a thicker outline.

the `Ontology Exportable` supertype for the ontological forms of the collectors themselves.

- Ontology spaces are mapped to the `Ontology Space` type in the object model. This type extends `Ontology Collector`. No specialized objects are specified for core and custom spaces, as it is simply up to specific implementations of this interface to realize the behavior of core and custom spaces by overriding methods such as `toOntologicalForm()` and `getOntology(key,merged)`.

- Scopes are bound to the `Scope` type in the object model. In this model, the `Scope` acts as a façade for its two ontology spaces, because in the ontology network model a scope inherits all the ontology references of its spaces. Handles to ontology spaces can be obtained via the `getCoreSpace()` and `getCustomSpace()` methods. Also recall that a `Scope` is a composite of `Ontology Space` objects, as its lifetime is tightly bound to the lifetimes of its spaces.

- Sessions are bound to the `Session` type in the object model type in the object model. Besides being an ontology collector by its own right, a `Session` also aggregates `Scope` objects, which correspond to items of type $S$ so that $\$_{f(S)} = (i_S, *) \in R$, where $R$ is the reference set of the session. These scope references are encapsulated using the `attachScope(id)`, `detachScope(id)` and `listAttachedScopes()` methods.

As shown by this overview, in some cases there is a direct binding between tiers in the multiplexing method and objects in the software model, while in others, such as core and custom spaces, the binding is implicit. Other artifacts are unbound, as discussed in the next section.

#### 5.3.1.1 Unbound artifacts

Identifier types in general are not bound and it is left up to concrete implementations to specify them. That holds for both ontology collector identifiers (which are encapsulated in the `Named Resource` type for convenience) and public keys of ontologies in the KB implementation (both primary keys and aliases). The only restriction is applied to public keys, which obviously must be of a type that implements ontology references of the type $\$_o = (i, v)$ (cf. Section 4.1.3.1. In a software engineering approach applied

to OWL 2, this restriction can be fulfilled by encapsulating the two IRIs used for identifying ontologies, via methods such as `getOntologyIRI()` and `getVersionIRI()`.

Likewise, ontologies and their images and networks, be they real or virtual, are not bound to specific types. This is mainly due to the fact that, per the assumptions made and restrictions considered for the scope of this work (*A8*, existence of software tools for interpreting OWL; *R2*, import graph pre-order visit, cf. Sections 3.4 and 3.6), the task of interpreting resource sets as ontology networks is left to client applications and service consumers. Therefore, from the software perspective, their object model of ontologies and ontology network applies.

### 5.3.2 Operations

Some of the methods illustrated in Figure 5.2 are not entirely straightforward with respect to how they represent virtual ontology network construction operations. These will be further explained in this section. As for ontology export methods, it is assumed that they realize an operative schema akin to the one illustrated in Section 4.2.5.

#### 5.3.2.1 Manage/unmanage ontology

Recall from 4.1.3.2 that an ontology collector is a pair $(I, R)$, where $I$ is an identifier and $R$ a set of references. Then from the architectural perspective on an ontology collector, to *manage* an ontology $o$ is to add *some* $\$_o^i$ to $R$, where $\$_o^i$ is a reference for $o$. Vice versa, to *unmanage* $o$ is to remove from $R$ *every* $\$_o^i$ so that $\$_o^i \in R$ and $\$_o^i$ is a reference for $o$.

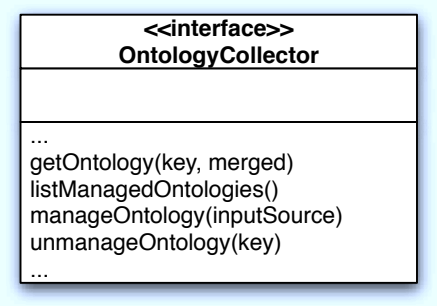


**Figure 5.3: Management methods of the ontology collector interface** - Parameter `key` can be of any type deemed suitable for encapsulating an ontology name. Parameter `merge` indicates whether, or up to which level, the dependencies of an ontology should be merged with it.

In Figure 5.3, the operations of managing and unmanaging an ontology map to the `manageOntology(inputSource)` and `unmanageOntology(key)`, respectively.

The following options apply to parameter `key`:

1. it can be of any type that can encapsulate the *public key* of an ontology, as in the definition of public key in Section 4.3.1;

2. it can be the primary key or any alias of the ontology referenced by it, even it that key was not used for managing the ontology by this collector in the first place.

Parameter `inputSource` is a wrapper for the following types:

1. an *ontology*;

2. a *public key*, in which case restriction 1 on parameter `key` applies;

3. an *ontology source*.

Managing an ontology may or may not imply a prior storage operation. Depending which type is being wrapped by `inputSource`, different conditions apply in order to determine whether the ontology should be stored at all. These conditions are summarized in Table 5.1. The general rationale is that, if the wrapped object is of the same type as public keys, then that key must match an entry in the knowledge base, otherwise the contrary must occur: any public keys automatically extracted from the ontology (i.e. that would be assigned to that ontology if it were to be stored) must not already exist as entries in the knowledge base.

Refinements to the policies expressed in Table 5.1 can apply. For instance, the policy for public keys can be either strict, i.e. the ontology collector is not set to manage the ontology with that public key if it matches an *orphan* entry (the key is registered but there is no associated ontology in the KB), or loose, i.e. the collector is set to manage the ontology by reference anyhow, assuming the actual ontology is provided in a second step.

| Wrapped type | Conditions |
|---|---|
| Ontology | (if named) Ontology name must not match any entry in the KB, unless the ontology itself is already stored. |
| Public Key | Public key must match an entry (either primary key or alias) in the KB.<br>(optional) Matching entry must not be orphan or uncharted. |
| Ontology source | Source must resolve to an ontology.<br>Ontology source locator or name (if applicable) must not match any entry in the KB. |

**Table 5.1: Conditions for storing an ontology in the knowledge base given its input source type**.

#### 5.3.2.2 Attach/detach scope

A session $z = (I_z, R_z)$, where $I_z = (i_z, *)$, has the ability to include scope identifiers in its reference set. From the object model perspective, scope identifiers in $R$ are handled by the `attachScope(id)`, `detachScope(id)` and `listAttachedScopes()` methods, as shown in Figure 5.4.

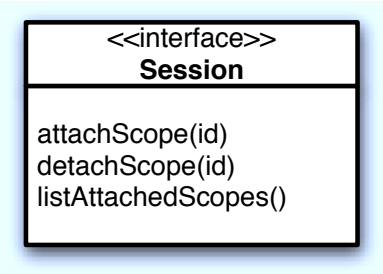


**Figure 5.4: Scope referencing methods of the session interface** - Parameter `id` can be of any type deemed suitable for encapsulating a collector identifier.

If $R_z$ is the reference set for session $z$, then the bindings of these operations with the ontology network model are as follows:

- `attachScope(id)`: adds a scope reference to $R_z$, i.e. $R_z := R_z \cup \$^{\texttt{id}}$, where $\$^{\texttt{id}}$ is the reference represented by object `id`.
- `detachScope(id)`: removes a scope reference from $R_z$, if present, i.e. $R_z := R_z \setminus \$^{\texttt{id}}$, where $\$^{\texttt{id}}$ is the reference represented by object `id`.

- `listAttachedScopes()`: returns (a representation of) a set of references $R'_z$ such that $R'_z \subseteq R_z$ and $\$_S \in R'_z$ iff $\$_S \in R_z$ and $S$ is a scope.

Note that parameter `id` does not have to be as the same type as the one that represents ontology references in the object model (the parameter `key` in ontology management operations). The only requirement is that both scope identifiers `id` and ontology public keys `key` are both accepted by the object that represents $R_z$. It is even acceptable that two separate objects store ontology public keys and scope identifiers separately. This choice is arbitrary and implementation-dependent, and as such it is not treated in this specification.

Attaching or detaching a scope does not imply any ontology storage operation, in general.

## 5.4 RESTful service interface

Design of the reference service platform that we used as a host began in 2009. Its requirement specification phase was thorough and articulated across multiple aspects of content management as well as general features. Among these, it emerged from the horizontal requirement specification [CES+10], that the industry was looking forward to integrating functionalities on a loose-coupling scheme that would not require drastic alteration of the architectures of existing software platforms. A solution was found in the then-emerging application of the REpresentational State Transfer (REST) paradigm [Fie00] to Web Services, which are therefore deemed RESTful [RR07].

Among other features, RESTful Web Services associate standards verbs of the HTTP protocol paradigm to expected functionalities that read or affect the state of resources directly responsible for providing Web Services. The OWL ontology language also normatively supports resolution of resources via the HTTP protocol and content negotiation for specific representation formats [SHK+09]. So do other Semantic Web specifications favor the adoption of the REST paradigm. Namely, the *uniform interface* constraint of the REST specification and the guiding principles of having identifiable resources sharing a base identifier are valuable allies of Linked Data [BB08]. These principles also proved to be compatible with a possible *representation* of our virtual ontology network structure as a set of resources, as well as convenient for designing their representation scheme. It was therefore desirable for us to develop a RESTful Web Service specification that allows client applications to interact with an ontology network provider and request ontology networks to be manipulated.

### 5.4.1 Service endpoints

This section provides an overview of the service endpoints for accessing ontology network components, managers and meta-level information that is not explicit in the networked ontologies themselves. Some of these RESTful resources are not fixed and their construction follows a certain naming pattern. This occurs whenever an item with an identifier is involved in locating a RESTful resource, as is the case of unmanaged ontologies, ontology collectors and images. In other cases, there are no variables involved, but this only occurs for accessing ontology network managers.

The fixed service endpoints and the naming schemes for variable ones are listed in Table 5.2. They are grouped as: (1) resources associated to objects that can be used for listing ontology network components, and manipulating them to a limited extent; (2) resources that represent specific ontology network components, each in turn an ontology source (cf. Section 4.1.1); (3) resources that enable access to meta-level features of ontologies, other than those declared in the annotations within the ontologies themselves. Each row represents a RESTful resource, its naming template being the way the URL of the resource is constructed and its type being the class of resource represented by that name. Each naming template begins with a slash ('/') character, and is intended to be concatenated to a common base URL (e.g. `http://localhost:8080/ontology` or `http://www.example.org/stanbol/ontology-network-manager`), which is omitted for simplicity, in order to obtain the full, unambiguous resource URL.

| | **Resource naming template** | **Resource type** |
|---|---|---|
| Manager | `/` | Ontology manager |
| | `/scope` | Scope manager |
| | `/session` | Session manager |
| Ontologies | `/{`*`publicKey`*`}` | Ontology entry |
| | `/scope/{`*`scopeID`*`}` | Scope |
| | `/scope/{`*`scopeID`*`}/core` | Core space |
| | `/scope/{`*`scopeID`*`}/custom` | Custom space |
| | `/scope/{`*`scopeID`*`}/{`*`publicKey`*`}` | {*scopeID*}-image |
| | `/session/{`*`sessionID`*`}` | Session |
| | `/session/{`*`sessionID`*`}/{`*`publicKey`*`}` | {*sessionID*}-image |
| Meta | `/{`*`publicKey`*`}/aliases` | Public key aliases |
| | `/{`*`publicKey`*`}/handles` | Handles on ontology |

**Table 5.2: Overview of REST endpoints for ontology network management** - These are the public resources that represent the components of ontology networks, both real and virtual, plus additional endpoints for management and meta-level access. Placeholders for variables are enclosed in curly braces. Note that `core` and `custom` are reserved and cannot be used as public keys.

These naming templates allow a client application to access the following resources:

- The **ontology manager** is a horizontal component that is aware of all the ontologies stored in the knowledge base of the host. It is able to list stored ontologies,

store new ones from heterogeneous sources and perform mass deletions. It does *not* serve virtual networks, but knows how many virtual network components are managing an ontology. These are also called *handles* on the ontology. Additionally, if an ontology belongs to a *real* (i.e. statically assembled at design time) ontology network, the ontology manager will also be able to list the other ontologies in the same network.

| **Ontology manager** | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | get meta-level ontology | |
| POST | load and store ontology | `file` : ontology content<br>`format` : MIME type of content (requires `file`)<br>`url` : URL of the ontology source |
| POST | load and store ontology | ontology content (as HTTP URL-encoded form) |

**Table 5.3: HTTP methods supported by the ontology manager REST resource.**

| **Ontology manager** | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | meta-level ontology |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| POST | 201 CREATED | primary key of stored ontology |
| | 400 BAD REQUEST | malformed `url` |
| | 409 CONFLICT | extracted public key matches existing entry |
| | 415 UNSUPPORTED MEDIA TYPE | invalid `format` |
| | 500 INTERNAL SERVER ERROR | failure to load ontology |

**Table 5.4: Response table of the ontology manager REST resource.**

- The **scope manager** is responsible for listing *scopes* (cf. Section 4.1.3.5), as well as their sizes in terms of the amount of ontologies managed by their spaces. It

is the only component to be aware of the activation status of registered scopes, which scopes themselves ignore.

| Scope manager | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | get meta-level scope ontology | |
| DELETE | clear all scopes | |

**Table 5.5: HTTP methods supported by the scope manager REST resource.**

| Scope manager | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | meta-level ontology |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| DELETE | 200 OK | |

**Table 5.6: Response table of the scope manager REST resource.**

- The **session manager** is responsible for listing *sessions* (cf. Section 4.1.3.6), as well as their sizes in terms of the amount of ontologies managed by them. It is also the component responsible for managing the lifetimes of registered sessions.

| Session manager | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | get meta-level session ontology | |
| POST | create session with automatically selected ID | |
| DELETE | clear all sessions | |

**Table 5.7: HTTP methods supported by the session manager REST resource.**

- An **ontology entry** is the native, unmanaged form of an ontology with the given public key. If an ontology with that public key is stored, its content can be returned by this endpoint. That is to say, that it is *not mandatory* for an entry to match a stored ontology. An entry can exist even if it is not mapped yet to any ontology *source* (in that case the entry is called *uncharted*), or if it is mapped to

| Session manager | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | meta-level ontology |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| POST | 201 CREATED | response entity is the endpoint URL of the new session |
| | 403 FORBIDDEN | maximum session quota reached |
| DELETE | 200 OK | |

**Table 5.8: Response table of the session manager REST resource.**

an invalid source, such as an RDF graph that does not exist or a URL that does not resolve to an ontology (in that case the entry is called *orphan*).

- A **scope** is a representation of an artifact of the type described in Section 4.1.3.5, accessible by its identifier (as $I$ in its formal definition $(I, R)$). Through this resource, a scope can be created (and automatically registered) or deleted (and therefore unregistered). A scope can be created and have its *core space* populated in a single operation, after which the core space is write-locked. A scope can be instructed to manage an ontology, which implies setting its *custom space* to manage it. However, the reverse operation is delegated to the resulting ontology image, as advised by the RESTful service guidelines[1]. As a scope is a valid ontology source in OWL, this resource supports content negotiation in multiple knowledge representation formats. If an RDF or OWL format is negotiated the ontological form of a scope imports those of its core and custom space. As with the other RESTful resources that represent ontological artifacts, a scope can be merged with all of its dependencies upon export.

- The **core space** of a scope lists the ontologies set to be managed by the scope at creation time, or while the scope is inactive. It does not support create, update or delete operations directly, as they have to be orchestrated by the scope owning this space. Being an ontology collector, the core space can be exported to RDF

[1] According to best practices in RESTful Web Services, any operation that deletes a resource should be invoked via an HTTP DELETE call on the resource itself [RR07]

| Ontology entry | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | get ontology | `merged` : if true, recursively merge dependencies |
| POST | load ontology content into entry | `file` : ontology content<br>`format` : MIME type of content (requires `file`)<br>`url` : URL of the ontology source |
| POST | load ontology content into entry | ontology content (as HTTP URL-encoded form) |
| POST | add alias for entry | `alias` : new public key |
| PUT | create uncharted ontology entry | |
| DELETE | - remove public key from aliases<br>- if there are no public keys left for the entry, delete the ontology | |

**Table 5.9: HTTP methods supported by the ontology entry REST resource.**

| Ontology entry | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| POST | 200 OK | new public keys not included in response |
| | 400 BAD REQUEST | malformed `url` |
| | 409 CONFLICT | extracted public key matches existing entry |
| | 415 UNSUPPORTED MEDIA TYPE | invalid `format` |
| | 500 INTERNAL SERVER ERROR | failure to load ontology |
| PUT | 201 CREATED | |
| | 404 NOT FOUND | entry is orphan (no stored ontology found) |
| | 409 CONFLICT | there are 1+ handles on corresponding entry |
| DELETE | 200 OK | |
| | 403 FORBIDDEN | there are 1+ handles on corresponding entry |

**Table 5.10: Response table of the ontology entry REST resource.**

| Scope | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | export to ontological form | `merged` : if true, recursively merge dependencies |
| POST | load ontology content and manage the corresponding ontology | `file` : ontology content<br><br>`format` : MIME type of content (requires `file`)<br>`stored` : public key of stored ontology entry (excludes `file`, `url`)<br>`url` : URL of the ontology source |
| POST | load ontology content and manage the corresponding ontology | ontology content (as HTTP URL-encoded form) |
| PUT | create and register scope, then populate its core space | `ontology` (0..*) : URLs of ontologies to manage in core space<br>`stored` (0..*) : public keys of stored ontologies to manage in core space<br>`activate` : if true, activate the scope immediately |
| DELETE | unregister and delete scope | |

**Table 5.11: HTTP methods supported by a scope REST resource.**

| Scope | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| POST | 200 CREATED | ontology image URL included in response |
| | 400 BAD REQUEST | malformed `url` |
| | 404 NOT FOUND | `stored` value is orphan or uncharted |
| | 409 CONFLICT | extracted public key matches an existing entry |
| | 415 UNSUPPORTED MEDIA TYPE | invalid `format` |
| | 500 INTERNAL SERVER ERROR | failure to load ontology |
| PUT | 201 CREATED | original request URL is returned |
| | 400 BAD REQUEST | failed to load at least one ontology into the core space |
| | 409 CONFLICT | a collector with that ID already exists |
| | 500 INTERNAL SERVER ERROR | other failure to create scope |
| DELETE | 200 OK | |

**Table 5.12: Response table of a scope REST resource.**

or native OWL, but since it belongs to the lowest tier of virtual networks, the ontological form of a core space will only import the ontologies managed by it.

| Core space | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | export to ontological form | `merged` : if true, recursively merge dependencies |

**Table 5.13: HTTP methods supported by a core space REST resource.**

| Core space | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |

**Table 5.14: Response table of a core space REST resource.**

- The **custom space** of a scope contains the set of ontologies set to be managed by the scope during its active state. As with the core space, this resource delegates create, update and delete operations to its owner scope. Custom space resources support the same content negotiation as core space resources, and their ontological forms import managed ontologies, plus an import statement that references the core space resource if the connectivity policy is set to `LOOSE`.

| Custom space | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | export to ontological form | `merged` : if true, recursively merge dependencies |

**Table 5.15: HTTP methods supported by a custom space REST resource.**

- An **ontology image with respect to a scope** is referenced by concatenating its encoded public key (either a primary key or an alias) to the resource name of the scope. The scope itself is responsible for preventing redundancies between its core and custom spaces, therefore this public key will be an unambiguous ontology reference. An ontology image can be exported either in standalone form, in which

| Custom space | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |

**Table 5.16: Response table of a custom space REST resource.**

case any dependencies are preserved and imports are simply rewritten to their corresponding images, or in merged form with all its dependencies. If the original ontology is managed by the custom space of the scope and the connectivity policy is set to `TIGHT`, the core space will also be set as a dependency and the appropriate import statement will be added. If the negotiated content type is a native OWL form, the virtual ontology network having this ontology image as its entry node will be interpreted prior to exporting. An OWL version IRI will be set for the ontology image, so as to match the URL of this resource. Clearly, `core` and `custom` are reserved terms and cannot be used as public keys of ontology entries.

| Ontology image wrt. Scope | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | export to ontological form | `merged` : if true, recursively merge dependencies |
| DELETE | instruct scope to stop managing the ontology | |

**Table 5.17: HTTP methods supported by an ontology image wrt. a scope REST resource.**

- A **session** lists its own managed ontologies and any scopes attached to it. If a session is removed via an HTTP DELETE request, it will attempt to remove the sources of all the ontologies it owns, i.e. those whose images it can set an OWL version IRI for. This occurs in accordance with its severance policies. When exported to a KR format and not set to be merged, its ontological form will import the endpoint URLs of all its managed ontologies. If the connectivity policy is set to `LOOSE` it will also import a any attached scopes.

| Ontology image wrt. Scope | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| DELETE | 200 OK | |
| | 403 FORBIDDEN | ontology is managed within the core space, and the scope is active |

**Table 5.18: Response table of an ontology image wrt. a scope REST resource.**

| Session | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | export to ontological form | `merged` : if true, recursively merge dependencies |
| POST | load ontology content and manage the corresponding ontology | `file` : ontology content<br>`format` : MIME type of content (requires `file`)<br>`stored` : public key of stored ontology entry (excludes `file`, `url`)<br>`url` : URL of the ontology source |
| POST | load ontology content and manage the corresponding ontology | ontology content (as HTTP URL-encoded form) |
| POST | select and attach scopes, detach any other scope | `scope` (0..*) : the IDs of the scopes to attach |
| POST | detach all scopes | |
| PUT | create and register session | |
| DELETE | unregister and delete session | |

**Table 5.19: HTTP methods supported by a session REST resource.**

| Session | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| POST | 200 CREATED | ontology image URL included in response |
| | 400 BAD REQUEST | malformed `url` |
| | 404 NOT FOUND | `stored` value is orphan or uncharted |
| | 409 CONFLICT | extracted public key matches an existing entry |
| | 415 UNSUPPORTED MEDIA TYPE | invalid `format` |
| | 500 INTERNAL SERVER ERROR | other failure to load ontology |
| PUT | 201 CREATED | original request URL is returned |
| | 403 FORBIDDEN | maximum session quota reached |
| | 409 CONFLICT | a collector with that ID already exists |
| | 500 INTERNAL SERVER ERROR | other failure to create session |
| DELETE | 200 OK | |

**Table 5.20: Response table of a session REST resource.**

- An **ontology image with respect to a session** is referenced by concatenating its encoded public key to the resource name of the session. Its ontological form follows the same rationale as images with respect to scopes, except that the `TIGHT` connectivity policy implies that, if the managing session has any scopes attached, the corresponding endpoint URLs will be imported, or included in the merge computation if the merge option is set.

| **Ontology image wrt. Session** | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | export to ontological form | `merged` : if true, recursively merge dependencies |
| DELETE | instruct session to stop managing the ontology | |

**Table 5.21: HTTP methods supported by an ontology image wrt. a session REST resource.**

| **Ontology image wrt. Session** | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED MEDIA TYPE | invalid MIME type in `Accept` header |
| DELETE | 200 OK | |

**Table 5.22: Response table of an ontology image wrt. a session REST resource.**

- The **aliases** endpoint of an ontology entry is a meta-level resource that only supports read operations for listing all the public keys associated with the ontology entry. It can be used to aid client applications in reconciling multiple names of an ontology, especially in case of a mismatch between its source location and logical name, or when a mnemonic name is used in lieu of the public key (e.g. '`foaf`' instead of "`http://xmlns.com/foaf/0.1/`").

- The **handles** endpoint of an ontology entry is a meta-level resource that only supports read operations for listing the identifiers of scopes and sessions managing that ontology, either by that public key or by any other associated public key. These identifiers form the set of *handles* of the ontology.

| Aliases | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | get aliases listing | |

**Table 5.23: HTTP methods supported by an aliases REST resource.**

| Aliases | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED | |
| | MEDIA TYPE | invalid MIME type in `Accept` header |

**Table 5.24: Response table of an aliases REST resource.**

| Handles | | |
|---|---|---|
| **Verb** | **Operation** | **Parameters** |
| GET | get handles listing | |

**Table 5.25: HTTP methods supported by a handles REST resource.**

| Handles | | |
|---|---|---|
| **Verb** | **Response code** | **Notes** |
| GET | 200 OK | |
| | 415 UNSUPPORTED | |
| | MEDIA TYPE | invalid MIME type in `Accept` header |

**Table 5.26: Response table of a handles REST resource.**

For the most part, the assembly of virtual ontology networks occurs by directly operating on the resources that represent ontology network components, or component candidates. There are, however, a limited number of exceptions. For example, the scope and session managers can be used to submit a new ontology for both storage and management in a single operation. In addition, the session manager can be used for creating new sessions when the client does not intend to specify their IDs.

There is no explicit way to address an entire ontology network as a single resource: to obtain a reference to an ontology network, through a single node that allows applications to visit the entire network, **a GET call on an ontology collector in the highest tier must be performed**. Therefore, if the ontology network includes a *session*, the GET call should be performed on the session resource; otherwise, if it includes a *scope* but no sessions, the call should be performed on the scope resource, and so on.

Note that a virtual ontology network *can* include more than one session, in principle. However, since with only 3 tiers there is no way to reference a session automatically from another ontology, it is not possible to visit more than one session in the virtual network, and this occurs only if the session represents the entry node for the virtual network visit.

# 6

# Implementing the model: the Stanbol ONM

To assess the feasibility of our proposed contribution, a reference implementation of our proposed solution was produced. We describe its features in this chapter.

Our logical model for virtual ontology networks, as well as one possible configuration of the corresponding architectural framework, was implemented as the **Apache Stanbol Ontology Network Manager**, which we shall refer to as simply ONM at times. Apache Stanbol [Apag] is a modular software framework that supports content management systems with Semantic-Web-based knowledge processing capabilities. It uses the component model and Java Virtual Machine management features of OSGi [All09b], stores ontologies using the uniform interface of sister project Apache Clerezza [Apaa], and exposes its content and knowledge through a full-fledged RESTful API [RR07, Bur09]. We implemented ontology network management and serving as RESTful Web Services that mimic ontology publishing via traditional Web servers, using the specification of Section 5.4. Stanbol is now fully endorsed as a top-level project of the Apache Software Foundation, and the ONM has been part of its codebase since its initial incubation period.

Although not digressing in depth into the coding details of the implementation, this chapter will provide sufficient grounds for the reader to understand whether and how development choices influenced the evaluation results, i.e. which potential bias is introduced or preserved, and which is eliminated. However, the presence of this chapter is imprescindible, as it documents the development path along which further

issues were discovered, which broadened the range of our research problem, in the meantime specializing it further.

## 6.1 Apache Stanbol overview

Apache Stanbol [Apag] is a Java 6 software project that provides a set of reusable components for semantic content management. An instance of Stanbol is intended for usage alongside a deployed content management system (CMS), possibly even in multiple instances. The Stanbol software project underwent incubation at The Apache Software Foundation (ASF) [Apae] in early 2011, after one year of design and active prototype development. In September 2012, less than two years since incubation, Stanbol was promoted to top-level project and fully endorsed by the ASF.

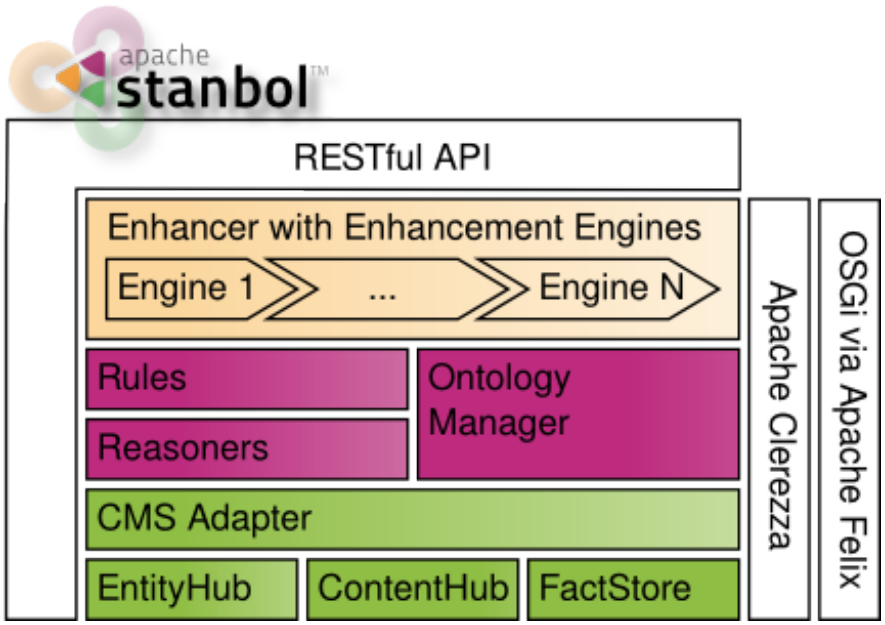


**Figure 6.1: Apache Stanbol component architecture** - Retrieved from Apache Stanbol Components.

A high-level abstraction of the Stanbol architecture is shown in Figure 6.1. Functionalities are grouped under the three major categories of persistence management (bottom layer), knowledge management (middle layer) and content enhancement pipelines (top layer). All these components run on top of an OSGi modular backend [All09b] using the OSGi implementation provided by the *Apache Felix* project [Apac]. A triple

store interface and binding is provided by sister project *Apache Clerezza* [Apaa], and all component functionalities are backed by a RESTful Web Service API.

Since its initial design phase, this framework already incorporated an early, stateless version of our proposed solution. During that phase, this module, along with additional modules for DL reasoning, non-ontological reengineering, rule management and execution, went by the collective name of Knowledge Representation and Reasoning System (KReS).

### 6.1.1 Relation to the software model

In Section 3.2.1 we had established the setting of our work from the standpoint of the software framework where our proposed method should be able to manage multiple ontology networks dynamically. The model of this software framework, called the *host*, had some features which we assumed to be dealing with. Let us now briefly review how these features are realized in Apache Stanbol.

- **shared knowledge base**: it is implemented in Stanbol as an Apache Clerezza backend. Clerezza is a software framework that can interoperate with a triple store of many possible types and present a uniform programming interface for performing CRUD operation on this triple store. The interface includes a single object, called *triple collection manager*, which allows multiple triple stores to register, each with its own *triple collection provider*. An *Apache Derby* database [Apab] is also present for storing content not mapped to RDF graphs.

- **Web Services**: these are implemented using *Jersey*, the reference implementation of RESTful Web Services in Java [Jer]. Stanbol implements a basic set of RESTful resource types in Jersey, which all the components can then specialize in order to wrap their APIs. In addition, Jersey endpoints are also used for providing these services as Web pages for human clients to use.

- **concurrency support**: it is guaranteed in Stanbol by the asynchronous implementation of RESTful services and multithreading support. Every component, including the underlying triple store, is responsible for maintaining transaction control where required.

- **locality**: it is natively implemented for some stateless components that perform in-memory processing without writing down their payload into Clerezza. A recent addition introduced user management and authentication support, but this feature still is not mapped to privileged user spaces.

### 6.1.2 Services

At the time of writing, Apache Stanbol provides a host of features that include the following:

- The **Enhancer** is a service register and executor that consumes generic content items and generates RDF triples for them. The nature of the RDF triples depends on which *Enhancement Engines* are registered with the enhancer, which pipeline has been chosen for enhancement and in which priority order the pipeline runs its engines. Enhancement engines exist for natural language processing, geographical Linked Data, sentiment analysis, language detection, RDF vocabulary translation and more. All the RDF triples are stored in a named graph maintained by the Apache Clerezza backend.

- The **Contenthub** is the content persistence component of documents submitted for Stanbol, which it is able to store, index using the *Apache Solr* search library [Apaf] and provide faceted search capabilities.

- A **SPARQL** endpoint for querying the graphs that selectively register themselves for querying, including the graph with all enhancement results managed by the Stanbol Enhancer and the Contenthub.

- The **Entityhub** is an optimized local cache for Linked Data as well as custom data, such as specialized topic thesauri, provided either in their entirety or as Apache Solr indices.

- The **Ontology Manager** is the facility entirely contributed by us. Besides the capabilities of the network manager (ONM) illustrated in this thesis, the Stanbol Ontology Manager also provides an ontology library management interface that extends the registry mechanism of the *ontologydesignpatterns.org* repository (cf. Section 2.4.3). It also has the ability to obtain ontologies by querying the Stanbol Entityhub or SPARQL endpoint.

- The **Rules** component is an environment for the definition, management and execution of inference rules. It has a custom language that can be adapted to SPARQL or SWRL depending on the required expressivity. In addition, its rule execution environment allows ontologies to be refactored in a stateless manner.
- The **Reasoner** module is a shared service interface for registering and invoking multiple DL reasoners. Reasoners with support for different logical fragments and OWL profiles can be registered, and each reasoner can declare its support for reasoning primitives such as consistency checking and classification.
- The **CMS Adapter** acts as a bridge between Stanbol content management systems and Stanbol. It can be used to map existing node structures from content repositories to RDF models or vice versa.

All the components listed above provide their services both through the OSGi declarative service model, and a RESTful API for Web Service clients.

## 6.2 Stanbol ONM

Although placed in specific layers from the architectural standpoint and with set dependencies on one another, the components of Apache Stanbol are established on a flat scheme service-wise. This means that there is no set service hierarchy, and any service exposed by one component does not *require* additional calls to sub-services exposed by other components. We can therefore discuss the component(s) of interest for this dissertation independently on others.

Technical documentation concerning the Stanbol ONM is available online [Apah] and in related technical reports [CSS+11, ABC+11, ABB+10].

### 6.2.1 Modules

Even from the perspective of the OSGi modular framework, the ONM is still a modular architecture and its modules, called *bundles*, are a subset of the bundles of the Stanbol Ontology Manager. A list of the Stanbol bundles of interest follows. For clarity, the bundle names have been shortened by removing the `org.apache.stanbol` prefix, therefore if a bundle name reads `bundlename`, it should actually be read as `org.apache.stanbol.bundlename`.

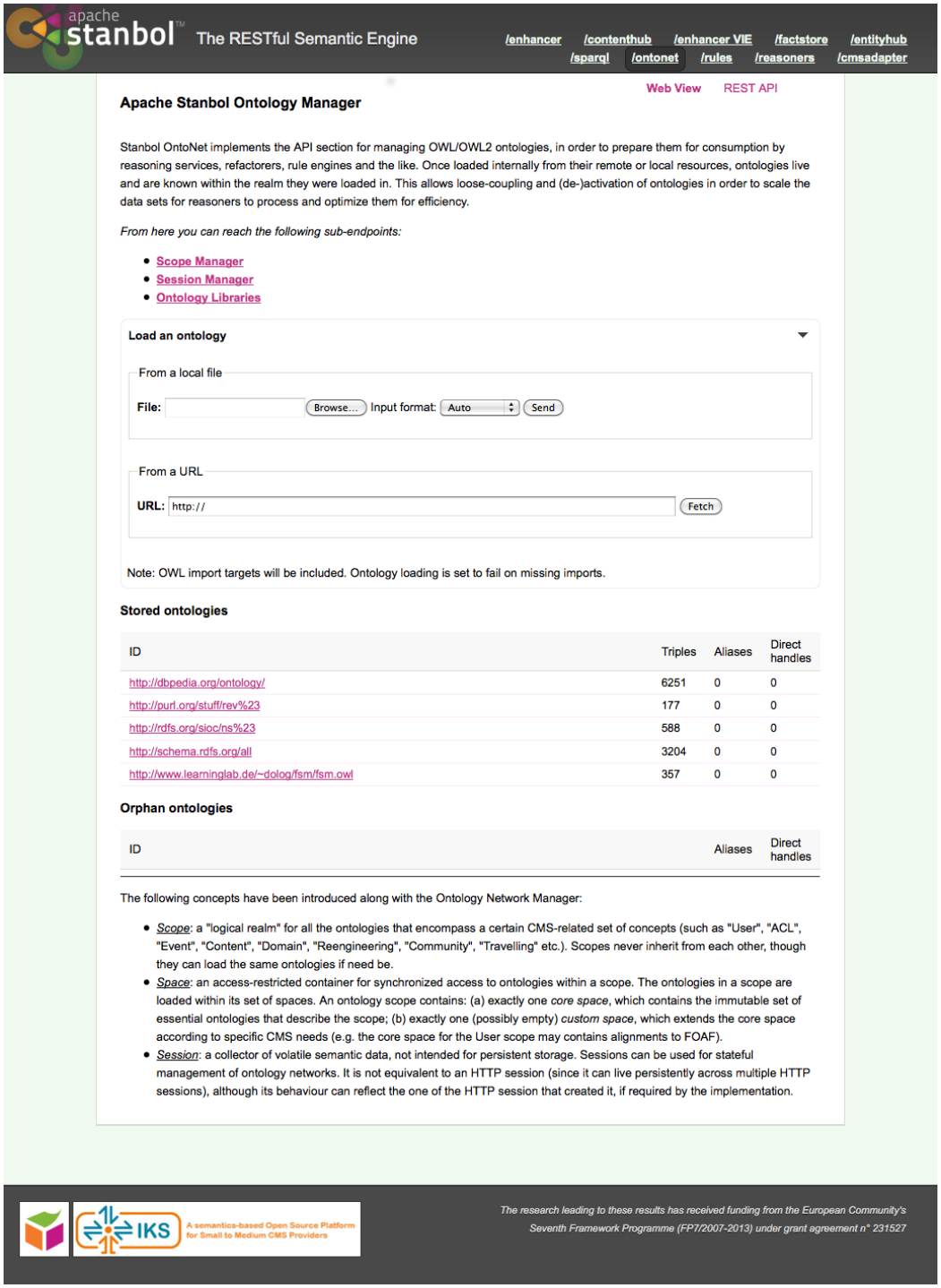


**Figure 6.2: Screenshot of the Apache Stanbol Ontology Network Manager** - Stanbol provides a Rich Internet Application for invoking its RESTful services from a Web browser. The ONM start page has features for loading ontologies from files, Web resources and ontology libraries. For each ontology entry, it shows the primary key and number of aliases and handles, and provides access to HTML managers for scopes and sessions.

- `commons.owl`: a set of utilities and primitives for managing OWL ontologies in general, and includes transformation primitives between the OWL API [OWLa] and Clerezza [Apaa] data models. This allows the ontological export of virtual ontology network components to occur as OWL API or Clerezza objects, no matter what specific implementation is used for the actual ONM.

- `ontologymanager.servicesapi`: the Java service interface of the virtual ontology network manager.

- `ontologymanager.core`: implements those interfaces of the `servicesapi` bundle that do not depend on a specific semantic backend.

- `ontologymanager.sources.clerezza`: implements Clerezza-based ontology input sources, i.e. classes that can be used by other OSGi components in order to feed native Clerezza objects to virtual ontology networks.

- `ontologymanager.sources.owlapi`: implements OWL API -based ontology input sources, i.e. classes that can be used by other OSGi components in order to feed native OWL API objects to virtual ontology networks.

- `ontologymanager.multiplexer.clerezza` this is the API binding bundle that implements the actual ONM in all the parts that depend on a specific storage backend. For our purpose, only the Clerezza binding is currently supported[1].

- `ontologymanager.registry`: contains both the API and the implementation of an ontology registry manager, which is able to maintain ontology libraries obtained by reading registry files. It extends the pattern registry model of *ontologydesignpatterns.org* repository (cf. Section 2.4.3).

- `ontologymanager.web`: the RESTful service wrapper for all the service interfaces of the `servicesapi` and `registry` bundles.

The actual ONM, i.e. the software responsible for assembling ontologies into virtual networks, is embedded in the following subset of Stanbol Ontology Manager bundles:

[1]The initial release of the Stanbol ONM was entirely based on an OWL API binding. However, this forced all virtual ontology networks to be persistent in memory, and further code was required in order to maintain some degree of synchrony with the underlying Clerezza persistence store.

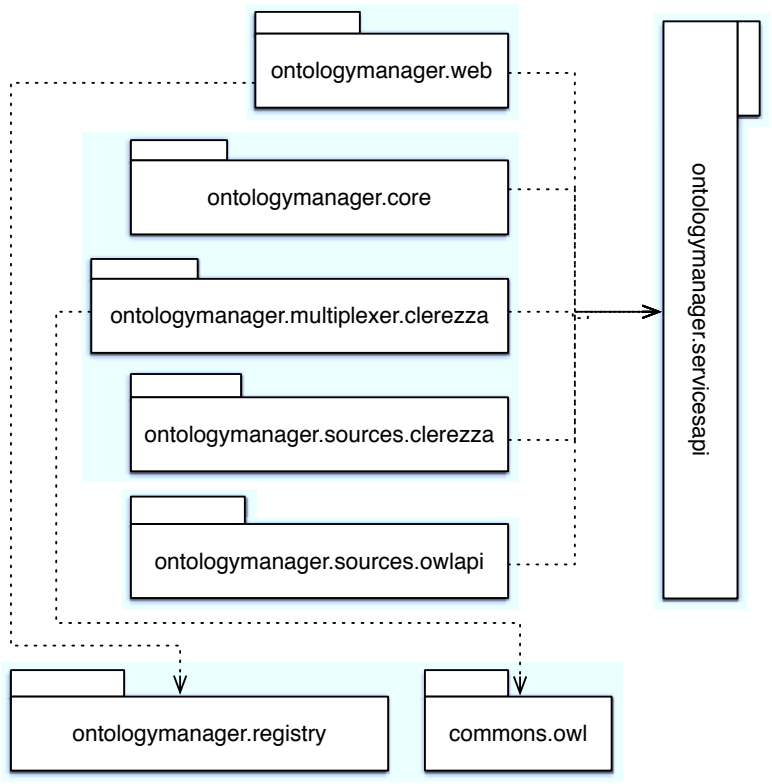


**Figure 6.3: Stanbol Ontology Manager bundle dependencies** - Since each bundle name matches the root package of all its classes, the UML package notation is used.

{`ontologymanager.servicesapi`, `ontologymanager.multiplexer.clerezza`, `ontologymanager.core`, `ontologymanager.web`}.

To represent dependency diagram across Stanbol Ontology Manager bundles, we have used a UML package diagram. This was possible with a slight abuse of notation, because the name of each bundle matches the name of the root package that contains all the classes in that bundle. The following remarks should be made on this structure:

1. `ontologymanager.web` depends on the two bundles that provide an API, but not on implementations, because the latter are resolved by the OSGi declarative service model at runtime.

2. The `ontologymanager.sources.clerezza` and `ontologymanager.sources.owlapi` bundles also register their service components and make their classes available at runtime, which is why no bundles depend on them (though if neither nor any other implementation were present in the OSGi environment, it would not be possible to activate components that require an ontology input source, including several Web Services).

### 6.2.2 Technology

From an engineering perspective, virtual ontology network management is comprised of several ontology, application and knowledge management functionalities, and is therefore subject to the availability of software tools and libraries that implement them and can perform their tasks. The technological stack of the Stanbol ONM is shown in Figure 6.4. We summarize these functionalities as follows, grouped by major areas:

- **User access**, i.e. the provision of an interactive (desktop, Web-based etc.) toolkit that allows human agents to manage ontology networks through a user interface.

- **Exposure** of ontology networks using standard formats and mechanisms, so that external client agents can consume them with no additional requirements on top of standard ontology language support.

- **Marshalling**, i.e. the combined functionalities of parsers and serializers able to read and write ontologies in canonical and non-canonical formats.

- **Retrieval** of ontologies distributed across multiple resources and types thereof, including Web and non-Web Internet protocols, local or distributed file systems, reengineered non-ontological resources, databases and triple/quad stores. Retrieval from heterogeneous sources should be seamless interface-wise.

- **Security** for privileged access to ontology networks and their components. Also cryptographic support for data exchange etc.

- **Multitenancy** for sharing ontologies across, and associating virtual ontology networks to, client accounts belonging to either users or applications.

- **Multiplexing** of shared ontologies across multiple virtual networks, i.e. the implementation of the logical model described in Chapter 4.

- **Deployment** of ontology-aware applications in the shared host platform; what ontology-related information and requirements should be published or removed upon activation or deactivation.

- **Persistence**, i.e. storage management. Functionalities in this group are required not only for performing CRUD operations on stored ontologies, but also for implementing statefulness and allowing ontology network configurations to be backed up and restored upon migrating or rebooting the host platform. Concurrency and transaction control functionalities also belong to this area.

Per Restriction R4 as described in Section 3.6, multitenancy and security transcended the scope of our work and were therefore not included in its reference implementation. Some persistence-related functionalities such as concurrency and transaction control are also delegated to the underlying storage implementation.

### 6.2.3 API implementations

Every application in the reference framework Apache Stanbol, i.e. every component providing an API for resource management, as opposed to simple software libraries, exposes its functionalities both via Java programming interfaces, and via Web Services [BHM+04].

The Stanbol ONM is no exception, and therefore provides two APIs:

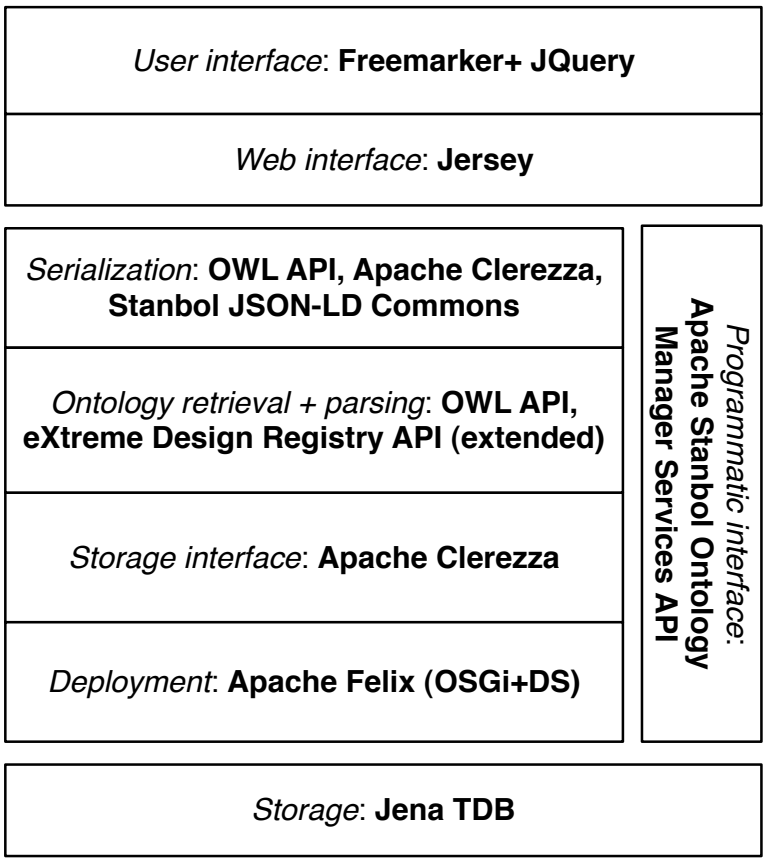


**Figure 6.4: Stanbol ONM technology stack (final iteration)** - Logical component names are indicated in italics, while the software libraries used to implement it are in boldface next to the component name. The programmatic interface allows an abstraction over all the internal component interfaces it encompasses.

- The Java object-oriented API implements the object model as defined in Section 5.3. The Java API can be accessed by including the Stanbol ONM packages and their dependencies in the Java classpath of the client application that needs to use the API, either manually by importing the corresponding set of bundle, or manually via the OSGi service component framework, by wrapping the client application as an OSGi component itself.

- The RESTful API was implemented in Java using the reference implementation of the JAX-RS specification for enabling RESTful Web Services in Java [Bur09]. The reference implementation of the JAX-RS specification is Jersey [Jer]. The RESTful API can be accessed by any client that handles requests and responses in the HTTP protocol. An HTML client application is also deployed by default, which allows users to interact with the service stack directly and manipulate virtual ontology network by using a Web browser (cf. Figure 6.2).

As far as ontology serialization formats are concerned, the RESTful API is able to consume and return ontologies (including images and ontological forms of collectors) in any of the nine formats listed in Section 2.2.5, i.e. RDF/XML, RDF/JSON, JSON-LD, N3, Turtle, N-Triples, OWL/XML, the OWL functional syntax and the Manchester OWL syntax.

## 6.3 Availability

As an integrating component of Apache Stanbol, the Ontology Manager source code and binary distributions are freely available and so are all the assets required to build from source. So are the ontology networking components going by the name of Stanbol ONM in the present work. All the Stanbol Ontology Manager components are available in source code form under the **Apache License 2.0** [Apad].

The Apache Stanbol project page [Apag] provides documentation, source code links and some binary releases of the whole Stanbol technology stack as well. Documentation on how to build the project from source is available on the same website.

Another source of the binary Stanbol Ontology Manager is the Interactive Knowledge Stack project page. The **Interactive Knowledge Stack** provides an alternative deployment of a semantic CMS support platform, which includes an Apache Stanbol

binary release. At the time of writing, the most recent release of this technology includes a version of the Stanbol Ontology Manager dating six months ahead of the most stable and feature-complete revision.

# 7

# Evaluation

We evaluated our combined logical and software approach by running a selection of tests and experiments on its reference implementation, i.e. the Stanbol ONM. As multiple aspects, both qualitative and quantitative, had to be evaluated, so was the evaluation method multifaceted. To assess which ontology network layouts generated by 3-tier multiplexing could guarantee highly expressive OWL interpretations, we devised a light theoretical framework based on graph theory. We then singled out the cases where very sparse axiom distributions could not be effectively laid out by the method. It was also necessary to give an assessment of the memory footprint of the ontology networks generated by this method. The resulting figures were used to verify not only whether the overhead introduced was negligible, but also whether less resource consumption could be guaranteed, compared to holding multiple in-memory ontology networks which share some ontologies.

## 7.1 OWL axiom interpretation

Unless a language-specific programming interface is used for supplying them to a client application, ontology networks need to be serialized to a consumable, presumably standard format and thereby exported as documents or, more precisely, Web resources. Some of these formats, described in Section 2.2.5, are not natively ontological, as is the case of RDF-based ones (RDF/XML, Turtle, JSON-LD, etc.). In such cases, the burden of parsing these documents and extracting knowledge from them naturally falls on client applications such as ontology editors, DL reasoners and indexing software.

Having already discussed and forewarned about the caveats and side effects of network structures on ontology interpretation in Chapter 3, we shall now proceed to evaluate how accommodating the network structures resulting from our solution proposal are, in this respect.

### 7.1.1 Method

Our validation work for this qualitative aspect of 3-tier multiplexing proceeded by the following steps:

1. Define a family of sets of ontology sources in a non-OWL format, whose raw statements are distributed according to certain criteria. These will be grouped into *distribution patterns*.

2. For every occurrence of such distribution patterns:

   (a) Determine what its optimal (i.e. most expressive, or whose axioms are of the highest possible order, cf. Section 4.1.1.2) OWL interpretation is.

   (b) Determine the way ontologies in the distribution pattern occurrence should be laid out in order to guarantee the optimal interpretation. These layout methods are grouped into *connectivity patterns*.

   (c) Verify whether an ontology network constructed through a naïve approach, i.e. a *trivial connectivity pattern*, can deliver the interpretation in (a) when fed to an OWL interpretation engine.

   (d) If not:

      i. Try to realize a layout according to the connectivity pattern in (b) using the Stanbol ONM, in order to guarantee (a) using the same OWL interpretation engine as in (c).

      ii. If (b) cannot be obtained, verify whether Stanbol ONM can realize a layout that is still more expressive than the naïve approach in (c).

3. Group distribution patterns in each (b) according to the possibility of obtaining them using the Stanbol ONM, so as to isolate *satisfiable* and *unsatisfiable* distribution patterns.

Distribution patterns reflect the ways axioms of different types can be distributed across multiple ontologies. Each resulting set of ontologies will be an occurrence of a distribution pattern. At least one example for each distribution pattern will be listed here, however, the test were performed on multiple occurrences, roughly *ten to twenty-five sets* each, which varied by the number of vertices (ontology sources) and entities used within each vertex. The core aspect to be tested for all distribution patterns, however, is **whether every usage in the ABox of a property that we expect to be an OWL object property, is interpreted as an OWL object property assertion**, instead of an annotation of data property assertion. Every other test is built upon this rationale.

The goal of step 2 is then to determine if it is possible to lay ontologies in a distribution pattern out by using 3-tier multiplexing, so that reading out these ontologies we can attain the highest expressivity possible (intuitively, the same level of expressivity reached by having all the statements in a single ontology). The possible, or rather, plausible ways to lay ontologies out in a network can themselves be conveniently grouped into classes, which we labeled connectivity patterns. Connectivity patterns will be enumerated and discussed in a dedicated section in this chapter. We can then summarily reformulate the goal of step 2 as: "for each distribution pattern, find at least one connectivity pattern that allows the resulting ontology network to be interpreted with the same expressivity as if it were a single ontology". We then determine whether any of the connectivity patterns found can be realized as a 3-tier multiplexing configuration in the Stanbol ONM. This proof is conducted by example. If none is found for at least one pattern occurrence, then the distribution pattern at hand is unsatisfiable.

As for the OWL interpretation engine mentioned in (2c-d), the reference engine was the OWL API 3 [OWLa], taken both as a standalone library for writing test programs, and as a core component of the Protégé 4[Pro] and NeOn Toolkit [NeOb] ontology editors. We chose this library because (i) it is a widespread and complete Java library with full support for the OWL 2 language features; (ii) it can parse RDF/XML, Turtle and Notation 3 ontology sources and interpret them in OWL; (iii) it has a visiting policy for ontology networks, that delivers different interpretations of the same ontologies if laid out differently[1].

[1]Per Assumption A9, we assumed not to know about this characteristic of the OWL API 3 prior to employing it, hence we verified it empirically.

We have set no limits on the expressivity to be attained by interpreting ontologies, other than assuming to stay within the normative profiles of OWL 2 listed in Section 2.2.4 [MGH+09]. Per Hypothesis H2 (cf. Section 3.5), our goal is to simply determine whether our method allows a natural way to lay out some ontology networks in order to obtain better results that the naïve approach. We use the naïve approach as a reference in accordance with Assumption A10 (cf. Section 3.4), which states that ontology consumers do not necessarily know how ontology networks should be built, let alone that their layout is logically relevant. Therefore, they will expectedly try the naïve approach in absence of a dedicated framework. Also, due to Assumption A3, they are typically not allowed to modify the original ontology sources, so network assembly occurs in an environment they can control, and that environment is the Stanbol ONM.

#### 7.1.1.1 Materials

We quickly review the technological setting in which we performed our experiments:

- **Hardware**: 2.3 GHz Intel Core i5, 8 GB DDR3 1,333 MHz
- **Operating System**: Mac OS 10.6.8, Ubuntu 12.04 on Linux kernel 3.2 (virtualized on Oracle VM VirtualBox), both systems for x86_64 architectures.
- **Runtime environment**: Java 1.6.0 (JDK), Hotspot Virtual Machine on both operating systems, 4GB heap space.
- **Testing platform version**: Apache Stanbol 0.10, built from source snapshot as of September 30, 2012.
- **OWL interpretation engine**: OWL API versions 3.2.x (both in Protégé 4.2 and standalone as used in plain Java testing mini-applications) and 3.1.0 (NeOn Toolkit 2.5).

### 7.1.2 Connectivity patterns

A **connectivity pattern** is a scheme for constructing connected graphs[1]. These graphs are built from the connectivity pattern by replication of vertices and arcs. Resulting

[1]The term "connectivity pattern" is of novel usage in this context, being lifted from an unrelated concept in the neuroscience domain [KKMM11]

graphs, in turn, may or may not be trees, that is, they may contain at least one cycle or none, respectively. In the context of this work, connectivity patterns are ways to connect networked ontologies with one another in order to assemble real or virtual ontology networks.

In general, graph patterns can be regarded as classes of graphs conveniently grouped by a given property, or a specific way to construct other graphs in the class starting from a base one, e.g. by replication of certain vertices and arcs. The following theoretical discussion refers to the fundamentals of graph theory [CZ04].

**Definition (connectivity pattern and entry node).** A *connectivity pattern* is a class of vertex-wise traversable digraphs. The starting vertex of one walk that traverses vertex-wise a digraph in each class is called the *entry node* for the connectivity pattern.

A digraph is a directed graph. "Vertex-wise traversable" means that there is at least one path, or *circuit*, that visits every vertex in the digraph, though not necessarily by passing though every arc. The relations and criteria by which digraphs are grouped into classes will be cleared in the remainder of this section for each connectivity pattern. For the general literature on graph classes we refer to Brandstädt et al. [BLS99]. A digraph that is a member of a class defined as a connectivity pattern is said to *instantiate* that pattern.

The above definition has a number of implications to be taken into account. To begin with, since all digraphs in a connectivity pattern are traversable vertex-wise, then they are necessarily connected. Recall from our definition of ontology network (cf. Section 4.1.2), that ontology networks are mappable to connected graphs via the chosen connectivity relation. As the requirements for a digraph to be traversable are out of the scope of this work, we refer to the literature on digraph traversibility [BL98] for details.

In addition, connectivity patterns do not necessarily construct trees, since cycles are generally allowed. Note, however, that for the sake of simplicity we shall make the reasonable assumption that loops, i.e. cycles from a vertex into itself (or, in OWL parlance, an ontology importing itself), have no impact on the interpretation of ontologies and can therefore be disregarded.

Finally, note that different selections of entry nodes imply, in general, different connectivity patterns. However, there can be cases, as are those of ring patterns, where they can reduced to a single pattern by isomorphism.

In the following, we will make use of some fundamental notions of graph theory, such as Hamiltonian and Eulerian graphs [CZ04]. To that end, we introduce the following corollary to the definition given earlier.

**Corollary 1.** Hamiltonian or Eulerian digraphs instantiate connectivity patterns.

*Proof.* Both Hamiltonian and Eulerian digraphs are traversable by their very definitions. Specifically, if a digraph G is Hamiltonian, then it has at least one cycle that contains every vertex in G. Let the entry node of G be any vertex of such a cycle. Likewise, if G is Eulerian, then it contains at least one circuit that passes through every vertex in G exactly once. Let the entry node of G be the first vertex of such a circuit. □

#### 7.1.2.1 Trivial connectivity pattern: flat with auxiliary root

The *trivial connectivity pattern* will be the reference scheme for ontology networks in this evaluation, i.e. for every class of ontology network that can be realized with 3-tier multiplexing, we will check whether it performs better than a network realized with the same ontologies using a flat connectivity pattern. In other words, it represents the naïve approach to the construction of ontology networks.

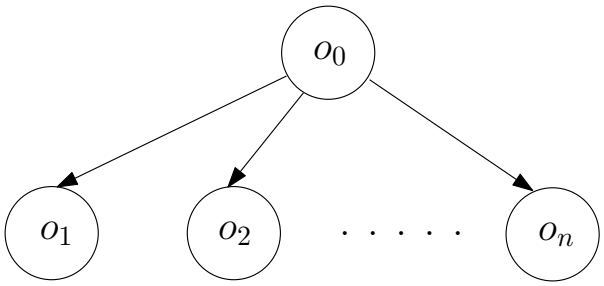


**Figure 7.1: Flat connectivity pattern for $n$ ontologies.** - Root ontology $o_0$ is introduced for importing the other two without modifying them permanently.

The flat connectivity pattern is illustrated in Figure 7.1. Vertices $\{o_1, o_2, \ldots o_n\}$ denote networked ontologies, while $o_0$ is an ontology devoid of logical axioms, such that a dependency between $o_0$ and each of $\{o_1, o_2, \ldots o_n\}$ holds.

The auxiliary root vertex $o_0$ is required in order to have an entry node. It is labelled auxiliary, in that it does not correspond to any ontology from the knowledge base of interest $\{o_1, o_2, \ldots o_n\}$, but its role is simply to connect them. In sheer topological terms, the root could also be chosen from that knowledge base. However, if the selected vertex denoted an ontology containing logical axioms, this would introduce a bias influenced by which ontology is selected: the connectivity pattern could no longer be trivial nor serve as the basis for the required naïve approach.

#### 7.1.2.2 Ring

A *ring connectivity pattern* is a class of the simplest Hamiltonian digraphs.

Given $n$ vertices, $n \geq 2$, any Hamiltonian digraph with no loops[1], such that:

- for each vertex $o_i,\ i = 1..n\ \text{indeg}(o_i) \leq 2$ and $\text{indeg}(o_i) \leq 2$, and
- for each $i$ and $j$ if there is an arc $(o_i, o_j)$ and $\text{outdeg}(j) = 2$, then $(o_j, o_i)$ also exists.

belongs to the ring connectivity pattern. In addition, the *ring* connectivity pattern (see Figure 7.2) comprises all and only the digraphs in the ring pattern, such that for each $o_i,\ i = 1..n, \text{indeg}(o_i) = \text{outdeg}(o_i) = 1$. Recall that $\text{indeg}(v)$ and $\text{outdeg}(v)$ are the number of inbound and outbound arcs of $v$, respectively. In other words, the one-way ring pattern is the class of all the Eulerian digraphs in the *tout-court* ring pattern.

If the connectivity relation is an OWL import relation, then ontology networks constructed on a ring pattern are generally an overkill, since in our experimental setting the axioms in an ontology are only interpreted on its first visit and not re-interpreted further. For this reason, we can disregard non-Hamiltonian cycles and consider one-way rings exclusively.

#### 7.1.2.3 Broken ring

A *broken ring connectivity pattern* is a non-Hamiltonian variant of the ring pattern.

Given $n$ vertices, $n \geq 2$, any digraph with no loops, such that:

- for each vertex $o_i,\ i = 1..n\ \text{indeg}(o_i) \leq 2$ and $\text{indeg}(o_i) \leq 2$, and

[1] Recall that a *loop* is a cycle from one vertex into itself.

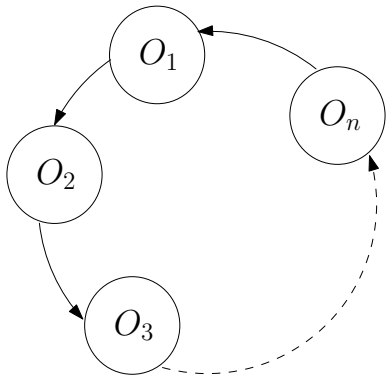


**Figure 7.2: (One-way) ring connectivity pattern** - All digraphs in this pattern contain a cycle that encompasses every vertex and every vertex has exactly one inbound arc and one outbound arc. The dashed arrow indicates the connection of the remaining vertices $o_4..o_{n-1}$.

- for *exactly two pairs* of vertices $i$ and $j$, there are only arcs of type $(o_i, o_j)$ or $(o_j, o_i)$, and at least one of either type exists.

In general, not all digraphs belonging to this pattern are traversable. A traversable variant of the broken ring is the *one-way broken ring* pattern, which is illustrated in Figure 7.3. In the one-way variant, $\text{indeg}(o_i) = \text{outdeg}(o_i) = 1$ for all the vertices, except for one where $\text{outdeg}(o) = 1, \text{indeg}(o) = 0$ and another one where $\text{outdeg}(o') = 0, \text{indeg}(o') = 1$, and $o \neq o'$.

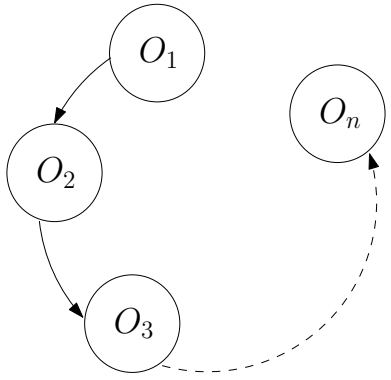


**Figure 7.3: (One-way) broken ring connectivity pattern** - The dashed arrow indicates the connection of the remaining vertices $o_4..o_{n-1}$.

#### 7.1.2.4 Multipartite

The *multipartite* connectivity pattern is one of the ways several static ontology networks are assembled by design. It is found, for example, in many ontology design patterns submitted to the *ODP.org* repository (cf. Section 2.4.3)[1].

Any *acyclic* digraph for whose vertices a *partition* exists, so that:

- if there is an outbound arc from any vertex in partition $i$, then that arc goes necessarily into a vertex of partition $j$ where $i \neq j$;
- there is a single vertex $o_0$ with no inbound arcs, and that vertex is the only member of partition 0;
- one of the following holds:
  - every arc belongs to some path of length $k$, where $k+1$ is the number of partitions[2] (*fixed path length*);
  - every arc belongs to some path of length $l, 1 \leq l \leq k$, where $k+1$ is the number of partitions (*variable path length*);

belongs to the *multipartite* connectivity pattern. If the digraph has $p$ partitions, then it is called *p-partite*, and so is the connectivity pattern it is built upon.

Figure 7.4 shows an example of a $k+1$-partite connectivity pattern with fixed path length. Clearly, $o_0$ is the entry point of this connectivity pattern. Note that all partitions can be ordered so that all arcs are from vertices in one partition to vertices in the partition "below". There are no arcs upwards, otherwise the digraph would have cycles. Also note that, as with multipartite graphs (non-directed), there are never arcs between two vertices in the same partition, as such a type of connection cannot be realized with the multiplexing technique described in this thesis.

Figure 7.5 shows an example of a $k+1$-partite connectivity pattern with variable path length. In the figure, vertices $o_{1,n}$ and $o_{2,2}$ have no outbound arcs, so paths of length 1 and 2 are shown.

[1] This holds if we rule out the content pattern annotation schema [Con], which is an ontology imported by every content pattern, but comprised of annotation property definitions only, therefore it can be disregarded from the DL point of view

[2] Recall that the length of a path is the number of *arcs* it has, counting repeated arcs as many times as they appear.

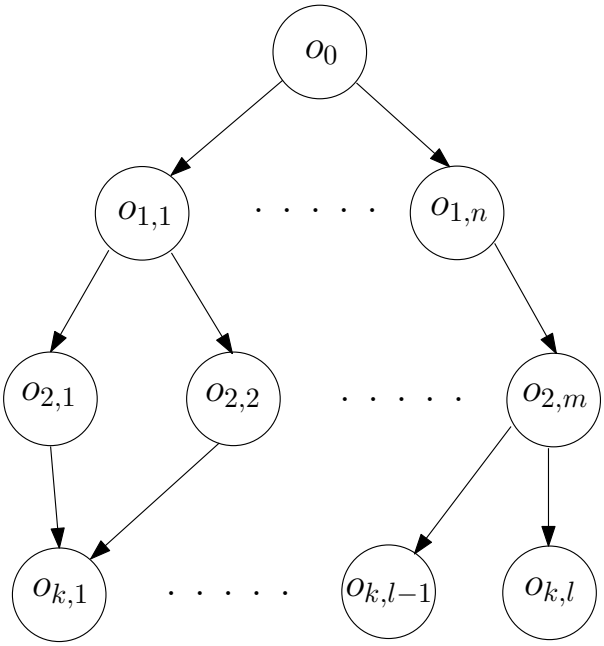


**Figure 7.4: Multipartite connectivity pattern with fixed path length.** - The figure depicts a $k+1$-partite connectivity pattern. Arcs only exist from vertices in a partition to vertices in a single other partition. All paths are of length $k$.

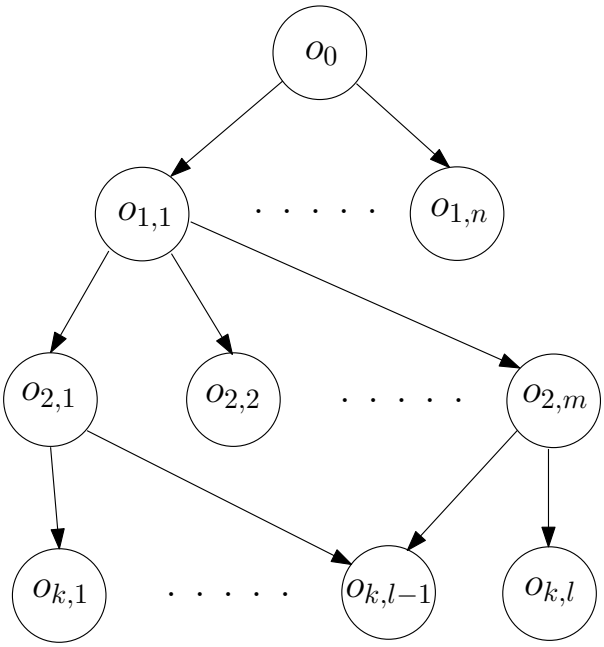


**Figure 7.5: Multipartite connectivity pattern with variable path length.** - The figure depicts a $k+1$-partite connectivity pattern. All paths are of length up to $k$.

#### 7.1.2.5 Tight replication with root cycling

The following connectivity pattern is a peculiar class of connected digraphs, in that it defines one of the structures that can be implemented by the Stanbol ONM.

Before introducing this pattern, it is useful to recollect the notion of *polytree* from graph theory [RP88]. A polytree is a digraph graph with exactly one undirected path between any two vertices. Note that a so-defined digraph is also acyclic, therefore it is also called a directed acyclic graph (DAG) whose underlying undirected graph is connected. Note that every graph built using the multipartite connectivity pattern is a polytree, but not every polytree can be built that way, as polytrees in general can have multiple roots.

Any connected digraph $G$ with no loops so that:

1. for every vertex $o_{i,j}$ but one (called *root*, $o_0$), $\text{indeg}(o_{i,j}) = 1$;
2. $\text{indeg}(o_0) > 1$;
3. the union of the shortest paths from $o_0$ to every other vertex forms a polytree $T$;
4. with the exception of the descendants of *one* child of $o_0$, for any leaf $o$ of $T$ there is an arc $(o, o_0)$ in $G$;
5. no other arcs than those in $T$ and those defined in (4) exist in $G$;

belongs to the *tight replication connectivity pattern with root cycling*, with $o_0$ as the entry node to the pattern.

A less formal definition can be given by construction as follows. Let $T$ be a polytree with only one source, and let that source be $o_0$. For every leaf but the descendants of one single child of $o_0$, add an arc from the leaf to the root. The digraph $G$ so obtained belongs to the connectivity pattern.

Figure 7.6 shows a construction scheme for members of this connectivity pattern. Root $o_0$ has $n$ children $\{o_{1,1}...o_{n,1}\}$ Each child has a (possibly empty) subtree, indicated by the dashed arrows and the dotted lines between their arcs. For each leaf in every subtree but one, there is an arc from that leaf to $o_0$. No other arcs exist.

Note that, because of the designated vertex in (4), which we shall call $o_{n,1}$, *none* of the graphs in this pattern is Hamiltonian, because there should be at least one

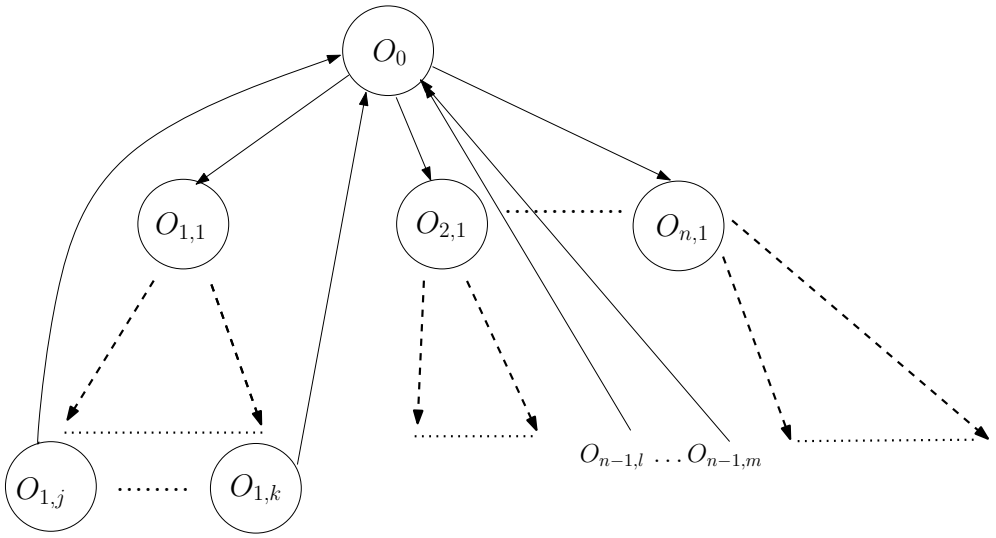


**Figure 7.6: Tight replication connectivity pattern with cycling on root vertex** - All arcs from a designated root $O_{0,1}$ recursively form a descending tree. However, for every leaf that is not a descendant of the last child of the root $O_{n,1}$, an arc from that leaf to $O_{0,1}$ is added.

Hamiltonian cycle passing through $o_{n,1}$ and its children. However, no such cycle exists as it is not possible to return to the root $o_0$ once $o_{n,1}$ is encountered. Also note that there is a Hamiltonian *path* (non-cycle) iff $o_{n,1}$ is a leaf.

#### 7.1.2.6 Tight replication with intermediate layer cycling

This is a variant of the tight replication connectivity pattern with root cycling. It is still used to realize cyclic graphs, but unlike the root cycling variant, all the cycles in these graphs no longer involve a single vertex necessarily. Once again, this connectivity pattern can be implemented using 3-tier multiplexing, and by extension the Stanbol ONM. Let us now formulate the definition of this connectivity pattern by means of the previous one.

Any connected digraph $G$ with no loops so that:

1. for every vertex $o_{i,j}$ but one (called *root*, $o_0$), $\text{indeg}(o_{i,j}) = 1$;
2. $\text{indeg}(o_0) = 0$;
3. the union of the shortest paths from $o_0$ to every other vertex forms a polytree $T$;

4. with the exception of the descendants of *one* child of $o_0$, for *some* leaf $o_{n,m}$ of $T$ there is an arc $(o_{n,m}, o_{k,m+1})$ in $G$, with $0 < k < n$ ;

5. no other arcs than those in $T$ and those defined in (4) exist in $G$;

belongs to the *tight replication connectivity pattern with intermediate layer cycling.*

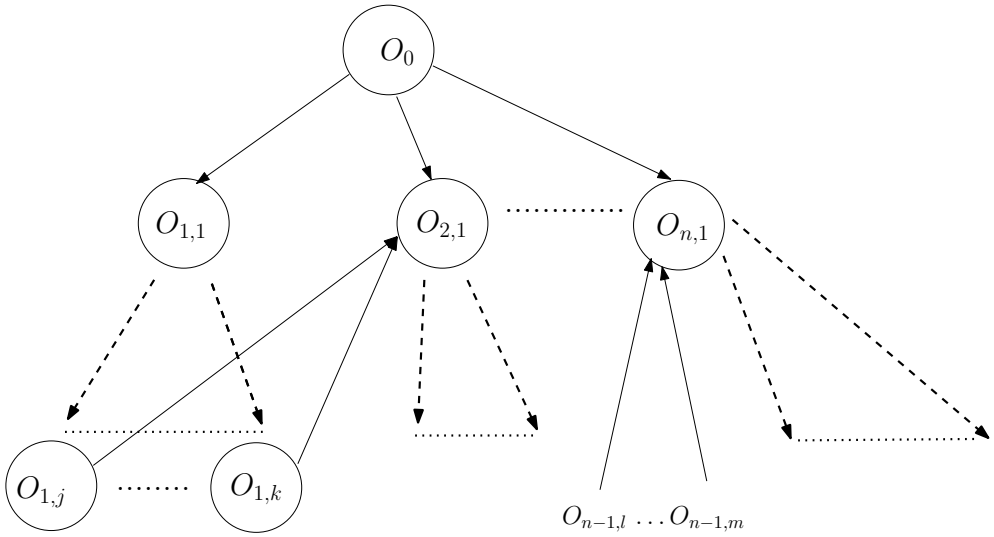


**Figure 7.7: Tight replication connectivity pattern with intermediate layer cycling** - Exactly one root $o_0$ has no inbound arcs. The shortest paths from $o_0$ form a polytree. However, for every leaf that is not a descendant of the last child of the root $o_{n,1}$, there is an arc from that leaf to an intermediate vertex.

#### 7.1.2.7 Rooted polytree, or loose replication

The *rooted polytree* connectivity pattern can be regarded as an acyclic variant of the tight replication pattern, i.e. one where there are no outbound arcs in correspondence of the leaves of the spanning tree. For this reason, we shall also call it *loose replication* connectivity pattern.

Recall that a polytree is a directed graph with exactly one undirected path between any two vertices. Thus, every polytree with exactly one source (the *root*) belongs to the *rooted polytree* connectivity pattern. This connectivity pattern is also distinguished from the multipartite pattern, since the former allows vertices whose inbound degree is greater than 1, whereas this one requires all of them to have inbound degree 1 except for the root, which has zero.

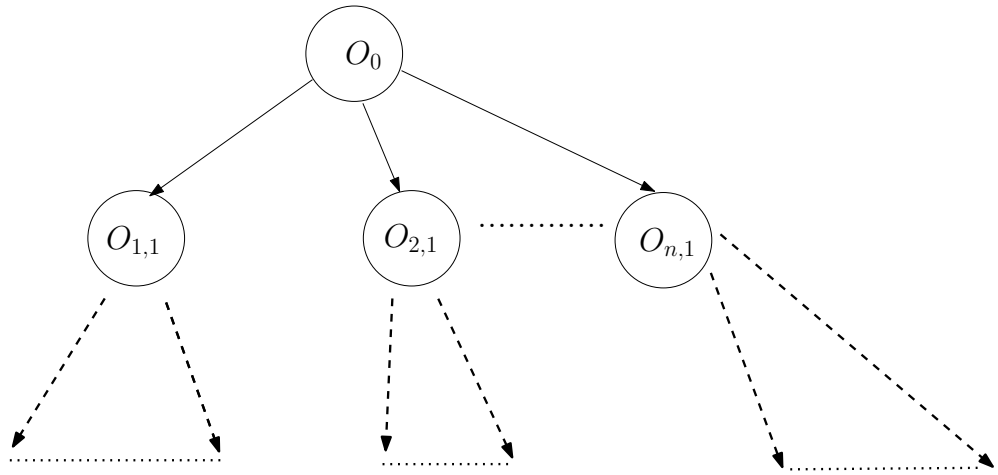


**Figure 7.8: Rooted polytree connectivity pattern** - For each leaf there is a (directed) path from a designated root vertex $o_0$ to that leaf. No arcs exists that are not part of one such path.

The construction scheme of digraphs in the rooted polytree pattern is shown in Figure 7.8. $o_0$ is the single source of the polytree. Its children, $\{o_{1,1}, ..., o_{n,1}\}$ in this scenario, can have any arbitrary subtrees, and no cycles are added on top of them. Ontology networks using this connectivity pattern can be implemented with the Stanbol ONM. Due to their desirable properties such as the lack of cycles, if an ontology network built using this scheme can be interpreted in order to deliver highly expressible axioms, then the original set of ontologies is informally said to be network-friendly.

### 7.1.3 Distribution patterns

Having described the essential connectivity patterns that shall be used in our demonstrations, let us not proceed to describe *distribution patterns*. Distribution patterns encode the possible ways *axioms* are distributed across multiple ontologies. If the ontologies are not represented natively in an ontology language, then distribution patterns are ways to distribute *raw statements* across multiple ontology sources. We have singled out some distribution patterns, and the goal is to determine whether for these distribution patterns there is an optimal ontology network layout that can be realized using 3-tier multiplexing, and by extension the Stanbol ONM.

In Section 4.1.2 we enunciated the difference between *connectivity* and *dependency* relations between ontologies in an ontology network. Typically, the example ontology

sets we used for representing distribution patterns are connected, but not dependent on one another. The rationale is to use dependency relations that are able to reflect the existing connectivity relations, as the latter we cannot modify due to having restricted ourselves to not using ontology modularization (cf. Restriction R5 in Section 3.6). As a matter of fact, to *realize* a connectivity pattern means to implement dependencies between ontologies, by using the same structure as the one that emerges from their intrinsic connectivity.

In the following, these prefixes, expressed in Turtle syntax, will be assumed to hold:

```
owl: <http://www.w3.org/2002/07/owl#> .
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
: <http://www.cs.unibo.it/~adamou/ontologies/patterns/entities#> .
```

We shall now describe, with some examples, the distribution patterns that have been singled out for analysis.

#### 7.1.3.1 TA-simple

This is the **simple TBox-ABox (TA)** distribution pattern. In it, the ontology set consists of two ontologies $\{o_1^T, o_1^A\}$, where $o_1^T$ and $o_1^A$ contain exclusively TBox and ABox axioms, respectively. This is a recurrent, albeit stringent, distribution pattern that is often instantiated in Linked Data. An example is to be found in the DBPedia {ontology, dataset} pair [DBP], but the pattern actually appears in any Linked Data set that does not export any terminological axioms associated with its data.

**Example occurrence - Set 1**

$o_1^T$ contains a single object property declaration:

```
:op1 a owl:ObjectProperty .
```

$o_1^A$ contains a single statement using the object property asserted $o_1^T$ as a predicate. Note that we are not in the position to call it an object property assertion at this stage.

```
:in1 :op :in2
```

The desired interpretation on OWL, i.e. the one that would be obtained of all the statements were in the same ontology, would be (in OWL functional syntax):

```
Declaration(ObjectProperty(:op1))
ObjectPropertyAssertion(:op1 :in1 :in2)
```

However, the naïve approach delivered the following:

```
Declaration(ObjectProperty(:op1))
AnnotationAssertion(:op1 :in1 :in2)
```

where `in2` is treated as an IRI but *not* as an OWL named individual, as IRIs that do not represent entities are still legal OWL annotation values [MPSP+09].

**Example occurrence - Set 2**

In this second example, the object property declaration is not explicit, but it should be inferred from property usage in a restriction applied to a class (which is never defined as such, but used in another ontology as a type). $o_1^T$ declares a universal restriction on property `op2` and entity `c1`, and applies it to a class `c2`.

```
:c2
  rdfs:subClassOf [ rdf:type owl:Restriction ;
    owl:onProperty :op2 ;
    owl:allValuesFrom :c1
  ] .
```

$o_1^A$ applies property `op2` between an instance `inc2` of class `c2` and another entity `inc1`.

```
:inc2 a :c2 ;
  :op1 :inc1 .
```

Again, the optimal interpretation would be:

```
ClassAssertion(:c2 :inc2)
SubClassOf(:c2 ObjectAllValuesFrom(:op2 :c1))
ObjectPropertyAssertion(:op1 :inc2 :inc1)
```

However, the naïve interpretation was instead:

```
ClassAssertion(:c2 :inc2)
SubClassOf(:c2 ObjectAllValuesFrom(:op2 :c1))
AnnotationAssertion(:op1 :inc2 :inc1)
```

Note that the universal restriction was correctly detected as an application of an object property, but its usage in the subsequent assertion did not respect the binding on the property type for `op1`.

#### 7.1.3.2 T-split

In a *T-split* distribution pattern, only the statements that are expected to be interpreted as TBox axioms are considered. A set of two or more ontologies whose aforementioned statements *(i)* are distributed across them; and *(ii)* are not repeated in any ontology in the set, are said to follow a T-split pattern.

**Example occurrence**

$o_{T_1}$ contains n object property declaration property declaration:

```
:op1 a owl:ObjectProperty .
```

$o_{T_2}$ declares a subproperty of the property declared in $o_{T_1}$.

```
:op2 rdfs:subPropertyOf :op1 .
```

$o_{T_3}$ uses the property declared in $o_{T_2}$ in a universal restriction.

```
:c2 rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty :op2 ;
  owl:allValuesFrom :c1
] .
```

The optimal interpretation of the above would be:

```
Declaration(ObjectProperty(:op1))
SubObjectPropertyOf(:op2 :op1)
SubClassOf(:c2 ObjectAllValuesFrom(:op2 :c1))
```

However, the naïve interpretation was instead:

```
Declaration(ObjectProperty(:op1))
SubAnnotationPropertyOf(:op2 :op1)
SubClassOf(:c2 ObjectAllValuesFrom(:op2 :c1))
```

According to the result of the naïve approach, `op1` is an object property because it is declared to be, and `op2` is also used as an object property in the anonymous superclass defined for `c2`. However, subsumption between the two properties is detected at the annotation property level, whereas `op1` and `op2` remain as sibling properties at the object property level. This means that, if we were to add the following statement somewhere:

```
:in1 :op2 :in2 .
```

there is a chance a DL reasoner might *not* be able to produce the following expected inference for the super property (which the HermiT reasoner [Her], for one, did not when we conducted out experiment):

```
ObjectPropertyAssertion(:op1 :in1 :in2)
```

#### 7.1.3.3 TA-retroactive

A set of two or more ontologies follows the *TA-retroactive* distribution pattern if some ontology contains TBox axioms and at least one ontology applies ABox axioms, whose corresponding individuals should influence the interpretation of the ontologies that contain TBox axioms.

### Example occurrence

$o_1$ contains a single RDFS subsumption statement between two undeclared properties.

```
:op1 rdfs:subPropertyOf :op0 .
```

$o_2$ declares two named individuals `:in1` and `:in2`, and establishes all possible predicates using subproperty `:op1`. In addition, it assigns a property range to super-property `:op0` over a nominal class made up by enumerating the newly declared individuals.

```
:op0 rdfs:range [ rdf:type owl:Class ; owl:oneOf ( :in1 :in2 ) ] .

:in1 a owl:Thing ;
  :op1 :in2 .

:in2 a owl:Thing ;
  :op1 :in1 .
```

Since property range axioms over OWL classes are only allowed for object properties, and due to the punning restriction on property types in OWL 2 [GWPS09], it is expected that both `:op0` and `:op1` be interpreted as object properties, and their uses in ABox axioms as object property assertions. Thus the optimal interpretation would be:

```
SubObjectPropertyOf(:op1 :op0)
ObjectPropertyRange(:op0 ObjectOneOf(:in2 :in1))
ClassAssertion(owl:Thing :in1)
ObjectPropertyAssertion(:op1 :in1 :in2)
ClassAssertion(owl:Thing :in2)
ObjectPropertyAssertion(:op1 :in2 :in1)
```

However, the naïve interpretation was instead:

```
SubAnnotationPropertyOf(:op1 :op0)
ObjectPropertyRange(:op0 ObjectOneOf(:in2 :in1))
ClassAssertion(owl:Thing :in1)
```

```
AnnotationAssertion(:op1 :in1 :in2)
ClassAssertion(owl:Thing :in2)
AnnotationAssertion(:op1 :in2 :in1)
```

which means that the object property range axiom in the interpreted $o_2$ did not propagate the type of `op1` to the one axiom in in $o_1$, and then back to the usage of the subproperty in $o_2$ itself.

#### 7.1.3.4 A-split

In the *A-split* distribution pattern, the TBox and ABox axioms are distributed across two or more ontologies, so that at least two ontologies contain some ABox axiom. Occurrences of this distribution pattern can be expected whenever there is a need to generate an ontology network that combines multiple portions of linked data, especially if they come from different datasets that are nevertheless built on top of the same vocabulary. An example are the various BBC media datasets which make use of the *DBTune* vocabulary [DBT].

Note that a set of ontologies can follow both the T-split and the A-split distribution patterns at the same time.

**Example occurrence 1**

Let $o_1$ be an ontology[1] that declares an OWL object property and uses it across two entities `in1` and `in2`. Clearly, we expect both `in1` and `in2` to be interpreted as OWL individuals, ultimately.

```
:op1 a owl:ObjectProperty .

:in1 :op1 :in2 .
```

Let now $o_2$ be an ontology that declares a further named individual `:in3`, this time explicitly. $o_2$ also relates `:in3` with another entity, incidentally the object of an object property assertion in $o_1$. The property used is a new subproperty `:op2` of property `:op1` explicitly declared in $o_1$.

[1]The notation using integers as subscripts instead of $A$ and $T$ indicates that the ontologies at hand are no longer exclusively comprised of ABox axioms or TBox axioms.

```
:op2 rdfs:subPropertyOf :op1 .

:in3 a owl:Thing ;
  :op2 :in2 .
```

The optimal interpretation would be:

```
Declaration(ObjectProperty(:op1))
SubObjectPropertyOf(:op2 :op1)
ClassAssertion(owl:Thing :in3)
ObjectPropertyAssertion(:op1 :in1 :in2)
ObjectPropertyAssertion(:op2 :in3 :in2)
```

The naïve interpretation was instead:

```
Declaration(ObjectProperty(:op1))
SubAnnotationPropertyOf(:op2 :op1)
ClassAssertion(owl:Thing :in3)
ObjectPropertyAssertion(:op1 :in1 :in2)
AnnotationAssertion(:op2 :in3 :in2)
```

**Example occurrence 2**

In this second example, two individuals are declared in separate ABox ontologies, and a property assertion uses them in a third ontology.

Ontology $o_{A_1}$ declares named individual `in1`.

```
:in1 a owl:Thing .
```

Ontology $o_{A_2}$ declares named individual `in2`.

```
:in2 a owl:Thing .
```

Finally, ontology $o_{A_3}$ applies an undeclared property `op1` to both individuals.

```
:in1 :op1 :in2 .
```

This particular occurrence does not pose a problem that our solution addresses, because the interpretation results proved to be unrelated to the layout of ontology networks. More precisely, when we merged all the three statements into a single ontology in order to derive the optimal interpretation, the result was as follows:

```
ClassAssertion(owl:Thing :in1)
ClassAssertion(owl:Thing :in2)
AnnotationAssertion(:op1 :in1 :in2)
```

where the property assertion for `op1` was interpreted as an annotation assertion, rather than an object property assertion as we expected, due to holding between two individuals. This was probably a conservative strategy adopted by the OWL API, whether a resource is treated as an IRI whenever it is not explicitly stated to be treated as an entity. We have not found documentation on whether the dual interpretation as an IRI and an OWL individual would violate OWL 2 punning or not [GWPS09, MPSP+09].

#### 7.1.3.5 *mn*-scatter

The ***mn*-scatter** distribution pattern is the generalization of split patterns where ABox and TBox axioms are distributed as sparsely as it makes sense for them to be, i.e. so that each entity appears at most once in every ontology i.e. they are *scattered*, hence the name. Variables $m$ and $n$ denote the number of ontologies across which the TBox and ABox are distributed, respectively.

**Example occurrence.**

The following is an example of a 22-scatter pattern occurrence, i.e. there are 2 ontologies that only contain TBox axioms and 2 ontologies that only contain ABox axioms.

$o_T^1$ declares a simple OWL object property.

```
:op1 a owl:ObjectProperty op1.
```

$o_T^2$ states a subproperty TBox axiom involving property `op1` which was typed in $o_T^1$.

```
:op2 rdfs:subPropertyOf :op1 .
```

$o_A^1$ contains a single statement that involves `op1`, and that should therefore be interpreted as an ABox axiom.

```
:in1 :op1 :in2 .
```

$o_A^2$ explicitly declares a new individual `in3` and adds a property axiom for it. If the other ontologies were to be interpreted first, that property axiom should be interpreted as an object property assertion, and therefore an ABox axiom.

```
:in3 a owl:Thing ;
  :op2 :in2 .
```

As with previous examples, none of the above ontologies depend on one another. However, they are connected, as the following connectivity pairs are formed due to the usage of properties and individuals: $c(o_A^1, o_A^2), c(o_A^1, o_T^2), c(o_A^1, o_T^1), c(o_A^2, o_T^2), c(o_A^2, o_A^1)$, along with their inverses.

The expected optimal interpretation is as follows:

```
Declaration(ObjectProperty(:op1))
SubObjectPropertyOf(:op2 :op1)

ClassAssertion(owl:Thing :in3)
ObjectPropertyAssertion(:op1 :in1 :in2)
ObjectPropertyAssertion(:op2 :in3 :in2)
```

However, the naïve interpretation produced the following:

```
Declaration(ObjectProperty(:op1))
SubAnnotationPropertyOf(:op2 :op1)

ClassAssertion(owl:Thing :in3)
AnnotationAssertion(:op1 :in1 :in2)
AnnotationAssertion(:op2 :in3 :in2)
```

The OWL functional code above shows that, despite the explicit object property declaration for `op1`, every usage of `op1` and its subproperty `op2` was interpreted as an annotation assertion, thereby violating global punning.

### 7.1.4 Results

We shall now proceeds to discuss the results obtained for the distribution patterns under consideration. For every distribution pattern, we indicated which connectivity patterns realizable via 3-tier multiplexing can accommodate its ontologies, and whether the resulting ontology network is interpreted in the most expressive possible way.

#### 7.1.4.1 TA-simple

Ontologies laid out using the *TA-simple* distribution pattern can be assembled into an ontology network using the *tight replication with intermediate layer cycling* connectivity pattern. To that end, a single scope $S$ can be used. The way to do so is depicted in Figure 7.9.

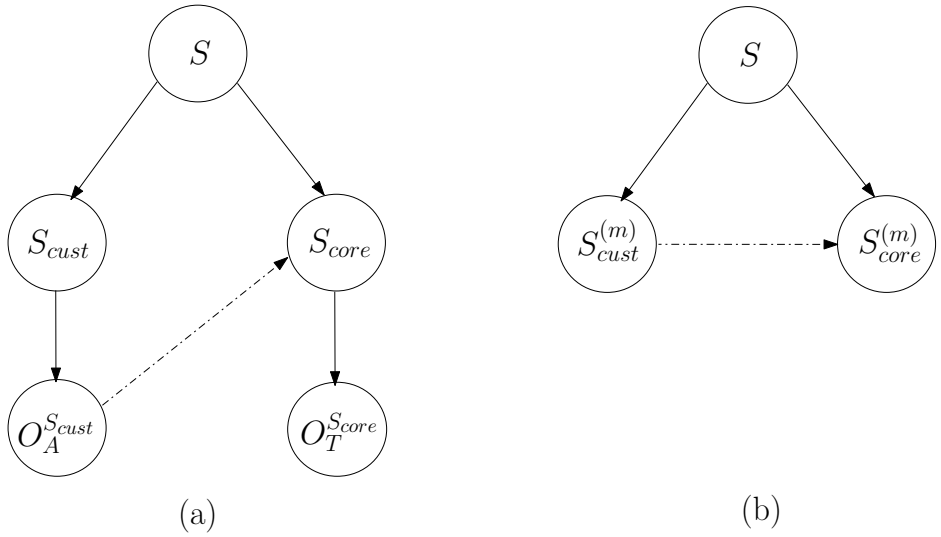


**Figure 7.9: Realization of an occurrence of the TA-simple distribution pattern** - In (a), the virtual ontology network is built using a tight replication scheme with scope $S$ serving as the auxiliary root. In (b), the vertices representing ontology images are collapsed by merging the images with their ontology collectors. The dash-dotted arcs denote references generated at OWL export time by the connectivity policy applied at runtime.

The layout in (a) can be realized in the Stanbol ONM and relies solely on a single instance of the two lower tiers of the model. These are represented by a single scope $S$. In (a), the TBox ontology $o_T$ is referenced by the core space of $S$ $S_{core}$, thus generating its $S_{core}$-image $o_T^{S_{core}}$, while its custom space $S_{cust}$ references the ABox ontology $o_A$, thus generating its $S_{cust}$-image $o_A^{S_{cust}}$. As setting a `TIGHT` connectivity policy makes every custom ontology reference the core space, so does $o_A^{S_{cust}}$ reference $S_{core}$. References so generated are indicated as dash-dotted arcs in the figure. Layout (b), which dos not follow the tight replication pattern, can be realized by following a similar process, except that the core and custom spaces of $S$ are merged with their children, thus leading to the merged spaces $S_{core}^{(m)}$ and $S_{cust}^{(m)}$. The subsequent application of the `TIGHT` policy makes $S_{cust}^{(m)}$ reference $S_{core}^{(m)}$ and depend on it.

Let us show how the first example occurrence is treated. Generated ontology networks were interpreted in the most expressive way possible, yet the global interpretation was ambiguous. The typing statement for `:op1` was interpreted unambiguously in $o_T$ as expected.

```
Declaration(ObjectProperty(:op1))
```

The statement that applies `:op1` in $o_A$ was interpreted as an object property assertion. We noted, however, a spurious declaration of `:op1` as an annotation property. This means that the punning restriction on properties in OWL 2, by which an IRI can be used for one kind of property only [GWPS09], is only globally valid, but locally violated.

```
Declaration(AnnotationProperty(:op1))
ObjectPropertyAssertion(:op1 :in1 :in2)
```

The annotation property declaration was not part of the original ontology, but was added by the rendering policy of the OWL API inherited by Apache Stanbol, which requires that all entities in an ontology signature be locally typed by an OWL declaration. Since the serialization of a stored graph into an RDF format is performed locally, `:op1` was exported as an annotation property and its application as an annotation (though implicitly, as required by RDF serialization). Only when visiting the whole ontology network could the `:op1` application be interpreted as an object property assertion.

This behavior was noted as a non-fatal flaw of the Stanbol ONM implementation, which could be improved by loading the entire imports closure in-memory before exporting an ontology, though at an increased computational cost. Such an upgrade would benefit from optimizations that were not in our research plan to implement. However, because the observed behavior did not disrupt processing by OWL API-based tools such as Protégé, the result was deemed satisfactory.

We also noted that no declarations were recorded for `:in1` and `:in2`, as there were no matching statements in the original source code. However, they appeared in the corresponding Manchester Syntax rendering, as the following excerpt shows:

```
Individual: :in1
Individual: :in2
```

The `LOOSE` variant of the replication pattern, which does not cycle back to an intermediate layer (thus being a rooted polytree), failed to deliver an equally expressive OWL interpretation. The application of `:op1` was interpreted as an annotation assertion, as shown by the resulting OWL functional code below:

```
Declaration(ObjectProperty(:op1))
Declaration(AnnotationProperty(:op1))
AnnotationAssertion(:op1 :in1 :in2)
```

Consequently, no implicit individuals were detected either. Overall, this occurrence of the TA-simple pattern was recorded as one case that requires a relatively dense network layout.

#### 7.1.4.2 T-split

Ontologies distributed by the T-split distribution pattern can be accommodated, under certain conditions, by also using the tight replication connectivity pattern with intermediate cycling. In Figure 7.10, we show how to guarantee an optimal interpretation of the example occurrence described in Section 7.1.3. For this specific case, opening a single scope $S$ is still sufficient. $o_{T_1}$ needs to be managed by its core space $S_{core}$, while $o_{T_2}$ and $o_{T_3}$ need to be managed by the custom space $S_{cust}$, so that the object property declaration in $o_{T_1}$ is inherited by the $S_{cust}$-images of $o_{T_2}$ and $o_{T_3}$.

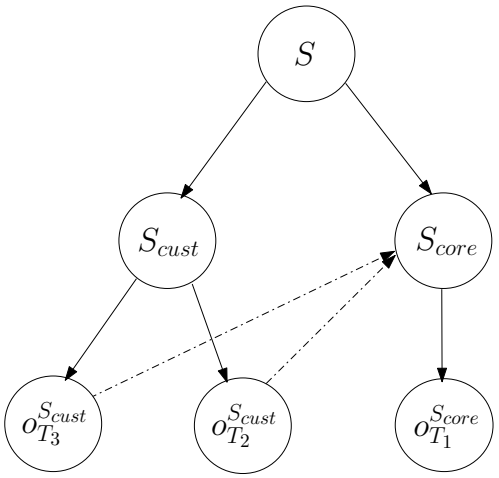


**Figure 7.10: Realization of an occurrence of the T-split distribution pattern** - A single scope $S$ is used, with the ontology $o_{T_1}$ containing the object property declaration managed by the core space $S_{core}$ and the other two ontologies managed by the custom space $S_{cust}$.

In cases where there a multiple TBox ontologies that need to inherit from $n$ ontologies where an object property is declared for each, it is possible to load the declaring ontologies into a core space and the remaining TBox ontologies in the corresponding custom space. If, for some reason (such as access rights restrictions, race conditions etc.) it is not possible to modify the core space further, additional scopes can be created, the declaring ontologies can be loaded in their core spaces, and any further TBox ontologies can be loaded in the corresponding custom spaces. Then a single session can be used to join all the scopes together. However, for more complex occurrences of the T-split pattern, this layout may no longer suffice. If the OWL inheritance chain were to be any longer, such as an object property declaration, followed by a subproperty axiom, followed in turn by a declaration of inverse properties, then a session would have to be involved for storing TBox ontologies as well, which is sub-optimal due to intended use of sessions as ABox ontology collectors. We have not yet found T-split pattern occurrences where more than the 3 tiers used by our method would be necessary in order to guarantee optimal OWL interpretation.

#### 7.1.4.3 TA-retroactive

This test proved inconclusive prior to attempting an ontology network assembly using the Stanbol ONM. When setting a direct import between $o_T$ and $o_A$, the subsumption relationship from `:op1` to `:op0` was interpreted as a sub-annotation property axiom, and their applications were interpreted as annotations despite the object property range axiom on super-property `:op0` and the explicit typing of individuals `:in1` and `:in2`.

```
SubAnnotationPropertyOf(:op1 :op0)
ObjectPropertyRange(:op0 ObjectOneOf(:in2 :in1))

ClassAssertion(owl:Thing :in1)
AnnotationAssertion(:op1 :in1 :in2)
AnnotationAssertion(:op1 :in1 :in1)

ClassAssertion(owl:Thing :in2)
AnnotationAssertion(:op1 :in2 :in1)
AnnotationAssertion(:op1 :in2 :in2)
```

Attempting to interpret a single, merged ontology with all the above statements together yielded the same clauses. The result of this experiment was therefore deemed positive by hypothesis negation.

#### 7.1.4.4 A-split

Ontology network layouts for the A-split distribution pattern can be created by instantiating the *multipartite* connectivity pattern. The maximum-length multipartite pattern that can be realized with 3-tier multiplexing is the 6-partite, spanning from the ontological form of a session to the ontology images with respect to core spaces. This is also the implementation that was used for the example occurrence 1 of this distribution pattern, as shown in Figure 7.11. By having a scope $S$ manage $o_{A_1}$ in its custom space, and a session $z$ connected to $S$ and managing the other ABox ontology $o_{A_2}$, we were able to obtain the desired interpretation of the resulting ontology network.

Note that it is possible to obtain a bipartite digraph out of the 5-partite one in the figure. To do so, one should simply collapse all the vertices representing ontology

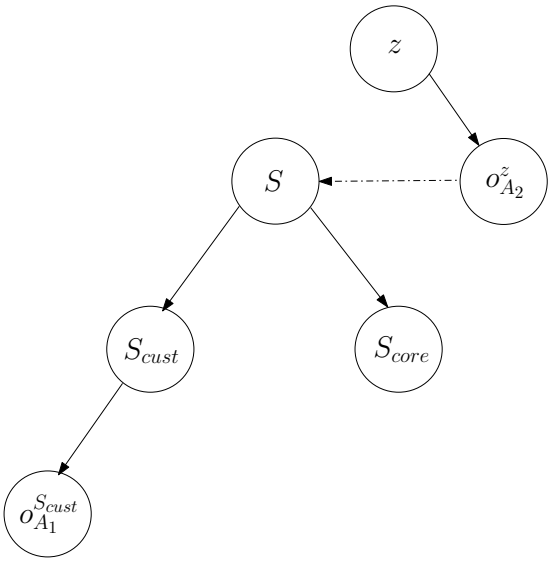


**Figure 7.11: Realization of an occurrence of the A-split distribution pattern** - The distribution pattern can be accommodated using a multipartite connectivity pattern with variable path length. In this example it can reduced to a bipartite graph if all the vertices representing ontology collectors are collapsed.

collectors by requesting merged versions for the $o_{A_1}$ and $o_{A_2}$ images. Note that a bipartite digraph also realizes a *one-way broken ring connectivity pattern*.

We have not found any other occurrence of the A-split distribution pattern alone, where it was not possible to obtain an optimal OWL interpretation of the resulting virtual network, unless those occurrences were combined with an unsatisfiable T-split distribution pattern occurrence. All other cases could be accommodated by distributing ABox ontologies between the custom space of $S$ and the session $z$, which is also the intended usage of the Stanbol ONM from an engineering perspective.

#### 7.1.4.5 *nm*-scatter

A set of ontologies distributed using the *nm*-scatter pattern can be combined with 3-tier multiplexing only under certain conditions concerning $n$, $m$ and the amount of connectivity relations between ABox and TBox ontologies, and between ontologies in the ABox and TBox. In particular, for $n > 2$ there are cases where the optimal ontology network layout cannot be realized using this method.

Figure 7.12 shows how the example occurrence in the *nm*-scatter pattern section

can be realized. The connectivity pattern used is the 3-partite. In (a), a *3-partite connectivity pattern* that can implement the network is displayed. In (b), a realization of that connectivity pattern using one session $z$ and two scopes $S_1$ and $S_2$. The two scopes share the TBox ontology $o_{T_1}$ in their custom spaces. However, $S_1$ manages the other TBox ontology $o_{T_2}$, while $S_2$ manages one ABox ontology $o_{A_2}$. Session $z$ is connected to both scopes and manages ABox ontology $o_{A_1}$. If all the vertices representing ontology collectors in (b) are collapsed, then we obtain (a).

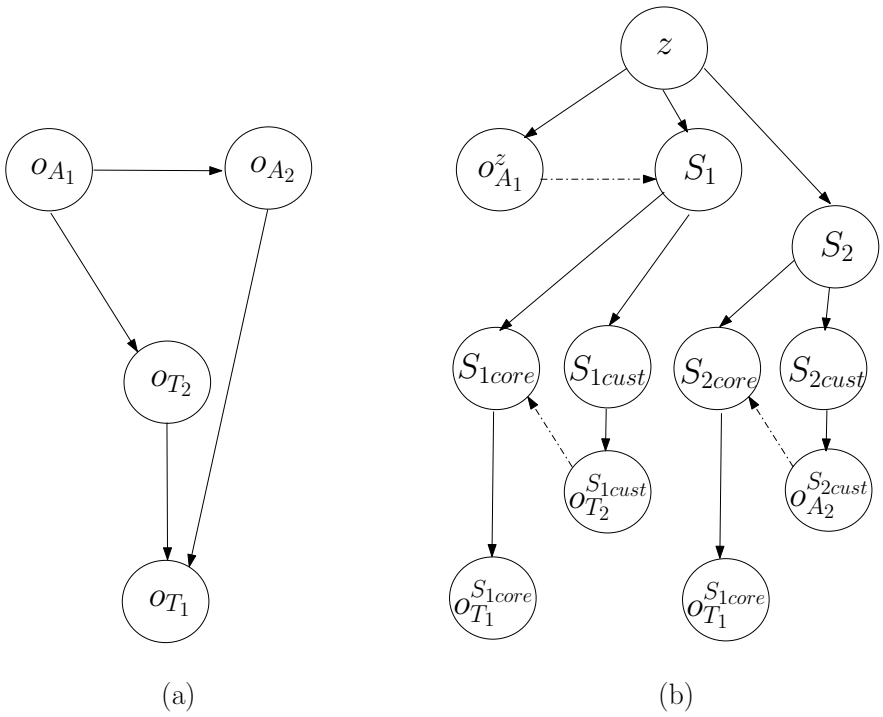


**Figure 7.12: Realization of an occurrence of the 22-scatter distribution pattern** - (a) the 3-partite connectivity pattern. (b) its realization in 3-tier multiplexing. The vertices representing ontology collectors in (b) can be collapsed to obtain (a).

Using this strategy, the resulting interpreted ontology network is as follows:

```
Declaration(ObjectProperty(:op1))
SubObjectPropertyOf(:op2 :op1)

ObjectPropertyAssertion(:op1 :in1 :in2)
ClassAssertion(owl:Thing :in3)
ObjectPropertyAssertion(:op2 :in3 :in2)
```

We note, however, that it was possible to obtain this interpretation because of a favorable condition that proved fundamental. In fact, it can be observed in Figure 7.12 (a), that in the multipartite connectivity pattern occurrence there are no paths longer than 2 that involve more than 2 TBox ontologies, the longest one s being from $o_{A_1}$ to $o_{T_1}$ and from $o_{A_2}$ to $o_{T_1}$. If the optimal layout has some path of length 3 or greater, then we are no longer guaranteed that it can be realized using our method, as it was proven by more complex experiments. In cases such as 22-scatter and 13-scatter, this can never happen. However, starting with distributions such as 31-scatter, the possibility of sub-optimal interpretation of the ontology network. This does not mean that the method automatically fails on ontology sets larger than 4, because repeated axioms or the reuse of the same entities across all ontologies, provided that each entity appears at most once per ontology, can require optimal layouts with path lengths that involve at most 2 TBox ontologies.

#### 7.1.4.6 Summary

Table 7.1 summarizes our findings with respect to the matches found between connectivity patterns and distribution patterns. What emerges from this table is that, for all distribution patterns where the optimal interpretation can be obtained, there were always cases where 3-tier multiplexing fared better than the naïve approach. Some of the initially hypothesized connectivity patterns, such as the ring and rooted polytree, despite being realizable with our method, did not prove to be particularly useful for accommodating ontology sets distributed under specific pattern.

The sets of connectivity patterns, and especially distribution patterns, are by no means intended to be complete, as they were merely identified by plausibility (with examples given as the patterns were illustrated) and the possible challenges brought along from the perspectives of OWL interpretation and ontology network management. More of these patterns could be identified through a dedicated study, at which point it can be of interest to locate their matches, real or potential, in the whole Linked Data cloud. This is far beyond the ambitions of this work, whose future evolution strategies we shall discuss in Section 8.3. However, with the present study we reasonably believe to have laid the basis of a general-purpose evaluation framework for ontology networks, concerning aspects that are normally not encompassed by existing evaluation strategies or normative documentation.

| **Distribution pattern** | **Satisfiable** | **Connectivity pattern(s)** | **Restriction** |
|---|---|---|---|
| **TA-simple** | yes | tight replication w/ intermediate cycling | |
| **T-split** | yes (conditional) | tight replication w/ intermediate cycling | Works for axiom inheritance chains up to 3. |
| **TA-retroactive** | not applicable | none | test cases failed to interpret properly |
| **A-split** | yes | multipartite w/ variable path length, broken ring | |
| $nm$**-scatter** | yes (conditional) | multipartite | Guaranteed for $n + m = 4$ except for $n = 3$. Can work for some cases with $n + m > 4$. Connectivity pattern is obtained by collapsing ontology collectors. |

**Table 7.1: Summary of qualitative evaluation**.

## 7.2 Quantitative evaluation

Having claimed that knowledge management methods alone are not sufficient for tackling ontology network management unless they are backed by an equally solid software approach, further aspects need to be taken into account when software enters the context. Measurements derived from ontology lifecycle management can help us establish the soundness of our model logics-wise, in effect validating it. However, there are factors related to computational efficiency that need at least to be singled out and kept under control.

As this work did not focus on heavy optimization of RDF graph management operations, we had Stanbol ONM rely on existing graph storage backends. This guarantees implementations to be versatile with respect to the ontology storage backend, but makes the computational cost greatly dependent on that of the one chosen. Yet, as our implemented approach constructs additional artifacts on top of the flat graph management middleware, the computational impact of these artifacts and their management had to be measured.

### 7.2.1 Setting

In Section 3.6 we stated the restriction under which our research work operated. Among these, R1 stated that we would not be considering time-efficiency as a parameter for evaluation, under the provision of a reasonable real-time responsiveness of the software platform in rendering ontology network components for Web Services. This is partly due to the fact that Apache Stanbol and Clerezza rely on third-party storage backends, which introduce variables that a high-level programming language like Java does not have full control of. Also, time optimization strategies would require dedicated software engineering work beyond the intended scope of our research goals. More specifically, the time efficiency of create, read, update and delete (CRUD) operations on ontologies loaded into Stanbol ONM was not considered due to the following reasons:

1. It greatly depends on the time efficiency of the underlying triple store performing these operations.

2. It depends on the performance of the hosting hardware, but even if it were to be normalized, it would still be greatly influenced by other factors, such as con-

current processes, operating system swapping/thrashing phenomena, and disk fragmentation.

The technological setting of our experiments was similar to that for the qualitative evaluation, with a few additional details:

- **Hardware**: 2.3 GHz Intel Core i5, 8 GB DDR3 1,333 MHz
- **Operating System**: Mac OS 10.6.8, Ubuntu 12.04 on Linux kernel 3.2 (virtualized on Oracle VM VirtualBox), both systems for x86_64 architectures.
- **Testing platform version**: Apache Stanbol 0.10, built from source snapshot as of September 30, 2012. Dedicated launcher containing only the Stanbol ONM and its dependencies.
- **Runtime environment**: Java 1.6.0_37; Java HotSpot(TM) 64-Bit Server VM
- **File systems**: HFS+ with journalling (Mac OS environment); ext4 (Linux environment)
- **Benchmark**: VisualVM 1.3.5
- **Virtual Machine arguments (vmargs)**: `-Xmx4g -XX:MaxPermSize=128m`

The maximum size of the memory pool for storing objects, or **heap**, was set to 4 GiB, which only slightly exceeds the address space of 32-bit architectures without having to swap memory. The maximum permanent generation size (cf. Section 7.2.2) is set to a maximum of 128 MiB instead of the 256 MiB that would normally be required by a full Stanbol launcher, because we used for a minimal launcher with all the dependencies of the ontology manager. Since only one further bundle is added for performing the tests, the permanent generation does not grow further once all the bundles have been loaded and activated.

### 7.2.2 Definitions

Before proceeding with the quantitative evaluation results, a few preliminary definitions concerning the principles of memory management and garbage collection are necessary. For further details, we refer to the existing literature [JL96].

**[Java] Virtual Machine ([J]VM).** A simulation of a real machine that runs bytecode [in the Java programming language].

**Garbage collection.** A memory management feature that reclaims the memory space occupied by objects that are no longer in use in the virtual machine. Can run periodically or upon explicit calls.

**Heap.** The memory area used by the VM for storing objects created by running programs.

**Retained size (of an object)** . The amount of memory that a VM object preserves from garbage collection. It includes the size of the object itself (**shallow size**), plus the retained sizes of all the objects referenced *only* by it. Note that this definition is recursive.

**Young, tenured and permanent generation.** Most Java VMs (including the *HotSpot*, which we chose for our measures due to its high circulation) manage multiple memory pools for storing objects and other data. Some of these pools are called *generations*. The initial allocation of most objects occurs in the *young generation*, in turn divided into an *eden space* and one or more *survivor spaces*, the latter being destined to store objects that survive a garbage collection run. The *tenured generation* holds objects that have reached a certain age in a survivor space. Finally, the *permanent generation* holds all the reflective data of the VM itself, such as the objects that represent the classes and methods themselves [Jav].

All sizes related to virtual machine (VM) objects are assumed to be specified in bytes unless otherwise noted.

### 7.2.3 Calibration

The computational impact of our proposed software architecture can only be evaluated by comparing the resource usage of its reference implementation against a set of reference measures. These measures will be taken as the "zeroes" of our evaluation process, i.e. the basic figures that equally bottleneck naïve systems and more elaborate systems.

In other words they are the limits, to improve upon which intervention on the low-level layers of ontology storage would be required. The process of determining zeroes for our quantitative evaluation is what we call *calibration*.

To calibrate on the memory usage of ontologies, we automatically generated very simple but potentially large RDF graphs, each containing entities whose size is almost fixed for nearly every triple in the graph. In order to create a graph of $N$ triples, where $N$ is an even integer $>= 4$, we proceeded as follows:

For $n = 0..(\frac{N}{2} - 2)$, the following pair of typing triples (i.e. raw statements that express the assignment of a type to a resource) were created, namely an individual declaration and a class assertion axiom in RDFS and OWL, plus the class definition of `foaf:Person` [GCB07] and a simple, non-versioned OWL ontology ID.

```
<http://www.cs.unibo.it/~adamou/ontologies/[time]> a owl:Ontology .

foaf:Person a owl:Class .

<http://www.cs.unibo.it/~adamou/data/person-[n]>
  a owl:NamedIndividual, foaf:Person .
```

thus totaling exactly $N$ triples. The DL expressivity of these ontologies is $\mathcal{AL}$ (*Attributive Concept Language*, without complements), i.e. very basic. This only has a computational impact if DL-aware processing applications are involved.

These were steps performed for calibrating the memory footprint of ontologies:

1. Generate one ontology as described above for each order of magnitude, in terms of size in triples, from 10 to 100,000.
2. Load each ontology once into Stanbol ONM.
3. For each number of concurrent requests to simulate:
   (a) Simulate the desired number of ontology requests, keeping the JVM from garbage-collecting the corresponding ontology objects.
   (b) Dump the JVM heap and compute the retained size of all objects therein.
   (c) Query the heap dump for every instance of the class representing ontology objects.

| Graph size (triples) | Concurrent requests | In-memory occurrences | Cumulative retained size (B) |
|---:|---:|---:|---:|
| 10 | 10 | 10 | 27,530 |
| 10 | 100 | 100 | 275,300 |
| 10 | 200 | 200 | 550,600 |
| 10 | 500 | 500 | 1,376,500 |
| 100 | 10 | 10 | 187,130 |
| 100 | 100 | 100 | 1,871,300 |
| 100 | 200 | 200 | 3,742,600 |
| 100 | 500 | 500 | 9,356,500 |
| 1,000 | 10 | 10 | 1,734,490 |
| 1,000 | 100 | 100 | 17,344,900 |
| 1,000 | 200 | 200 | 34,689,800 |
| 1,000 | 500 | 500 | 86,724,500 |
| 10,000 | 10 | 10 | 16,921,370 |
| 10,000 | 100 | 100 | 169,213,700 |
| 10,000 | 200 | 200 | 338,427,400 |
| 10,000 | 500 | 500 | 846,068,500 |
| 100,000 | 10 | 10 | 176,982,170 |
| 100,000 | 100 | 100 | 1,769,821,700 |
| 100,000 | 200 | 200 | 3,539,643,400 |
| 100,000 | 500 | N/A | OOM |

**Table 7.2: Memory footprint of multiple simulated requests for an ontology** - Clerezza in-memory (class `org.apache.clerezza.rdf.core.impl.SimpleMGraph`).

Each ontology was loaded from its data stream onto the Stanbol ONM. This means that, once loaded, each ontology only had one public key of type ($ontologyIRI$).

Table 7.2 shows the calibration results for the Apache Clerezza [Apaa] storage backend when requested for in-memory images of its stored RDF graphs. In the JVM heap dump, they are objects of type `org.apache.clerezza.rdf.core.impl.SimpleMGraph`.

We observed that the cumulative retained size of all the ontology objects is totally linear with respect to the number of concurrent requests for the same ontology. This means that, when Clerezza creates multiple memory images of the same RDF graphs, it replicates their triples completely. Even though this implies a potentially large res-

| Graph size (triples) | Concurrent requests | In-memory occurrences | Cumulative retained size (B) |
|---:|---:|---:|---:|
| 10 | 10 | 10 | 15,113 |
| 10 | 100 | 100 | 151,130 |
| 10 | 200 | 200 | 302,260 |
| 10 | 500 | 500 | 755,650 |
| 100 | 10 | 10 | 627,930 |
| 100 | 100 | 100 | 6,279,300 |
| 100 | 200 | 200 | 12,558,600 |
| 100 | 500 | 500 | 31,396,500 |
| 1,000 | 10 | 10 | 5,306,330 |
| 1,000 | 100 | 100 | 53,063,300 |
| 1,000 | 100 | 100 | 106,126,600 |
| 1,000 | 500 | 500 | 265,316,500 |
| 10,000 | 10 | 10 | 51,373,530 |
| 10,000 | 100 | 100 | 513,735,300 |
| 10,000 | 200 | 200 | 1,027,470,600 |
| 10,000 | 500 | 500 | 2,568,676,500 |
| 100,000 | 10 | 10 | 532,525,530 |
| 100,000 | 100 | N/A | OOM |
| 100,000 | 200 | N/A | OOM |
| 100,000 | 500 | N/A | OOM |

**Table 7.3: Memory footprint of multiple simulated requests for an ontology** - OWL API (class `org.semanticweb.owlapi.model.OWLOntologyImpl`).

ident memory occupation, it also gives us the liberty to treat the images of the same ontology with respect to multiple ontology collectors as separate objects, and therefore to manipulate their contents independently, in accordance to the rules described in Section 4.2.5. We also observed that, by using the Clerezza implementation, we could store over 200 occurrences of a graph with 100,000 triples, amounting to $20 * 10^6$ total resident triples. Larger numbers of simulated requests led to out-of-memory (OOM) errors.

Table 7.3 shows the calibration results for the OWL API [OWLa] when requested for in-memory images of ontologies converted from Clerezza in-store RDF graphs. In the

JVM heap, they are objects of type `org.semanticweb.owlapi.model.OWLOntologyImpl`. We observed that concurrent requests are not cached by the OWL API either, however the memory footprint of OWL API objects is much larger than the Clerezza equivalent, starting with graphs of 100 triples or above, leading to a resident memory occupation of less than $8 * 10^6$ triples, with out-of-memory errors occurring thereafter. This is to be expected, since the OWL API encodes the entire OWL semantics of its objects. However, these expressed semantics would be partly list when the ontology is serialized to a non-native OWL format. These were the considerations that led us to implement the RESTful interface of Stanbol ONM using Clerezza in-memory graphs as middleware.

Multiple concurrent requests for the same ontology were then simulated. To do so, a simple application was plugged into the Stanbol framework as an OSGi bundle. This application constructs an array of ontology objects, whose size is specified by the value of the *concurrent requests* column in Tables 7.2 and 7.3.

### 7.2.4 Framework caching

When the Stanbol ONM receives a request for the image of a multiplexed ontology via the Java API, it returns an object which conforms to the type requested by the call, by means of a feature of the Java language called *generics* [NW08]. When the RESTful API is used for obtaining the ontology image instead, the intermediate type is no longer relevant since it has to be serialized into text content for the HTTP response. The only exception is content negotiation: while certain formats, such as RDF/XML and Turtle, are supported by both implementations, others are supported exclusively by the OWL API (e.g. OWL/XML, Manchester and functional syntax) or Clerezza (e.g. RDF/JSON and JSON-LD).

In Figure 7.13 we plotted the results from Table 7.2, which represented the resulting figures of our JVM calibration on the selected Clerezza-based implementation. A linear fit was performed [DS66] with respect to the amount of concurrent requests for an ontology, and computed on the size of the corresponding ontology sources in RDF triples, what parameter $t$ is in the figure. As the result was a perfect fit, with determination coefficient $R^2 = 1$ for all values of $t$, it was verified that no caching phenomena occurred.

The results of the calibration process described earlier brought unfavorable evidence on the possibility of expecting that the combined Apache Stanbol+Clerezza framework

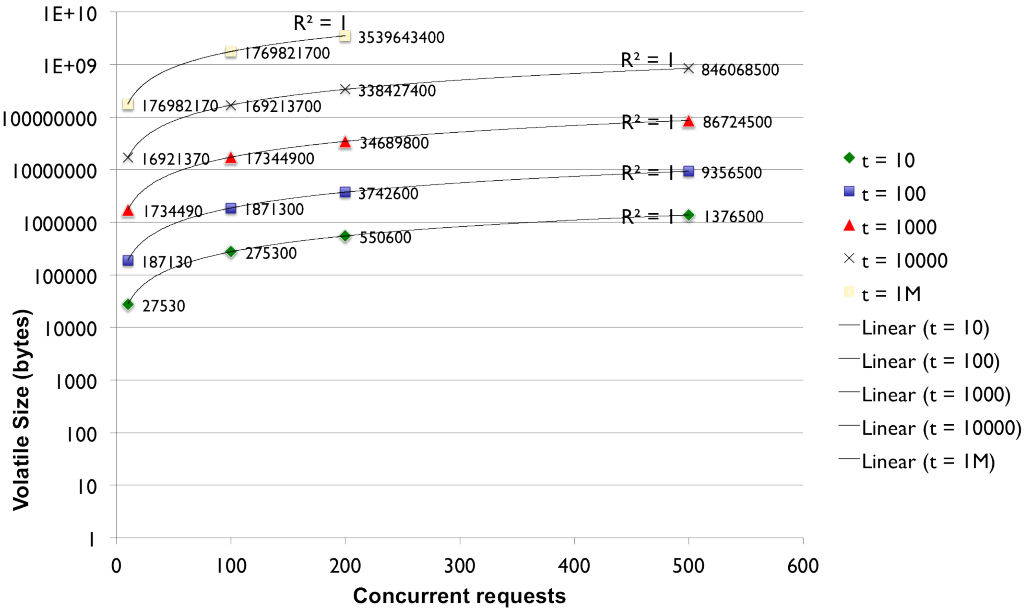


**Figure 7.13: Linear fit on the volatile size of multiple ontology occurrences generated by multiple concurrent requests** - Fit is calculated on the number of triples $t$ per ontology source. Regression is plotted on a lin-log scale.

would cache ontology images in order to maintain a single copy. In Clerezza, two identical RDF triples are separate Java objects, unless CRUD operations are being performed on the original stored graph. This is not desirable for our purpose, as we needed to manipulate each ontology image with its own import statements, ontology IDs, and even TBox and ABox axioms (when a native OWL format is requested). Early experiments using this technique showed that any modification on cached ontology images, although not affecting the unmanaged form of the ontology, was affecting all the other images created by other ontology collectors, thus violating Hypothesis H1 (cf. Section 3.5).

We therefore opted for relaxing Hypothesis H4 instead, which concerned the expected reduction of the memory footprint of $n$ combined ontology images by at least one third of $n$ times the footprint of the unmanaged ontology. As a matter of fact, in the Stanbol ONM this occurs only under specific conditions, which depend almost exclusively on the density of raw statements that can be unequivocally interpreted as annotations in OWL.

The only caching mechanism detected was the one we implemented ourselves, which

encompasses a restricted amount of raw statement types. Roughly, the caching mechanism operated as follows: when the image $o^C$ of an ontology $o$ with respect to a collector $C$ is requested, the Stanbol ONM checks whether $o$ is being managed by other collectors. If so, it loads the source of $o$, $s_o$, from the Clerezza store, but *filtering out all the triples that unequivocally denote annotations*, such as those using `rdfs:label` or `rdfs:comment` predicates. These triples are kept in a separate in-memory `SimpleMGraph`. They are added back to the ontology image only at the time it has to be serialized into a RESTful response object. This only happens after the ontology image has been exported using the method as per Section 4.2.5. This way, should there be any need for managing multiple memory images of the same multiplexed ontology, at least the memory occupation of common annotations could be saved.

To verify that this was the only relevant caching phenomenon, we proceeded by creating multiple ontology sources differing in size by various orders of magnitude, as those 7.2.3. However, depending on the required annotation density (being e.g. 25% one every four ABox axioms), se added RDFS annotation of the form:

```
<http://www.cs.unibo.it/~adamou/data/person-[n]>
  rdfs:label "fixed length annotation"@en .
```

to selected entities of the $n$ declared in the ontology source.

As expected, the memory space saved showed to be linear on the desired annotation density, therefore H4 could only be verified starting with densely annotated ontologies. The advantage of this large memory footprint, though, is that it is not resident: when a HTTP request for an ontology is received on a REST endpoint, the corresponding RESTful resource object is created on-the-fly, the ontology object created and returned, and then the resource disposed. This way, the in-memory ontology object is also disposed on the next garbage collection occurrence. Once the RESTful requests have been served and a JVM garbage collection has occurred, the only resident objects remaining are those that represent ontology collectors, which constitute the memory overhead of the framework discussed in the next section.

### 7.2.5 Framework overhead

According to the measurements performed during the calibration process, the size of an ontology in terms of the triples in its source graph, i.e. the atomic units of its origin,

is related to its memory footprint by a linear law. This is true whether the ontology is interpreted (i.e. loaded as an OWL API object) or encapsulated in raw form (i.e. loaded as a Clerezza object). Even though this is not desirable if all the ontologies managed by the Stanbol ONM had to be resident, it stops being a problem in a Web Service environment. In such contexts an ontology, once serialized and returned to the client, is marked for garbage-collection, and therefore its occupied space is freed up every few seconds, depending on the garbage collector settings. Obviously, this gain goes at the expense of computational time-efficiency, but in our Web Service environment this would only be a problem if the ontology had to be returned to the client in an interpreted form, e.g. in Manchester syntax of functional OWL syntax, which is generally not required by Semantic Web applications that consume RDF.

While the hurdle mentioned above can be overcome, it is also necessary to keep the memory footprint of virtual ontology network artifacts low and under control. This holds for both the ontology collectors themselves, which are resident along the whole service lifetime, and the possible increase in size of ontologies managed by these collectors. This is called the **(space) overhead** of the framework for virtual ontology networks.

The overhead was once more measured by comparing JVM heap dumps of a running Apache Stanbol launcher versus snapshots taken on live samples. Upon realization that, once the ontologies were loaded, the memory footprints of the artifacts of interest no longer changed over time, we replaced sample snapshot with heap dumps taken on simulated service calls.

In order to measure the resident size of **scopes**, we sampled memory measurements on a finite range of configurations that match and exceed the expected workload of the framework. Scopes were measured atomically, because in the standard Stanbol ONM setting every scope has exactly two ontology spaces, and there are no ontology spaces that do not depend on any active scope. Given that $(n, m)$ is the configuration of a scope with $n$ ontologies in its core space and $m$ ontologies in its custom space, the strategy was to set either $n$ or $m$ to zero and vary along two exponential scales without exceeding three orders of magnitude. Therefore, the following configurations were considered:

- the *zero-configuration* $(0, 0)$

- configurations of type $(n, 0)$, with $n = 2^k, k = 0..10$
- configurations of type $(n, 0)$, with $n = 10^k, k = 0..3$
- configurations of type $(0, m)$, with $m = 2^k, k = 0..10$
- configurations of type $(0, m)$, with $m = 10^k, k = 0..3$

Given this setting, the measurements proceeded as follows. For every configuration $(n, m)$:

1. Create a new scope whose ID is of a fixed length (16 UTF-8 characters were chosen in our experiments).
2. Populate the scope with ontologies of the same size (i.e. same amount of triples with URIs of fixed length), according to the configuration. If the configuration is of type $(n, 0)$, populating will occur alongside creation (1).
3. Dump the JVM heap.
4. Compute the retained size of the corresponding object that is an instance of `ScopeImpl`[1]. The resulting value in bytes is the **resident scope size** $M(n, m)$. Recall that, due to the fact that a scope preserves its two spaces from garbage collection, the resulting value will be greater than the sum of their two retained sizes.
5. Compute the average **weight** in bytes $w(n, m)$ of the ontologies in the scope using the following formula: $w(n, m) = \frac{M(n,m)-M(0,0)}{n+m}$.

As shown by the results in Table 7.4, we measured that the resident scope size is essentially the same whether all the ontologies are managed by its core space or its custom space, with a slight bias of 8 bytes in favor of scopes whose custom space is populated. We also noted that the average impact each managed ontology has on the resident size of the scope (measured as the *ontology weight*) decreases significantly once 10 or more ontologies are being managed. In addition, with up to 10 ontologies loaded in a scope, its resident size stays within the 4 kiB mark, while for the less realistic

[1] Canonical name: `org.apache.stanbol.ontologymanager.multiplexer.clerezza.impl.ScopeImpl`.

| Core ontologies (#) | Custom ontologies (#) | Resident scope size (B) | Ontology weight (B) |
|---:|---:|---:|---:|
| 0 | 0 | 2,550 | N/A |
| 0 | 1 | 2,727 | 177 |
| 0 | 2 | 2,851 | 151 |
| 0 | 10 | 3,847 | 130 |
| 0 | 100 | 16,931 | 144 |
| 0 | 1,000 | 142,871 | 140 |
| 1 | 0 | 2,735 | 185 |
| 2 | 0 | 2,859 | 155 |
| 10 | 0 | 3,855 | 131 |
| 100 | 0 | 16,939 | 144 |
| 1,000 | 0 | 142,879 | 140 |

**Table 7.4: Memory overhead of scopes by ontology population** - Values are computed for populations whose fixed graph size varies from 10 to 100,000 triples. The only free variable of each test is either the size of the core space, or the size of the custom space.

value of 1000 managed ontologies the resident memory footprint tops at $142,875 \pm 4$ B (less that 140 kiB). By comparison, this number is of the same order as an in-memory Clerezza graph of 500 triples.

To measure the resident size of **sessions**, we proceeded in a similar fashion as for scopes, except that, since sessions are not divided into further ontology collectors, session configurations have only one parameter. Given that $(n)$ is the configuration of a session with $n$ ontologies, the strategy was to vary $n$ along two exponential scales without exceeding three orders of magnitude. Therefore, the following configurations were considered:

- the *zero-configuration* (0)
- configurations of type $(n)$, with $n = 2^k, k = 0..10$
- configurations of type $(n)$, with $n = 10^k, k = 0..3$

The measurements proceeded as follows. For every configuration $(n)$:

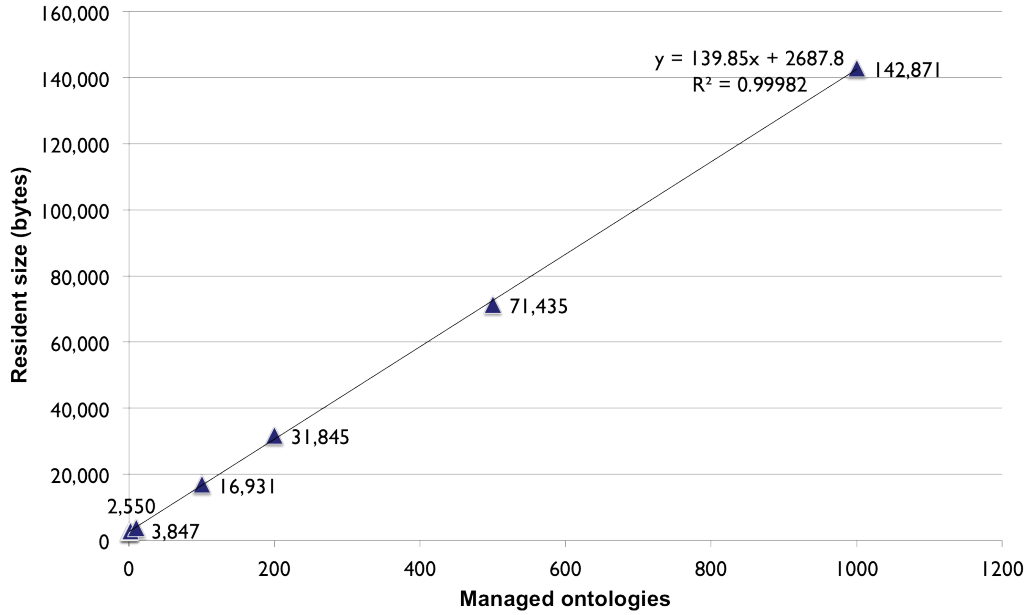


**Figure 7.14: Linear fit on the resident size of a custom ontology space with respect to its number of ontologies** - Regression is plotted on a lin-lin scale.

1. Create a new session whose ID is of a fixed length (16 UTF-8 characters were chosen in our experiments).
2. Populate the session with ontologies of the same size (i.e. same amount of triples with URIs of fixed length), according to the configuration.
3. Dump the JVM heap.
4. Compute the retained size of the corresponding object that is an instance of `SessionImpl`[1]. The resulting value in bytes is the **resident scope size** $M(n)$.
5. Compute the average **weight** in bytes $w(n)$ of the ontologies in the scope using the following formula: $w(n) = \frac{M(n)-M(0)}{n}$.

As shown by the results in Table 7.5, we measured that the resident session size is comparable to the resident scope size when many ontologies are being managed. In this context, the results for the average impact of each ontology (measured as the *ontology weight*) is fluctuating, but still within the range of the most positive i.e. lowest values

[1]Canonical name: `org.apache.stanbol.ontologymanager.multiplexer.clerezza.impl.SessionImpl`.

| Ontologies (#) | Resident session size (B) | Ontology weight (B) |
|---:|---:|---:|
| 0 | 1,717 | N/A |
| 1 | 1,865 | 148 |
| 2 | 1,989 | 136 |
| 10 | 2,981 | 126 |
| 100 | 16,061 | 143 |
| 1,000 | 141,997 | 130 |

**Table 7.5: Memory overhead of sessions by ontology population** - Values are computed for populations whose fixed graph size varies from 10 to 100,000 triples.

for the ontology weight measured on scopes. With up to 10 ontologies loaded in a session, its resident size stays within the 3 kiB mark, while for the less realistic value of 1000 managed ontologies the resident memory footprint tops at $141,997$ B, which is comparable to the resident size of a scope managing the same ontologies.

The linear growth trend of the resident size of ontology collectors was once again proven by performing a linear fit, part of which is displayed in Figure 7.14. For simplicity, only the results for scope configurations of type $(0, m)$ are shown, with $m$ on the X-axis. Figures for $(n, 0)$ configurations, as well as for the resident size of sessions, are greatly similar and not shown. As the determination coefficient was computed as $R^2 \approx 1$, namely 0.99982 for $(0, m)$ configurations, we were able to obtain a nearly-perfect fit, with a zero-overhead (i.e. for no ontologies managed) below 3 kilobytes.

Another overhead factor we need to measure is how larger or smaller ontology images are than their unmanaged forms. To that end, we measured the retained size of the same Clerezza objects of type `SimpleMGraph` (short) as in the calibration phase, first taken standalone and then as managed by the core space of a scope. The overhead was measured for graphs of sizes within all the orders of magnitude considered.

The results of this last overhead measurement are shown in Table 7.6. We observed that ontology images, whose size is shown in the *Size in tier 1* column, added a fixed amount of memory overhead to their unmanaged counterparts, whose size is shown in the *Plain size* column. This value is shown in the *Delta* column, and its impact remains significant, of 1% or above, only for small ontologies whose sources have up to 100 triples. The impact is measured as a percentage in the *Normalized delta* column.

| Graph size (triples) | Plain size (B) | Size in tier 1 (B) | Delta (B) | Normalized delta (%) |
|---:|---:|---:|---:|---:|
| 10 | 2,753 | 3,477 | 724 | 26.29858 |
| 100 | 18,713 | 19,437 | 724 | 3.86896 |
| 1,000 | 173,449 | 174,173 | 724 | 0.41741 |
| 10,000 | 1,692,137 | 1,692,861 | 724 | 0.04278 |
| 100,000 | 17,698,217 | 17,698,941 | 724 | 0.00409 |

**Table 7.6: Memory overhead of ontologies obtained from a scope versus their plain forms** - Delta value indicates the overhead. The normalized delta is calculated with respect to the plain size of the source graph.

If we set as tolerance a cutoff value of 0.5 percent of the byte-size of the original ontology, it is safe to state that the overhead of scoping the ontology becomes negligible for ontologies whose graph source is one thousand triples large, and orders above.

# 8

# Conclusions

We have treated the problem of dynamic ontology network management from a dual perspective, i.e. a logical one and a software-engineering one. The choice of tackling this problem was the result of an in-depth analysis of the scientific and technological background in ontology lifecycle management and, to an extent, in the discipline of ontology networking which is still developing. This analysis has a deep technological foundation, as it also results from our experience in the usage of ontology-based applications and the development of new ones with the existing software development tools [AP12, APG10, Ada09, Ada08]. A number of limitations emerged from the aforementioned study, which we list as follows.

- Semantic Web applications and ontology-driven application are often regarded as two things not only distinct, but also mutually exclusive, as shown by the the reluctancy to regard linked data as ontologies.

- Although the concept of ontology network was mainly discussed in the engineering department [SFGPMG12b], no formal specification or theoretical framework of ontology networks was found at the time this thesis was written.

- Due to their knowledge engineering background, ontology networks are seen as artifacts which only need to be assembled as the networked ontologies are developed. There is no formal doctrine for the practice of assembling ontology networks at runtime even beyond the scope envisioned by the ontology authors. However, these practices become more necessary from time to time, as use cases

and applications surface, where there is a need to combine heterogeneous Linked Data and ontologies in general.

- In some scenarios, the *layout* of ontology networks can affect the result of interpretation procedures even when the actual content of ontologies remains unchanged. This was verified for the OWL API and its applications, when the connectivity mechanism is the usage of `owl:imports` in OWL ontologies rendered in RDF.

- Application developers that need to consume ontologies are generally not as knowledgeable in the subject matter as ontology engineers are. Therefore, they cannot be presented with the problem of assembling ontology networks from the same perspective as if they belonged to the latter category.

- The problem of interpreting ontology network is often underestimated, especially on the side of ontology repositories which provide ontologies only in their native forms.

- The introduction of new features in the OWL 2 language, such as *punning*, has a potential for side effects which could even violate the restrictions placed on the features themselves, unless unrealistically resource-intensive computations are undertaken.

- Tool support that is aware of ontology networks as knowledge items by their own right is scarce.

- Plugin-based software frameworks, being as modularized as ontology networks are, have an intrinsic potential for being practically involved with ontology network management. However, to date this potential goes underexpressed for the most part.

Each detected limitation configures a problem per se, in either scientific terms or technological. While solving them altogether would be overly ambitious for the scope of a doctoral thesis, we have tackled every single one of them with different degrees of detail, and proposed a general framework which we expect to serve as the basis for full-fledged solution proposals to each problem. To that end, we consolidated our overall aim into the following sub-objectives: (1) a theoretical framework for expressing

connectivity relations in ontology networks; (2) a method for defining and constructing ontology networks dynamically, with an attention to problems in ontology interpretation; (3) a software-engineering approach that describes how applications that either provide ontology networks or consume them should accommodate such a method; (4) a Web service interface specification for manipulating ontology networks dynamically; (5) the development of an ontology network management application that implements this method and service interface.

The remainder of this chapter recaps the contributions to the state of the art described in this thesis, then reviews them from the standpoint of the objectives and hypotheses formulated in Chapter 3. The potential for future evolution strategies of this work is then discussed and concludes this dissertation.

## 8.1 Summary

The contributions delivered by this research work are as follows.

- We provided an overview of the state of the art in ontology management, including repository software, related representation languages, methodologies and notable exemplary instances. Whenever these aspects specifically tackled ontology networks, be they given that name or not, it was made explicit in the course of our treatise. The relationship between ontology management and Linked Data was also cleared, in order to resolve the frequent terminological ambiguity around these notions and single out the binding between Linked Data combinations and ontology network management.

- Having acknowledged that a formal, i.e. not grounded on engineering, specification of ontology network was missing at the time this work began, we proceeded to establish a theoretical framework that defines ontologies, connections and ontology networks, using essential algebraic constructs. On top of that framework, we added a novel set of definitions, such as virtual ontology network, ontology source and image. These were introduced specifically for a better understanding of our dynamic assembly method, but also make sense out of this context, since they reflect practices which already belong to ontology lifecycle management, especially to the extent where repository software is concerned.

- A method was described, called *3-tier multiplexing*, which places the combined bulk of all the ontologies required by all virtual ontology networks along three tiers of an abstract infrastructure. These are the tiers of *core spaces*, *custom spaces* and *sessions*, so named after the types of artifacts that are used for collecting ontologies. For practical convenience, since the two lower tiers are strongly linked, we also unified them using a further class of artifacts, which are called *scopes*. We provided proof that the constructs generated by applying this method are still ontology networks compliant with the theoretical framework established earlier.

- A software architecture was introduced, in order to specify a possible *binding* between elements of a virtual ontology network and software components that are responsible for either providing the ontology networks, or ordering their assembly and consuming them. We resorted to classical software engineering practices such as activity specifications and design and communication patterns such as façades and publish/subscribe.

- Ontology languages such as OWL assume a protocol binding with HTTP URIs, as well as a correspondence between the physical locations of ontologies and their names and versions. In order to preserve this assumed binding, we devised a Web Service interface specification that maps to a set of methods for reading, creating, updating and deleting ontologies, both in their unmanaged form and as images created by artifacts of our multiplexing method. Due to the existing relationship with the HTTP method, the Web Service interface conforms to the current standard of REST.

- The 3-tier multiplexing method and its RESTful interface were implemented in an ontology network provider application. This software is the Ontology Network Manager (ONM) of Apache Stanbol, a stack of semantic services for supporting content management systems. Since it is an OSGi application, it is inherently modular and can run on a single service platform where client applications responsible for ordering ontology network setups can also be registered. Alternatively, these applications can be simple Web clients and take advantage of the RESTful Web Services of the Stanbol ONM. The ONM has been a part of Apache Stanbol since its incubation in late 2010, and still constitutes the core framework for

managing ontologies, as well as a dependency of reasoning services and rule-based content enhancers, even now that Apache Stanbol has been promoted to top-level project status.

- The implementation itself was evaluated from two different angles. On the qualitative side, we had to verify for which distributions of axioms across multiple ontologies the Stanbol ONM was able to put these ontologies together in a network whose layout guaranteed the most expressive interpretation possible. To that end, we had to identify scenarios where, given a set of axioms distributed in a certain fashion across multiple ontologies, an optimal layout exists at all, regardless of whether it can be realized by the Stanbol ONM or not. This prompted us to establish our own evaluation framework, which comprises *distribution patterns* and *connectivity patterns*. The evaluation framework itself is grounded on the fundamentals of graph theory. This is an intuitive yet relatively novel type of approach as applied to ontology networks, since we had so far only acknowledged its application to single ontologies treated as graphs [GCCL05]. Results showed that, although certain optimal layouts would require a more articulated multiplexing strategy, which however would be hard to justify on the side of software and services, many distribution patterns had a corresponding layout realizable with the Stanbol ONM. These combined distribution patterns and layouts, in turn, guaranteed more expressive interpretations than common or brute-force layouts, when submitted to the OWL API and its applications.

- On the quantitative side of the evaluation, the approach was more traditional. After calibrating the memory footprints of sample ontology networks, we measured the resident occupation of multiplexed ontologies by computing their retained sizes in dumps of the Java Virtual Machine heap. We figured out that, of the two extreme strategies that could be adopted, the safest one was to perform no caching and save the computational cost of maintaining memory images in sync with changes in the ontologies. This implies that multiple concurrent request on a networked ontology mean a proportional growth of its memory footprint. However, this could only be a problem in the interval between two JVM garbage collections, as the memory images of networked ontologies never persist. In addition, we measured the resident sizes of the ontological artifacts introduced by our

method, and which remain resident during the lifetime of the ontology networks that use them. Results on this front were far more promising: not only do these new artifacts have a negligible memory overhead, but also that overhead is not increasing linearly as more ontology networks use them.

## 8.2 Relation to stated goals

In Sections 3.5 and 3.3 our research hypotheses were formulated, which drove our overall line of work. How the hypotheses were verified and the objectives achieved is quickly recapped in this section.

### 8.2.1 Objectives

**O1** was a prerequisite for most of the objectives that followed, which is why it was the first to be achieved and described in Section 4.1. Many concepts revolving around ontology network management were formalized in this phase, while others were reformulated, like "ontology interpretation" becoming the resolution function from ontology sources to ontologies.

**O2** was fulfilled in the final part of Section 4.1, which introduced novel concepts such as virtual ontology networks, ontology collectors and images, and in Section 4.2, which described a method to combine established and novel concepts for generating ontology networks in OWL 2.

**O3** was achieved in Chapter 5, where both a reference software architecture for ontology network providers and a specification of how software components can relate to them are described.

**O4** concerned the specification of an effective yet conservative RESTful interface for manipulating ontology networks. This specification is outlined in Section 5.4 and adds full CRUD operation support to ontology networks. The fulfillment of the conservativeness requirement was corroborated by the fact that the specification respects OWL 2 conventions, save a few borderline cases, and does not propose any extension of the language in order to comply with the RESTful interface.

**O5** was the technological objective that focused on a reference implementation. This was developed in Java as the Ontology Network Manager component suite for Apache Stanbol (Stanbol ONM), a top-level Apache project, which included the ONM since its very first implementation. This framework was described in Chapter 6 (sans the RESTful API specification). Development of the Stanbol ONM had two iterations due to the need to find a trade-off between the persistence of ontology network structures, the versatility for managing RDF and native OWL formats, and the resource usage of these features. The memory efficiency of this solution was discussed in Section 7.2 and is reprised in the discussion over hypothesis H3 in the next section. The objective can be considered to be fulfilled, modulo the memory footprint considerations discussed.

### 8.2.2 Hypotheses

**H0,** the null hypothesis, was empirically disproven in the preliminary phase of our work. There, we verified that in common real-world OWL 2 applications, the way ontologies are organized in an ontology network has an impact on how the whole network is interpreted as an ontology, with potential disruptive side-effects (cf. Section 3.5.1). This aspect is underspecified in the normative documentation of OWL 2, and the potential consequences can even violate the restrictions applied to OWL 2 features such as the application of *punning*. It was because of this result that we decided to tackle this research problem and formulate the other hypotheses.

**H1** concerned the possibility to create standalone ontology networks artificially. The first part of Chapter 4 covers this aspect theoretically, by providing definitions of artifacts from which there is a way to obtain OWL 2 ontologies. Proof was then given, that the resulting ontologies formed ontology networks compliant with the theoretical background we gave early in the chapter, and that we claim to be plausible as it reflects the notions of ontology network given in practical studies on ontology engineering. The independent, or standalone, nature of these ontology network was verified partially. On the one hand, multiplexing allows us to deploy multiple images of an ontology, each belonging to its own ontology network without even referencing the original ontology. On the other hand, if an ontology is removed from the persistent knowledge base, its images in each ontology network are also lost, leading to unsatisfied dependencies. This

trade-off was a necessary evil, as it was opted for in order to avoid an uncontrollable expansion and clutter of the persistent knowledge base.

**H2** stated that the method described in Section 4.2 can achieve better results, in terms of interpreting a set of ontology sources as a single ontology, than by simply having all the ontologies imported by a root. This proved to be true for most cases where the method can be applied at all. First, in Chapter 4 we gave an intuitive notion of what a "better result" would be. Then, in the qualitative evaluation phase described in Section 7.1, we analyzed which connectivity patterns would be optimal for ontology networks with a certain distribution of their statements. Although not all such connectivity patterns can be implemented using our method, those that can have shown promising results once applied to "critical" distribution patterns.

**H3** is the space-efficiency hypothesis, which gradually came under question as our solution strategies evolved. In cases where ontologies can be provided in a native OWL format (e.g. in Manchester syntax or OWL functional syntax), it was possible to maintain ontology networks in memory and save space, but the computational cost of synchronizing memory images with changes in the ontology networks turned out to be high. By switching to a pure RDF API for implementing multiplexers, we could guarantee that the majority of the *resident* memory footprint was limited to a small overhead given by ontology collectors, as their ontology images were no longer maintained. This means that the memory required to serve multiple request is proportional to the size in triples of the requested ontologies, but this only concerns the interval between two garbage collections by the Java Virtual Machine. The quantitative evaluation described in Section 7.2 was conducted in order to verify this hypothesis.

As part of the experiments on space-efficiency, we also observed that, in some reasonably likely scenarios, the storage of ontologies in Apache Stanbol had a tremendous impact on their nonvolatile fingerprint, namely the on-disk occupation. Every created ontology was mapped to a standalone file system directory, which stored triple contents, indices and meta-level information. In the Mac OS X environment used for experiments, each directory was at least 200 MiB large upon creation, no matter how large the ontology that was stored. As a result, 5 ontologies with 10 axioms each could occupy almost 1 GiB of disk storage, which was not an acceptable result even

on production server systems. The reason was the Mac OS HFS+ file system, which has a policy of immediately allocating the *maximum* size allowed for files when they are created. Other filesystems, such as ext4 for GNU/Linux, or NTFS for Windows, allocate bytes dynamically as the file size actually needs to be increased. This was not a shortcoming of the Stanbol ONM, which could only have worked around it by storing all ontologies in a single graph, but a limitation of the underlying Clerezza implementation. This issue was ultimately solved after Clerezza implemented named graph support in its storage backend.

## 8.3 Future work

This dissertation concludes with a discussion on possible lines of future work, some of which are underway at the time of this writing.

On the strictly technological side, it should be noted that my role in the Apache Software Foundation [Apae] represents, as anyone else's, a personal commitment to fostering the evolution of the Stanbol project. Therefore, this committership is an independent yet strong driver for the technological advancement of the ontology network management platform, of which I am the key contributor, as well as dedicated support for specific use cases.

**Automatic assembly.** An obvious direction where the evolution of this work could be heading is clearly automation. This dissertation has laid out the principles and rationale behind using multi-tiered multiplexing for constructing ontology networks given a set of originally non-networked ontologies. However, the selection of which ontologies should be loaded in a certain tier, and how they should be spread across ontology collectors within the same tier, is assumed to be performed either manually under the guidance of common sense, or procedurally by the client application. The next step would be to automate this process: given an arbitrary set of ontology resources, a lookahead method could be devised for calculating the optimal layout that accommodates all of them and guarantees the highest possible level of interpretation of their raw statements as axioms. As a matter of fact, the studies on connectivity patterns used for qualitatively evaluating the method (cf. Section 7.1) constitute preliminary work towards this direction we intend to pursue. The connectivity patterns studied

here can help construct classes of virtual ontology network layouts. What is currently under investigation is the possibility to combine the existing connectivity pattern work with a logic theory that models how a set of non-OWL-native statements should be interpreted depending on how the graph is visited and what output the visit has produced until those statements are encountered. As far as this dissertation is concerned, this theory was replaced empirically with the observation of how OWL libraries of wide adoption operated.

**Exploiting ontology modularization.** Another aspect that was not considered, as for the initial restrictions we placed (cf. Section 3.6), is ontology modularization. The techniques used to explore this field are numerous [TDL$^+$11, dSSS09, OY12, SCG12], however those techniques and studies concerning the binding of ontologies with distributed systems [DTPP09] and TBox/ABox scalability [WM12, MW09] are especially interesting for the context of this work. Decomposition techniques would allow mixed ontologies, which could have an unnecessarily negative impact on the performance of ontological computations, to be replaced with more scalable equivalents. On the other hand, heavily modularized ontology networks, which imply very sparse distribution patterns 7.1.3, could require a network layout that cannot be realized with our method in order to be interpreted optimally. One line of research would be to analyze how ontology multiplexing techniques can be combined with tailored modularization schemes.

**Increasing multiplexing tiers.** We decided to implement a method that spreads networked ontologies across three logical tiers. The choice of three tiers was a trade-off between the impact of each tier on the probability of obtaining higher-order ontology interpretations, and practical justifications for the usage of each tier from an ontology engineering perspective. It would have been difficult to provide an intuitive guideline as to how a greater number of tiers could be used by practitioners. However, this is still to be verified through at least a user study. In addition, we would have to assess up to what point the addition of further multiplexing tiers brings a tangible benefit to ontology networks on the side of logical interpretation. If these issues are overcome, we can then easily extend the method presented here.

**Improved virtualization.** Some of the limits encountered while implementing and testing our solution proposal are technological. An apparent ground for ample improvement is the introduction of cutting-edge caching capabilities, which would enable us to keep time-efficiency in consideration as well. One limitation we encountered is that, if there is a need for manipulating ontologies as simultaneous images, as described in Section 4.2, some degree of in-memory replication of the ontology contents is necessary. We eventually chose to partially surrender caching and adopt quasi-total replication for short bursts of time, but alternative approaches are being explored. Replacing the underlying triple store with a native OWL axiom store would bring some benefits, such as the possibility to maintain in-memory ontology images and virtual networks in their OWL-interpreted state, thereby easing the burden of client applications. Some work concerning native OWL storage exists [Red10, HKG09], but it would be necessary to verify (i) what computational impact there would be concerning the synchrony between stored ontologies and cached images; and (ii) whether this strategy would not impair interoperability with RDF and Linked Data applications.

**Integration with ontology repository systems.** There are multiple possible integration points with the existing world of ontology repositories described in Section 2.4. Some include: proposing the Stanbol ONM as an OWL-safe backend for managing user spaces in Cupboard; exporting virtual ontology network metadata using the OMV vocabulary, to achieve interoperability with Oyster registries; implementing our theoretical framework as part of the Open Ontology Repository. In addition, having the Stanbol ONM as a support infrastructure for the ODP.org repository is a task already underway, since we extended and implemented the eXtreme Design library management features, used within ODP.org, as the default mechanism for Stanbol to fetch multiple distributed ontologies selectively.

**Extension to other ontology networking paradigms.** Our method was applied specifically to OWL. More precisely, it created ontology networks based on one explicit networking feature of OWL which is the import-by-location scheme, and a change management feature of OWL 2, i.e. the version IRI, which we manipulate in order to reflect management policies within the virtual ontology network. Imports are just one way of representing connectivity and dependency relations in ontology networks, and since

they are non-selective, they represent a sub-optimal method. It is however part of the OWL recommendation and used by many applications. One interesting course of evolution for our work would be to explore other possibilities for connecting ontology images into virtual networks, both explicit, as those based on ontology alignment [RVNLE09], and implicit, as the extension of OWL with $\varepsilon$-connections [GPS06].

# References


[Ada08] Alessandro Adamou. SAscha: A RIA approach for supporting semantic web services in the italian interoperability framework. In Aldo Gangemi, Johannes Keizer, Valentina Presutti, and Heiko Stoermer, editors, *SWAP*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[Ada09] Alessandro Adamou. SAscha: Supporting the Italian public cooperation system with a rich internet application for semantic web services. In Aroyo et al. [ATC+09], pages 796–800.

[Ada12] Alessandro Adamou. Software architectures for scalable ontology networks. In Simperl et al. [SCP+12], pages 844–848.

[AdM09] Carlo Allocca, Mathieu d'Aquin, and Enrico Motta. DOOR - towards a formalization of ontology relations. In Jan L. G. Dietz, editor, *KEOD*, pages 13–20. INSTICC Press, 2009.

[Amb05] Scott W. Ambler. *The Elements of UML 2.0 Style*. Cambridge University Press, Cambridge, UK, 2005.

[AMH08] Herman J. Ader, Gideon J. Mellenbergh, and David J. Hand. *Advising on Research Methods: a Consultant's Companion*. Johannes Van Kessel Publishing, 2008.

[AP12] Alessandro Adamou and Valentina Presutti. Customizing your interaction with Kali-ma. In Suárez-Figueroa et al. [SFGPMG12b], chapter 15, pages 319–342.

[APG10] Alessandro Adamou, Valentina Presutti, and Aldo Gangemi. Kali-ma: A semantic guide to browsing and accessing functionalities in plugin-based tools. In Cimiano and Pinto [CP10], pages 483–492.

[APH+12] Alessandro Adamou, Raúl Palma, Peter Haase, Elena Montiel-Ponsoda, Guadalupe Aguado de Cea, Asunción Gómez-Pérez, Wim Peters, and Aldo Gangemi. The NeOn ontology models. In Suárez-Figueroa et al. [SFGPMG12b], chapter 4, pages 65–90.

## REFERENCES


[ATC+09] Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Paslaru Bontas Simperl, editors. *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, volume 5554 of *Lecture Notes in Computer Science*. Springer, 2009.

[ATS+07] Richard Arndt, Raphaël Troncy, Steffen Staab, Lynda Hardman, and Miroslav Vacura. COMM: Designing a Well-Founded Multimedia Ontology for the Web. In *ISWC*, 2007.

[BB08] Robert Battle and Edward Benson. Bridging the Semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semant.*, 6(1):61–69, February 2008.

[BBRC09] Yazid Benazzouz, Philippe Beaune, Fano Ramparany, and Laure Chotard. Context data-driven approach for ubiquitous computing applications. In Bill Grosky, Frédéric Andrès, and Pit Pichappan, editors, *ICDIM*, pages 235–240. IEEE, 2009.

[BC06] Laura Bocchi and Paolo Ciancarini. On the impact of formal methods in the SOA. *Electr. Notes Theor. Comput. Sci.*, 160:113–126, 2006.

[BCM99] Trevor J. M. Bench-Capon and Grant Malcolm. Formalising ontologies and their relations. In Trevor J. M. Bench-Capon, Giovanni Soda, and A. Min Tjoa, editors, *DEXA*, volume 1677 of *Lecture Notes in Computer Science*, pages 250–259. Springer, 1999.

[BCMP12] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Web data reconciliation: Models and experiences. In Stefano Ceri and Marco Brambilla, editors, *SeCO Book*, volume 7538 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2012.

[BDHB06] John G. Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. SIOC: an approach to connect web-based communities. *IJWBC*, 2(2):133–142, 2006.

[BGH+03] Paolo Bouquet, Fausto Giunchiglia, Frank Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL: Contextualizing ontologies. In *ISWC*, pages 164–179, 2003.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. W3C working group note, World Wide Web Consortium (W3C), February 2004.

[BHN80] Jayanta Banerjee, David K. Hsiao, and Fred K. Ng. Database transformation, query translation, and performance analysis of a new database computer in supporting hierarchical database management. *IEEE Trans. Software Eng.*, 6(1):91–109, 1980.

[BHS08] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier, 2008.

[BJ08] Christian Bizer and Anupam Joshi, editors. *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, October 28, 2008*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[BL98] Hajo Broersma and Xueliang Li . Isomorphisms and traversability of directed path graphs. *Memorandum / University of Twente, Faculty of Applied Mathematics*, 1433, 1998.

[BL06] Tim Berners-Lee. Linked Data - design issues. *W3C*, 09(20), 2006.

[BLCPS05] Tim Berners-Lee, Dan Connolly, Eric Prud'hommeaux, and Yosi Scharf. Experience with N3 rules. In *Rule Languages for Interoperability*. W3C, 2005.

[BLHL01] Tim Berners-Lee, James A. Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the Web of Data. *J. Web Sem.*, 7(3):154–165, 2009.

[Blo09] Eva Blomqvist. OntoCase – automatic ontology enrichment based on ontology design patterns. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, volume 5823 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2009.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[BPDG10] Eva Blomqvist, Valentina Presutti, Enrico Daga, and Aldo Gangemi. Experimenting with eXtreme Design. In Cimiano and Pinto [CP10], pages 120–134.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

## REFERENCES


[BS09] Kenneth Baclawski and Todd Schneider. The Open Ontology Repository initiative: Requirements and research challenges. In Tania Tudorache, Gianluca Correndo, Natasha Noy, Harith Alani, and Mark Greaves, editors, *Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, volume 514. CEUR Workshop Proceedings, 2009.

[BT05] Max Bramer and Vagan Terziyan. *Industrial Applications of Semantic Web: Proceedings of the 1st International IFIP/WG12.5 Working Conference on Industrial Applications of Semantic Web, ... Federation for Information Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Bur09] Bill Burke. *RESTful Java with JAX-RS*. O'Reilly Media, 2009.

[Can06] Linda Cantara. Encoding controlled vocabularies for the Semantic Web using SKOS Core. *OCLC Systems & Services*, 22(2):111–114, 2006.

[CFLGPLC03] Óscar Corcho, Mariano Fernández-López, Asunción Gómez-Pérez, and Angel López-Cima. Building legal ontologies with METHONTOLOGY and WebODE. In V. Richard Benjamins, Pompeu Casanovas, Joost Breuker, and Aldo Gangemi, editors, *Law and the Semantic Web*, volume 3369, pages 142–157, 2003.

[CGL+11] Mélanie Courtot, Frank Gibson, Allyson L. Lister, James Malone, Daniel Schober, Ryan R. Brinkman, and Alan Ruttenberg. Mireot: The minimum information to reference an external ontology term. *Applied Ontology*, 6(1):23–33, 2011.

[CGP+03] Paolo Ciancarini, Riccardo Gentilucci, Marco Pirruccio, Valentina Presutti, and Fabio Vitali. Metadata on the Web: On the integration of RDF and topic maps. In *Extreme Markup Languages®*, 2003.

[CHL07] Jorge Cardoso, Martin Hepp, and Miltiadis D. Lytras, editors. *The Semantic Web: Real-World Applications from Industry*, volume 6 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2007.

[CMBR11] Mélanie Courtot, Chris Mungall, Ryan R. Brinkman, and Alan Ruttenberg. Building the obo foundry - one policy at a time. In Olivier Bodenreider, Maryann E. Martone, and Alan Ruttenberg, editors, *ICBO*, volume 833 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[Com00] Douglas E. Comer. *Internetworking with TCP/IP: Principles, protocols, and architectures*. Prentice Hall, Upper Saddle River NJ, USA, 4th edition, 2000.

[CP10] Philipp Cimiano and Helena Sofia Pinto, editors. *Knowledge Engineering and Management by the Masses - 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*, volume 6317 of *Lecture Notes in Computer Science*. Springer, 2010.

[CZ04] Gary Chartrand and Ping Zhang. *Introduction to Graph Theory*. The Walter Rudin Student Series in Advanced Mathematics. McGraw-Hill, 2004.

[DDLS01] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder policy specification language. In Morris Sloman, Jorge Lobo, and Emil Lupu, editors, *POLICY*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2001.

[dEDL09] Mathieu d'Aquin, Jérôme Euzenat, Chan Le Duc, and Holger Lewen. Sharing and reusing aligned ontologies with Cupboard. In Gil and Noy [GN09], pages 179–180.

[Den95] A.M. Denisov. Local and global uniqueness of a solution to the problem of determining a nonlinear coefficient in a system of partial differential equations. *Siberian Mathematical Journal*, 36:55–65, 1995.

[DFv02] John Davies, Dieter Fensel, and Frank van Harmelen, editors. *On-To-Knowledge: Semantic Web enabled Knowledge Management*. J. Wiley and Sons, 2002.

[dL09] Mathieu d'Aquin and Holger Lewen. Cupboard - a place to expose your ontologies to applications and the community. In Aroyo et al. [ATC+09], pages 913–918.

[dM11] Mathieu d'Aquin and Enrico Motta. Watson, more than a Semantic Web search engine. *Semantic Web*, 2(1):55–63, 2011.

[DMR11] Alicia Díaz, Regina Motz, and Edelweis Rohrer. Making ontology relationships explicit in a ontology network. In Pablo Barceló and Val Tannen, editors, *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[DRS09] Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors. *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[DS66] Norman Richard Draper and Harry Smith. *Applied regression analysis*. Wiley series in probability and mathematical statistics. Wiley, 1966.

## REFERENCES


[dSSS09] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, and Marta Sabou. Criteria and evaluation for ontology modularization techniques. In Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors, *Modular Ontologies*, volume 5445 of *Lecture Notes in Computer Science*, pages 67–89. Springer, 2009.

[DTPP09] Paul Doran, Valentina A. M. Tamma, Terry R. Payne, and Ignazio Palmisano. Using ontology modularization for efficient negotiation over ontology correspondences in MAS. In Peter McBurney, Iyad Rahwan, Simon Parsons, and Nicolas Maudet, editors, *ArgMAS*, volume 6057 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2009.

[DW03] Ian Dickinson and Michael Wooldridge. Towards practical reasoning agents for the Semantic Web. In *AAMAS*, pages 827–834. ACM, 2003.

[Ere00] Marc Ereshefsky. *The Poverty of the Linnaean Hierarchy: A Philosophical Study of Biological Taxonomy.* Cambridge Studies in Philosophy and Biology. Cambridge University Press, 2000.

[FBD+02] Dieter Fensel, Christoph Bussler, Ying Ding, Vera Kartseva, Michel Klein, Maksym Korotkiy, Borys Omelayenko, and Ronny Siebes. Semantic Web application areas. In *Proc. 7th Int. Workshop on Applications of Natural Language to Information Systems (NLDB 2002)*, Stockholm, Sweden, 2002.

[FBS02] Charles J. Fillmore, Collin F. Baker, and Hiroaki Sato. The FrameNet database and software tools. In *LREC*. European Language Resources Association, 2002.

[FGPJ97] Mariano Fernández, Asunción Gómez-Pérez, and Natalia Juristo. METHONTOLOGY: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, 1997.

[Fie00] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, Irvine, CA, United States, 2000.

[FMPA12] Javier D. Fernández, Miguel A. Martínez-Prieto, and Mario Arias. Scalable management of compressed semantic Big Data. *ERCIM News*, 2012(89), 2012.

[FO04] Roberta Ferrario and Alessandro Oltramari. Towards a computational ontology of mind. In *Formal Ontology in Information Systems, Proceedings of the International Conference FOIS 2004*, Amsterdam, The Netherlands, 2004. IOS Press.

[FvHH+01] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[Gam95] E. Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.

[Gan08] Aldo Gangemi. Norms and plans as unification criteria for social collectives. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(3):70–112, 2008.

[GCB07] Mike Graves, Adam Constabaris, and Dan Brickley. FOAF: Connecting people on the Semantic Web. *Cataloging & Classification Quarterly*, 43(3-4):191–202, 2007.

[GCCL05] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. A theoretical framework for ontology evaluation and validation. In Paolo Bouquet and Giovanni Tummarello, editors, *SWAP*, volume 166 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[GF94] Michael Gruninger and Mark S. Fox. The role of competency questions in enterprise engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.

[GG06] Roberto García and Rosa Gil. An OWL copyright ontology for semantic digital rights management. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *OTM Workshops (2)*, volume 4278 of *Lecture Notes in Computer Science*, pages 1745–1754. Springer, 2006.

[GG09] Roberto García and Rosa Gil. Copyright licenses reasoning using an OWL-DL ontology. In J. Breuker, P. Casanovas, M.C.A. Klein, and E. Francesconi, editors, *Law, Ontologies and the Semantic Web: Channelling the Legal Information Flood*, volume 188 of *Frontiers in Artificial Intelligence and Applications*, pages 145–162. IOS Press, Amsterdam, NL, 2009.

[GGM+02] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening Ontologies with DOLCE. In *Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, volume 2473 of Lecture Notes in Computer Science, page 166 ff, Sigüenza, Spain, Oct. 1–4 2002.

[GJ10] Faiez Gargouri and Wassim Jaziri. *Ontology Theory, Management, and Design: Advanced Tools and Models*. IGI Global research collection. IGI Publishing, 2010.

[GK12] Michael Grüninger and Megan Katsumi. Specifying ontology design patterns with an ontology repository. In Eva Blomqvist, Aldo Gangemi, Karl Hammar, and María del Carmen Suárez-Figueroa, editors, *WOP*, volume 929 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

## REFERENCES


[GM03] Aldo Gangemi and Peter Mika. Understanding the Semantic Web through Descriptions and Situations. In *Proc. of the International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2003)*, Catania, Italy, November 3-7 2003.

[GN09] Yolanda Gil and Natasha Fridman Noy, editors. *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), September 1-4, 2009, Redondo Beach, California, USA*. ACM, 2009.

[GP09] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on Ontologies, 2nd Ed.*, International Handbooks on Information Systems. Springer, 2009.

[GPS06] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining OWL ontologies using epsilon-connections. *J. Web Sem.*, 4(1):40–59, 2006.

[GPSF09] Asunción Gómez-Pérez and Mari Carmen Suárez-Figueroa. Scenarios for building ontology networks within the NeOn Methodology. In Gil and Noy [GN09], pages 183–184.

[Gru93a] Thomas R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In Nicola Guarino and Roberto Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[Gru93b] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[Gru09] Thomas R. Gruber. Ontology. In *Encyclopedia of Database Systems*, pages 1963–1965. Springer-Verlag, 2009.

[Guh91] Ramanathan Guha. *Contexts: a formalization and some applications*. PhD thesis, Stanford University, Stanford, CA, USA, 1991.

[HB11a] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[HB11b] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[HDG+06] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The Manchester OWL syntax. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWLED*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[HDS06] Tom Heath, John Domingue, and Paul Shabajee. User interaction and uptake challenges to successfully deploying Semantic Web technologies. In *Third International Semantic Web User Interaction Workshop (SWUI 2006), Athens, GA, USA*, 2006.

[Hen10] James A. Hendler. Web 3.0: The dawn of semantic search. *IEEE Computer*, 43(1):77–80, 2010.

[Hep08] Martin Hepp. GoodRelations: An ontology for describing products and services offers on the Web. In Aldo Gangemi and Jérôme Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2008.

[HF11] Ahmad Hawalah and Maria Fasli. Improving the mapping process in ontology-based user profiles for Web personalization systems. In Joaquim Filipe and Ana L. N. Fred, editors, *ICAART (1)*, pages 321–328. SciTePress, 2011.

[HG85] Carole D. Hafner and Kurt Godden. Portability of syntax and semantics in Datalog. *ACM Trans. Inf. Syst.*, 3(2):141–164, 1985.

[HHK+10] Bjørn Jervell Hansen, Jonas Halvorsen, Svein Ivar Kristiansen, Rolf Rasmussen, Marianne Rustad, and Geir Sletten. Recommended application areas for semantic technologies. Technical report, Norwegian Defence Research Establishment (FFI), February 2010.

[HKG09] Jörg Henss, Joachim Kleb, and Stephan Grimm. A database backend for OWL. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[HM00] James A. Hendler and Deoprah L. McGuinness. The DARPA agent markup language. *IEEE Intelligent systems*, 15(6):67–73, 2000.

[HM08] Tom Heath and Enrico Motta. Revyu: Linking reviews and ratings into the Web of Data. *J. Web Sem.*, 6(4):266–273, 2008.

[HP11] Harry Halpin and Valentina Presutti. The identity of resources on the Web: An ontology for Web architecture. *Applied Ontology*, 6(3):263–293, 2011.

[HPPR11] Aidan Hogan, Jeff Z. Pan, Axel Polleres, and Yuan Ren. Scalable OWL 2 reasoning for Linked Data. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter F. Patel-Schneider, editors, *Reasoning Web*, volume 6848 of *Lecture Notes in Computer Science*, pages 250–325. Springer, 2011.

## REFERENCES


[HW07] Peter Haase and Yimin Wang. A decentralized infrastructure for query answering over distributed ontologies. In Yookun Cho, Roger L. Wainwright, Hisham Haddad, Sung Y. Shin, and Yong Wan Koo, editors, *SAC*, pages 1351–1356. ACM, 2007.

[IRS09] Luigi Iannone, Alan Rector, and Robert Stevens. Embedding knowledge patterns into OWL. In *6th Annual European Semantic Web Conference (ESWC2009)*, pages 218–232, June 2009.

[JL96] Richard E. Jones and Rafael Dueire Lins. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley, 1996.

[KA05] Alexander Kleshchev and Irene Artemjeva. An analysis of some relations among domain ontologies. *International Journal on Information Theories and Applications*, 12(1):85–93, 2005.

[Kag02] Lalana Kagal. Rei : A Policy Language for the Me-Centric Project. Technical report, HP Labs, September 2002.

[KC06] Eunhoe Kim and Jaeyoung Choi. An ontology-based context model in a smart home. In *Computational Science and Its Applications - ICCSA 2006, International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part IV*, LNCS, pages 11–20. Springer, 2006.

[Kif08] Michael Kifer. Rule Interchange Format: The framework. In Diego Calvanese and Georg Lausen, editors, *RR*, volume 5341 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2008.

[KKMM11] Takahiko Koike, Shigeyuki Kan, Masaya Misaki, and Satoru Miyauchi. Connectivity pattern changes in default-mode network with deep non-REM and REM sleep. *Neuroscience Research*, 69(4):322 – 330, 2011.

[Kle04] Michel Christiaan Alexander Klein. *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, 2004.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.

[KSKM07] Kouji Kozaki, Eiichi Sunagawa, Yoshinobu Kitamura, and Riichiro Mizoguchi. A framework for cooperative ontology construction based on dependency management of modules. In Liming Chen, Philippe Cudré-Mauroux, Peter Haase, Andreas Hotho, and Ernie Ong, editors, *ESOE*, volume 292 of *CEUR Workshop Proceedings*, pages 33–44. CEUR-WS.org, 2007.

[Ld10] Holger Lewen and Mathieu d'Aquin. Extending open rating systems for ontology ranking and reuse. In Cimiano and Pinto [CP10], pages 441–450.

[LJ12] Alexandros Labrinidis and H. V. Jagadish. Challenges and opportunities with Big Data. *PVLDB*, 5(12):2032–2033, 2012.

[LK11] Roman Lukyanenko and Sherrie Y. X. Komiak. Designing recommendation agents as extensions of individual users: similarity and identification in web personalization. In Dennis F. Galletta and Ting-Peng Liang, editors, *ICIS*. Association for Information Systems, 2011.

[LLNW11] Thorsten Liebig, Marko Luther, Olaf Noppens, and Michael Wessel. OWLlink. *Semantic Web*, 2(1):23–32, 2011.

[LPP+10] Jihyun Lee, Jeong-Hoon Park, Myung-Jae Park, Chin-Wan Chung, and Jun-Ki Min. An intelligent query processing for distributed ontologies. *Journal of Systems and Software*, 83(1):85–95, 2010.

[LvHN05] Thorsten Liebig, Friedrich W. von Henke, and Olaf Noppens. Explanation support for OWL authoring. In Thomas Roth-Berghofer and Stefan Schulz, editors, *ExaCt*, volume FS-05-04 of *AAAI Technical Report*, pages 86–93. AAAI Press, 2005.

[Mal88] James L. Malone. *The Science of Linguistics in the Art of Translation: Some Tools from Linguistics for the Analysis and Practice of Translation*. SUNY Series in Linguistics. State University of New York Press, 1988.

[Mar83] James Martin. *Managing the data-base environment*. (The James Martin books on computer systems and telecommunications). Prentice-Hall, 1983.

[MB06] Jing Mei and Harold Boley. Interpreting SWRL rules in RDF graphs. *Electr. Notes Theor. Comput. Sci.*, 151(2):53–69, 2006.

[MBS12] Nor Azlinayati Abdul Manaf, Sean Bechhofer, and Robert Stevens. The current state of SKOS vocabularies on the Web. In Simperl et al. [SCP+12], pages 270–284.

[McB04] Brian McBride. The Resource Description Framework (RDF) and its vocabulary description language RDFS. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 51–66. Springer, 2004.

[McC93] John McCarthy. Notes on formalizing context. In *IJCAI*, pages 555–562, San Mateo, California, 1993. Morgan Kaumann.

[MFHS02] Deborah L. McGuinness, Richard Fikes, James A. Hendler, and Lynn Andrea Stein. DAML+OIL: An ontology language for the Semantic Web. *IEEE Intelligent Systems*, 17(5):72–80, 2002.

[Min81] Marvin Minsky. A framework for representing knowledge. In John Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, Cambridge, MA, 1981.

[Mit09] Prasenjit Mitra. Dewey decimal system. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 808–809. Springer US, 2009.

[MJ09] Wolfgang Maass and Sabine Janzen. A pattern-based ontology building method for ambient environments. In Eva Blomqvist, Kurt Sandkuhl, Francois Scharffe, and Vojtech Svatek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009.*, volume 516. CEUR Workshop Proceedings, 2009.

[MK12] Wolfgang Maass and Tobias Kowatsch, editors. *Semantic Technologies in Content Management Systems: Trends, Applications and Evaluations*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2012.

[MMS+03] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. An infrastructure for searching, reusing and evolving distributed ontologies. In *WWW*, pages 439–448, 2003.

[MMWB05] Alistair Miles, Brian Matthews, Michael Wilson, and Dan Brickley. SKOS Core: simple knowledge organisation for the web. In *Proceedings of the 2005 international conference on Dublin Core and metadata applications: vocabularies in practice*, DCMI '05, pages 1:1–1:9. Dublin Core Metadata Initiative, 2005.

[MTG09] Jihen Majdoubi, Mohamed Tmar, and Faïez Gargouri. Using the MeSH thesaurus to index a medical article: Combination of content, structure and semantics. In Juan D. Velásquez, Sebastián A. Ríos, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (1)*, volume 5711 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 2009.

[Mut12] Raghava Mutharaju. Very large scale OWL reasoning through distributed computation. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *International Semantic Web Conference (2)*, volume 7650 of *Lecture Notes in Computer Science*, pages 407–414. Springer, 2012.

[MW09] Yuxin Mao and Guiyi Wei. Sub-ontology modularization for large-scale Web ontologies. In *IRI*, pages 145–150. IEEE Systems, Man, and Cybernetics Society, 2009.

[NGCP10] Andrea Giovanni Nuzzolese, Aldo Gangemi, Paolo Ciancarini, and Valentina Presutti. Fine-tuning triplification with Semion. In Valentina Presutti, Francois Scharffe, and Vojtech Svatek, editors, *Proceedings of the 1st Workshop on Knowledge Injection into and Extraction from Linked Data (KIELD-2010)*, volume 632, pages 2–14. CEUR Workshop Proceedings, 2010.

[NJH01] Stuart J. Nelson, W. Douglas Johnston, and Betsy L. Humphreys. Relationships in medical subject headings. In Carol A. Bean and Rebecca Green, editors, *Relationships in the Organization of Knowledge*, pages 171–184. Kluwer Academic Publishers, New York, NY, USA, 2001.

[NM04] Natalya Fridman Noy and Mark A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[NN06] Claire Nedellec and Adeline Nazarenko. Ontologies and information extraction. *CoRR*, abs/cs/0609137, 2006.

[NRB09] Nadejda Nikitina, Sebastian Rudolph, and Sebastian Blohm. Refining ontologies by pattern-based completion. In Eva Blomqvist, Kurt Sandkuhl, Francois Scharffe, and Vojtech Svatek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009*, volume 516. CEUR Workshop Proceedings, 2009.

[NW08] M. Naftalin and P. Wadler. *Java Generics and Collections*. Java Series. O'Reilly Media, 2008.

[NWQ+02] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *WWW*, pages 604–615, 2002.

[NWS+03] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario T. Schlosser, Ingo Brunkhorst, and Alexander Löser. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *WWW*, pages 536–543, 2003.

[OGA+06] Shumao Ou, Nektarios Georgalas, Manooch Azmoodeh, Kun Yang, and Xiantang Sun. A model driven integration architecture for ontology-based context modelling and context-aware application development. In *Model Driven*

*Architecture – Foundations and Applications*, volume 4066 of *LNCS*. Springer, 2006.

[OSV05] Daniel Oberle, Steffen Staab, and Raphael Volz. Three dimensions of knowledge representation in WonderWeb. *KI*, 19(1):31–, 2005.

[OY12] Sunjoo Oh and Heon Y. Yeom. A comprehensive framework for the evaluation of ontology modularization. *Expert Syst. Appl.*, 39(10):8547–8556, 2012.

[PAA+11] Valentina Presutti, Lora Aroyo, Alessandro Adamou, Balthasar A. C. Schopman, Aldo Gangemi, and Guus Schreiber. Extracting core knowledge from linked data. In Olaf Hartig, Andreas Harth, and Juan Sequeda, editors, *COLD*, volume 782 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[PAG+11] Valentina Presutti, Lora Aroyo, Aldo Gangemi, Alessandro Adamou, Balthasar A. C. Schopman, and Guus Schreiber. A knowledge pattern-based method for linked data analysis. In Mark A. Musen and Óscar Corcho, editors, *K-CAP*, pages 173–174. ACM, 2011.

[Pal09] Raúl Palma. *Ontology Metadata Management in Distributed Environments*. PhD thesis, Universidad Politécnica de Madrid, Madrid, Spain, 2009.

[Pal11] Raman Pal. Secure Semantic Web ontology sharing. Master's thesis, University of Southampton, January 2011.

[PBDG12] Valentina Presutti, Eva Blomqvist, Enrico Daga, and Aldo Gangemi. Pattern-based ontology design. In Suárez-Figueroa et al. [SFGPMG12b], chapter 3, pages 35–64.

[PBKL06] Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. Fresnel: A browser-independent presentation vocabulary for RDF. In *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 158–171. Springer-Verlag, 2006.

[PDGB09] Valentina Presutti, Enrico Daga, Aldo Gangemi, and Eva Blomqvist. extreme design with content ontology design patterns. In Eva Blomqvist, Kurt Sandkuhl, François Scharffe, and Vojtech Svátek, editors, *WOP*, volume 516 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[Pel12] Tassilo Pellegrini. Integrating linked data into the content value chain: a review of news-related standards, methodologies and licensing requirements. In Valentina Presutti and Helena Sofia Pinto, editors, *I-SEMANTICS*, pages 94–102. ACM, 2012.

[PGG08] Davide Picca, Alfio Massimiliano Gliozzo, and Aldo Gangemi. LMM: an OWL-DL metamodel to represent heterogeneous lexical knowledge. In *LREC*. European Language Resources Association, 2008.

[PHGP06] Raúl Palma, Peter Haase, and Asunción Gómez-Pérez. Oyster: sharing and re-using ontologies in a peer-to-peer community. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *WWW*, pages 1009–1010. ACM, 2006.

[PMPdCGP06] Wim Peters, Elena Montiel-Ponsoda, Guadalupe Aguado de Cea, and Asunción Gómez-Pérez. Localizing ontologies in OWL. In *In Proceedings of the OntoLex07 Workshop at the 6th International Semantic Web Conference*, pages 13–22, 2007-06.

[PNC12] Perrine Pittet, Christophe Nicolle, and Christophe Cruz. Guidelines for a dynamic ontology - integrating tools of evolution and versioning in ontology. *CoRR*, abs/1208.1750, 2012.

[Pre12] Valentina Presutti. Essential requirements for semantic CMS. In Maass and Kowatsch [MK12], pages 91–107. 10.1007/978-3-642-24960-0_8.

[PSM08] Jinsoo Park, Kimoon Sung, and Sewon Moon. Developing graduation screen ontology based on the METHONTOLOGY approach. In Jinhwa Kim, Dursun Delen, Jinsoo Park, Franz Ko, and Yun Ji Na, editors, *NCM (2)*, pages 375–380. IEEE Computer Society, 2008.

[PVSFGP10] María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Reusing ontology design patterns in a context ontology network. In Eva Blomqvist, Vinay K. Chaudhri, Óscar Corcho, Valentina Presutti, and Kurt Sandkuhl, editors, *WOP*, volume 671 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[Rat93] Christian Rathke. Object-oriented programming and frame-based knowledge representation. In *Fifth International Conference on Tools with Artificial Intelligence*, pages 95–98, 1993.

[RDE+07] Kurt Rohloff, Mike Dean, Ian Emmons, Dorene Ryder, and John Sumner. An evaluation of triple-store technologies for large data stores. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *OTM Workshops (2)*, volume 4806 of *Lecture Notes in Computer Science*, pages 1105–1114. Springer, 2007.

[Red10] Timothy Redmond. An open source database backend for the OWL API and Protege 4. In Evren Sirin and Kendall Clark, editors, *OWLED*, volume 614 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

## REFERENCES


[RM04] Dnyanesh Rajpathak and Enrico Motta. An ontological formalization of the planning task. In *International Conference on Formal Ontology in Information Systems*, Valencia, Spain, 2004.

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

[Roh12] Edelweis Rohrer. Formal specification of ontology networks. In Simperl et al. [SCP+12], pages 818–822.

[RP88] George Rebane and Judea Pearl. The recovery of causal poly-trees from statistical data. *Int. J. Approx. Reasoning*, 2(3):341, 1988.

[RR07] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2007.

[RT04] Sandra K. Roe and Alan R. Thomas. *The Thesaurus: Review, Renaissance, and Revision*. Cataloging and Classification Quarterly Series. Haworth Press, 2004.

[RVNLE09] Marcos Martínez Romero, José Manuel Vázquez-Naya, Javier Pereira Loureiro, and Norberto Ezquerra. Ontology alignment techniques. In Juan R. Rabuñal, Julian Dorado, and Alejandro Pazos, editors, *Encyclopedia of Artificial Intelligence*, pages 1290–1295. IGI Global, 2009.

[SBLH06] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The Semantic Web revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[SBW02] Shigeo Sugimoto, Thomas Baker, and Stuart Weibel. Dublin Core: Process and principles. In Ee-Peng Lim, Schubert Foo, Christopher S. G. Khoo, Hsinchun Chen, Edward A. Fox, Shalini R. Urs, and Costantino Thanos, editors, *ICADL*, volume 2555 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 2002.

[SCG12] Amir Souissi, Walid Chainbi, and Khaled Ghédira. Proposal of a new approach for ontology modularization. In Thomas Schneider and Dirk Walther, editors, *WoMO*, volume 875 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[SCP+12] Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho, and Valentina Presutti, editors. *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, volume 7295 of *Lecture Notes in Computer Science*. Springer, 2012.

[SF10] Mari Carmen Suárez-Figueroa. *NeOn Methodology for Building Ontology Networks: Specification, Scheduling and Reuse*. PhD thesis, Universidad Politécnica de Madrid, Madrid, Spain, June 2010.

[SFGP09] Mari Carmen Suárez-Figueroa and Asunción Gómez-Pérez. NeOn methodology for building ontology networks: a scenario-based methodology. In Services & Semantic Technologies (S3T 2009) International Conference on Software, editor, *Proceedings of the International Conference on Software, Services & Semantic Technologies (S3T 2009)*, 2009.

[SFGPMG12a] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi. Introduction: Ontology engineering in a networked world. In *Ontology Engineering in a Networked World* [SFGPMG12b], chapter 1, pages 1–6.

[SFGPMG12b] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi, editors. *Ontology Engineering in a Networked World*. Springer Berlin Heidelberg, 2012.

[SHL10] Heru Agus Santoso, Su-Cheng Haw, and Chien-Sing Lee. Change detection in ontology versioning: A bottom-up approach by incorporating ontology metadata vocabulary. In Yanchun Zhang, Alfredo Cuzzocrea, Jianhua Ma, Kyo-Il Chung, Tughrul Arslan, and Xiaofeng Song, editors, *FGIT-DTA/BSBT*, volume 118 of *Communications in Computer and Information Science*, pages 37–46. Springer, 2010.

[SLL+04] Dagobert Soergel, Boris Lauser, Anita C. Liang, Frehiwot Fisseha, Johannes Keizer, and Stephen Katz. Reengineering thesauri for new applications: The agrovoc example. *J. Digit. Inf.*, 4(4), 2004.

[SM11] Martin Svoboda and Irena Mlýnková. Linked data indexing methods: A survey. In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *OTM Workshops*, volume 7046 of *Lecture Notes in Computer Science*, pages 474–483. Springer, 2011.

[Sni03] Gordon L. Snider. Nosology for our day: its application to chronic obstructive pulmonary disease. *Am. J. Respir. Crit. Care Med.*, 167(5):678–83, 2003.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[SS02] York Sure and Rudi Studer. On-To-Knowledge methodology. In Davies et al. [DFv02], chapter 3, pages 33–46.

## REFERENCES


[SS09] Anne Schlicht and Heiner Stuckenschmidt. Distributed resolution for expressive ontology networks. In Axel Polleres and Terrance Swift, editors, *RR*, volume 5837 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2009.

[ST05] Luciano Serafini and Andrei Tamilin. DRAGO: Distributed reasoning architecture for the Semantic Web. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 361–376. Springer, 2005.

[TBJ+03] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei and Ponder. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 419–437. Springer, 2003.

[TDL+11] Dhavalkumar Thakker, Vania Dimitrova, Lydia Lau, Ronald Denaux, Stan Karanasios, and Fan Yang-Turner. A priori ontology modularisation in ill-defined domains. In Chiara Ghidini, Axel-Cyrille Ngonga Ngomo, Stefanie N. Lindstaedt, and Tassilo Pellegrini, editors, *I-SEMANTICS*, ACM International Conference Proceeding Series, pages 167–170. ACM, 2011.

[UBJ+03] A. Uszok, J. M. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of Policy*, Como, Italy, June 2003. AAAI.

[vAMMS06] Mark van Assem, Véronique Malaisé, Alistair Miles, and Guus Schreiber. A method to convert thesauri to SKOS. In York Sure and John Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2006.

[Var02] Vasudeva Varma. Building large scale ontology networks. In *Language Engineering Conference*, page 121. IEEE Computer Society, 2002.

[VPST05] Denny Vrandecic, H. Sofia Pinto, York Sure, and Christoph Tempich. The DILIGENT knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, October 2005.

[VSS+08] Stijn Verstichel, Matthias Strobbe, Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. Distributed reasoning for context-aware services through design of an OWL meta-model. In *ICAS*, pages 70–75. IEEE Computer Society, 2008.

[VT12] Boris Villazón-Terrazas. *A Method for Reusing and Re-Engineering Non-Ontological Resources for Building Ontologies*. Studies on the Semantic Web. IOS Press, 2012.

[VTMH12] Kim Viljanen, Jouni Tuominen, Eetu Mäkelä, and Eero Hyvönen. Normalized access to ontology repositories. In *ICSC*, pages 109–116. IEEE Computer Society, 2012.

[WM12] Sebastian Wandelt and Ralf Möller. Towards ABox modularization of semi-expressive Description Logics. *Applied Ontology*, 7(2):133–167, 2012.

[WNR+06] Hai H. Wang, Natasha Noy, Alan Rector, Mark Musen, Timothy Redmond, Daniel Rubin, Samson Tu, Tania Tudorache, Nick Drummond, Matthew Horridge, and Julian Seidenberg. Frames and OWL side by side. In *9th International Protégé Conference*, 2006.

[WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using OWL. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:18, 2004.

# Document references


[ABB+10] Alessandro Adamou, Eva Blomqvist, Concetto Elvio Bonafede, Enrico Daga, Andrea Giovanni Nuzzolese, and Valentina Presutti. Knowledge representation and reasoning system (KReS) - alpha version report. IKS Deliverable 5.2a, IKS Consortium, August 2010.

[ABC+11] Alessandro Adamou, Eva Blomqvist, Paolo Ciancarini, Enrico Daga, Alberto Musetti, Andrea Giovanni Nuzzolese, Valentina Presutti, Sebastian Germesin, and Massimo Romanelli. Knowledge representation and reasoning system (KReS) - beta version report. IKS Deliverable 5.2b, IKS Consortium, June 2011.

[ABG+10] Alessandro Adamou, Eva Blomqvist, Aldo Gangemi, Andrea Giovanni Nuzzolese, Valentina Presutti, Wernher Behrendt, Violeta Damjanovic, and Alex Conconi. Ontological requirements for industrial CMS applications. IKS Deliverable 3.2, IKS Consortium, 2010.

[All09a] The OSGi Alliance. OSGi service platform release 4 version 4.2, compendium specification. Committee specification, Open Services Gateway initiative (OSGi), September 2009.

[All09b] The OSGi Alliance. OSGi service platform release 4 version 4.2, core specification. Committee specification, Open Services Gateway initiative (OSGi), September 2009.

[All10] The OSGi Alliance. OSGi service platform release 4 version 4.2, enterprise specification. Committee specification, Open Services Gateway initiative (OSGi), March 2010.

[BBL11] David Beckett and Tim Berners-Lee. Turtle - terse RDF triple language. W3C Team Submission, World Wide Web Consortium (W3C), March 2011.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[BEL+11] Peter Brown, Jeff A. Estefan, Ken Laskey, Francis G. McCabe, and Danny Thornton. OASIS reference architecture foundation for service oriented architecture 1.0.

Committee Specification Draft 03, Organization for the Advancement of Structured Information Standards (OASIS), July 2011.

[BG04] Dan Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[BJC11] Chris Bizer, Anja Jentzsch, and Richard Cyganiak. State of the LOD cloud. Technical report, Freie Universität Berlin, September 2011.

[BL06] Tim Berners-Lee. Design issues: Linked Data. Technical report, World Wide Web Consortium (W3C), July 2006.

[BLC11] Tim Berners-Lee and Dan Connolly. OWL web ontology language overview. Notation3 (N3): A readable RDF syntax, World Wide Web Consortium (W3C), March 2011.

[BLFM05] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic syntax. Request for Comments 3986, Internet Engineering Task Force, January 2005.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. W3C note, World Wide Web Consortium (W3C), March 2001.

[CCW06] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. Multi-tenant data architecture. Technical report, The Microsoft Developer Network (MSDN), June 2006.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath) version 1.0. W3C recommendation, World Wide Web Consortium (W3C), November 1999.

[CES+10] Fabian Christ, Gregor Engels, Stefan Sauer, Gokce B. Laleci, Erdem Alpay, Tuncay Namli, Ali Anil Sinaci, and Fulya Tuncer. Requirements specification of the horizontal use case. IKS Deliverable 2.2, IKS Consortium, 2010.

[CSS+11] Fabian Christ, Stefan Sauer, Ali Anil Sinaci, Suat Gonul, Andrea Nuzzolese, Alessandro Adamou, Enrico Daga, Alberto Musetti, Szabolcs Grünwald, and Sebastian Germesin. IKS final. IKS Deliverable 5.0, IKS Consortium, December 2011.

[DS05] Martin Duerst and Michel Suignard. Internationalized Resource Identifiers (IRIs). Request for Comments 3987, Internet Engineering Task Force, January 2005.

[FL07] Joel Farrell and Holger Lausen. Semantic annotations for WSDL and XML Schema. W3C recommendation, World Wide Web Consortium (W3C), August 2007.

[GBM04] Jan Grant, Dave Beckett, and Brian McBride. RDF test cases. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[GOH+12] Birte Glimm, Chimezie Ogbuji, Sandro Hawke, Ivan Herman, Bijan Parsia, Axel Polleres, and Andy Seaborne. SPARQL 1.1 entailment regimes. W3C Candidate Recommendation, World Wide Web Consortium (W3C), November 2012.

[GPP12] Paul Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 update. W3C Proposed Recommendation, World Wide Web Consortium (W3C), November 2012.

[Gro09] OWL Working Group. OWL 2 Web Ontology Language: Document overview. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[GWPS09] Christine Golbreich, Evan K. Wallace, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language: New features and rationale. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[HPPSR09] Pascal Hitzler, Markus Krötzsch Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language: Primer. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium (W3C), May 2004.

[MB09] Alistair Miles and Sean Bechhofer. SKOS simple knowledge organization system reference. W3C Recommendation, World Wide Web Consortium (W3C), August 2009.

[MGH+09] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, and Carsten Lutz. OWL 2 Web Ontology Language: Profiles. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[MM04] Frank Manola and Eric Miller. RDF primer. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[MPPS+12] Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, Sean Bechhofer, Bernardo Cuenca Grau, Achille Fokoue, and Rinke Hoekstra. OWL 2 Web Ontology Language: XML serialization (second edition). W3C Recommendation, World Wide Web Consortium (W3C), December 2012.

[MPSP+09] Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler,

and Mike Smith. OWL 2 Web Ontology Language: Structural specification and functional-style syntax. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language overview. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[NPJN08] Mikael Nilsson, Andy Powell, Pete Johnston, and Ambjörn Naeve. Expressing Dublin Core metadata using the Resource Description Framework (RDF). DCMI Recommendation, Dublin Core Metadata Initiative, January 2008.

[PBAS+12] Eric Prud'hommeaux, Carlos Buil-Aranda, Andy Seaborne, Axel Polleres, Lee Feigenbaum, and Gregory Todd Williams. SPARQL 1.1 federated query. W3C Proposed Recommendation, World Wide Web Consortium (W3C), November 2012.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, World Wide Web Consortium (W3C), January 2008.

[PSMG+09] Peter F. Patel-Schneider, Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, Alan Ruttenberg, and Michael Schneider. OWL 2 Web Ontology Language: Mapping to RDF graphs. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[RA07] Yves Raimond and Samer Abdallah. The Event Ontology. Technical report, Centre for Digital Music, Queen Mary University of London, October 2007.

[Sea10] Andy Seaborne. SPARQL 1.1 property paths. W3C Working Draft, World Wide Web Consortium (W3C), January 2010.

[SHK+09] Michael Smith, Ian Horrocks, Markus Krötzsch, Birte Glimm, Sandro Hawke, Matthew Horridge, Bijan Parsia, and Michael Schneider. OWL 2 Web Ontology Language: Conformance. W3C Recommendation, World Wide Web Consortium (W3C), October 2009.

[SL12] Manu Sporny and Dave Longley. RDF Graph Normalization. Draft community group specification, World Wide Web Consortium (W3C) Community Group, September 2012.

[vAGS06] Mark van Assem, Aldo Gangemi, and Guus Schreiber. RDF/OWL representation of WordNet. W3C Working Draft, World Wide Web Consortium (W3C), June 2006.

[Yil06] Burcu Yildiz. Ontology evolution and versioning, the state of the art. Technical report, Vienna University of Technology, October 2006.

# Web references


[Ali] Alignment API and Alignment Server. http://alignapi.gforge.inria.fr.

[Apaa] Apache Clerezza (incubating). http://incubator.apache.org/clerezza/.

[Apab] Apache Derby. http://db.apache.org/derby/.

[Apac] Apache Felix. http://felix.apache.org.

[Apad] The Apache License, version 2.0. http://www.apache.org/licenses/LICENSE-2.0.html.

[Apae] The Apache Software Foundation. http://apache.org.

[Apaf] Apache Solr. http://solr.apache.org.

[Apag] Apache Stanbol. http://stanbol.apache.org.

[Apah] Apache Stanbol Ontology Manager. http://stanbol.apache.org/docs/trunk/components/ontologymanager/.

[Apai] Apache Stanbol Rule Language. http://stanbol.apache.org/docs/trunk/components/rules/language.html.

[Bro] Meta S. Brown. The Big Data blasphemy: why sample? http://smartdatacollective.com/metabrown/47591/big-data-blasphemy-why-sample.

[COL] COLORE. http://code.google.com/p/colore/.

[Con] Content pattern annotation schema (ontology). http://ontologydesignpatterns.org/schemas/cpannotationschema.owl.

[CVS] Concurrent Versions System. http://cvs.nongnu.org/.

[DBL] DBLP Bibliography Database. http://dblp.l3s.de/d2r.

[DBP] DBPedia home. http://dbpedia.org.

[DBT] DBTune home. http://dbtune.org.

## WEB REFERENCES


[Des] Descriptive Ontology for Linguistic and Cognitive Engineering. http://www.loa-cnr.it/DOLCE.html.

[Dew] Dewey Decimal Classification / Linked Data. http://dewey.info.

[Dol] Peter Dolog. The Ontology for State Machines. http://people.cs.aau.dk/~dolog/fsm/.

[FaC] FaCT++. http://owl.man.ac.uk/factplusplus/.

[Fra] FrameNet project home. https://framenet.icsi.berkeley.edu.

[Fre] Free and open-source software. http://en.wikipedia.org/wiki/Free_and_open-source_software.

[Gana] Aldo Gangemi. DOLCE UltraLite. http://www.ontologydesignpatterns.org/ont/dul/DUL.owl.

[Ganb] Aldo Gangemi. Plan Ontology. http://www.loa-cnr.it/ontologies/Plans.owl.

[Ganc] Aldo Gangemi. Plan Ontology (lite version). http://www.loa-cnr.it/ontologies/PlansLite.owl.

[Gand] Aldo Gangemi. The Time-Indexed Participation ontology design pattern (submission overview). http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_participation.

[Goo] Google Knowledge Graph. http://www.google.com/insidesearch/features/search/knowledge.html.

[Her] HermiT OWL reasoner. http://www.hermit-reasoner.com.

[Int] The international classification of diseases (ICD). http://www.who.int/classifications/icd/en/.

[Isa] IsaViz: a visual authoring tool for RDF. http://www.w3.org/2001/11/IsaViz/.

[Jav] Java SE 6 HotSpot$^{TM}$ Virtual Machine garbage collection tuning. http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html.

[Jer] Jersey. http://jersey.java.net.

[JSO] JSON for Linking Data. http://json-ld.org.

[Lin] Linked Data. http://linkeddata.org.

[Lon] Longwell. http://simile.mit.edu/wiki/Longwell.

[NeOa] NeOn (Networked Ontologies). http://www.neon-project.org.

[NeOb] NeOn Toolkit. http://neon-toolkit.org.

[ont] ontologydesignpatterns.org. http://ontologydesignpatterns.org.

[Opea] The open ontology repository initiative. http://openontologyrepository.org.

[Opeb] The open ontology repository q&a. https://wiki.nci.nih.gov/pages/viewpage.action?pageId=56799816.

[Opec] Openrefine. https://github.com/OpenRefine.

[OWLa] The OWL API. http://owlapi.sourceforge.net.

[OWLb] The OWLlink API. http://owllink-owlapi.sourceforge.net.

[OWLc] The OWLOntologyID class specification in OWL API 3.x. http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLOntologyID.html.

[Pig] Piggy Bank. http://simile.mit.edu/wiki/Piggy_Bank.

[Prea] Valentina Presutti. The Agent Role ontology design pattern (submission overview). http://ontologydesignpatterns.org/wiki/Submissions:AgentRole.

[Preb] Valentina Presutti. The Classification ontology design pattern (submission overview). http://ontologydesignpatterns.org/wiki/Submissions:Classification.

[Prec] Valentina Presutti. The Object Role ontology design pattern (submission overview). http://ontologydesignpatterns.org/wiki/Submissions:Objectrole.

[Pro] The protégé ontology editor. http://protege.stanford.edu.

[Rac] Racerpro. http://www.racer-systems.com/products/racerpro/index.phtml.

[RDF] RDF/JSON. http://docs.api.talis.com/platform-api/output-types/rdf-json.

[Res] Resource Description Framework (RDF). http://www.w3.org/RDF/.

[Sch] Schema.org. http://schema.org.

[Sem] Semantic Web standards. http://semanticweb.org/wiki/Semantic_Web_standards.

[SPA] SPARQL Working Group Wiki. http://www.w3.org/2009/sparql/wiki.

[STH] Ryan Shaw, Raphaël Troncy, and Lynda Hardman. LODE: An ontology for Linking Open Descriptions of Events. http://linkedevents.org.

## WEB REFERENCES


[TONa] Thinking ONtologiES (TONES) Project home. http://www.inf.unibz.it/tones/.

[TONb] The TONES Repository. http://owl.cs.manchester.ac.uk/repository/.

[Top] TopBraid Composer. http://www.topquadrant.com/products/TB_Composer.html.

[W3Ca] All W3C Standards and Drafts. http://www.w3.org/TR/.

[W3Cb] W3C Semantic Web activity. http://www.w3.org/2001/sw/.

[Wik] Wikipedia home. http://www.wikipedia.org.

[Woo] David Wood. The state of RDF and JSON. http://www.w3.org/blog/SW/2011/09/13/the-state-of-rdf-and-json/.