Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN
INFORMATICA
Ciclo XXVI

Settore Concorsuale di afferenza: 01/B1
Settore Scientifico disciplinare: INF01

Knowledge Patterns for the Web: extraction, tranformation and reuse

Presentata da: Andrea Giovanni Nuzzolese

Coordinatore Dottorato:
Maurizio Gabbrielli

Relatore:
Paolo Ciancarini

Esame finale anno 2014

To my family.

Abstract

This thesis aims at investigating methods and software architectures for discovering what are the typical and frequently occurring structures used for organizing knowledge in the Web. We identify these structures as *Knowledge Patterns* (KPs), i.e., small, well connected units of meaning which are task-based, well-grounded, and cognitively sound. KPs are an abstraction of *frames* as introduced by Fillmore [51] and Minsky [101]. KP discovery needs to address two main research problems: the heterogeneity of sources, formats and semantics in the Web (i.e., the knowledge soup problem) and the difficulty to draw relevant boundary around data that allows to capture the meaningful knowledge with respect to a certain context (i.e., the knowledge boundary problem). Hence, we introduce two methods that provide different solutions to these two problems by tackling KP discovery from two different perspectives: (i) the transformation of *KP-like artifacts* (i.e., top-down defined artifacts that can be compared to KPs, such as FrameNet frames [11] or Ontology Design Patterns [65]) to KPs formalized as OWL2 ontologies; (ii) the bottom-up extraction of KPs by analyzing how data are organized in Linked Data. The two methods address the knowledge soup and boundary problems in different ways. The first method provides a solution to the two aforementioned problems that is based on a purely syntactic transformation step of the original source to RDF followed by a refactoring step whose aim is to add semantics to RDF by select meaningful RDF triples. The second method allows to draw boundaries around RDF in Linked Data by analyzing *type paths*. A type path is a possible route through an RDF that

takes into account the types associated to the nodes of a path. Unfortunately, type paths are not always available. In fact, Linked Data is a knowledge soup because of the heterogeneous semantics of its datasets and because of the limited intentional as well as extensional coverage of ontologies (e.g., DBpedia ontology [1], YAGO [133]) or other controlled vocabularies (e.g., SKOS [99], FOAF [28], etc.). Thus, we propose a solution for enriching Linked Data with additional axioms (e.g., `rdf:type` axioms) by exploiting the natural language available for example in annotations (e.g. `rdfs:comment`) or in corpora on which datasets in Linked Data are grounded (e.g. DBpedia is grounded on Wikipedia). Then we present K∼ore, a software architecture conceived to be the basis for developing KP discovery systems and designed according to two software architectural styles, i.e, the Component-based and REST. K∼ore is the architectural binding of a set of tools, i.e., K∼tools, which implements the methods for KP transformation and extraction. Finally we provide an example of reuse of KP based on Aemoo, an exploratory search tool which exploits KPs for performing entity summarization.

[1] `http://dbpedia.org/ontology`

Acknowledgements

Now that my Ph.D. is going to an end and this dissertation is finalized it is time to write acknowledgements. I know that, as it usually happens in writing acknowledgements, I will miss someone whose support has been very important during these years, but I am sure that they will understand that these acknowledgements are also for them.

First of all, I would like to thank Pamela for her love that has made my life marvelous. This achievement is mine and yours as well.

I would like to thank my parents that always and unconditionally endured, supported and encouraged me in everything.

A big thanks to my brother Paolo who introduced me in Computer Science some years ago and gave me, together with his wife Erika, my wonderful nephew Aurora.

I would like to express my deep gratitude to my tutors, prof. Aldo Gangemi and dr. Valentina Presutti, who have involved me in their extraordinary research group and who have patiently guided and encouraged me during my Ph.D.

I would like to offer my special thanks to my advisor, prof. Paolo Ciancarini, for his frank, valuable and constructive suggestions and useful critiques to my research activities.

I wish to acknowledge prof. Paola Mello for having always been ready to discuss with me about my Ph.D. topics.

My grateful thanks are also extended to the referees, i.e., prof Enrico Motta, prof. Lora Aroyo and prof. Robert Tolksdorf, for their precious and careful comments and

advises.

I would also like to extend my thanks to all the people that during these years have been part of my research group, namely, Alberto Musetti, Francesco Draicchio, Silvio Peroni, Angelo Di Iorio, Enrico Daga, Alessandro Adamou, Eva Blomqvist, Diego Reforgiato, Sergio Consoli, Daria Spampinato and Stefania Capotosti.

Special thanks go to the people who shared with me this hard but amazing Ph.D. program, namely, Ornela Dardha, Alexandru Tudor Lascu, Giulio Pellitta, Francesco Poggi, Roberto Amadini and Gioele Barabucci.

Another big thanks to Michele, Katia and Alfonso for their cheerfulness and the great fun we have had so far and we will still have.

Last but not least, I would like to thank all my friends that during these years have been simply friends.

Contents

List of Tables

List of Figures

List of Publications

The following is the list of peer-reviewed articles that have been published at conferences and workshops so far during the Ph.D. program.

- **A. G. Nuzzolese**, A. Gangemi, V. Presutti, P. Ciancarini. Towards the Natural Ontology of Wikipedia. In *International Semantic Web Conference (Posters & Demos)*. CEUR-WS, pp. 273-276, Sydney, New South Wales, Australia. 2013

- A. Gangemi, F.Draicchio, V. Presutti, **A. G. Nuzzolese**, D. Reforgiato. A Machine Reader for the Semantic Web. In *International Semantic Web Conference (Posters & Demos)*. CEUR-WS, pp. 149-152, Sydney, New South Wales, Australia. 2013

- **A. G. Nuzzolese**, A. Gangemi, V. Presutti, F.Draicchio, A. Musetti, P. Ciancarini. Tìpalo: A Tool for Automatic Typing of DBpedia Entities. In *Proceedings of the 10th Extended Semantic Web Conference*. Springer, pp. 253-257, Montpellier, France. 2013

- F.Draicchio, A. Gangemi, V. Presutti, **A. G. Nuzzolese**. FRED: From Natural Language Text to RDF and OWL in One Click. In *Proceedings of the 10th Extended Semantic Web Conference*. Springer, pp. 263-267, Montpellier, France. 2013

- A. Di Iorio, **A. G. Nuzzolese**, S. Peroni. Identifying Functions of Citations with CiTalO. In *Proceedings of the 10th Extended Semantic Web Conference*. Springer, pp. 231-235, Montpellier, France. 2013

- A. Di Iorio, **A. G. Nuzzolese**, S. Peroni. Characterising Citations in Scholarly Documents: The CiTalO Framework. In *Proceedings of the 10th Extended Semantic Web Conference*. Springer, pp. 66-77, Montpellier, France. 2013

- A. Di Iorio, **A. G. Nuzzolese**, S. Peroni. Towards the automatic identification of the nature of citations. In *Proceedings of the 3rd Workshop on Semantic Publishing (SePublica 2013) of the 10th Extended Semantic Web Conference*. CEUR-WS, pp. 63-74, Montpellier, France. 2013

- **A. G. Nuzzolese**, V. Presutti, A. Gangemi, A. Musetti, P. Ciancarini. Aemoo: exploring knowledge on the web, In: *Proceedings of the 5th Annual ACM Web Science Conference*. ACM, pp. 272-275, Paris, France, 2013.

- A. Gangemi, **A. G. Nuzzolese**, V. Presutti, F. Draicchio, A. Musetti, P. Ciancarini. Automatic typing of DBpedia entities. In: *J. Heflin, A. Bernstein, P. Cudré-Mauroux, editors, Proceedings of the 11th International Semantic Web Conference (ISWC2012)*. Springer, pp. 65-91, Boston, Massachusetts, US, 2012.

- **A. G. Nuzzolese**. Knowledge Pattern Extraction and their usage in Exploratory Search. In: *J. Heflin, A. Bernstein, P. Cudré-Mauroux, editors, Proceedings of the 11th International Semantic Web Conference (ISWC2012)*. Springer, pp. 449-452, Boston, Massachusetts, US, 2012.

- **A. G. Nuzzolese**, A. Gangemi, V. Presutti, P. Ciancarini. Type Inference through the Analysis of Wikipedia Links. In: *C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, editors, Proceedings of the WWW workshop on Linked Data on the Web (LDOW2012)*. CEUR-WS, Lyon, France, 2012

- **A. G. Nuzzolese**, A. Gangemi, V. Presutti, P. Ciancarini. Encyclopedic knowledge patterns from wikipedia links. In: *L. Aroyo, N. Noy, C. Welty, editors, Proceedings of the 10th International Semantic Web Conference (ISWC2011)*. Springer, pp. 520-536, Bonn, Germany, 2011.

- **A. G. Nuzzolese**, A. Gangemi, and V. Presutti. Gathering Lexical Linked Data and Knowledge Patterns from FrameNet. In *M. Musen, O. Corcho, editors, Proceedings of the 6th International Conference on Knowledge Capture (K-CAP)*, pp. 41-48. ACM, Banff, Alberta, Canada, 2011.

Chapter 1

Introduction

In the vision of the Semantic Web agents are supposed to interact with Web knowledge in order to help humans in solving knowledge-intensive tasks. Though Linked Data is a breakthrough in Semantic Web it is still hard to build contextualized views over data, which would allow to select relevant knowledge for a specific purpose, i.e., to draw relevant boundaries around data. Let's suppose we are interested in events involving Arnold Schwarzenegger in the artistic context. For example, the movies in which Arnold Schwarzenegger starred before starting his political career.

There are state of the art tools that provide semantic mash-ups, for example Sig.ma [1] [136] provides a view on the knowledge available in the Web of Data about Arnold Schwarzenegger as shown in Figure 1. The mash-up proposes all the RDF triples from Linked Data for the selected topic (i.e., Arnold Schwarzenegger). However, it is difficult to select the RDF triples that are important in Arnold Schwarzenegger's artistic context.

In fact, a system should be able to recognize "starring" situations over the knowledge about Arnold Schwarzenegger available on the Web. Such situations are represented as structures that relate entities and concepts according to a unifying view, e.g., Arnold Schwarzenegger having the role of actor in movies during a time period.

[1] Sig.ma: `http://sig.ma`

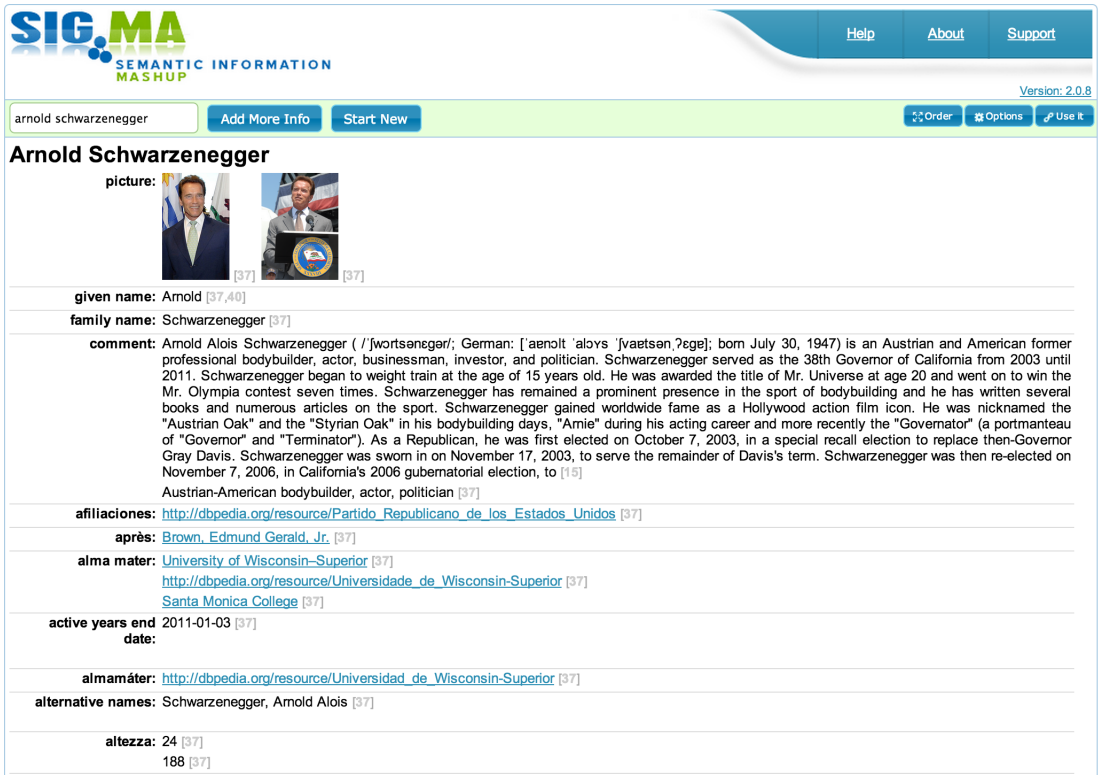


Figure 1.1: The semantic mash-up proposed by Sig.ma for the entity "Arnold Schwarzenegger".

Such structures can be exploited for supporting a variety of tasks at the knowledge level (in the sense of Newell [103]), such as decision support, content recommendation, exploratory search, content summarization, question answering, information visualization, interface design, etc. These knowledge structures have been identified and described by Fillmore [51] and Minsky [101], who proposed to conceptualize them as *frames*. Frames, known as *Knowledge Patterns* (KPs), have been reproposed in Semantic Web [66]. A KP can be informally defined as "a formalized schema representing a structure that is used to organize our knowledge, as well as for interpreting, processing or anticipating information". KPs would allow to view Linked Data under a common unifying view. In our opinion the need of unifying views is getting popular also in the mainstream Web. For example Google Search is starting to provide mash-up snapshots about search topics by exploiting the Knowl-

edge Graph. Figure 1 shows the summarization proposed for the entity "Arnold Schwarzenegger" by Google Search.

Figure 1.2: Information from the Google Knowledge Graph for the entity "Arnold Schwarzenegger".

The aim of this thesis is to formalize and implement methods for enabling KP discovery from the Web of Data. The main problems we have to address are two and have been identified in the KP vision proposed by[66], i.e.:

- the *knowledge soup problem*. The Web of Data is a knowledge soup because because of the heterogeneous semantics of its datasets: real world facts (e.g. geo data), conceptual structures (e.g. thesauri, schemes), lexical and linguistic

data (e.g. wordnets, triples inferred from NLP algorithms), social data about data (e.g. provenance and trust data), etc.;

- the *boundary problem*. How to establish the boundary of a set of triples that makes them meaningful, i.e. relevant in context, so that they constitute a KP? How do the very different types of data (e.g. natural language structures, RDFa, database tables, etc.) that are used by semantic web techniques contribute to carve out that boundary?

Our research is mainly empiric, as KPs are empirical objects [52, 66]. Furthermore, with Linked Data, for the first time in the history of knowledge engineering, we have a large set of realistic data, created by large communities of practice, on which experiments can be performed, so that the semantic web can be founded as an empirical science. This means that the methods we formalize in this dissertation enable us to make KPs emerge from the Web of Data by looking for recurrent data organization schemata. Discovered KPs will be formalized as OWL2 ontologies and published into a catalogue (i.e. ontologydesignpatterns.org) for reuse. The reuse is generally the main requirement at the base of the need for patterns in software engineering as well in knowledge engineering. Whereas there are existing resources that make available artifacts that can be compared to KPs we want to provide a method that allows their formalization as OWL2 ontologies. We call these artifacts *KP-like artifacts* and they are, for example, FrameNet [11] frames, Ontology Design Patterns [65] (ODPs) in the ODP portal [2], or Components in the Component Library [13] (CLIB). KP-like artifacts are generally defined with a top-down approach, hence their nature is not empiric. An important research direction that KP discovery will enable is the validation of top-down defined KPs with respect to those emerging in bottom-up fashion from data.

The structure of the dissertation is the following:

Chapter 2 - Background. This Chapter outlines the seminal work, bodies of standards, including a quick introduction to the Semantic Web. Mentions of related

[2]ODP portal: `http://www.ontologydesignpatterns.org`.

work are also featured in other chapters.

Chapter 3 - Knowledge Patterns for the Web. Knowledge Patterns (KPs) are extensively introduced in this Chapter. Here we provide a definition for KP, we outline the literature about KPs and we identify the sources that we want to use for KP discovery.

Chapter 4 - Knowledge Pattern transformation from KP-like sources. We propose a method for the transformation of KP-like artifacts expressed in heterogeneous formats and semantics to KP formalized as OWL2 ontologies.

Chapter 5 - Knowledge Pattern extraction from the Web. Linking things to other things is a typical cognitive action performed used by humans on the Web for organizing knowledge. In this chapter we show how to use links among Linked Data for KP extraction.

Chapter 6 - Enrichment of sources for Knowledge Pattern extraction. In some cases Linked Data cannot be sufficient for KP extraction. For example, the limited extensional as well intentional coverage of Linked Data ontologies and vocabularies is a problem that is part of the knowledge soup of the Web of Data. In this chapter we present a method for solving this issue by exploiting the richness of the natural language used for describing things il Linked Data. For example, descriptions available in annotations or in textual corpora that are eventually related to Linked Data (e.g., Wikipedia [3].

Chapter 7 - A software architecture for KP discovery and reuse. In this Chapter we focus on K∼ore and K∼tools. K∼ore is a software architecture we have designed for addressing KP discovery. K∼tools are a set of tools that implement K∼ore.

[3]Wikipedia: `http://en.wikipedia.org/`.

Chapter 8 - Aemoo: Exploratory search based on Knowledge Patterns. We present a tool, i.e., Aemoo [108] [4] which implements an exploratory search system based on the exploitation of KPs. Hence, we provide an example of KP reuse in KP-aware appliaction.

Chapter 8 - Conclusion and future work. In this Chapter we outline final remarks and ideas for future work.

[4] Aemoo: `http://www.aemoo.org`

Chapter 2

Background

In this Chapter we introduce the background that this work is based on.

2.1 The Semantic Web

In the early sixties the concept of Semantic Network Model (SNM) emerged from different communities like cognitive sciences [38], linguistics [120] and psychology [39]. A SNM was conceptually introduced as a means for representing semantically structured knowledge. In 2001, Berners-Lee, Hendler and Lassila published an article [18] that followed the direction outlined by SNMs and anticipated an ongoing and foreseen transformation of the Web as it was known then. Namely, this vision extended the network of hyperlinked human-readable Web pages by inserting machine-readable metadata about pages and how they were related to each other. The term coined for this vision was **Semantic Web**, meaning in the authors' own words:

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

The Semantic Web is a vision for a Web in which computers become capable of analyzing all the data, the contents, the links, and the transactions between people and computers. 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. In fact, the Semantic Web brings structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. For example, an agent coming to the clinic's Web page will know not just that the page has keywords such as "treatment, medicine, physical, therapy" (as might be encoded in HTML) but also that a certain Dr. Hartman works at this clinic on Mondays, Wednesdays and Fridays and that a script takes a date range in yyyy-mm-dd format and returns appointment times.

Another simple example is about 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, while 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 [30, 49, 25, 72].

Architecturally the Semantic Web is thought to be an extension of the traditional Web. Figure 2.1 shows 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)** [17];

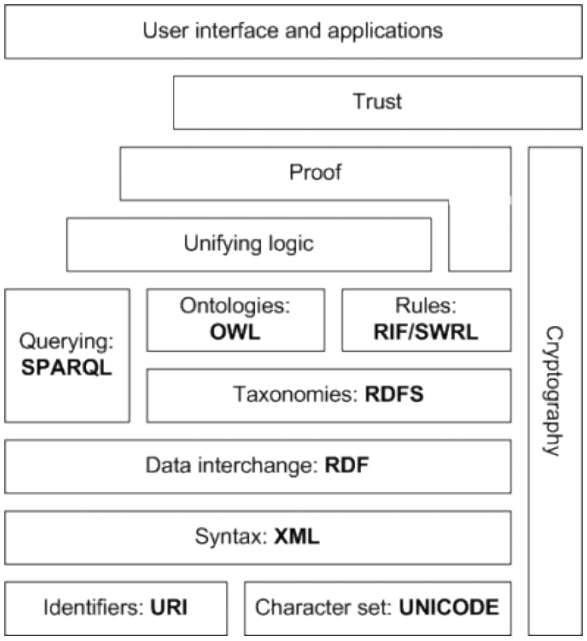


Figure 2.1: Status of the Semantic Web stack implementation as of 2013.

- a data interchange format based on a simple linguistic paradigm (**RDF**) [93];
- a vocabulary for the above interchange format that allows a simple organization form for knowledge (**RDFS**) [27];
- languages for representing complex knowledge (**OWL**) [2] and inference rules for execution (**SWRL**) [82] and interchange (**RIF**) [85];
- a language for querying data in the agreed interchange format (**SPARQL**) [119].

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.

[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 [144].

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 [109] and interactive [76] 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.

Figure 2.1 shows the architectural stack, which includes a set of high-level knowledge representation structures. 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 of ontologies and their evolving trend towards networked, interconnected structures has encouraged us to further study and support this field in the present work.

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) and curating cross-links between things [16]. 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" [2].

[2]Linked Data: `http://linkeddata.org`

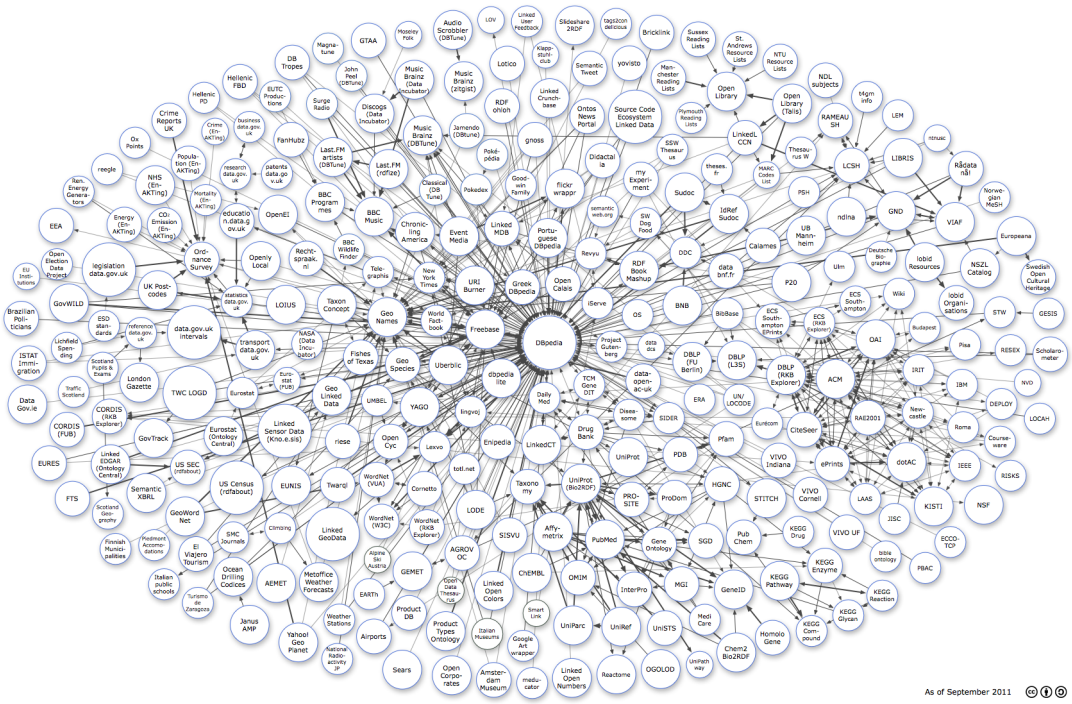


Figure 2.2: The latest graphical snapshot taken by the Linking Open Data community.

The most recent report from the LOD group is summarized by 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 [20].

2.2 Ontologies and Ontology Design Patterns

Historically ontology, listed as part of metaphysics, is the philosophical study of the nature of being, becoming, existence, or reality, as well as the basic categories of being and their relations. Ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences.

While the term ontology has been rather confined to the philosophical sphere in the recent past, it is now gaining a specific role in a variety of fields of Computer Science, such as Artificial Intelligence, Computational Linguistics, and Database Theory and Semantic Web. In Computer Science the term loses part of its metaphysical background and, still keeping a general expectation that the features of the model in an ontology should closely resemble the real world, it is referred as a formal model consisting of a set of types, properties, and relationship types aimed at modeling objects in a certain domain or in the world. In early '90s Gruber [68] gave an initial and widely accepted definition:

> *an ontology is a formal, explicit specification of a shared conceptualisation. An ontology is a description (like a formal specification of a program) of the concepts and relationships that can formally exist for an agent or a community of agents*

According to Guarino [71] a conceptualization

> *contains many "world structures", one for each world. It has both extensional and intentional components*

The initial definition of ontology was further elaborated upon by Gruber, who in 2009 wrote [69]:

> *...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.

Computational ontologies in the context of information systems are artifacts that encode a description of some world (actual, possible, counterfactual, impossible, desired, etc.), for some purpose. They have a (primarily logical) structure, and must match both domain and task: they allow the description of entities whose attributes and relations are of concern because of their relevance in a domain for some purpose, e.g. query, search, integration, matching, explanation, etc. [65] Ontology in classical knowledge bases are typically composed by a terminological component and an assertional component, i.e., TBox, and ABox respectively. The TBox specifies the terminology used for modeling a specified conceptualisation of the world. The ABox expresses TBox-compliant statements that describe the population of that world. The distinction between TBox and ABox is not strict and still an open issue in knowledge representation being a matter of distinguishing between *intensional*, i.e., part of the TBox, and *extensional*, i.e., part of the ABox, knowledge [131].

Ontologies can be distinguished in terms of the level of knowledge they capture and represent. Traditionally there are two macro-categories of ontologies: (i) foundational or top-level ontologies and (ii) domain or lower-level ontologies.

2.2.1 Ontology Design Patterns

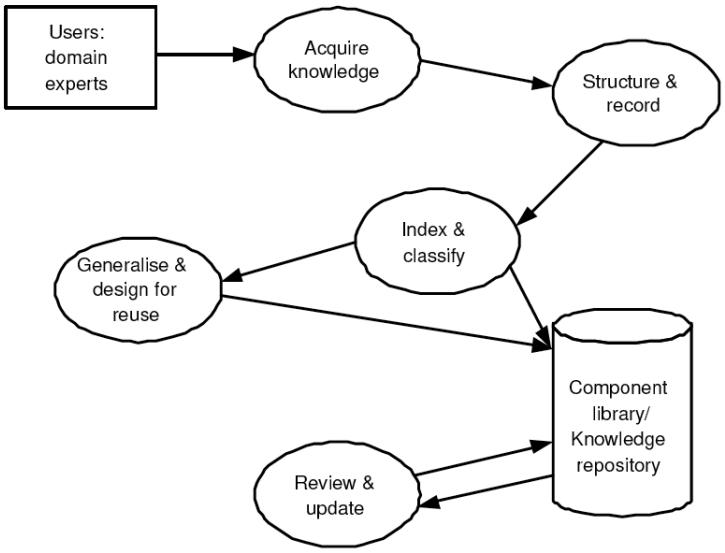
Before introducing the notions of design patterns and ontology design patterns it is useful to focus on the main motivation that drove forward design patterns and, then, ontology design patterns. This motivation is the need for reusable solution archetypes at design time. The concept of reuse has been deeply investigated in literature, especially in Software Engineering [110, 84, 86]. The main benefit of reuse is to improve the quality of a work by reducing the complexity of the task and thereby possibly also the time and effort needed to perform it. The concept of reuse plays a crucial role in software engineering processes and it is a practice that a good designer should adopt for optimizing the quality of the software. Sutcliffe in [110]

gives an overview about the concept of reuse: why it is so desirable, but also why it is a so hard to achieve. The author states that the real motivators for reuse are:

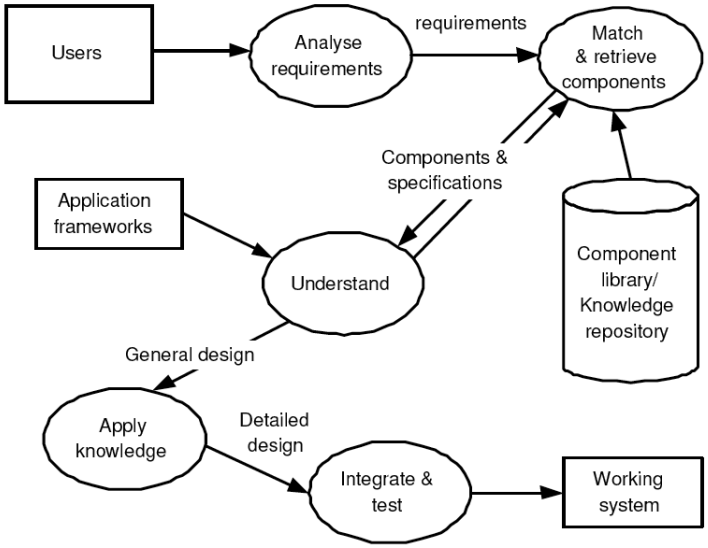
- to share good ideas;
- to save time and costs by developing products from ready-made components;
- to improve the design process by reuse of good ideas and templates;
- to improve quality by certified, reliable components.

The main cause of failure of a reuse process seems to depend on social and behavioural aspects related to the designer, e.g., the lack of motivation among designers. The effects of the lack of reuse might result devastating and compromise the final success of a software. For example, it is disheartening the tendency towards "reinventing the wheel" of many designer based on the belief that re-design solutions is inherently better suited, socially convenient, more secure or more controlled than reusing existing solutions. The reuse process according to Sutcliffe [110] is like a coin whose faces are the "design by reuse" process and the "'design for reuse" process (cf. figure 2.3). On one hand, the design for reuse is the process which expects the generation of reusable artifacts and the population of knowledge base of reusable artifacts by users (typically domain experts). On the other hand, the design by reuse is the process which expects the reuse of the artifacts that comply with certain requirements and are available in a shared knowledge repository of components. Hence, both processes are complementary as they contribute together in the achievement of the reuse of knowledge,

The term *design pattern* was introduced in the seventies by the architect and mathematician Christopher Alexander. Alexander in [4] argues that a good architectural design can be achieved by means of a set of rules that are packaged in the form of patterns, such as "courtyards which live", "windows place", or "entrance room". Design patterns are then assumed as archetypal solutions to common and frequently occurring design problems. In this idea the notion of reuse of patterns is implicit. In fact, the architectural task that designers have to address can be easily



(a) The design for reuse process.



(b) The design by reuse process.

Figure 2.3: The two faces of the coin consisting in the reuse process [110].

solved by reusing and applying solutions that have commonly accepted, validated and recommended for the same problem.

Software Engineering [57, 143, 96] has eagerly borrowed this notion of design patterns (especially in the scope of software reuse) in order to support good design practice, to teach how to design good software, and to provide standards for software design. Some authors state that design patterns exist because they emerge from the experience and the practice of designers in a sort of bottom-up fashion. For example, Gamma et al. [57] argue that

> *... It is likely that designer do not think about the notation they are using for recording the design. Rather, they look for patterns to match against plans, algorithms, data structures, and idioms they have learned in the past. Good designer, it appears, rely on large amount of desing experience, and this experience is just as important as the notations for recording designs and the rules for using those notations...*

We bear out this observation as a general hypothesis we espouse in this thesis is the cognitive grounding of using patterns for organizing knowledge by humans. Evidence of these patterns can emerge from a variety of different cognitive tasks, such as software design, ontology design, knowledge organization, knowledge interaction, etc.

Coming back to the notion of design patten in literature, we observe how the intended meaning of a design pattern coming from different area, e.g., architecture, software engineering, etc.), is stated as a reusable solution to a commonly occurring design problem within a given domain or context. It is important to remark that design patterns in software engineering do not specify implementation details, but give only to the designer an abstract solution schema, which in turn can be adopted for implementing a multiplicity of software systems that share the same design problems. This ensure wide applicability and reuse of patterns in software engineering especially for object-oriented design.

Ontologies are artifacts that encode a description of some world. Like any artifact, ontologies have a lifecycle: they are designed, implemented, evaluated, fixed,

exploited, reused, etc. Despite the original ontology engineering approach, when ontologies were seen as "portable" components [68], and its enormous impact on SemanticWeb and interoperability, one of the most challenging areas of ontology design is reusability [65]. Presutti et al. [118] define an ontology design pattern (ODP) as

> *... a modeling solution to solve a recurrent ontology design problem. It is an dul:InformationObject that dul:expresses a DesignPatternSchema (or skin). Such schema can only be satisfied by DesignSolutions. Design solutions provide the setting for oddata:OntologyElements that play some ElementRole(s) from the schema. ...*

In the definition above `dul` and `oddata` identify the namespace prefixes which belong to the DOLCE+DnS ontology [3] [60] and to the C-ODO ontology [4] [61] respectively. An information object is a piece of information encoded in some language, and a design pattern schema is the description of an ontology design pattern. Several types of ODPs have been identified so far [118, 65], and basically they can be grouped into six families (cf. Figure 2.4): Structural OPs, Correspondence OPs, Content OPs (CPs), Reasoning OPs, Presentation OPs, and Lexico- Syntactic OPs. Lexico-syntactic patterns allows to generalize and extract some conclusions about the meaning expressed by language constructs. They connect language constructs, mainly in natural language, to ontological constructs and consist of certain types of words that follow a specific order. They are formalized basing on some notations used for describing the syntax of languages, e.g., the BNF notation, and have been proposed by Hearst [74]. Structural pattern can be either logical patterns or architectural patterns. Logical patterns are aimed at specifying certain logical structures that help to overcome design expressivity problems directly connected with the lack or the limitation of the primitives of the representation language, e.g., it does not directly support certain logical constructs. If the representation language is OWL,

[3] DOLCE+DnS: `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`
[4] C-ODO: `http://www.loa.istc.cnr.it/ontologies/OD/odSolutions.owl`

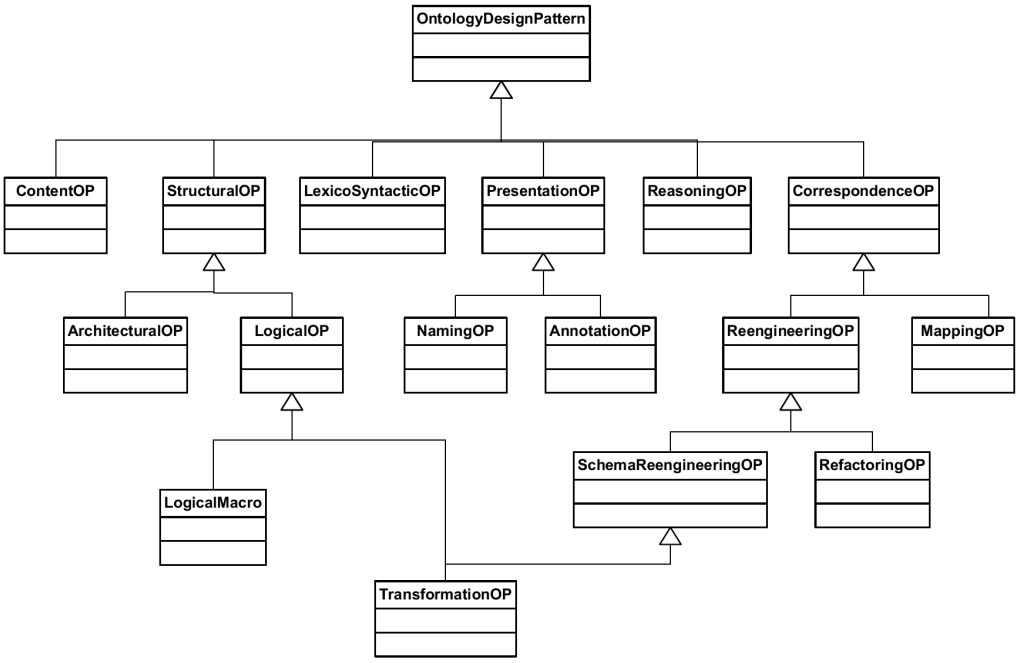


Figure 2.4: ODPs families as defined by [118].

the canonical example of a logical pattern is the n-ary relation pattern (cf. figure 2.5), which overcomes the intrinsic limitation of the OWL language primitives of representing only binary relations. Architectural patterns are defined in terms of composition of logiacal patterns and they address the problem of helping designer in expressing how the ontology they are modeling should look like. Example of architectural patterns are the Taxonomy and the Modular Architecture. Presentation patterns deal with usability and readability of ontologies from a user perspective. They include naming conventions an annotation schemas for ontologies, and are to be seen as good practices on how to document ontologies and their elements. Reasoning patterns provide facilities in order to obtain certain kind of reasoning over ontologies, such as classification, subsumption, innheritance, etc. Reasoning patterns have been also called normalizations by Vrandečić [141]. Correspondence patterns can be reengineering patterns or mapping patterns, where reengineering patterns focus on correspondences between different formalisms for transforming some source model,

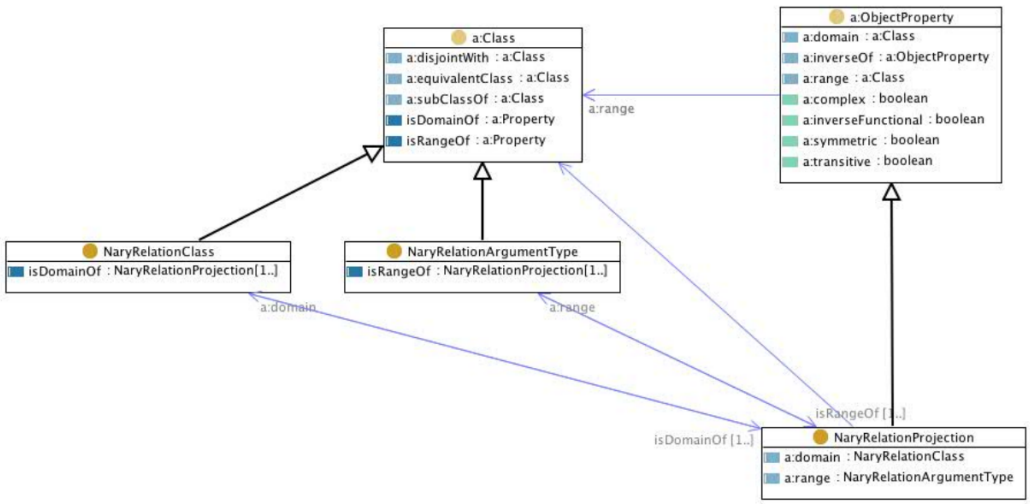


Figure 2.5: The N-ary relation logical pattern expressed with UML notation.

even a non-ontological model, into another ontology representation. Mapping patterns on the other hand are related to the possible correspondences between two or more ontologies, as investigated by Scharffe et al. [125]. Content patterns encode conceptual, rather than logical design patterns. In other words, while logical patterns solve design problems independently of a particular conceptualization, content patterns propose patterns for solving design problems for the domain classes and properties that populate an ontology, therefore addressing content problems [58]. modeling. Content Patterns help in solving problems coded as specific tasks (or use case) in specific domains. A certain domain may deal with several use cases, which can be seen as different scenarios. Similarly a certain use case can be found in different domains, e.g., different domains with a same "expert finding" scenario. A typical way of capturing use cases is by means of so called competency questions [70]. Competency questions are aimed at validating the association between content patterns and specific tasks. The relation between competency questions and contente patterns is strict. In fact, Gangemi and Presutti [65] define content patterns based on their ability to address a specific set of competency questions, which represent the problem they provide a solution for. An example of a content pattern

is the agent role (cf. figure 2.6). The pattern states allows to model ontologies that require to classify agents by their roles and to. The agent role content pattern is a specialization of the object role pattern.

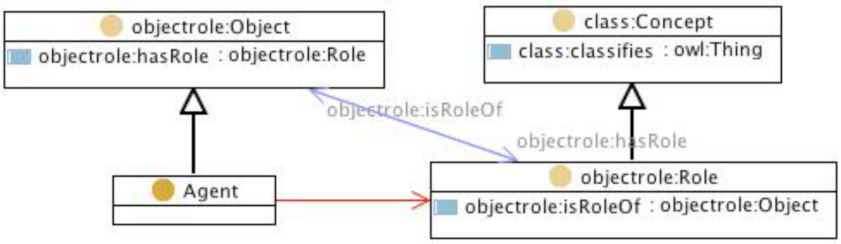


Figure 2.6: The Agent-Role content pattern expressed with UML notation.

2.2.2 Pattern-based methodologies

ODPs enable *pattern-based methodologies* in ontology engineering. These methodologies formalize approaches and provides facilities aimed at the extensive re-use of ODPs for modeling ontologies. For example, ODPs can reused by means of their specialization or composition according to their types and to the scope of the new ontology that is going to be modeled.

The Ontology Pre-Processor Language (OPPL) [45] is a domain-specific language based on the Manchester OWL Syntax [80]. OPPL allows to specify modeling instructions (macros) in order to manipulate ontologies in terms of add/remove axioms on to/from entities. The result of the composition of this axioms on entities are called modelling modules, i.e., ODPs, that can be applied or re-used in ontology modeling processes.

The eXtreme Design (XD) methodology [115, 22] is another state of the art pattern-based design methodology that adopts competency questions as a reference source for the requirement analysis. XD associate ODPs with generic use cases (cf. figure 2.7) in the solution space. Instead, the problem space is composed of local use cases that provide descriptions of the actual issues. Use cases represent problems to

which ODPs provide solutions for. Both global and local use cases are competency questions expressed in natural language. The separation of use cases and the way in which the latter are expressed make possible to match local use cases against global use cases. This matching conveys suitable ODPs to be exploited for solving modelling problems.

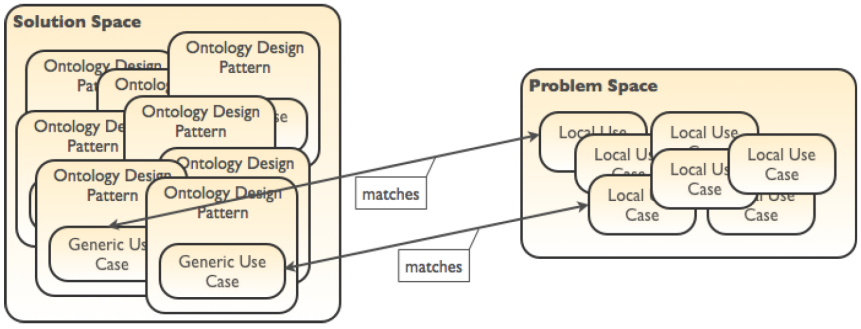


Figure 2.7: The eXtreme Design methodology [115].

Finally, the Pattern-based Ontology Building Method for Ambient Environments (POnA) [91] consists of four engineering phases: Requirements, Design, Implementation and Test. Each phase is subdivided into activities, contains decision points, and provides clearly defined outcomes. The re-use of ODPs is integrated into the design phase. Again, the more suitable ODPs for modeling an ontology for a specific problem are selected during the design phase by exploiting competency questions. Competency questions that outline problems are matched against those associated to Prototypical Ontology Design Patterns (PODPs), which are grounded to ODPs.

2.3 Ontology Mining

Research focusing on feeding the Web of Data is typically centered on extracting knowledge from structured sources and transforming it into Linked Data. Notably, [88] describes how DBpedia is extracted from Wikipedia, and its linking to other Web datasets.

Another perspective is to apply knowledge engineering principles to linked data in order to improve its quality. [132] presents YAGO, an ontology extracted from Wikipedia categories and infoboxes that has been combined with taxonomic relations from WordNet. Here the approach can be described as a reengineering task for transforming a thesaurus, i.e. Wikipedia category taxonomy, to an ontology, which required accurate ontological analysis.

Relevant research has been conducted in Ontology learning and population typically by hybridizing Natural Language Processing (NLP) and Machine Learning (ML) methods. These approaches usually require large corpora, sometimes manually annotated, in order to induce a set of probabilistic learning rules. [42] provides an exhaustive survey of *Ontology Mining* (OM) techniques. OM includes Ontology Learning (OL). OL research aims to develop algorithms that extract ontological elements from different kinds of input, i.e. "learning" since many approaches apply some kind of machine learning (ML), and semi-automatically compose an ontology from those elements.

Maedche [92] and Cimiano [32] argue that OL is composed of a set of methods and algorithms which enable to extract more expressive elements and include them in a proposed ontology.

Most of the OL rely on techniques developed in other research fields, e.g., Natural Language Processing, Machine Learning, Data Mining, etc. Some of the most important basic ideas are presented below in brief.

The text expressed as natural language is a typical input for extracting ontological terms. Most commonly NLP-based methods start by extracting terms according to their frequency of occurrence in the texts, although this frequency count is usually modified to represent some notion of *relevance*. For example Navigli et al. [102] use a classical TFIDF-measure [10] (term frequency, inverse document frequency) from Information Retrieval (IR) in order to filter out words that are common in all texts, thus not domain specific, concepts that are only used in one single document and not in the whole corpus, and terms that are simply not frequent enough.

Different approaches to term detection are those based on recognizing existing

linguistic patterns that are part of important linguistic constructs [145] and those based on discovering multi-word terms. About the latter a relevant example is [54] that proposes C/NC-value method, which enables to assess the "termhood" of a set of words by studying both its frequency in the text, but also its occurrence as a subset of of other candidate terms, the number of such occurrences, and the length of the candidate term. The NC-value in addition incorporates the term contexts, surrounding words, in the assessment. By using such methods not only single word terms but compound terms can be extracted, as candidate lexical realisations of concepts.

Another important problem of OM is the synonym detection, i.e., the problem of clustering terms into sets of synonyms. WordNet [48] is the most used resource for synonym detection. WordNet collects terms in so called "synsets". A synset is a collection of terms that have a similar meaning in a certain context. Until recently concept formation has been mostly seen as the process of term clustering and synonym detection. Recent approaches have however attempted to extract more complex concept definitions from text. An example of such a method is the LExO method suggested by Völker [138] where complex concept definitions and restrictions are extracted from natural language definitions and descriptions of terms, for example in sentences extracted from dictionaries.

Another problem in OM that is the taxonomy induction. The taxonomy induction is the task of inducing a taxonomy for example starting from a natural language text or from a given ontology, i.e. extract subsumption relations between the formed concepts. One of the first solution proposed to taxonomy induction is SVETLAN [43], which divides a text into into so called thematic units, i.e. different contexts. The sentences in the texts are parsed and specific patterns of words are extracted, the nouns in these sentence-parts are then aggregated into groups depending on their use with the same verb and the thematic unit or group of thematic units it belongs to. Gamallo et al. [56] use a similar method, where sentences are parsed, syntactic dependencies extracted, which can then be translated into semantic relations using predefined interpretation rules. Formal concept analysis (FCA)

is another approach that has been used in addition to the similarity and clustering techniques listed above, as for example presented by Cimiano et al. [33], and Völker and Rudolph [140]. In order to apply FCA for hierarchy induction the verbs used in connection with the terms, representing the concepts, to be ordered are collected as attributes of the term. Applying FCA on these attribute vectors will construct a concept lattice, which is transformed to a concept hierarchy, where the leaves are the terms and the intermediate nodes are named by the verbs applicable to the terms below the node in the concept hierarchy.

A very common approach to relation extraction is the used of lexico-syntactic patterns. Such patterns were first proposed in 1992 by Hearst [74], and are today usually referred to as "Hearst-patterns". For example a subsumption relation is recognized in the sentence "animals such as cats and dogs" (we can conclude that cats are a kind of animal) by matching a lexicosyntactic pattern like $NP_0\{NP_1, NP_2, ...(and \mid or)\}NP_n$, where NP_i stands for an arbitrary noun phrase. Similar patterns could also be developed for other types of relations, and even for domain specific relations. Another popular pattern, or rather heuristic, is the commonly used so called "head heuristic", sometimes also denoted "vertical relations heuristic". This heuristic is very simple but quite useful for OM. The basic idea is that modifiers are added to a word in order to construct more specific terms, i.e. the term "graduate student" consists of the term "student" and the modifier "graduate". Using the head heuristic we can derive that a "graduate student" is a kind of "student". Text2Onto [34] is a system which generates a class taxonomy and additional axioms from textual documents and it is intended to support both constructing ontologies and maintaining them. The idea is to get better extraction results and to reduce the need for an experienced ontology engineer by using several different extraction approaches and then combining the results. FRED [117] is an online tool and an algorithm for converting text into internally well-connected and quality linked-data-ready ontologies in web-service acceptable time. It implements a novel approach for ontology design from natural language sentences by combining Discourse Representation Theory (DRT), linguistic frame semantics, and

Ontology Design Patterns (ODP). The tool is based on Boxer which implements a DRT-compliant deep parser. The logical output of Boxer enriched with semantic data from Verbnet or Framenet frames is transformed into RDF/OWL by means of a mapping model and a set of heuristics following ODP best-practice of OWL ontologies and RDF data design

2.4 Knowledge patterns

We adopt a notion of KP that derives from the notion of frame [101] and is defined by [66]. A KP can briefly be defined as "a formalized schema representing a structure that is used to organize our knowledge, as well as for interpreting, processing or anticipating information". [66] argues that Knowledge Patterns (KPs) are basic elements of the Semantic Web as an empirical science, which is the vision motivating our work. [23, 114] present experimental studies on KPs, focusing on their creation and usage for supporting ontology design with shared good practices. Such KPs are usually stored in online repositories[5]. Contrary to what we present in this work, KPs are typically defined with a top-down approach, from practical experience in knowledge engineering projects, or extracted from existing. Examples are the Component Library [13], which provides formal representations of frames; the FrameNet project [11], a lexical resource that collects linguistic frames, each described with its semantic roles, and lexical units (the words evoking a frame); and the Ontology Design Patterns portal [6], which provides a collection of ontology patterns and a collaborative platform for discussing about them.

Knowledge Patterns will be extensively discussed in next chapter (cf. Chapter 3)

[5]E.g. the ontology design patterns semantic portal, http://www.ontologydesignpatterns.org
[6]http://www.ontologydesignpatterns.org

Chapter 3

Knowledge Patterns for the Web

This Chapter is aimed at clarifying what we mean by Knowledge Pattern (KP). Hence, we:

- provide a definition for KP. This definition is the result of work we did in [106] (cf. Section 3.1);
- introduce the various definitions for KP available in literature (cf. Section 3.2);
- discuss the sources we want to use for KP discovery, i.e., KP-like artifacts already available (e.g. FrameNet frames) and Linked Data (cf. Section 3.3).

3.1 A definition for Knowledge Pattern

The Web is the largest knowledge repository ever designed by humans and also a melting pot of incompatible platforms, multiple structuring levels, many presentation formats, myriads of conceptual schemata for data, and localized, peculiar content semantics. This heterogeneity has been referred to as the *knowledge soup* of the Web [66].

In the vision of the Semantic Web [18] agents are supposed to help humans in accessing, interacting and exploiting Web knowledge. Linked Data [19] is a breakthrough in the Semantic Web providing access and query support to a number of structured data sets based on URIs [17] and RDF [73]. Nonetheless, it is hard to

build contextualized views over data, which would allow to select relevant knowledge for a specific purpose, i.e., to draw relevant boundaries around data.

In the example about Arnold Schwarzenegger introduced in Chapter 1 we argued about the need for knowledge structures (i.e., KPs) able to select relevant knowledge and to relate entities and concepts according to a unifying view. The aim of this work is to identify methods for the discovery of KPs from the knowledge soup found in the Web.

The following is the definition that we introduced elsewhere [106] and that we will use in this thesis to refer to KPs.

Definition 1 *A Knowledge Pattern is a small, well connected and frequently occurring unit of meaning, which provides a symbolic schema with a semantic interpretation. The unit of meaning a KP identifies results*

1. *task-based;*
2. *well-grounded;*
3. *cognitively sound.*

The first requirement comes from the ability of associating ontologies, vocabularies or schemas with explicit tasks. These tasks are often called *competency questions* [70] and are the basis for a rigorous characterization of the problems that a schema is able to solve. Hence, if a schema is able to answer a typical question that an expert or a user would like to make, it is a useful schema.

The second requirement is related to the fact that KPs are recurrent emerging schemata used for organizing knowledge in the Web. They are grounded in every manifesation of a schemata of which they provide a formalization. This manifesation can be expressed in a variety of Web formats, such as an RDF graph, a textual document, an XML file, etc. Hence, KPs provide a symbolic schema, formal semantics consisting in the meaning of the pattern as well as access to big data.

The third requirement comes from the expectation that schemas that more closely mirror the human ways of organizing knowledge are better. Unfortunately,

evidence for this expectation is only episodic until now for RDF or OWL vocabularies[66]. Nevertheless, a recent crowdsourced experiment [53] seems to prove the cognitive soundness of FrameNet [11] frames. FrameNet frames are defined with a top-down approach and the experiment shows that the same frames as in FrameNet can emerge in a bottom-up fashion by crowdsourcing through the annotation of the frame elements. In Figure 3.1 it is depicted an example of a KP for representing cooking situations. Such KP, represented by using an UML notation, allows us to express the main concepts that are typically associated to cooking situations, such as cook, recipe, ingredient, etc. Additionally the KP has different and heterogeneous manifestations over data, such as a picture of a cook in a cooking act, a recipe in natural language, or an HTML page with RDFa [1] about a recipe. A KP has its own logical form and representation, which allows (i) to extensionally access the heterogeneous manifestations of a KP over data, and (ii) to give an intensional interpretation to heterogeneous symbolic patterns that formalize similar conceptualizations with respect to a KP. Addressing semantic heterogeneity requires to provide homogeneous access to heterogeneous resources by focusing on the "knowledge-level", as it was introduced by Newell [103], who contrasted it to the basic "symbol level" of data and content. We believe that enabling the exploitation of KPs in the Web opens new perspectives towards the realization of the vision of the Semantic Web figured out by Tim Berners-Lee [18]. Hence, we focus on analyzing the Web knowledge in order to make KP empirically emerge.

3.2 Knowledge Patterns in literature

A general formal theory for Knowledge Patterns (KPs) still does not exist. Different independent theories have been developed so far and KPs have been proposed with different names and flavours across different research areas, such as linguistics [51], artificial intelligence [101, 13], cognitive sciences [15, 55] and more recently in the Semantic Web [66]. According to [66] it is possible to identify a shared meaning for KPs across these different theories, that can be informally summarized as "a

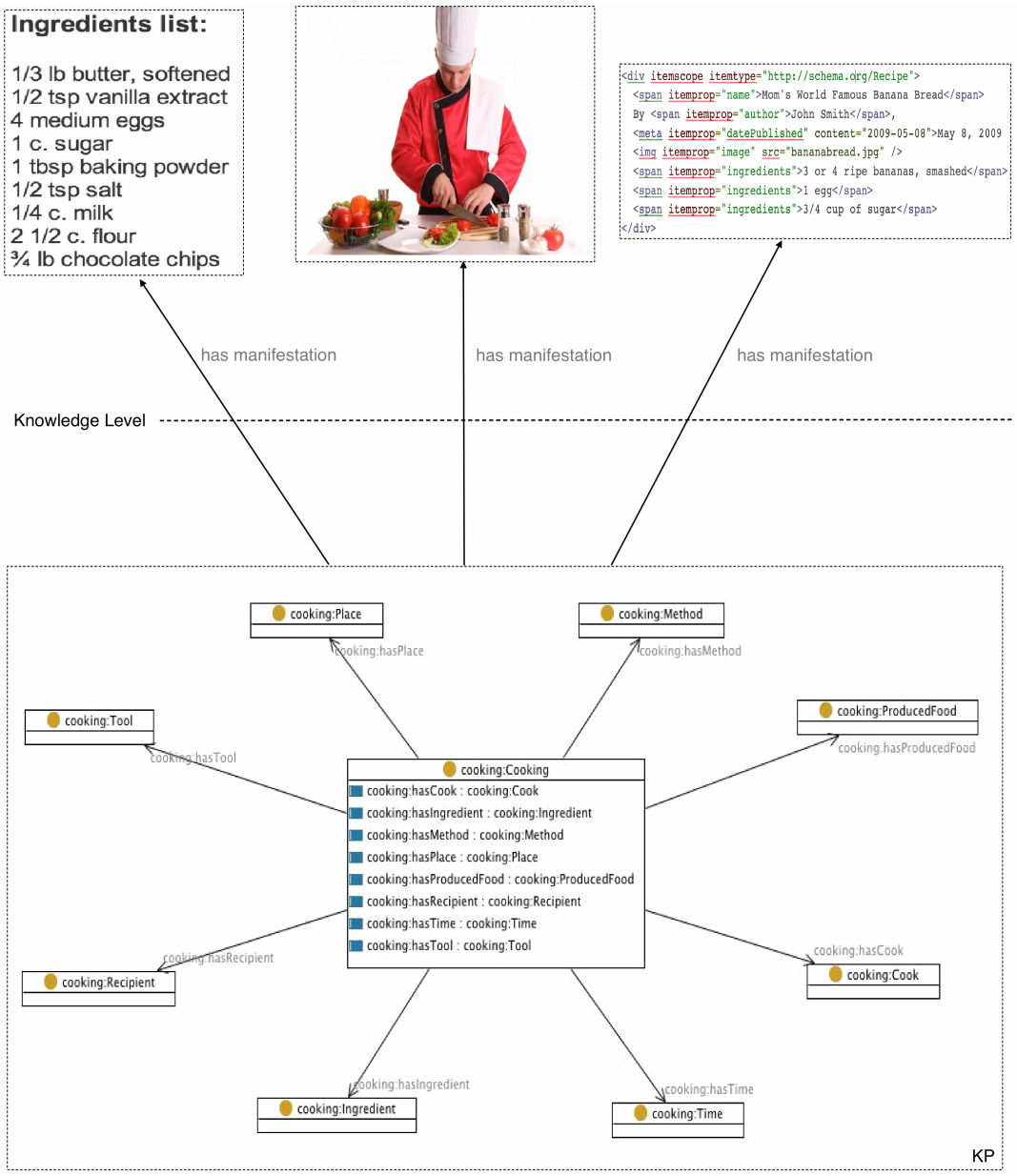


Figure 3.1: An example of a KP for representing cooking situation and its possible manifestation over data.

structure that is used to organize our knowledge, as well as for interpreting, processing or anticipating information". In linguistics KPs were introduced as *frames* by Fillmore in 1968 [51] in his work about *case grammar*. In a case grammar, each verb selects a number of deep cases which form its *case frame*. A case frame describes important aspects of semantic valency, verbs, adjectives and nouns. Fillmore elaborated further the initial theory of case frames and in 1976 he introduced *frame semantics* [52]. According to the author a frame is

> *any system of concepts related in such a way that to understand any one of them you have to understand the whole structure in which it fits; when one of the things in such a structure is introduced into a text, or into a conversation, all of the others are automatically made available.*

A frame is comparable to a cognitive schema. It has a prototypical form than can be applied to a variety of concrete cases that fit this prototypical form. According to cognitive science theories [15] humans are able to recognize frames, to apply them several times, in what are called manifestations of a frame, and to learn new frames that became part of they background. Hence, frames (aka KPs) are cognitively relevant, since they are used by humans to successfully interact with their environment, when some information structuring is needed. This holds for perceiving, searching, browsing, understanding a scene, planning an event, talking about some facts, etc. For example, if a human enters a room with people sitting around a table, and a person speaking to the group while projecting some slides, that human would immediately recognize that there is a working meeting going on in the room. On the other hand, if, when entering a room, the setting is constituted by people sitting and eating around a table, and a person having some packed gifts at hand, the human would typically recognize a birthday party situation. The cognitive mechanism that makes a human easily and quickly recognize those situations is based on her ability to associate them to more abstract patterns that she has learned by experience.

In computer science frames were introduced by Minsky [101], who recognized that frames convey both cognitive and computational value in representing and

organizing knowledge. The notion of frame, aka knowledge pattern, was formalized by Minsky [101] as:

> *a remembered framework to be adapted to fit reality by changing details as necessary. A frame is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party.*

In knowledge engineering the term Knowledge Pattern was used by Clark [36]. However, the notion of KP Clark introduces is slightly different from frames as introduced by Fillmore and Minsky. In fact, according to Clark, KPs are first order theories which provide a general schema able to provide terminological grounding and morphisms for enabling mappings among knowledge bases that use different terms for representing the same theory. Though Clark recognizes KPs as general templates denoting recurring theory schemata, his approach is similar to the use of theories and morphisms in the formal specification of software. Moreover, Clark's KPs lack the cognitive value as it is for frames. This makes difficult to use this formalization of KPs for representing contextual relevant knowledge.

More recently Knowledge Patterns have been reproposed in the context of the Semantic Web by Gangemi and Presutti in [66]. Their notion of KPs encompasses those proposed by Fillmore and Minsky and goes further envisioning KPs as the research objects of the Semantic Web as an empirical science. The empirical nature of KPs was envisaged by Fillmore in [52], who stated that frame semantics comes out of traditions of empirical semantics rather than formal semantics. Gangemi and Presutti argue that a KP can be modeled as a polymorphic relation that takes arguments from a number of *façades*, i.e., a type of knowledge that can be associated with a frame, and can be used to motivate, test, discover, and use it.

Figure 3.2 shows a representation of a KP with its façades. Namely, they are [66]:

- *Vocabulary*: a set of terms that can be structured with informal relations, for example, for a KP about researchers, the following set of terms could be activated: Person, Role, ResearchInterest, ResearchGroup, Name;

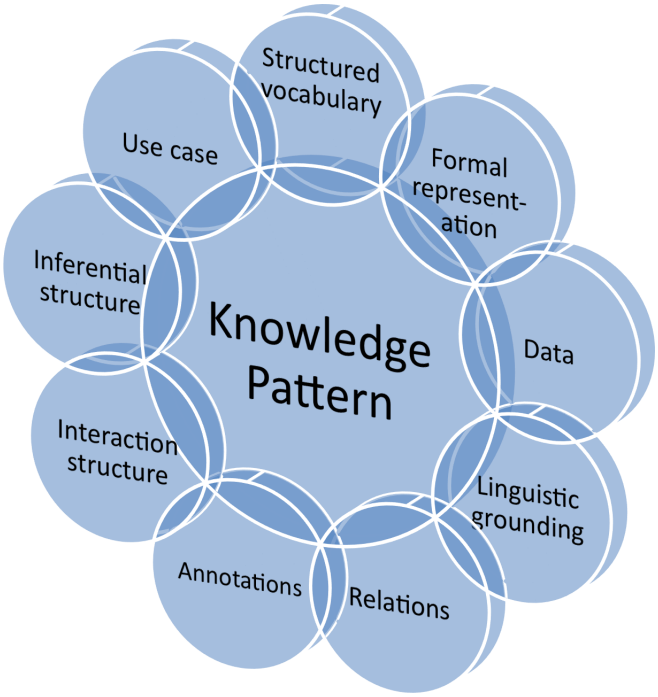


Figure 3.2: The façades of a knowledge pattern [66].

- *Formal representation*: axioms that provide a formal semantics to the vocabulary;
- *Inferential structure*: rules that can be applied to infer new knowledge from the formal representation of the KP;
- *Use case*: requirements addressed by the KP. They can be expressed in various forms e.g., including one or more competency questions [70];
- *Data* that can be used to populate an ontology whose schema is a formal representation for the KP;
- *Linguistic grounding*: textual data that express the meaning of the KP;
- *Interaction structure*: mappings between elements in the formal representation of a KP, and interface or interaction primitives;
- *Relations* to other KPs;

- Annotations: provenance data, comments, tags, and other informal descriptions not yet covered by any existing façades.

3.3 Sources of Knowledge Patterns

We want to design and develop methods for discovering KPs from the Web soup. We distinguish between two main sources of KPs in the Web:

- sources that already provide formalizations of KP-like artifacts. These are typically designed with a top-down approach and are expressed in heterogeneous formats, such as, frames in FrameNet [11], Ontology Design Patterns [65], or components in the Component Library [13]. We refer to methods based on these kind of sources as *KP transformation*. In fact, these methods are aimed at homogenizing existing alternative conceptual representations of a KP-like artifact under a unifying view, i.e. a KP. a common ;

- the Web of Data aka Linked Data [19], which provides large datasets designed by communities of experts, described with RDF and linked among them by exploiting URIs. We refer to methods based on these kind sources as *KP extraction* because they are grounded on bottom-up approaches that require empirical analysis for identifying recurrent and contextual relevant structures by observing how data are organized in order to infer what is part of a certain KP.

Figure 3.3 shows the basic idea of this distinction. The box named Source enrichment shown in figure identifies the methods that deal with the heterogeneity of The Web of Data. This heterogeneity is about semantics, formats and also about Linked Data that are not necessarily clean, optimized, or extensively structured. In next sections we detail the approach we have used for KP transformation and extraction. Then we remand to Chapter 4 for details about KP transformation, to Chapter 5 for details about KP extraction, and to Chapter 6 for details about source enrichment.

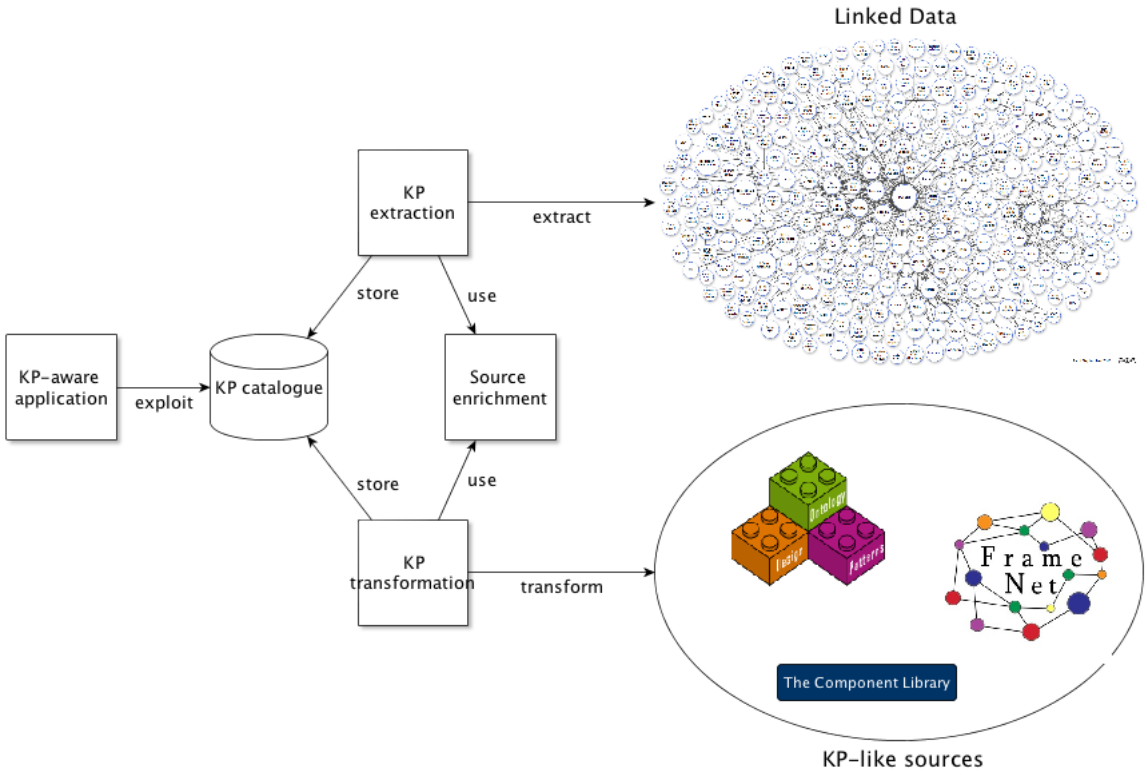


Figure 3.3: Graphical representation of the methodology for KP transformation and extraction.

3.3.1 KP-like repositories

Some existing resources either already provide KPs, e.g., in different formats or languages, or provide artifacts from which KPs could emerge. The first problem with existing schemas deals with the heterogeneity of formats used for expressing schemas, i.e. RDF, OWL, UML, XSD, RDBMS schemas, etc., and possible solutions in order to to represent them homogeneously. The second goes back to the knowledge boundary problem [66] and has to answer to simple questions like: "what part of a certain schema is really meaningful?". Our aim is to investigate methods able to solve these two problems. Our method for performing the transformation of existing schemas to KPs will be discussed in detail in chapter 4. Now we want to help readers

to better understand what are the possible sources of KPs.

Typically KP-like artifacts available in existing sources are modelled in a top-down fashion. This means that they are typically designed by domain experts, which tries to formalize the semantics of a certain domain moving by refinements of the intentional layer, i.e., the T-Box, towards the extensional one, e.g., the A-Box. A canonical example of sources of KP-like artifacts are the repositories of ontologies, i.e., locations where the content of ontologies is stored [137]. The main goal of organizing ontologies in repositories and making them available is their reuse for ontology engineering. There are many ontology repositories at the state of the art that differ one from the other depending on specific functions they provide. In fact, some repositories are only flat container aimed only at storing ontologies, others directly provide support for their reuse, some others are thought for collaboratively building ontologies, etc. Besides these differences, which may result effective in ontology engineering, we hypothesize they all can be rich source for gathering KPs. Examples of these repositories are:

- **ontologydesignpatterns.org** (ODP.org) [116] [1] is a repository of OWL ontologies targeted at the lifecycle management of ontology design patterns. It provides services for publishing and reusing ontology design patterns and it is the reference ontology repository for the XD tools [2], which implements the XDpattern-based methodology [115]. New ODPs can be published into the repository, being included into an official catalogue of patterns, after they have been submitted by a registered users and positively reviewed by other users.

- The **Component Library** (CLIB) [14] [3] is a repository targeted at allowing users with little experience in knowledge engineering to build knowledge bases by instantiating and composing components. Components are general concepts formally expressed in the Knowledge Machine language (KM) [35]

[1] http://www.ontologydesignpatterns.org
[2] http://stlab.istc.cnr.it/stlab/XDTools
[3] http://www.cs.utexas.edu/users/mfkb/RKF/tree/

(which in turn is defined in first-order logic) as coherent collections of axioms that can be given an intuitive label, usually a common English word. The composition of components is achieved by specifying relations among instantiated components. The main division in the Component Library is between entities, i.e., things that are, and events, i.e., things that happen. Events are states and actions. States represent relatively static situations brought about or changed by actions. Entities can be associated to values by properties or connected to Events by a small set of relations inspired by the case roles in linguistics [12].

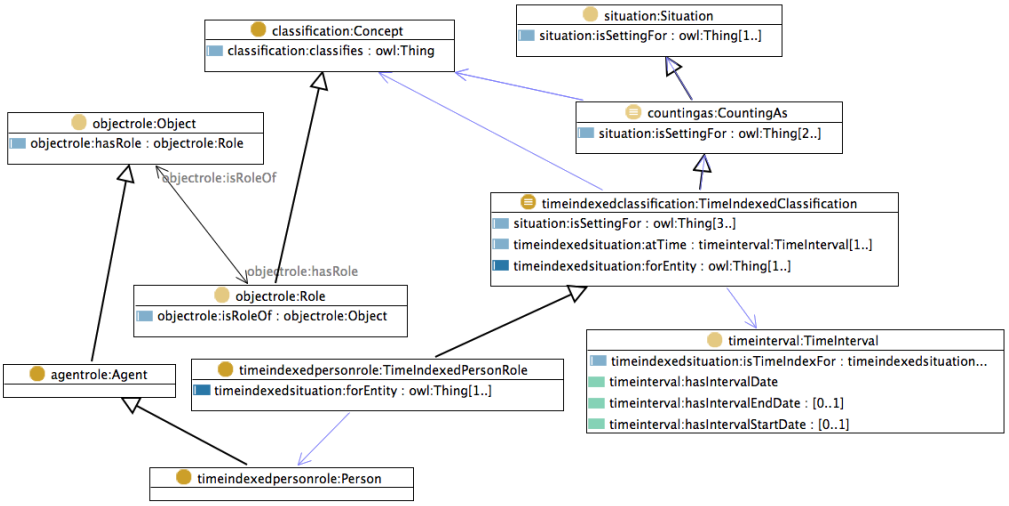
- **FrameNet** [11] is a XML lexical knowledge base, consisting of a set of *frames*, which are based on a theory of meaning called frame semantics. A frame in FrameNet is a collection of facts that specify characteristic features, attributes, and functions of a denotatum, and its characteristic interactions with things necessarily or typically associated with it.

As previously stated, the main issue in KP transformation is about dealing with the heterogeneity of formats and semantics used for representing conceptual schemata in the various source repositories. For example Figure 3.3.1 shows three different schemata represented as UML class diagrams, which describe a common conceptualization that can be used as the formal representation façade (see Figure 3.2) of a KP able to describe the knowledge about entities and their roles over time. However:

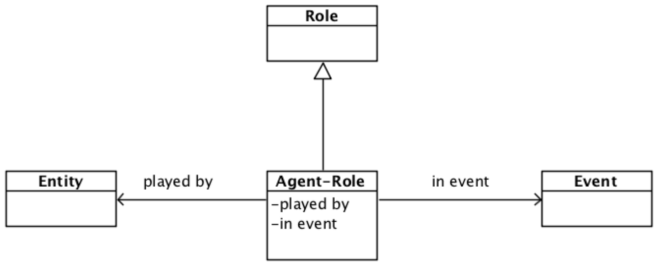
- the TimeIndexedPersonRole Content Pattern (cf. Figure 3.4(a)) provides a formal description of the domain with the OWL logical language;

- the semantics of the Agent-Role component (cf. Figure 3.4(b)) is operational and defined with KM;

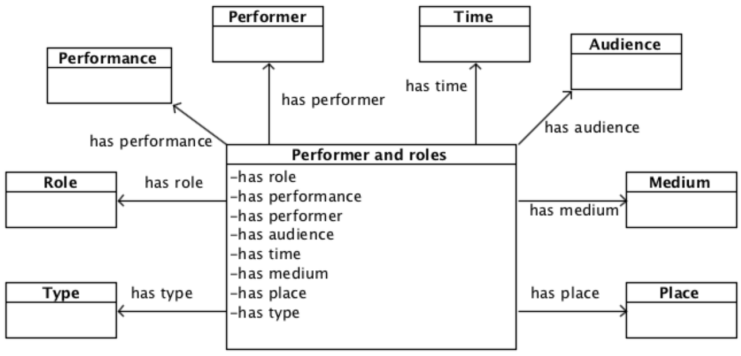
- the semantics of the Performers_and_roles frame (cf. Figure 3.4(c)) is more informal as FrameNet is a lexical knowledge base which represents frames by using XML [26].



(a) The TimeIndexedPersonRole Content Pattern from ontologydesignpatterns.org



(b) The Agent-Role component from the CLIB



(c) The Performers_and_roles frame from FrameNet

Figure 3.4: Three examples of different schemata which describe a common conceptualization about entities and their roles.

We want to overcome this issue by designing and developing a method for enabling KP transformation to OWL2 ontologies without loss of semantics with respect to the original source.

3.3.2 The Web of Data

The Web is evolving from a global information space of linked documents to one where both documents and data are linked. Underpinning this evolution is a set of best practices for publishing and connecting structured data on the Web known as Linked Data [19]. The Linked Open Data (LOD) project is bootstrapping the Web of Data by converting into RDF and publishing existing datasets available under open licenses.

The Web of Data is an ideal platform for empirical knowledge engineering research, since it has the critical amount of data for making KPs emerge. These KPs can be then reused in the knowledge engineering practice and for the design, maintenance, and consumption of data. Linked data and social web sources such as Wikipedia give us the chance to empirically study what are the patterns in organizing and representing knowledge, i.e., what are the knowledge patterns. Linked Data contains rich structured data, which are generally grounded on well defined and sometimes consistent ontologies, such as DBpedia [88]. Furthermore, Linked Data provides a rich linking structure composed by RDF statements, i.e., subject-predicate-object triples, that connect entities to other entities and literals (i.e., strings, numbers, and any other data type) internally in a single dataset and among datasets. The latter point enables connections among communities of experts and domains that before Linked Data evolved independently. We want to exploit the linking structure made available by Linked Data in order to define a method which allows us to draw contextual-relevant boundaries around data (i.e., RDF triples).

Unfortunately, the quality of Linked Data is unpredictable. In fact, it is possible to deal with incomplete data, wrong datatypes or partial intensional or extensional coverage with respect to a give reference ontology, e.g., Yago and the DBpedia ontology for DBpedia. For example, it would be hard to answer a simple question

like "What are the entities typed as PhD-Student?" if the coverage of a certain class PhD-Student is extensionally limited, i.e., entities that should be typed as PhD-Students are indeed untyped. This issue is part of the knowledge soup problem we want to address. Hence, we think that KP extraction can be classified as a specialization of the techniques used in ontology mining [42]. Ontology mining identifies all the activities aimed at discovering ontological hidden knowledge from non-formal data sources by using inductive approaches based on data mining and machine learning techniques. The following are relevant works in the area of ontology mining. [41] proposes an extension of the *k-Nearest Neighbor* algorithm for Description Logic KBs based on the exploitation of an *entropy*-based dissimilarity measure, while [21, 47] makes use of Support Vector Machine (SVN) [31] rather than *k-Nearest Neighbor* to perform automatic classification. SVM performs instance classification by implicitly mapping (through a kernel function) training data to the input values in a higher dimensional feature space where instances can be classified by means of a linear classifier. [139] proposes an approach to generating ontologies from large RDF data sets referred to as Statistical Schema Induction (SSI). SSI acquires firstly the terminology, i.e., the vocabulary used in the data set, by posing SPARQL queries to the repository's endpoint. The second step of SSI is the construction of transaction tables for the various types of axioms that the user would like to become part of the ontology. Based on those transaction tables are mined the association rules that allow to translate the tables into OWL 2 EL axioms. Further extensional approaches to generating or refining ontologies based on given facts can be found in the area of Formal Concept Analysis (FCA) or Relational Exploration, respectively. OntoComP [9] supports knowledge engineers in the acquisition of axioms expressing subsumption between conjunctions of named classes. A similar method for acquiring domain and range restrictions of object properties has been proposed later by [122]. In both cases, hypotheses about axioms potentially missing in the ontology are generated from existing as well as from interactively acquired assertions. Finally, FRED [117] is an online tool for converting text into internally well-connected and quality linked-data-ready ontologies in web-service-

acceptable time. It implements a novel approach for ontology design from natural language sentences by combining Discourse Representation Theory (DRT), linguistic frame semantics, and Ontology Design Patterns (ODP). The tool is based on Boxer which implements a DRT-compliant deep parser. The logical output of Boxer enriched with semantic data from Verbnet or Framenet frames is transformed into RDF/OWL by means of a mapping model and a set of heuristics following ODP best-practice [65] of OWL ontologies and RDF data design.

Chapter 4

Knowledge Pattern transformation from KP-like sources

In this Chapter we present:

- a method we have defined for transforming KP-like artifacts from heterogeneous sources to KPs expressed as OWL2 ontologies. This method addresses the knowledge soup and boundary problems by applying a purely syntactic transformation step of a KP-like artifact to RDF followed by a refactoring step whose aim is to add semantics and to make a KP emerge by selecting meaningful RDF triples (cf. Section 4.1);
- a case study we conducted in [104] for transforming FrameNet frames to KPs. This case study is based on the method defined in this Chapter and allowed us to transform 1024 frames to Linked Data and KPs formalized as OWL2 ontologies (cf. Section 4.2)

4.1 Method

The method we have developed for transforming existing KP-like repositories to OWL KPs is based on Semion, a methodology and a tool (cf. Section 7.4.2) for transforming non-RDF data sources to RDF that we have presented in [105]. The Semion methodology consists of two main steps:

1. a purely syntactic and completely automatic transformation of the data source to RDF datasets according to an OWL ontology that represents the data source structure, i.e. the source meta-model. For example, the OWL ontology for a relational database would include the classes "table", "column", and "row". The ontology can be either provided by the user, or reused from a repository of existing ones. The transformation is free from any assumptions on the domain semantics. This step allows us to homogenize heterogeneous sources to RDF;

2. a semantic refactoring that allows us to express the RDF triples according to specific domain ontologies, e.g., SKOS, DOLCE, FOAF, the Gene Ontology, or anything indicated by the user. This last action results in a RDF dataset, which expresses the knowledge stored in the original data source, according to a set of assumptions on the domain semantics, as selected and customized by the user. When applied to KP transformation this step allows us to identify to what extent RDF triples coming from step 1 can be part of a KP.

In order to exemplify the approach, let us consider a very simple example of a relational database that stores information about people and their roles in universities as that represented in Table 4.1. The table Person stores information about people with their first name, last name and two references (i.e., foreign keys) to their university and their role in the university respectively. The table University stores information about the ID of a university (UID) and its name. The table Role stores information about the ID of a role (RID) and its title.

Figure 4.1 shows two samples of RDF graphs obtained by applying the reengineering step over the sample database. More in detail, on one hand Figure 4.1(a) shows a sample of the database schema after its transformation to RDF, on the other Figure 4.1(a) shows the data in the database after their conversion to RDF. The RDF objects are distinguished in the graphs by prefixing the following notation to labels inside boxes: (i) orange circles for classes, (ii) purple diamonds for individuals, and (iii) green rectangles for literals. Both schema and data are expressed as RDF triples whose terminological layer is determined by an ontology able to capture the

(a) The table about people.

First Name	Last Name	University	Role
Paolo	Ciancarini	1	1
Oscar	Corcho	2	2
Fabio	Vitali	1	2
Anna Lisa	Gentile	3	3
Andrea Giovanni	Nuzzolese	1	4

(b) The table about universities.

UID	Name
1	University of Bologna
2	Universidad Politécnica de Madrid
3	University of Sheffield

(c) The table about roles.

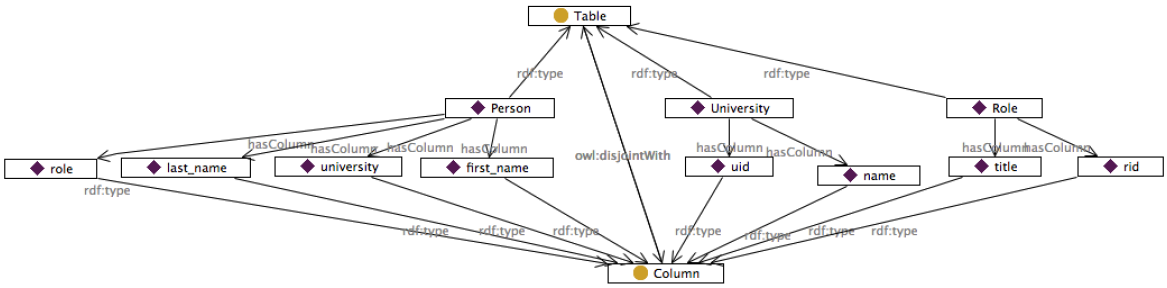
RID	Title
1	Full Professor
2	Associate Professor
3	Research Associate
4	Ph.D. Student

Table 4.1: Tables Person (a), University (b) and Role (c) for a sample database about people and their roles in universities.

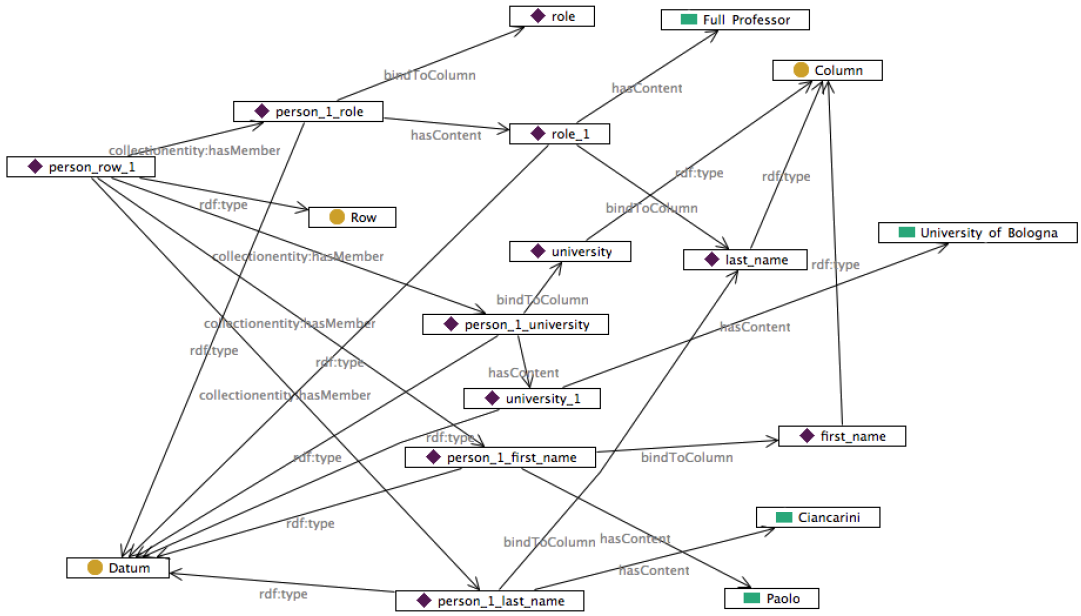
semantics of the original data schema and data. An example of such an ontology is shown in Figure 4.2. This ontology is only one of the several admitted, as it is thought to be one of the input parameter of the reengineering step [1]. The ontology allows to represent schema objects, i.e., tables, columns and keys, data objects, i.e., single data in field and rows, and their relations.

The refactoring step is the result of a non-trivial knowledge engineering work that requires a good knowledge of the target domain semantics. For that reason the refactoring is semi-automatic as it requires the design of transformation rules by a user. More exhaustively, the refactoring is performed by means of transformation rules of the form "*condition* $\rightarrow$ *consequent*" whose aim is to apply a transformation (specified in the consequent) in the RDF graph only if the condition is satisfied with respect to the knowledge expressed in the source graph. A set of rules which co-occur for the finalization of a transformation process is called a *transformation recipe*. During the refactoring step transformation recipes are interpreted as SPARQL CONSTRUCT that allow to model RDF triples to a desired format. We have defined a language for expressing transformation rules in order to have a simpler syntax than SPARQL. The Backus-Naur Form (BNF) of such a language

[1] By default the Semion tool provides reengineering modules for XML and RDBMS. It needs to be extended in order to handle other formats. Refer to Chapter 7 for details.



(a) A sample of RDF derived from the database schema.



(b) A sample of RDF derived from the database data.

Figure 4.1: The result of the reengineering applied to the sample database shown in Table 4.1.

is available in Appendix 9. The following is an example of the rules needed for modeling RDF data of the previous example in order to obtain a terminological component (TBox) able to capture the semantics of people playing a certain role in a university.

```
...
is(dbs:Table, ?x) -> is(owl:Class, ?x) .
is(dbs:Column, ?x) -> is(owl:ObjectProperty, ?x) .
```

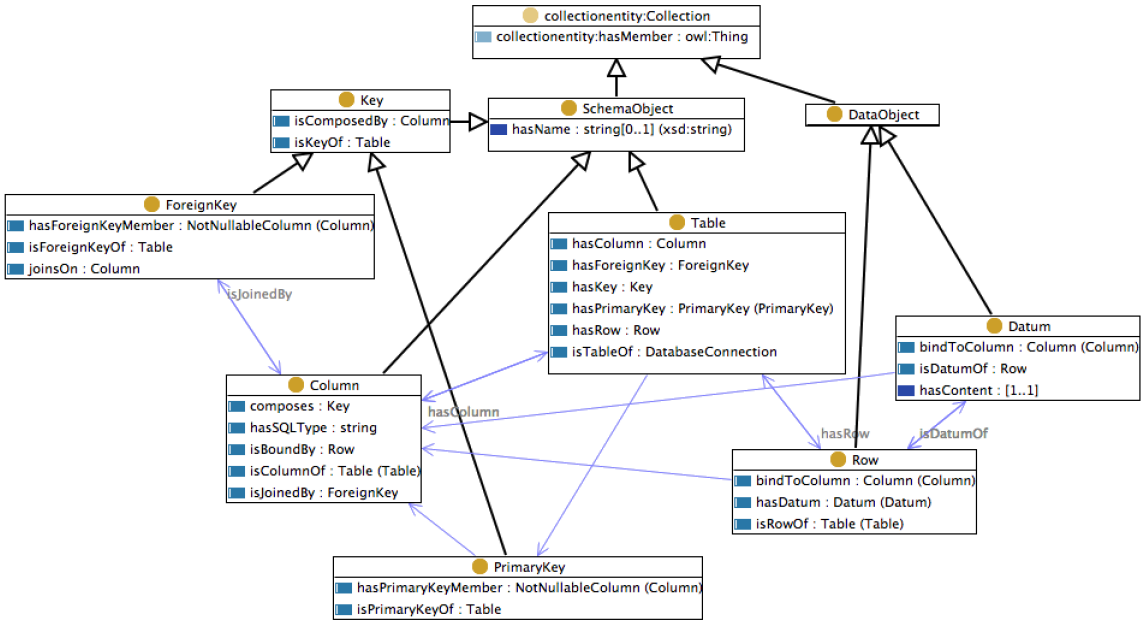


Figure 4.2: An example of an ontology describing concepts about the structure of relational databases. The ontology is represented by adopting an UML notation.

```
is(dbs:Row, ?x) -> is(owl:NamedIndividual, ?x) .
has(dbs:hasRow, ?x, ?y) . is(dbs:Table, ?x) . is(dbs:Row, ?x) -> is(?x, ?y)
...
```

The rules have to be read in the following way:

- everything before the arrow `(->)` is the precondition, i.e., the head of the rule, to verify in order to apply the consequent (everything after the arrow), i.e. the body of the rule;
- isA relations are expressed with the atom `is`$(\cdot, \cdot)$, where the first argument is the type and the second the individual to which the type has to be assigned or verified;
- object properties and datatype properties are expressed by means of `has`$(\cdot, \cdot, \cdot)$ atoms, where the first argument specifies the property, the second the subject and the third the object;
- variables are indicated with the suffix `?`.

The semantics of these rules is the following:

- the first rule allows to model each individual of the class `dbs:Table` [2] as an `owl:Class`;
- the second rule models each individual of the class `dbs:Column` as an `owl:ObjectProperty`;
- the third rule models each individual of the class `dbs:Row` as an `owl:NamedIndividual`;
- the fourth rule allows to add `rdf:type` statements between individual and classes if the precondition holds. The precondition verifies the existence of a relation `dbs:hasRow` between an individual that represents a table and another that represents a row.

As sample of the RDF resulting from the refactoring step over RDF data from the previous example is that shown in Figure 4.3. This graph is modeled by representing the individuals `Paolo_Ciancarini`, `University_of_Bologna` and `Full_Professor` as an instances of the classes `Person`, `University` and `Role` respectively.

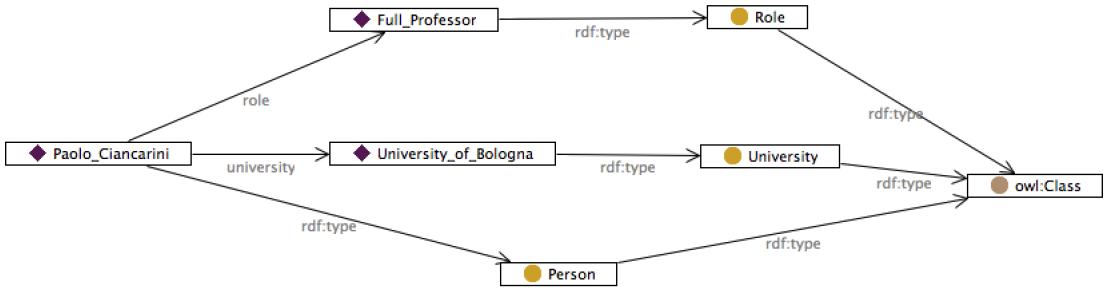


Figure 4.3: Sample RDF graph resulting after the refactoring step on the RDF data about people, universities and roles.

The refactoring step can be iterated as many times as needed. We believe that a good configuration of the Semion methodology in order to extract KPs from KP-like sources is that one depicted in Figure 4.4. This configuration assumes that KP-like

[2]The prefix dbs: stands for the namespace `http://www.ontologydesignpatterns.org/ont/semion/dbs.owl#`, which is the ontology used in the example.

artifacts populate the assertional component (ABox) of the original source. If this assumption does not hold, e.g., the KP-like artifacts populate the terminological component (TBox), this configuration can be easily remapped by removing one of the refactoring step. The figure has to be read in the following way:

- the first container represents the original source, the others represent the result of a step in the Semion methodology;
- each container is divided into three components, i.e., (i) the meta-model box (MBox), e.g., the language used for representing schema elements, (ii) the TBox, e.g., a specific schema for a relational database, and (iii) the ABox, e.g., the data in a database;
- the arrows among containers represent the transformations.

The configuration is the following:

- the reengineering, which performs the syntactic transformation of the original source;
- the ABox refactoring, which gathers RDF data modeled according to an ontology expressing the semantics of the TBox in the original source;
- the TBox refactoring, which is the process of gathering a new ontology schema (a TBox), which actually is a KP, from data (ABox).

In next section we explain how we have used this particular configuration of the Semion methodology for transforming FrameNet frames to KPs.

4.2 A case study: transforming KPs from FrameNet

This case study comes a work we presented in [104]. In sub-section 4.2.1 we give an overview about FrameNet and in sub-section 4.2.2 we discuss the results obtained from the transformation of frames into KPs.

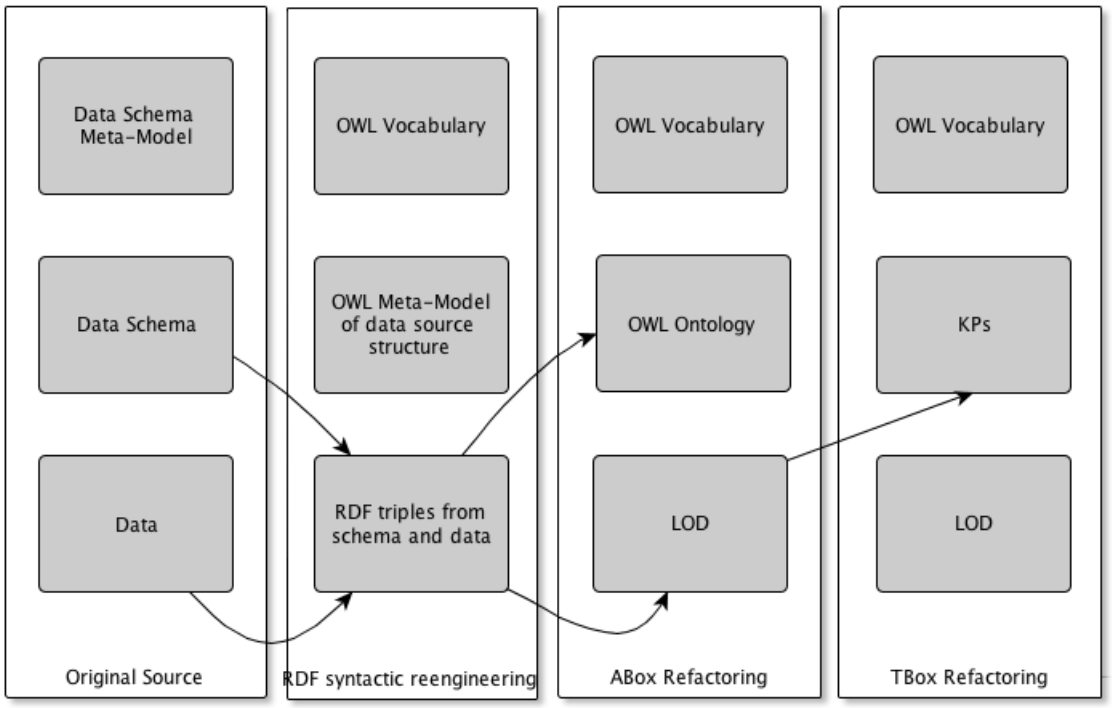


Figure 4.4: Semion tranformation: key concepts.

4.2.1 FrameNet

FrameNet [11] is an XML lexical knowledge base, consisting of a set of *frames*, which have proper *frame elements* and *lexical units*, which pair words (*lexemes*) to frames. As described in the FrameNet Book [123]:

> a lexical unit (LU) is a pairing of a word with a meaning. Typically, each sense of a polysemous word belongs to a different semantic frame, a script-like conceptual structure that describes a particular type of situation, object, or event along with its participants and properties. For example, the **Apply Heat** frame describes a common situation involving a **Cook**, some **Food**, and a **Heating** Instrument, and is evoked by words such as *bake, blanch, boil, broil, brown, simmer, steam*, etc. We call these roles frame elements (FEs) and the frame-evoking words are LUs in the Apply heat frame.

FrameNet has a rich internal structure and makes some cognitive and semantic assumptions that makes it unique as a lexical resource. The basic assumptions are reported here:

- frame elements are mostly unique to their frame;
- a frame usually has only some of its roles actually lexicalized in texts;
- frames can be *lexicalized* or not: non-lexicalized ones typically encode *schemata* from cognitive linguistics;
- frames, as well as frame elements, are related between them, e.g. through the *subframe* compositional relation, through inheritance relations, etc.

The semantic part of FrameNet is enriched by *semantic types* assigned to frames (e.g. **Artifact**), frame elements (e.g. **Sentient**), and lexical units (e.g. **Biframal_LU**). FrameNet also contains a huge amount of manual annotations (*annotation sets*) of sentences from textual corpora with frames, frame elements and lexical units, which make word *valences* (syntactic and semantic combinatory of words) emerge.

4.2.2 Result

The approach followed for the creation of a LOD dataset from FrameNet[3] is both derived from the transformation method implemented by Semion [105] and based on an iterative evaluation of the quality of the output produced with respect to the semantics of FrameNet formalized into a "gold standard" ontology[4] that we have used for the evaluation. Based on that, the transformation of FrameNet v.1.5 from XML to RDF consisted of two steps: (i) the syntactic transformation of the XML source to RDF according to the OWL meta-model that describes the structure of the source[5], (ii) the design and the application of a refactoring recipe for the ABox

[3]The dataset can be accessed through the SPARQL endpoint at http://bit.ly/fnsparql, as framenet_dataset

[4]http://ontologydesignpatterns.org/cp/owl/fn/framenet.owl

[5]http://www.ontologydesignpatterns.org/ont/iks/oxml.owl

refactoring on the RDF produced in the first step. The recipe was derived generalizing and revising some of the common transformation practices from existing tools (i.e. XML2OWL [24], TopBraid Composer[6], Rhizomik ReDeFer [121]). For example we used the following mappings: (i) a `xsd:ComplexType` is mapped to an `owl:Class`, (ii) a `xsd:SimpleType` is mapped to an `owl:DatatypeProperty` and (iii) a `xsd:Element` is mapped either to an `owl:ObjectProperty` or to a `owl:DatatypeProperty`. Further details can be found in [24]. As an example, according to the syntax of the rules for the Semion refactoring, we have that the mapping (i) is expressed as

```
is(oxsd:ComplexType, ?type)
    ->
is(owl:Class, ?classNode)
```

and maps any individual of the class *oxsd:ComplexType* to a *owl:Class*. We refer to the Semion wiki[7] for more information about the tool and the syntax of the rules. The relevance of a syntactic transformation and a following refactoring can be clarified saying that it is designed as a semi-automatic approach which allows, via the refactoring rules, to better address the domain semantics of the original source. As an example, we can consider a simple frame-to-frame relation like

```
Inherits_from(Abounding_with, Locative_relation)
```

which expresses the fact that the frame *Abounding_with* inherits the schematic representation of a situation involving various participants, properties, and other conceptual roles from the frame *Locative_relation*. This relation is expressed in the XML FrameNet notation as:

```
<frame name="Abounding_with" ... ID="262">
  ...
```

[6]http://www.topbraidcomposer.com
[7]http://stlab.istc.cnr.it/stlab/Semion

```
  <frameRelation type="Inherits from">
    <relatedFrame>
      Locative_relation
    </relatedFrame>
  </frameRelation>
  ...
</frame>
```

and, with most of the existing tools, it is transformed to the RDF schematized in Figure 4.5. It is easy to notice how the `Inherits from` frame-to-frame relation is realized through the reification of the relation `RelatedFrame_i`, that expresses its type and the related frames, i.e. `Inherits from` and `Locative_relation`, which are expressed as literals.

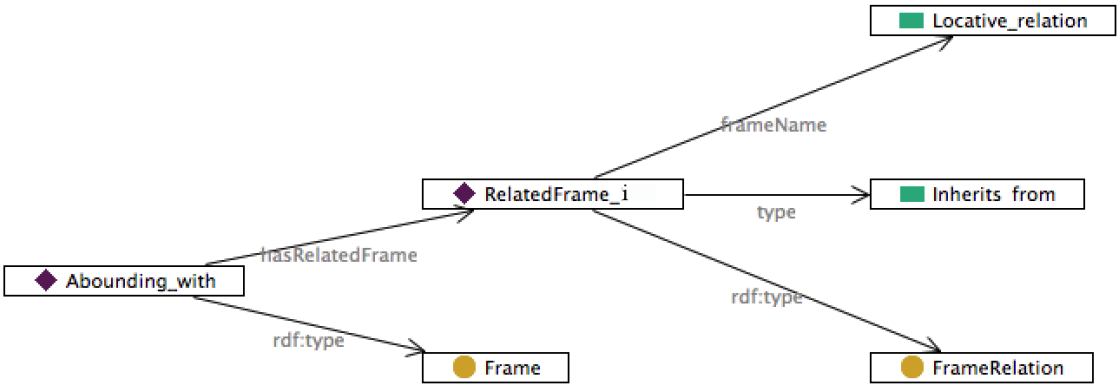


Figure 4.5: The "Inherits from" relation mapped to RDF with a common transformation recipe. Literals are identified by the icons drawn as green rectangles

Instead, by adopting the syntactic transformation of Semion, we have produced firstly an RDF graph, which is depicted in Figure 4.6[8].

In the figure, *fntbox:Frame* is no longer an *owl:Class*, but an *oxsd:Element* and *fnabox:Abounding_with* is an *oxml:XMLElement* related to *fntbox:Frame* through *oxsd:hasElementDeclaration*.

[8] `oxsd` and `oxml` are the default ontologies of Semion for XSD and XML data structures.

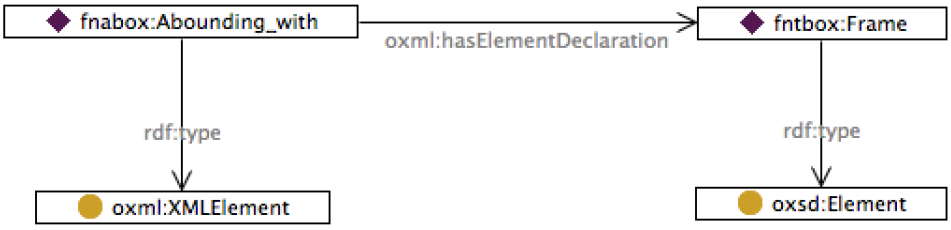


Figure 4.6: Example of reengineering of the frame "Abounding with" with its XSD definition.

After having syntactically converted FrameNet from XML to RDF, we applied the general recipe with the Semion Refactorer, in order to derive a LOD dataset for FrameNet. As the recipe is based on a general conversion from XML to OWL, the result was far from being a good formalization of the semantics of FrameNet. For that reason, we have incrementally refined the recipe in order to fill the gap between the semantics expressed by the output produced by the refactoring and the gold standard we had previously defined. We remark that the aim of the refactoring is to transform one RDF source to another trying to preserve either explicit or implicit domain semantics of the original source without information loss.

For example, the rule which allows to avoid the reification of frame-to-frame relations is shown in Figure 4.7. The rule shown in Figure 4.7 transforms all the frame-to-frame relations into binary relations between frames. The rule extracts the type of the relation from the `nodeValue` associated with the `type` attribute of a frame. Then it creates a new object property as a sub-property of `hasFrameRelation`, and resolves the name of the related frame that is expressed as a literal in the `relatedFrame` element, as shown in the XML code before. We remark that the model accessed by rules is not anymore the original XML source, but its syntactic translation to RDF. Figure 4.8 shows the RDF of the `inherits from` relation between the frames `Abounding with` and `Locative relation` obtained by applying the refactoring recipe with Semion. Figure 4.9 shows the core fragment of the OWL schema of FrameNet used as a vocabulary for the data from FrameNet.

```
...
values(oxml:nodeValue, ?xmlAttr, ?value) .
values(oxml:nodeName, ?xmlAttr, "type"^^xsd:string) .
has(composite:child, ?xmlElem, ?child) .
values(oxml:nodeValue, ?child, ?childValue) .
let(?relatioURI, concat(namespace(?frame), trim(?value)) .
let(?frameURI, concat(namespace(?frame),
                        concat("frame/",
                               ?childValue
                        )
                )
) .

newNode(?frameRelation, ?frameURI) .
newNode(?relatedFrame, ?frameURI)

   ->

is(owl:ObjectProperty, ?frameRelation) .
has(rdfs:subPropertyOf, ?frameRelation,
    fns:hasFrameRelation) .
has(?frameRelation, ?frame, ?relatedFrame)
```

Figure 4.7: Rule which allows to express frame-to-frame relation as binary relations.

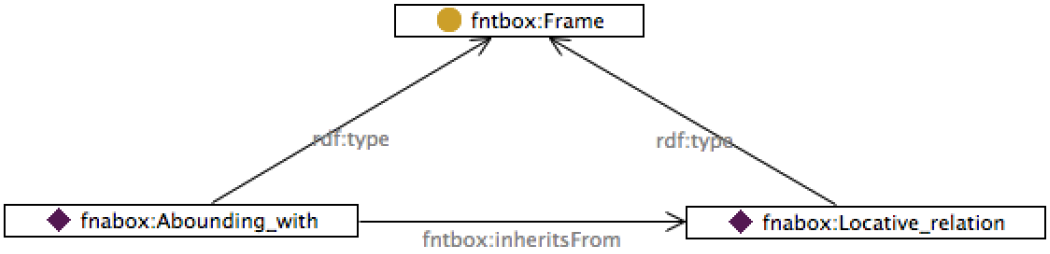


Figure 4.8: The "Inherits from" frame-to-frame relation between the frames "Abounding with" and "Locative relation" after the refactoring.

The complete refactoring recipe[9] is composed by 58 transformation rules in forward-chaining inference mode.

An important feature of FrameNet as a dataset in the LOD cloud, that will be investigated as part of our ongoing work, is the mapping of its frames and frame elements to other lexical resources, e.g. WordNet. WordNet is available as a LOD dataset since 2006 as a result of the W3C working draft [126]. Such mappings can

[9]http://stlab.istc.cnr.it/stlab/FrameNetKCAP2011# tab=ABoxRefactoring

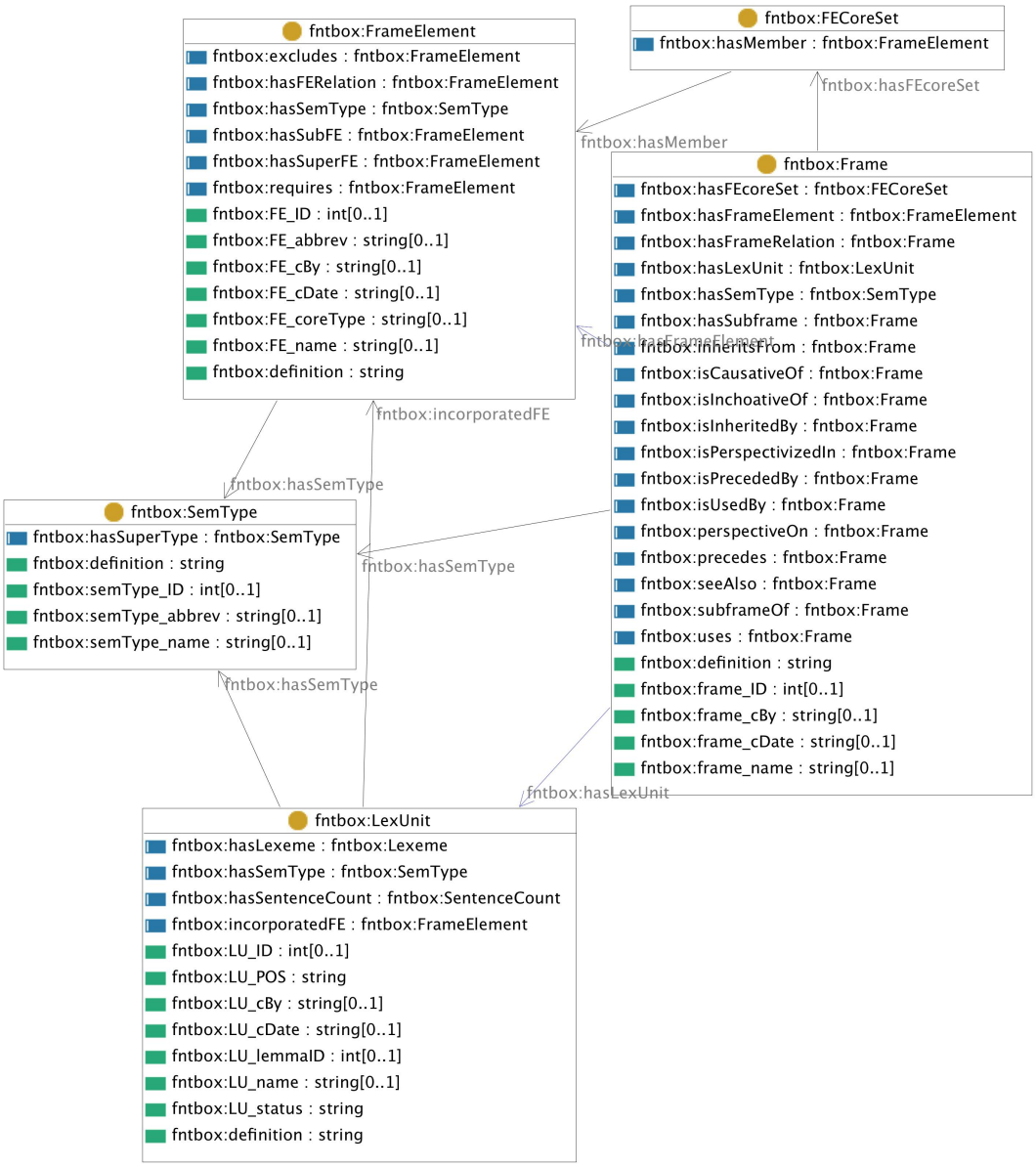


Figure 4.9: A fragment of the FrameNet OWL schema.

be obtained from VerbNet [127], a lexical resource that incorporates both semantic and syntactic information about English verb semantics. The VerbNet 3.1 XML database provides mappings between VerbNet classes, FrameNet frames, and WordNet synsets.

For example, from the VerbNet mappings converted to RDF:[10]

```
vnclass:accompany skos:exactMatch
wndata:synset-accompany-verb-2

vnclass:accompany skos:exactMatch
frame:Cotheme
```

The VerbNet dataset excerpt is intended to demonstrate linkings between lexical resources. An official release will be published in the near future.

In addition to the production of FrameNet as a LOD lexical dataset that can be accessed and queried over the Web of Data, our aim is to provide an interpretation of frames as Knowledge Patterns (KPs). In other words, following [66], we promote frames to relevant units of meaning for knowledge representation.
With reference to Figure 4.4, we have called this process TBox refactoring, because a new ontology schema (a TBox), is obtained starting from data (ABox).
The main problem with TBox refactoring is deciding the formal semantics to assign to the classes from the FrameNet LOD dataset schema. Since this is a relatively arbitrary process, SemionRules and recipes are useful to configure alternative choices or to compare the different assumptions made by knowledge engineers. Here we present a refactoring experience that exemplifies the design method behind such process, and how Semion is useful in supporting it. The recipe exemplified here is part of a larger project carried out together with FrameNet developers in Berkeley in order to optimize the refactoring from lexical frames to KPs: as such, it certainly bears validity, but it is mainly intended as a methodological and pragmatical description of refactoring recipes (also called *correspondence patterns* in [134]).
Besides the basic assumptions reported in Section 4.2.1, this process is guided by the FrameNet Book [123], which is quite explicit about possible formal semantic

[10] prefixes: skos: http://www.w3.org/2004/02/skos/core#; vnclass: http://www.ontologydesignpatterns.org/ont/vn/class/; wndata: http://www.w3.org/2006/03/wn/wn20/instances/; frame: http://www.ontologydesignpatterns.org/ont/framenet/ frame/

choices:

> The most basic summarization of the logic of FrameNet is that Frames describe classes of situations, the semantics of LUs are subclasses of the Frames, and (...) FEs are classes that are arguments of the Frame classes. An annotation set for a sentence generally describes an instance of the subclass associated with an LU as well as instances of each of its associated FE classes (...) The term "Frame Element" has two meanings: the relation itself, and the filler of the relation. When we describe the Coreness status of an FE (...) we are describing the relation; when we describe the Ontological type on an FE (...) we mean the type of the filler.

According to these statements, a fragment of the **Desiring** frame is transformed into OWL as follows (in Manchester syntax):

```
Ontology: odpfn:desiring.owl
Annotations:
    cpannoschema:specializes odp:situation.owl
Class: desiring:Desiring
    SubClassOf:
        desiring:hasEvent some desiring:Event,
        desiring:hasExperiencer some desiring:Experiencer,
        desiring:hasDegree some desiring:Degree,
        desiring:hasReason some desiring:Reason,
Class: desiring:covet.v
    SubClassOf: desiring:Desiring
Class: desiring:Event
    SubClassOf: semtype:State_of_Affairs
Class: desiring:Experiencer
    SubClassOf: semtype:Sentient
```

Notice that the uniqueness (*locality*) of frame elements and lexical units for a frame is obtained simply by means of a specific namespace (denoted by the *desiring*

prefix in the example, see below for possible namespace policies), while a frame is interpreted as an owl:Class, lexical units as its subclasses, frame elements as both an owl:Class (e.g. *Event*) and an owl:ObjectProperty (e.g. *hasEvent*), the relation between a frame and a frame element as a rdfs:subClassOf an owl:Restriction, and the semantic type assignments to frame elements as additional subclass axioms. All knowledge patterns derived from frames are considered specialization of the generic pattern `odp:situation.owl`[11], which generalizes the situation semantics suggested by Berkeley linguists.
A central role in FrameNet is played by *inheritance* assumptions. In [123], inheritance is viewed as

> the strongest relation between frames, corresponding to is-a in many ontologies. With this relation, anything which is strictly true about the semantics of the Parent must correspond to an equally or more specific fact about the Child. This includes Frame Element membership of the frames (except for Extrathematic FEs), most Semantic Types, frame relations to other frames, relationships among the Frame Elements, and Semantic Types on the Frame Elements.

This means that additional axioms must be wrapped into ontologies derived from frames, e.g. these two sample axioms are derived from the *inheritsFrom* relation between the **Aesthetics** and **Desirability** frames as well as from the *subFE* relation between some of their frame elements:

```
Ontology: odpfn:aesthetics.owl
Annotations:
    cpannoschema:specializes odpfn:desirability.owl
Class: aesthetics:Aesthetics
    SubClassOf: desirability:Desirability
Class: aesthetics:Degree
    SubClassOf: desirability:Degree
```

[11] odp:http://www.ontologydesignpatterns.org/cp/owl/,
odpfn:http://www.ontologydesignpatterns.org/cp/owl/fn/

The implementation of TBox refactoring is performed as a Semion refactoring, where the recipe includes rules for the mapping between FrameNet LOD dataset and KPs. Figure 4.10 shows an overview of TBox refactoring for deriving KPs from frames. The notation attempts to make rules intuitively understandable: arrows

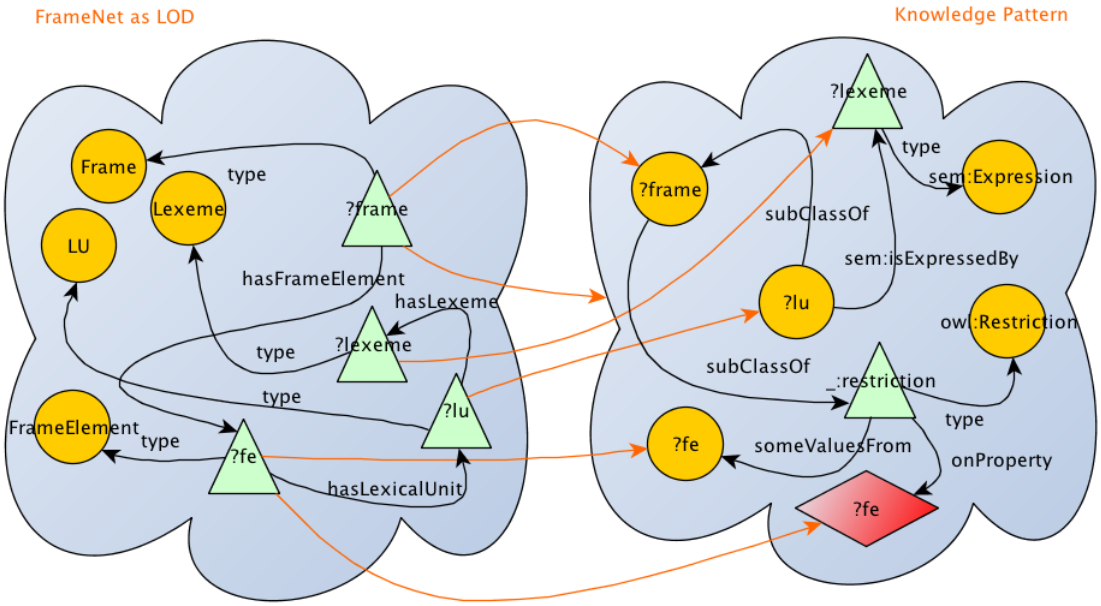


Figure 4.10: Diagram of the transformation recipe used for the production of knowledge patterns from FrameNet LOD.

between the clouds represent mappings from entities in the cloud "FrameNet as LOD" to entities in the cloud "Knowledge Pattern", classes are represented as circles, individuals as triangles, object properties as diamonds, and structural properties as labeled arcs. Each *Frame* is mapped both to an `owl:Ontology` that identifies the KP and to an `owl:Class`. The mapping takes into account the refactoring of the frame URI intended either as an ontology or as a class. Each *FrameElement* maps both to an `owl:Class` and to an `owl:ObjectProperty`. Again frame elements follow a renaming policy for the two interpretations, but in this case the situation is more complex. In fact, URI policy can follow from different interpretations:

1. *Locality* of frame elements within their frames (compatible to locality state-

ments in the Book, with some exceptions that cannot be discussed here). E.g. given the frame:
`http://someuri/Judgment.owl#Judgment`
we obtain the frame element:
`http://someuri/Judgment.owl#Cognizer`
interpreted as a class and
`http://someuri/Judgment.owl#hasCognizer`
interpreted as an object property;[12]

2. *Globality* of frame elements, abstracted from their contextual binding to a frame, e.g. given the frame:
`http://someuri/Judgment.owl#Judgment`
we obtain the frame element:
`http://someuri/class/Cognizer`
interpreted as a class and
`http://someuri/property/hasCognizer`
interpreted as an object property.

Lexical units are refactored as subclasses of the classes derived from the frames they are lexicalizations of, e.g.
`lexunit:cool.a SubClassOf: desirability:Desirability`
Lexemes are refactored as individuals of the class `semantics:Expression`; each lexical unit is related to a lexeme through the property `semantics:isExpressedBy`. Finally, each frame has `owl:someValuesFrom` restrictions accounting for the semantic roles implicit in frame elements (see example above).
Locality and globality alternatives required two refactoring recipies each of one composed by 4 rules in forward-chaining inference mode. The complete TBox refactoring recipe can be found in the wiki page[13].

[12] An OWL2 alternative is also possible, with multiple interpretations for the same constant.
[13] http://stlab.istc.cnr.it/stlab/FrameNetKCAP2011# tab=TBoxRefactoring

4.2.3 Evaluation

We carried out two evaluations of the transformation method implemented by Semion:

- one based on a "gold standard" ontology[14] that was formalized with respect to the semantics of FrameNet and that we used as the terminological layer for representing frames in the A-Box refactoring;
- another based on the isomorphism of the method, i.e., the capability of the method to be reversible.

The first evaluation was performed by transforming the original FrameNet source expressed as XML to Linked Data and observing the compliance of the generated RDF with respect the gold standard ontology. We remark that the gold standard ontology reflects the semantics of FrameNet, hence we expected to collect the same number of frames, frame elements, lexical units, etc. as in the original XML source. The results obtained are excellent as the number of obtained RDF objects after the A-Box refactoring followed the expectations (cf. Table 4.2).

FrameNet OWL class	**# of individuals in FrameNet LOD**	**# of individuals as in the original source**
fntbox:Label	340,856	340,856
fntbox:Layer	185,896	185,896
fntbox:AnnotationSet	29,928	29,928
fntbox:SentenceCount	11,942	11,942
fntbox:FrameElement	9,633	9,633
fntbox:LexUnit	9,515	9,515
fntbox:Lexeme	8,030	8,030
fntbox:Sentence	5,946	5,946
fntbox:Frame	1,024	1,024
fntbox:FECoreSet	240	240
fntbox:CorpDoc	78	78
fntbox:Document	78	78
fntbox:FullTextAnnotation	78	78
fntbox:Header	78	78
fntbox:SemType	74	74

Table 4.2: Number of obtained and expected individuals after the A-Box refactoring.

[14] http://ontologydesignpatterns.org/cp/owl/fn/framenet.owl

Additionally, we performed an inverse refactoring aimed at evaluating the isomorphism of our method. This was tested by first inverting the T-Box refactoring, hence by generating FrameNet LOD (we refer to the result of this step as to FrameNet LOD^{-1}) from the Knowledge Patterns and then by inverting the A-Box refactoring, hence by generating FrameNet XML (we refer to the result of this step as to FrameNet XML^{-1}) from FrameNet LOD^{-1}. Both the inverse refactoring steps were applied by reverting the directions of the rules. This was obtained by interpreting the head of the original rule as the body and vice versa. After having completed the inverse refactoring we compared FrameNet LOD^{-1} with respect to FrameNet LOD and FrameNet XML^{-1} with respect to FrameNet XML.

The first comparison was performed by means of a SPARQL query aimed at identifying possible differences among the two RDF graphs consisting in the two versions of FrameNet LOD. The query executed was the following:

```
SELECT *
FROM <framenet-lod>
WHERE{?s ?p ?o
   MINUS{
       SELECT ?s ?p ?o
       FROM <framenet-lod-inverse>
       WHERE{?s ?p ?o}
   }
}
```

where `<framenet-lod>` and `<framenet-lod-inverse>` identify the RDF graphs available in the triplestore for FrameNet LOD and FrameNet LOD^{-1} respectively. The query expresses a negation between the two graphs by means of the `MINUS` operator available from SPARQL1.1.

The execution of the SPARQL query above generated an empty result set, which means that FrameNet LOD and FrameNet LOD^{-1} are the same RDF graph and then the T-Box refactoring is isomorphic.

The second comparison was performed by first generating a single XML file from the collection of XML files available in the original version of FrameNet and then detecting the difference between the latter and FrameNet XML^{-1}. The difference were obtained by applying the X-Diff algorithm [142], which uses an unordered model (only ancestor relationships are significant) to compute the difference between two XML documents. The result of X-Diff was that no change was performed from FrameNet XML and FrameNet XML^{-1} meaning that the two XML documents were equivalent, hence the A-Box isomorphic.

Chapter 5

Knowledge Pattern extraction from the Web of Data

In this Chapter we present:

- a method we have defined for extracting KPs from Linked Data. This method is grounded on the hypothesis that the linking structure of Linked Data conveys a rich knowledge that can be exploited for an empirical analysis aimed at drawing boundaries around data and consequently formalizing patterns of frequently occurring knowledge organization schemata, i.e. KPs (cf. 5.1).

- a case study we conducted in [106] for extracting KPs from links in the english Wikipedia [1]. The result of the case study are 184 Encyclopedic Knowledge Patterns (EKPs). These KPs are called Encyclopedic for emphasizing that they are grounded in encyclopedic knowledge expressed as linked data, i.e. DBpedia [2] [88], and as natural language text, i.e. Wikipedia (cf. Section 4.2).

- an evaluation of extracted EKPs based on a user study whose aim is to validate the cognitive value of EKPs in providing an intuitive schema for organizing knowledge (cf. Section 4.2).

[1] DBpedia provides a dataset for Wikipedia links (i.e., `dbpedia_page_links_en`)
[2] DBpedia: `http://dbpedia.org`

5.1 Method

In recent years, the Web has evolved from a global information space of linked documents to one where both documents and data are linked. Underpinning this evolution is a set of best practices for publishing and connecting structured data on the Web. These best practices are known as Linked Data [19] principles and they can be paraphrased in the following way:

- Use URIs to denote things;
- Use HTTP URIs so that these things can be referred to and looked up by people and user agents;
- Provide useful information about the thing when its URI is dereferenced, leveraging standards such as RDF, SPARQL;
- Include links to other related things (using their URIs) when publishing data on the Web.

The aim of Linked Data is to bootstrap the Web of Data by identifying existing data sets that are available under open licenses, converting them to RDF according to [19], and publishing them on the Web. Thus, Linked Data provide a large set of realistic data, created by large communities of practice, on which experiments can be performed. By analyzing Linked Data we want to empirically understand what are the typical and frequently occurring patterns, i.e., KPs, used for organizing knowledge. We hypothesize that the linking structure of Linked Data can be used for our purpose as linking things to other things is a typical action done by humans for describing something in the Web.

The method we have designed is based on three phases:

1. data analysis;
2. boundary induction;
3. KP formalization.

We detail these phases in the next subsections.

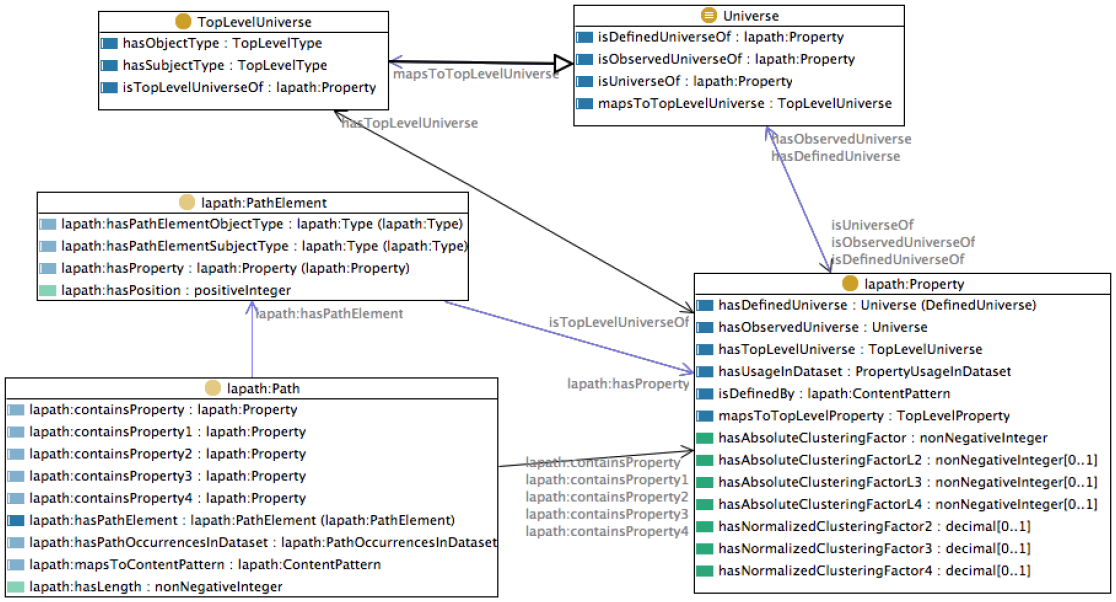


Figure 5.1: Core classes of the knowledge architecture ontology represented with an UML notation.

5.1.1 Data analysis

The data analysis phase is based on a specialization of the method defined by Presutti et al. in [113] that presents an approach aimed at modeling, inspecting, and summarizing datasets, by drawing the so-called dataset knowledge architecture, This relies on the notions of paths, i.e., distinct ordered type-property sequences that can be traversed in an RDF graph. The analysis allows to build an abstraction over a dataset that highlights its knowledge organisation and core components. This abstraction is built by gathering and consequently representing RDF paths according to the *knowledge architecture ontology* [3]. Figure 5.1 shows the core of such an ontology in which a `Path` is seen as the composition of path elements represented as individuals of `PathElement`. These are basically views over RDF triples which allow to gather the `Property` and the `Universe` (i.e., the couple identifying the types associated to the subject and to the object) of a triple.

The representation of the dataset according to the knowledge architecture ontology enables to build a prototypical querying layer that in our case is used for

[3] `http://www.ontologydesignpatterns.org/ont/lod-analysis-properties.owl`

gathering path statistics in a dataset.

As we want to extract terminological components, i.e., the KPs, in a bottom-up fashion from data, we extend the notion of *property path*[4]. We call this extension *type path*, whose definition is the following

Definition 2 (Type path) *A type path is a property path (limited to length 1 in this work, i.e. a triple pattern), whose occurrences have (i) the same* `rdf:type` *for their subject nodes, and (ii) the same* `rdf:type` *for their object nodes. It is denoted here as:*

$$P_{i,k,j} = [S_i, p_k, O_j]$$

where S_i is a subject type, p is a property, and O_j is an object type of a triple.

We extract KPs by analysing type paths (see Definition 3), however in order to formalize them, we perform a heuristic procedure to reduce multi-typing, and to avoid redundancies. In practice, given a triple `s p o`, we construct its path as follows:

- the subject type S_i is set to the most specific type(s) of s;
- the object type O_j is set to the most specific type(s) of o;
- the property p_k is the property used in the triple.

For example, the triple:

```
dbpedia:Andre_Agassi dbpprop:winnerOf dbpedia:Davis_Cup
```

would count as an occurrence of the following path:

$P_{Agassi,winnerOf,Davis} = [$`dbpo:TennisPlayer`, `dbpprop:winnerOf`, `dbpo:TennisLeague`$]$

[4]In SPARQL1.1 (http://www.w3.org/TR/sparql11-property-paths/) property paths can have length n, given by their route through the RDF graph.

Figure 5.2 depicts such a procedure for the path $P_{Agassi,winnerOf,Davis}$. The path (represented in Figure 5.2(b)) comes from the following observations (cf. Figure 5.2(a)):

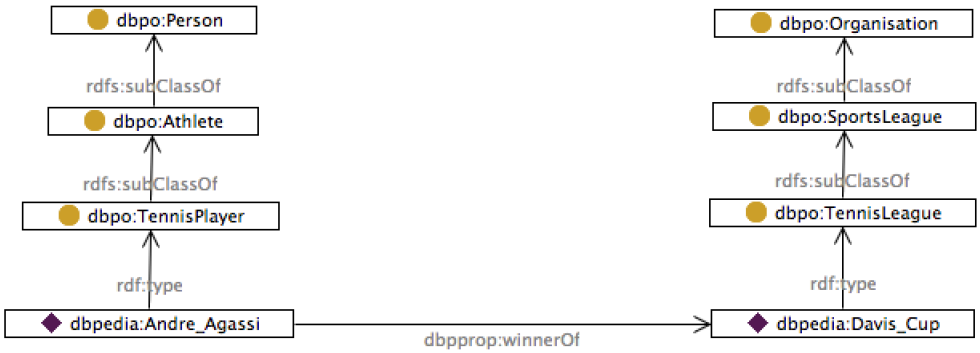
- `dbpo:TennisPlayer` is the subject type because it is the most specific type of `dbpedia:Andre_Agassi`, i.e., `dbpo:TennisPlayer` $\sqsubseteq$ `dbpo:Person`;
- `dbpo:TennisLeague` is the object type because it is the most specific type of `dbpedia:Davis_Cup`, i.e., `dbpo:TennisLeague` $\sqsubseteq$ `dbpo:SportsLeague` $\sqsubseteq$ `dbpo:Organisation`
- `dbpprop:winnerOf` is the property of the path because it is the property which links `dbpedia:Andre_Agassi` to `dbpedia:Davis_Cup`.

5.1.2 Boundary induction

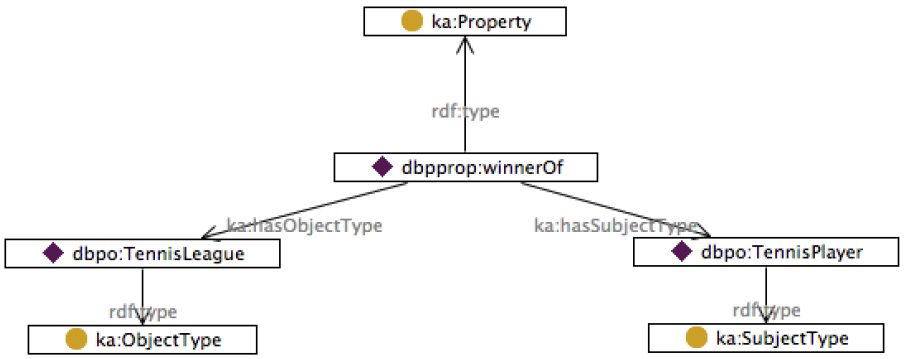
Knowledge patterns are not only symbolic patterns: they also have an interpretation, be it formal, or cognitive [66]. Hence, we need a boundary, which enables to select a set of triples that make a KP meaningful. In order to choose the boundary we have defined a set of indicators that provide path related statistics. These indicators are described in Table 5.1.

Indicator	**Description**
$nRes(C)$	number of resources typed with a certain class C, $\lvert\{r_i$ `rdf:type` $C\}\rvert$
$nSubjectRes(P_{i,k,j})$	number of distinct resources that participate in a path as subjects, $\lvert\{(s_i$ `rdf:type` $S_i) \in P_{i,k,j} = [S_i, p_k, O_j]\}\rvert$
$pathPopularity(P_{i,k,j}, S_i)$	The ratio of how many distinct resources of a certain type participate as subject in a path to the total number of resources of that type. Intuitively, it indicates the popularity of a path for a certain subject type, $nSubjectRes(P_{i,k,j} = [S_i, p_k, O_j])$ divided by $nRes(S_i)$
$nPathOcc(P_{i,k,j})$	number of occurrences of a path $P_{i,k,j} = [S_i, p_k, O_j]$
$nPath(S_i)$	number of distinct paths having a same subject type S_i, e.g. the number of paths having `dbpo:TennisPlayer` as subject type
$AvPathOcc(S_i)$	sum of all $nPathOcc(P_{i,k,j})$ having a subject type S_i divided by $nPath(S_i)$ e.g. the avarage number of occurrences of paths having `dbpo:Philosopher` as subject type

Table 5.1: Indicators used for empirical analysis of wikilink paths.



(a) RDF graph extracted from DBpedia representing the triple `dbpedia:Andre_Agassi dbpprop:winnerOf dbpedia:Davis_Cup` and the types and the related taxonomies associated to the subject and the object of the triple.



(b) Path discovered. The types and the property are represented as individuals of classes of the knowledge architecture ontology via OWL2 punning.

Figure 5.2: Path discovered from the triple `dbpedia:Andre_Agassi dbpprop:winnerOf dbpedia:Davis_Cup`.

We choose the boundary of a KP by defining a threshold t for $pathPopularity(P_{i,k,j}, S_i)$. Namely, the $pathPopularity(P_{i,k,j}, S_i)$ is the ratio of how many distinct resources of a certain type participate as subject in a path to the total number of resources of that type. For example the $pathPopularity(P_{i,k,j}, S_i)$ for the path $P_{Agassi,winnerOf,Davis}$ (indicated as P^* for space reasons) is calculated in the following way [5]:

$$pathPopularity(P^*, S_{TennisPlayer}) = \frac{nSubjectRes(P^*)}{nRes(TennisPlayer)}$$

[5]This example is based on real data extracted from DBpedia 3.6

$$= \frac{443}{1630}$$
$$= 0.2717$$

In fact, the number of distinct `TennisPlayer` individuals that participate in the path $P_{Agassi,winnerOf,Davis}$ is 443 and the number of resources typed with the class `TennisPlayer` is 1630. Intuitively, the $pathPopularity(P_{i,k,j}, S_i)$ gives an idea about how much is frequent a certain type path in data.

Accordingly, we give the following definition of $KP(S_i)$ for a type S_i [6].

Definition 3 (KP Boundary) *Let S_i be a type (i.e. `rdf:type`) of a Web resource, O_j $(j = 1, .., n)$ a list of types, $P_{i,j} = [S_i, p, O_j]$ and t a threshold value. Given the triples:*

$$\texttt{subj pred obj}$$
$$\texttt{subj rdf:type } S_i$$
$$\texttt{obj rdf:type } O_j$$

we state that $KP(S_i)$ is a set of paths, such that

$$P_{i,j} \in KP(S_i) \iff pathPopularity(P_{i,j}, S_i) \geq t \tag{5.1}$$

In Section 5.2 we will show an approach for the selection of a good value for t that we used for the extraction of KPs from Wikipedia links.

5.1.3 KP formalization

Based on the previous steps we can store paths and their associated indicators in a dataset, according to the knowledge architecture ontology. Then, we are able to generate the KPs by performing a refactoring of the knowledge architecture data into OWL2 ontologies. Given a certain namespace `kp:` and an $KP(S_i) =$

[6] We assume that a type is associated to a resource with a `rdf:type` statement.

$[S_i, p_1, O_1], \ldots, [S_i, p_n, O_n]$, we formalize it in OWL2 by applying the following translation procedure:

- the name of the OWL file is `kp:` followed by the local name of S e.g., `kp:TennisPlayer.owl`. Below we refer to the namespace of a specific KP through the generic prefix `kpS:`;
- S_i and O_j $j = 1, \ldots, n$ are refactored as `owl:Class` entities (they keep their original URI);
- p_j keep their original URI and are refactored as `owl:ObjectProperty` entities;
- for each O_j we create a sub-property of p_{i+n}, $kpS:O_j$ that has the same local name as O_j and the `kpS:` namespace; e.g. `kp:TennisPlayer.owl# TennisLeague`.
- for each $kpS:O_j$ we add an `owl:allVauesFrom` restriction to S_i on $kpS:O_j$, with range O_j.

For example, if $Path_{Agassi,winnerOf,Davis}$ (cf. Figure 5.2) is part of an EKP, it gets formalized as follows:

```
Prefix: dbpo: <http://dbpedia.org/ontology/>
Prefix:
   kpS: <http://www.ontologydesignpatterns.org/kp/TennisPlayer.owl#>
Ontology: <http://www.ontologydesignpatterns.org/kp/TennisPlayer.owl>
Class: dbpo:TennisPlayer
   SubClassOf:
       kpS:TennisLeague only dbpo:TennisLeague
Class: dbpo:TennisLeague
ObjectProperty: kpS:TennisLeague
   SubPropertyOf: dbpo:Organisation
...
```

5.2 A case study: extracting KPs from Wikipedia links

Wikipedia [7] is a peculiar source for KP extraction. In fact, it is particularly suitable because it has an RDF dump in Linked Data, i.e., DBpedia [88] [8] and is built according to some design guidelines that make KP investigation easier. The guidelines are the following:

- each wiki page describes a single topic;
- each topic corresponds to a single resource in DBpedia;
- wikilinks relate wiki pages. Hence each wikilink links two DBpedia resources;
- each resource in DBpedia can be associated to a type (with an `rdf:type` statement);

For these reasons we have used Wikipedia (and DBpedia) as a case study for KP extraction.

5.2.1 Matherial

We have extracted KPs from a subset of the DBpedia wikilink dataset (*dbpedia_page_links_en*), and have created a new dataset (*DBPOwikilinks*) including only links between resources that are typed by DBpedia ontology version 3.6 (DBPO) classes (15.52% of the total wikilinks in *dbpedia_page_links_en*). *DBPOwikilinks* excludes a lot of links that would create semantic interpretation issues, e.g. images (e.g. `dbpedia:Image:Twitter_2010_logo.svg`), Wikipedia categories (e.g. `dbpedia:CAT:Vampires_in_comics`), untyped resources (e.g. `dbpedia:%23Drogo`), etc. DBPO includes 272 classes, which are used to type 10.46% of the resources involved in *dbpedia_page_links_en*. We also use *dbpedia_instance_types_en*, which contains type axioms, i.e. `rdf:type` triples. This dataset contains the materialization

[7] Wikipedia: http://en.wikipedia.org
[8] DBpedia: http://dbpedia.org

of all inherited types. Table 5.2 summarizes the figures described above. The reason of the usage of absolute values in table is twofold (i) to give to the reader an idea about the size of data managed and (ii) to keep the distiction among the datasets.

Dataset	Description	Indicator	Value
DBPO	DBpedia ontology	Number of classes	272
dbpedia_instance_types_en	Resource types i.e. rdf:type triples	Number of resources having a DBPO type rdf:type triples	1,668,503 6,173,940
dbpedia_page_links_en	Wikilinks triples	Number of resources used in wikilinks Number of wikilinks	15,944,381 107,892,317
DBPOwikilinks	Wikilinks involving only resources typed with DBPO classes	Number of resources used in wikilinks Number of wikilinks	1,668,503 16,745,830

Table 5.2: Dataset used and associated figures.

5.2.2 Obtained results

We have extracted 33,052 paths from the English wikilink datasets, however many of them are not relevant either because they have a limited number of occurrences, or because their subject type is rarely used. In order to select the paths useful for KP discovery (our goal) we have considered the following criteria:

- **Usage in the wikilink dataset**. The resources involved in *dbpedia_page_links_en* are typed with any of 250 DBPO classes (out of 272). Though, we are interested in *direct types*[9] of resources in order to avoid redundancies when counting path occurrences. For example, the resource `dbpedia:Ludwik_Fleck` has three types `dbpo:Scientist;dbpo:Person;owl:Thing` because type assertions in DBpedia are materialized along the hirerachy of DBPO. Hence, only `dbpo:Scientist` is relevant to our study. Based on this criterion, we keep only 228 DBPO classes and the number of paths decreases to 25,407.

[9]In current work, we are also investigating indirectly typed resource count, which might lead to different KPs, and to empirically studying KP ordering.

- **Number of resources typed by a class C (i.e., $nRes(C)$)**. Looking at the distribution of resource types, we have noticed that 99.98% of DBPO classes have at least 30 resource instances. Therefore we have decided to keep paths whose subject type C has at least $nRes(C)=30$.

- **Number of path occurrences having a same subject type (i.e., $nPathOcc(P_{i,k,j})$)**. The average number of outgoing wikilinks per resource in *dbpedia_page_links_en* is 10. Based on this observation and on the previous criterion, we have decided to keep paths having at least $nPathOcc(P_{i,k,j})$=30*10=300.

After applying these two criteria, only 184 classes and 21,503 paths are retained. For example, we have discarded the path `[Album,Drug]` [10], which has 226 occurrences, and the type `dbpo:AustralianFootballLeague`, which has 3 instances.

5.2.3 KP discovery

At this point, we had each of the 184 classes used as subject types associated with a set of paths, each set with a cardinality ranging between 2 and 191 (with 86.29% of subjects bearing at least 20 paths). Our definition of KP requires that its backbone be constituted of a small number of object types, typically below 10, considering the existing resources of models that can be considered as KPs (see later in this subsection for details). In order to generate KPs from the extracted paths, we need to decide what threshold should be used for selecting them, which eventually creates appropriate *boundaries* for KPs. In order to establish some meaningful threshold, we have computed the ranked distributions of $pathPopularity(P_{i,k,j}, S_i)$ for each selected subject type, and measured the correlations between them. Then, we have fine-tuned these findings by means of a user study (cf. Section 5.2.4), which had the dual function of both evaluating our results, and suggesting relevance criteria for generating the KP resource. Our aim is to build a *prototypical* ranking of the $pathPopularity(P_{i,k,j}, S_i)$ of the selected 184 subject types, called

[10] In this use case we represent paths as couples `[SubjetType,ObjectType]` because in the *dbpedia_page_links_en* dataset the only property used is `dbpo:wikiPageWikiLink`.

$pathPopularity_{DBpedia}$, which should show how relevant paths for subject types are typically distributed according to the Wikipedia crowds, hence allowing us to propose a threshold criterion for any subject type. We have proceeded as follows.

1. We have chosen the top-ranked 40 paths ($P_{i,k,j}$) for each subject type (S_i), each constituting a $pathPopularity(P_{i,k,j}, S_i)$. Some subject types have less than 40 paths: in such cases, we have added 0 values until filling the gap. The number 40 has been chosen so that it is large enough to include not only paths covering at least 1% of the resources, but also much rarer ones, belonging to the long tail.

2. In order to assess if a prototypical ranking $pathPopularity_{DBpedia}$ would make sense, we have performed a multiple correlation between the different $pathPopularity(P_{i,k,j}, S_i)$. In case of low correlation, the prototypical ranking would create odd effects when applied to heterogeneous rank distributions across different S_i. In case of high correlation, the prototype would make sense, and we can get reassured that the taxonomy we have used (DBPO in this experiment) nicely fits the way wikilinks are created by the Wikipedia crowds.

3. We have created a prototypical distribution $pathPopularity_{DBpedia}$ that is representative for all S_i distributions. Such a distribution is then used to hypothesize some thresholds for the relevance of $P_{i,k,j}$ when creating boundaries for KPs. The thresholds are used in Section 5.2.4 to evaluate the proposed KPs with respect to the rankings produced during the user study.

In order to measure the distribution from step 2, we have used the Pearson correlation measure ρ, ranging from -1 (no agreement) to +1 (complete agreement), between two variables X and Y i.e. for two different S_i in our case. The correlation has been generalized to all 16,836 pairs of the 184 $pathPopularity(P_{i,k,j}, S_i)$ ranking sets ($184 * 183/2$), in order to gather a multiple correlation. The value of such multiple correlation is 0.906, hence excellent.

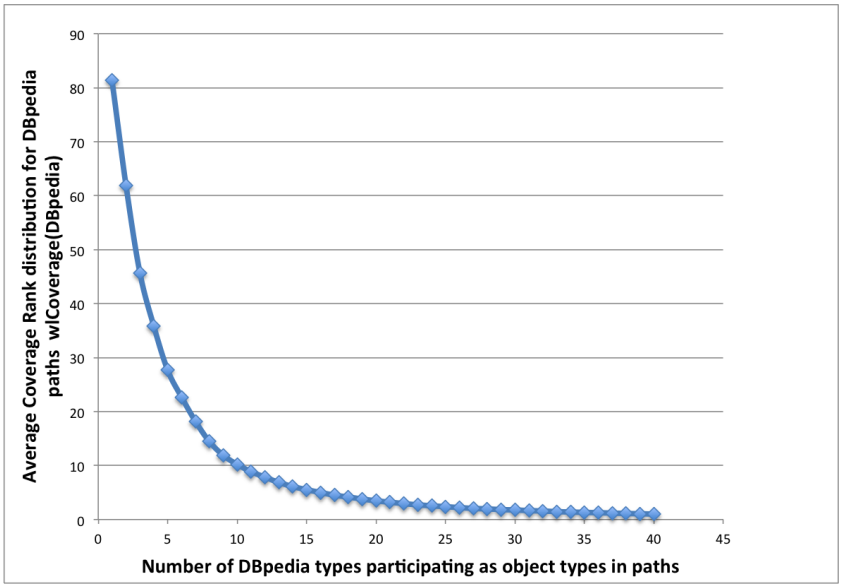


Figure 5.3: Distribution of $pathPopularity_{DBpedia}$: the average values of popularity rank i.e., $pathPopularity(P_{i,k,j}, S_i)$, for DBpedia paths. The x-axis indicates how many paths (on average) are above a certain value t of $pathPopularity(P, S)$.

Once reassured on the stability of $pathPopularity(P_{i,j,j}, S_i)$ across the different S_i, we have derived (step 3) $pathPopularity_{DBpedia}$, depicted in Figure 5.3.

In order to establish some reasonable relevance thresholds, $pathPopularity_{DBpedia}$ has been submitted to K-Means Clustering, which generates 3 small clusters with popularity ranks above 22.67%, and 1 large cluster (85% of the 40 ranks) with popularity ranks below 18.18%. The three small clusters includes seven paths: this feature supports the buzz in cognitive science about a supposed amount of 7 ± 2 objects that are typically manipulated by the cognitive systems of humans in their recognition tasks [100, 98]. While the 7 ± 2 conjecture is highly debated, and possibly too generic to be defended, this observation has been used to hypothesize a first threshold criterion: since the seventh rank is at 18.18% in $pathPopularity_{DBpedia}$, this value of $pathPopularity(P_{i,j}, S_i)$ will be our first guess for including a path in an KP. We propose a second threshold based on FrameNet [11], a lexical database, grounded in a textual corpus, of situation types called *frames*. FrameNet is currently the only cognitively-based resource of potential knowledge patterns (the frames, cf. [104]). The second threshold (11%) is provided by the average number of frame el-

Path	$nPathOcc(P_{i,k,j})$	$nSubjectRes(P_{i,k,j})$	$pathPopularity(P_{i,k,j}, S_i)$ (%)
`[Album,Album]`	170,227	78,137	78.89
`[Album,MusicGenre]`	108,928	68,944	69.61
`[Album,MusicalArtist]`	308,619	68,930	69.59
`[Album,Band]`	125,919	62,762	63.37
`[Album,Website]`	62,772	49,264	49.74
`[Album,RecordLabel]`	56,285	47,058	47.51
`[Album,Single]`	114,181	29,051	29.33
`[Album,Country]`	40,296	25,430	25.67

Table 5.3: Sample paths for the subject type `Album`: number of path occurrences, distinct subject resources, and popularity percentage value. Paths are expressed as couples `[SubjectType,ObjectType]` because in the *dbpedia_page_links_en* dataset the only property used is `dbpo:wikiPageWikiLink`.

ements in FrameNet frames (frame elements roughly correspond to paths for KPs), which is 9 (the ninth rank in $pathPopularity_{DBpedia}$ is at 11%). The mode value of frame elements associated with a frame is 7, which further supports our proposal for the first threshold. An example of the paths selected for a subject type according to the first threshold is depicted in Tab. 5.3, where some paths for the type `Album` are ranked according to their $pathPopularity(P_{i,k,j}, S_i)$. In Subsection 5.2.4 we describe an evaluation of these threshold criteria by means of a user study.

Threshold criteria are also used to enrich the formal interpretation of KPs. Our proposal, implemented in the OWL2 KP repository, considers the first threshold as an indicator for an existential quantification over an OWL restriction representing a certain path. For example, `[Album,MusicGenre]` is a highly-popular path in the `Album` KP. We interpret high-popularity as a feature for generating an existential interpretation, i.e.: `Album` $\sqsubseteq$ ($\exists$`MusicGenre.MusicGenre`). This interpretation suggests that each resource typed as an `Album` has at least one `MusicGenre`, which is intuitively correct. Notice that even if all paths have a $pathPopularity(P_{i,j}, S_i)$ of less that 100%, we should keep in mind that semantic interpretation over the Web is made in open-world, therefore we feel free to assume that such incompleteness is a necessary feature of Web-based knowledge (and possibly of any crowd-sourced knowledge).

We have stored paths and their associated indicators in a dataset, according to an OWL vocabulary called *knowledge architecture*[11]. Then, we have formalized the KPs as OWL2 ontologies by applying the translation recipe explained in Section 5.1.3. The result of the formalization is a collection of 184 KPs called Encyclopedic Knowledge Patterns [12](EKPs). This name emphasizes that they are grounded in encyclopedic knowledge expressed as linked data, i.e. DBpedia, and as natural language text, i.e. Wikipedia.

5.2.4 Evaluation

Although our empirical observations on DBpedia could give us means for defining a value for the threshold t (see Definition 3 and Section 5.2.2), we still have to prove that emerging KPs provide an intuitive schema for organizing knowledge. Therefore, we have conducted a user study for making users identify the KPs associated with a sample set of DBPO classes, and for comparing them with those emerging from our empirical observations.

User study

We have selected a sample of 12 DBPO classes that span social, media, commercial, science, technology, geographical, and governmental domains. They are listed in Table 5.4. For each class, we indicate the number of its resources, the number of paths it participates in as subject type, and the average number of occurrences of its associated paths. We have asked the users to express their judgement on how relevant were a number of (object) types (i.e., paths) for describing things of a certain (subject) type. The following sentence has been used for describing the user study task to the users:

> *We want to study the best way to describe things by linking them to other things. For example, if you want to describe a person, you might want*

[11] `http://www.ontologydesignpatterns.org/ont/lod-analysis-path.owl`
[12] EKP are available on line at `http://ontologydesignpatterns.org/ekp/`

DBPO class type	**nRes(S)**	$nPath(S_i)$	$AvPathOcc(S_i)$
Language	3,246	99	29.27
Philosopher	1,009	112	18.29
Writer	10,102	172	15.30
Ambassador	286	85	15.58
Legislature	453	83	25.11
Album	99,047	172	11.71
Radio Station	16,310	151	7.31
Administrative Region	31,386	185	11.30
Country	2,234	169	35.16
Insect	37,742	98	9.16
Disease	5,215	153	12.10
Aircraft	6,420	126	10.32

Table 5.4: DBPO classes used in the user-study and their related figures.

> *to link it to other persons, organizations, places, etc. In other words, what are the most relevant types of things that can be used to describe a certain type of things?*

We asked the users to fill a number of tables, each addressing a class in the sample described in Table 5.4. Each table has three columns:

- *Type 1* indicating the class of things (subjects) to be described e.g. `Country`;
- A second column to be filled with a relevance value for each row based on a scale of five relevance values, Table 5.5 shows the scale of relevance values and their interpretations as they have been provided to the users. Relevance values had to be associated with each element of Type 2;
- *Type 2* indicating a list of classes of the paths (i.e. the object types) in which *Type 1* participates as subject type. These were the suggested types of things that can be linked for describing entities of *Type 1* e.g. `Administrative Region`, `Airport`, `Book`, etc.

By observing the figures of DBPO classes (cf. Table 5.4) we realized that the entire list of paths associated with a subject type would have been too long to be proposed to the users. For example, if *Type 1* was `Country`, the users would have been submitted 169 rows for *Type 2*. Hence, we decided a criterion for selecting

Relevance score	Interpretation
1	The type is irrelevant;
2	The type is slightly irrelevant;
3	I am undecided between 2 and 4;
4	The type is relevant but can be optional;
5	The type is relevant and should be used for the description.

Table 5.5: Ordinal (Likert) scale of relevance scores.

User group	Average inter-rater agreement
Group 1	0.700
Group 2	0.665

Table 5.6: Average coefficient of concordance for ranks (Kendall's W) for the two groups of users.

a representative set of such paths. We have set a value for t to 18% and have included, in the sample set, all $P_{i,j}$ such that $pathPopularity(P_{i,k,j}, S_i) \geq 18\%$. Furthermore, we have also included an additional random set of 14 $P_{i,j}$ such that $pathPopularity(P_{i,k,j}, S_i) < 18\%$.

We have divided the sample set of classes into two groups of 6. We had ten users evaluating one group, and seven users evaluating the other group. Notice that the users come from different cultures (Italy, Germany, France, Japan, Serbia, Sweden, Tunisia, and Netherlands), and speak different mother tongues. In practice, we wanted to avoid focusing on one specific language or culture, at the risk of reducing consensus.

In order to use the KPs resulting from the user study as a reference for next steps in our evaluation task, we needed to check the inter-rater agreement. We have computed the Kendall's coefficient of concordance for ranks (W), for all analyzed DBPO classes, which calculates agreements between 3 or more rankers as they rank a number of subjects according to a particular characteristic. Kendall's W ranges from 0 (no agreement) to 1 (complete agreement). Table 5.6 reports such values for the two groups of users, which show that we have reached a good consensus in both cases. Additionally, Table 5.7 reports W values for each class in the evaluation sample.

DBPO class	Agreement	Reliability	DBPO class	Agreement	Reliability
Language	0.836	0.976	Philosopher	0.551	0.865
Writer	0.749	0.958	Ambassador	0.543	0.915
Legislature	0.612	0.888	Album	0.800	0.969
Radio Station	0.680	0.912	Administrative Region	0.692	0.946
Country	0.645	0.896	Insect	0.583	0.929
Disease	0.823	0.957	Aircraft	0.677	0.931

Table 5.7: Inter-rater agreement computed with Kendall's W (for all values $p < 0.0001$) and reliability test computed with Cronbach's alpha

Evaluation of emerging DBpedia KPs

Through correlation with user-study results we want to answer the following question: how good is DBpedia as a source of KPs? The second step towards deciding t for the generation of KPs has been to compare DBpedia KPs to those emerging from the users' choices. DBpedia $KP(S_i)$ would result from a selection of paths having S_i as subject type, based on their associated $pathPopularity(P_{i,k,j}, S_i)$ values (to be $\geq t$). We had to compare the $pathPopularity(P_{i,k,j}, S_i)$ of the paths associated with the DBPO sample classes (cf. Table 5.4), to the relevance scores assigned by the users. Therefore, we needed to define a mapping function between $pathPopularity(P_{i,k,j}, S_i)$ values and the 5-level scale of relevance scores (Table 5.5).

We have defined the mapping by splitting the $pathPopularity_{DBpedia}$ distribution (cf. Figure 5.3) into 5 intervals, each corresponding to the 5 relevance scores of the Likert scale used in the user-study. Table 5.8 shows our hypothesis of such mapping. The hypothesis is based on the thresholds defined in Section 5.2.2. The mapping function serves our purpose of performing the comparison and identifying the best value for t, which is our ultimate goal. In case of scarce correlation, we expected to fine-tune the intervals for finding a better correlation and identifying the best t. Based on the mapping function, we have computed the relevance scores that DBpedia would assign to the 12 sample types, and calculated the Spearman correlation value (ρ) wich ranges from -1 (no agreement) to $+1$ (complete agreement) by using the *means* of relevance scores assigned by the users. This measure gives us an indication on how precisely DBpedia wikilinks allow us to identify KPs as compared to those drawn by the users. As shown in Table 5.9, there is a good

$pathPopularity_{DBpedia}$ **interval**	**Relevance score**
[18, 100]	5
[11, 18[	4
]2, 11[	3
]1, 2]	2
[0, 1]	1

Table 5.8: Mapping between $wlCoverage_{DBpedia}$ intervals and the relevance score scale.

User group	**Correl. with DBpedia**
Group 1	0.777
Group 2	0.717

Table 5.9: Average multiple correlation (Spearman ρ) between users' assigned scores, and $pathPopularity_{DBpedia}$ based scores.

DBPO class	**Correl. users / DBpedia**	**DBpedia type**	**Correl. users / DBpedia**
Language	0.893	Philosopher	0.661
Writer	0.748	Ambassador	0.655
Legislature	0.716	Album	0.871
Radio Station	0.772	Administrative Region	0.874
Country	0.665	Insect	0.624
Disease	0.824	Aircraft	0.664

Table 5.10: Multiple correlation coefficient (ρ) between users's assigned score, and $pathPopularity_{DBpedia}$ based score.

correlation between the two distributions. Analogously, Table 5.10 shows the multiple correlation values computed for each class, which are significantly high. Hence, they indicate a satisfactory precision.

We can conclude that our hypothesis is supported by these findings, and that Wikipedia wikilinks are a good source for KPs. We have tested alternative values for t, and we have found that our hypothesized mapping (cf. Table 5.8) provides the best correlation values among them. Consequently, we have set the threshold value for KP boundaries (cf. Definition 3) as $t = 11\%$.

Chapter 6

Enrichment of sources for Knowledge Pattern extraction

In this Chapter we present:

- a method for enriching the limited intensional as well as extensional coverage of Linked Data with respect to ontologies and controlled vocabularies. Such a method exploits the natural language for generating new axioms like `rdf:type` from entity definitions. In fact, Linked Data is rich of natural language, for example annotations (e.g. `rdfs:comment`) or corpora on which datasets are grounded on (e.g. DBpedia is grounded on Wikipedia).

- a case study (conducted in [64]) about the automatic typing of DBpedia entities based on entity definitions available in Wikipedia abstracts. The result of the case study is an algorithm and a tool (cf. Section 7.4.2) called Tìpalo [1];

- an evaluation of Tìpalo based on a golden standard and a user study that shows a good accuracy in selecting the right types (with related taxonomies and WordNet synsets in order to gather the most correct sense) for an entity (cf. Section 6.2.3).

- a case study (conducted in [83, 44]) about the automatic identification of the nature of citations in scholarly articles. In fact citations can be seen as

[1] CiTalO: `http://wit.istc.cnr.it/stlab-tools/tipalo`

links between articles and we want to be able to capture the reasons behind their usage. The result of this case study is an algorithm and a tool (cf. Section 7.4.2) called CiTalO [2];

- an evaluation of CiTalO based on a user study that shows how our algorithm still needs improvements (cf. Section 6.3.2).

6.1 Enriching links with natural language

In previous chapters we have discussed about how to gather Knowledge Patterns either from existing KP-like sources or from Linked Data. In the method introduced in chapter 4 we have identified two main steps for KP transformation, i.e., (i) the reengineering of the source to pure RDF triples compliant to the original schema and data; (ii) the refactoring, a customized, task-oriented way to address KP semantics. Besides the format transformation, the reengineering step is a method for source enriching as it allows to generate RDF triples from non-RDF structured data.

The method introduced in chapter 5 is based on the notion of type paths (i.e., sequences of connected triple patterns whose occurrences have (i) the same `rdf:type` for their subject nodes, and (ii) the same `rdf:type`) for their object nodes) and allows to extract KPs by analyzing the linking structure of Linked Data. Unfortunately, the usage of ontologies and controlled vocabularies in Linked Data is limited. For example, the ontological coverage of the two *de facto* reference ontologies for DBpedia, DBpedia, i.e., DBPO [3] and YAGO [133], is partial. This partiality is both extensionally (number of typed resources), and intensionally (conceptual completeness), since they rely on Wikipedia categories, and infoboxes (that are not included in all Wikipedia pages).

For example, it is not possible to identify a type path between the two entities `dbpedia:Vladimir_Kramnik` and `dbpedia:Russia` from the RDF graph depicted in Figure 6.1. In fact, neither a type is associated to `dbpedia:Vladimir_Kramnik`

[2] CiTalO: `http://wit.istc.cnr.it:8080/tools/citalo`
[3] http://dbpedia.org/ontology/

nor a type from such an entity can be inferred from the universe of the property `dbpo:country` as no restriction for domain and range are provided for this property.

Figure 6.1: An example of limited extensional coverage, which prevents the identification of a type path between the entities `dbpedia:Vladimir_Kramnik` and `dbpedia:Russia`."

If we want to extend our KP extraction method in order to take into account HTML hyperlinks, the problem is even more complex. In fact, hyperlinks are unlabeled and there is no ontological coverage that can be exploited for the identification of type paths. Hence, we need techniques aimed at enhancing links in the case of lack of type information.

Our hypothesis are the following:

Hypothesis 1 *The natural language available in annotations of Linked Data resources, e.g.,* ***`rdfs:comment`****,* ***`rdfs:label`****, etc., can be exploited for enriching these resources with additional metadata, e.g.,* ***`rdf:type`****.*

Hypothesis 2 *The natural language that surrounds an hypelinks can be exploited for enhancing them and their linked entities to RDF.*

According to the hypothesis 1 and 2 the natural language should convey a rich knowledge that might be exploited in order to enrich data with additional metadata that we need for KP extraction. We have identified a method based on the following steps:

1. Natural language deep parsing of text;
2. Graph-pattern matching;
3. Word-sense disambiguation;

4. Ontology alignment.

In next sections we give details about these steps.

6.1.1 Natural language deep parsing of text

This step maps the natural language to a logical form, i.e. OWL. In order to accomplish this task we use **FRED**[4] [117]. FRED performs ontology learning by relying on Boxer [40], which implements *computational semantics*, a deep parsing method that produces a logical representation of NL sentences in DRT. FRED implements an alignment model and a set of heuristics for transforming DRT representations to RDF and OWL representations. In the context of our method, FRED is in charge of "reading" an entity NL definition, and producing its OWL representation, including a taxonomy of types. For example, let us suppose to enrich the entity `dbpedia:Vladimir_Kramnik` by adding `rdf:type` axioms by interpreting the natural language available from `dbpo:abstract` properties. Hence, given the following abstract

Vladimir Kramnik is a Russian chess grandmaster.

FRED returns the OWL graph depicted in Figure 6.2.

6.1.2 Graph-pattern matching

Once the natural language has been mapped to a logical form, it is possible to recognize specific typical forms of the language, i.e., definitions, facts, etc. This forms are recognized by applying a list of known graph-patterns (GPs) on FRED output. GPs can be expressed by means of SPARQL queries. For example the following graph pattern [5]:

[4]FRED is available online at `http://wit.istc.cnr.it/stlab-tools/fred`

[5]wt: `http://www.ontologydesignpatterns.org/ont/wikipedia/type/`

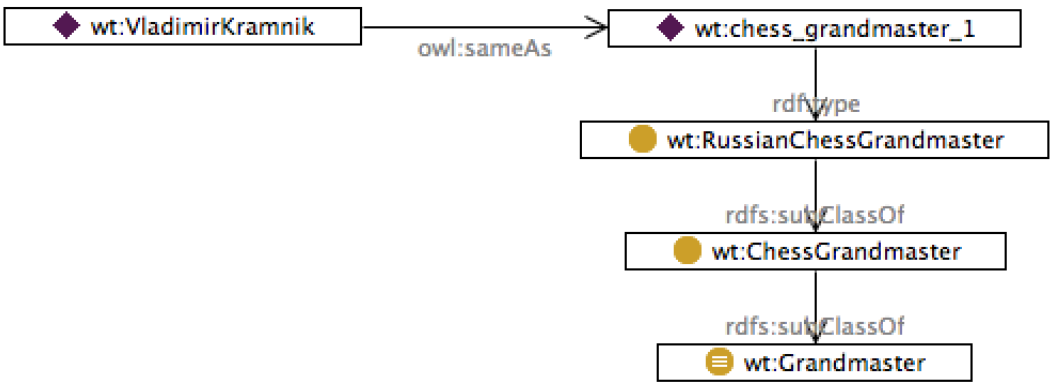


Figure 6.2: FRED result for the definition "Vladimir Borisovich Kramnik is a Russian chess grandmaster."

```
CONSTRUCT {?type rdfs:subClassOf ?subclass}
WHERE {
    wt:Vladimir_Kramnik owl:sameAs ?x .
    ?x rdf:type ?type .
    ?type rdfs:subClassOf+ ?subclass
}
```

returns the type and its related taxonomy from the logical representation of the abstract associated to the entity `dbpedia:Vladimir_Kramnik`. The result taxonomy is the following:

```
wt:RussianChessGrandmaster rdfs:subClassOf wt:ChessGrandmaster
        wt:ChessGrandmaster rdfs:subClassOf wt:Grandmaster
```

The identification of the GPs is task-specific. We will see how to model graph patterns for identifying candidate terms for the tasks of typing DBpedia entities (cf. Section 6.2.2) and citations in scholarly articles (cf. Section 6.3.1).

6.1.3 Word-sense disambiguation

After having identified the concepts expressing the types of an entity and their taxonomical relations, we have to gather their correct sense: we need word-sense disambiguation. A possible way to achieve this goal is to identify alignments between the type terms and WordNet terms. A first solution involves tools like UKB [3], a tool which returns the WordNet synset for a term, looking for the one that fits best the context given by the entity definition. UKB provids good results in terms of precision and recall although its speed performance needs improvement in order to apply it on large datasets, e.g., DBpedia. A second solution is to select the most frequent WordNet sense for a given term, which is very efficient in terms of speed, but shows lower precision and recall. This step allows us to assign a WordNet type (corresponding to the identified synset) to an entity. Referring to the above example (i.e., definition of `Vladimir_Kramnik`), the word-sense disambiguation would allow to produce the following additional triples[6]:

```
wt:Grandmaster owl:equivalentTo wn30syn:synset-grandmaster-noun-1
```

Where the synset `wn30syn:synset-grandmaster-noun-1` identifies in WordNet *a player of exceptional or world class skill in chess or bridge.*

6.1.4 Ontology alignment

So far the typing process produces a set of newly defined concepts, and disambiguates them to a WordNet sense. The final step consists in linking such concepts to other Semantic Web ontologies, in order to support shared interpretation and linked data enrichment. The alignment task can be performed in several ways, such as by providing manual mappings between WordNet synsets and other ontologies

[6] wn30syn: = `http://purl.org/vocabularies/princeton/wn30/instances/`

or by exploiting more sophisticated tools like the Alignment API [46], which allows to discover, expressing and sharing ontology alignments. The need of aligning ontologies is simple and can be summarized as the need to achieve interoperability among heterogeneous systems within the Semantic web. However, as the ontologies underlying two systems are not necessarily compatible, they may in turn need to be reconciled. In order to exemplify this task we provide an example of an alignment among `wt:Grandmaster`, WordNet synsets and some foundational ontology classes from Dolce [60]. The alignment produces the following triples:

```
wt:Grandmaster rdfs:subClassOf dul:Person
wt:Grandmaster rdfs:subClassOf wn30:supersense-noun_person
```

meaning that the term "grandmaster" associated with the WordNet sense `wn30syn:synset-grandmaster-noun-1` (as provided by the WSD) is aligned to the class `dul:Person` of Dolce [7] ontology. Analogously, the same term is aligned to the WordNet super sense "person". Finally it is easy to associate the entity `wt:VladimirKramnik` to `dbpedia:Vladimir_Kramnik` for example with a named entity recognizer [8] (NER). Hence, a final RDF graph for our example can be that depicted in Figure 6.3.

6.2 Automatic typing of DBpedia entities

In this section we present a case study which applies the method described in the previous section for automatically typing DBpedia entities. The contribution of the case study are (i) a natural ontology of Wikipedia [9] (cf. Section 6.2.4) and (ii) an algorithm and a tool (cf. Section 7.4.2), Tìpalo, which is the result of a work presented in [64] and that allow to generate RDF types with related taxonomies

[7] Dolce: `http://www.ontologydesignpatterns.org/ont/d0.owl`.
[8] FRED has an internal NER that links recognized entities to DBpedia.
[9] It is natural as it is extracted from natural language definitions of DBpedia entities.

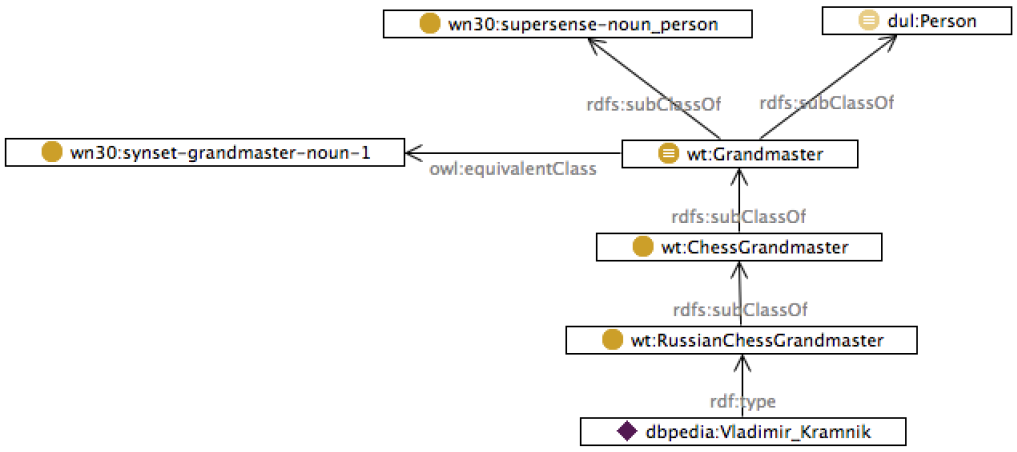


Figure 6.3: An example of the enrichment of the entity `dbpedia:Vladimir_Kramnik` based on its natural language definition available from the property `dbpo:abstract`.

given given a certain entity in DBpedia. We refer to Chapter 7 for architectural and implementation details about Tìpalo.

6.2.1 Material

In the context of this case study, we have used and produced a number of resources.

Wikipedia and DBpedia

Wikipedia is a collaboratively built multilingual encyclopedia on the Web. Each Wikipedia page usually refers to a single entity, and is manually associated to a number of categories. Entities referenced by Wikipedia pages are represented in DBpedia, the RDF version of Wikipedia. Currently, English Wikipedia contains 4M articles[10], while DBpedia wikilink dataset counts ∼15M distinct entities (as of version 3.6). One main reason for this big difference in size is that many linked resources are referenced by *sections* of Wikipedia pages, hence lacking explicit categorization or infoboxes. However they have a URI, and a NL description, hence they

[10]Source: `http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`

are a rich source for linked data. Out of these ∼15M resources, ∼2.7 are typed with YAGO classes and ∼1.83M are typed with DBpedia classes. We use Wikipedia page contents as input for the *definition extraction* step (cf. Figure 6.4), for extracting entity definitions.

WordNet 3.0 and WordNet 3.0 supersense RDF

WordNet[11] is a large database of English words. It groups words into sets of synonyms, called *synsets*, each expressing a different concept. Although WordNet includes different types of words such as verbs and adjectives, for the sake of this work we limit the scope to nouns. Words that may express different meanings, i.e. polysemous words, are related to different synsets. In this work, we use the WordNet 3.0 RDF porting[12] in order to identify the type of an entity. Hence when such type is expressed by a polysemous word we need to identify the most appropriate one. To this aim we exploit a WSD engine named UKB [3]. Furthermore, WordNet 3.0 includes relations between synsets and supersenses, which are broad semantic categories. WordNet contains 41 supersenses, 25 of which are for nouns. We have produced a resource named *WordNet 3.0 Supersense RDF*[13] that encodes such alignments as RDF data. This RDF dataset is used by the *graph-pattern matching* step for producing triples relating entities and supersenses.

OntoWordNet (OWN) 2012

OWN 2012 is a RDF resource that updates and extends OWN[63]. OWN is an OWL version of WordNet, which includes semantic alignments between synsets and DULplus types. DULplus[14], extends DUL[15], which is the OWL light version of DOLCE + DnS [59] foundational ontology. OWN 2012 contains mappings between 859 general synsets and 60 DULplus classes. Such mappings have been propagated

[11] WordNet, http://wordnet.princeton.edu/

[12] http://semanticweb.cs.vu.nl/lod/wn30/

[13] http://www.ontologydesignpatterns.org/wn/wn30/wordnet-supersense.rdf

[14] Dolce Ultra Lite Plus ontology, `http://www.ontologydesignpatterns.org/ont/wn/dulplus.owl`

[15] Dolce Ultra Lite ontology, `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`

through the transitive closure of the hyponym relation in order to cover all ∼82000 synsets. In the context of this work, we have updated OWN to the WordNet 3.0 version, and performed a revision of the manual mapping relations. Furthermore, we have defined a lightweight foundational ontology called Dolce Zero[16], whose classes generalize a number of DULplus classes used in OWN. We have used a combination of 23 Dolce Zero and DULplus classes for building a sample Wikipedia ontology. The reduction to 23 classes has been made in order make it comparable to the WordNet supersense set, and to simplify the task of evaluators.

6.2.2 Typing entities

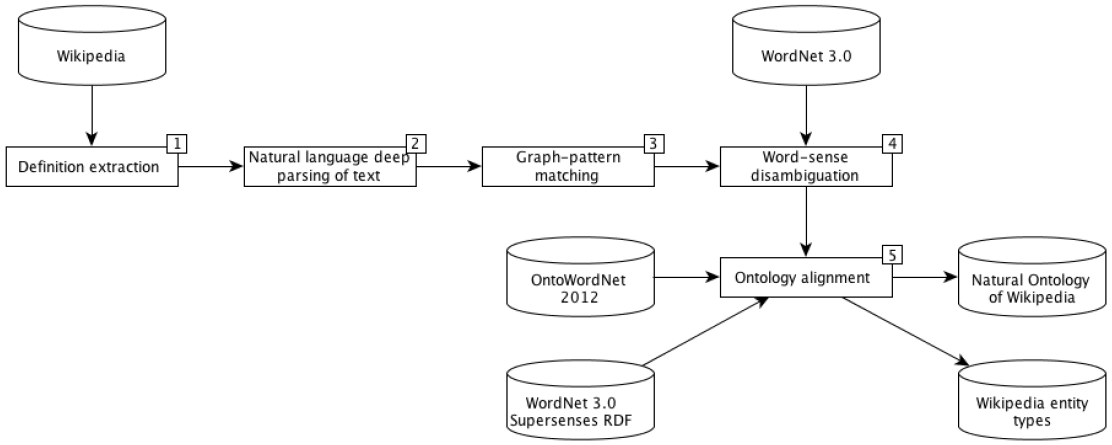


Figure 6.4: Pipeline implemented for automatic typing of DBpedia entities based on their natural language descriptions as provided in their corresponding Wikipedia pages. Numbers indicate the order of execution of a component in the pipeline. The output of a component i is passed as input to the next $i + 1$ component.

For this case study the method has been adapted in order to result as a pipeline of components and data sources, described below, which are applied in the sequence illustrated in Figure 6.4. We omit the description about steps 2, 4, 5 in the figure since they have been already described in Section 6.1.3. The step 5 is realized by exploiting the alignments between WordNet synsets and WordNet Super-senses

[16] `http://www.ontologydesignpatterns.org/d0.owl`

available from *WordNet 3.0 Supersense RDF* and between WordNet synsets and DULplus available form *OntoWordNet 2012*. Instead, in next paragraphs we give details about:

- how we actually capture entity definitions (step 1);
- how we have designed the graph patterns (step 3).

Definition extraction

The first step is the definition extraction, which consists in extracting the definition of a DBpedia entity from its corresponding Wikipedia page abstract. Instead of using the *dbpedia_short_abstracts_en* dataset of DBpedia, which provides abstract of DBpedia entities by means of `dbpo:abstract` datatype properties, we prefer to use HTML Wikipedia pages as we also need some markup for identifying the subject entity (the entity to which the Wikipedia page is referred to) within a sentence. We identify the shortest text including information about the entity type. Typically, an entity is defined in the first sentence of a Wikipedia page abstract, but sometimes the definition is expressed in one of the following sentences, can be a combination of two sentences, or even implicit. We rely on a set of heuristics based on lexico-syntactic patterns [75] and Wikipedia markup conventions in order to extract such sentences. A useful Wikipedia convention is the use of bold characters for visualizing the name of the referred entity in the page abstract: for example consider the Wikipedia page referring to "Vladimir Kramnik"[17] and the first paragraph of its abstract, depicted in Figure 6.5. Let us represent such paragraph as a sequence of n sentences $\{s_1, ..., s_n\}$. Typically, the bold words referring to the entity (*bold-name*) are included in a sentence $s_i, (i = 1, ..., n)$ that provides its definition according to a syntactic form of the type: "*bold–name <copula><predicative nominal||predicative adjective>*" (where *<copula>* is usually a form of the verb *to be*) e.g., "**Vladimir Borisovich Kramnik** is a Russian chess grandmaster". However, this is not always the case: sometimes, the sentence s_i containing the *bold-name* does not include any *<copula>*, while a

[17] `http://en.wikipedia.org/wiki/Vladimir_Kramnik`

Vladimir Kramnik

From Wikipedia, the free encyclopedia

Vladimir Borisovich Kramnik (Russian: Влади́мир Бори́сович Кра́мник; born 25 June 1975) is a Russian chess grandmaster. He was the Classical World Chess Champion from 2000 to 2006, and the undisputed World Chess Champion from 2006 to 2007. He has also won the two strongest tournaments (by rating strength) in chess history: the 2009 Mikhail Tal Memorial and the 2010 Grand Slam Masters Final. He has won three team gold medals and three individual medals at Chess Olympiads.[2]

Figure 6.5: First paragraph of the Wikipedia page abstract for the entity "Vladimir Kramnik".

<*copula*> can be found together with a co-reference to the entity, in one of the following sentences s_j. In such cases, we extract the entity definition by concatenating these two sentences i.e. $s_i + s_j$. If the abstract does not contain any *bold-name*, we inspect s_1: if it contains a <*copula*> we return s_1, otherwise we concatenate s_1 with the first of the next sentences e.g., s_i, containing a <*copula*> (i.e. $s_1 + s_i$). If none of the above is satisfied, we return s_1. We also apply additional heuristics for dealing with parentheses, and other punctuation. For the example in Figure 6.5 we return s_1: *Vladimir Borisovich Kramnik is a Russian chess grandmaster*, which contains the *bold-name* as well as a <*copula*>.

Graph-pattern matching

This step requires to identify, in FRED output graph, the paths providing typing information about the analyzed entity, and to discard the rest. Furthermore, we want to distinguish the case of an entity that is represented as an individual e.g. **Vladimir Kramnik**, from the case of an entity that is more appropriately represented as a class e.g., **Chess piece**. FRED output looks differently in these two situations as well as depending on the type of definition e.g., including a *copula* or parenthetic terms. For example, consider the entity **Chess piece**, which is a class entity, and is defined by *"Chess pieces, or chessmen, are the pieces deployed on a chessboard to play the game of chess."*. FRED output graph for such definition is depicted in Figure 6.6[18]. In this case, the graph paths encoding typing information comply

[18]For space reasons, we include only the portion of the graph of interest in this context. Readers interested in visualizing the complete graph can submit the sentence to FRED online `http://wit.`

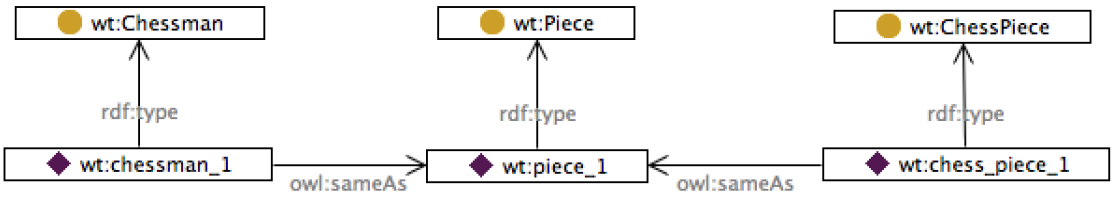


Figure 6.6: FRED result for the definition "Chess pieces, or chessmen, are the pieces deployed on a chessboard to play the game of chess."

with a different pattern from the one in Figure 6.2. The role of the graph-pattern matching is to recognize a set of graph patterns that allow to distinguish between an entity being a class or an individual, and to select the concepts to include in its graph of types. To implement a graph-pattern matching mechanism, we have identified a set of graph patterns (GP), and defined their associated heuristics by following similar criteria as lexico-syntactic patterns [75], extended with the exploitation of RDF graph topology and OWL semantics. Currently, we use 10 GPs: 4 of them identifying class entities, and 6 for individual entities. Firstly, the GP matching step distinguishes if an entity is either an individual or a class entity: given an entity e, it is an individual if it participates in a graph pattern of type e `owl:sameAs` x, it is a class if it participates in a graph pattern of type x `rdf:type` e. As empirically observed, these two situations are mutually exclusive. After performing this distinction, the the algorithm follows a priority order for GP detection and executes the heuristics associated with the first matching GP. Tables 6.1 and 6.2 respectively report the GP sets and their associated heuristics by following the priority order used for detection, for individual entities and class entities.

The rationale behind GP priority order resides in ontology design choices as well as in the way the current implementation of Tìpalo works. Sometimes, an entity definition from Wikipedia includes typing information from a "domain-level" as well as a "meta-level" perspective. For example, from the definition[19] *"Fast chess is a type of chess game in which each side is given less time to make their moves than under the normal tournament time controls of 60 to 180 minutes per player"*. We

`istc.cnr.it/stlab-tools/fred`.

[19] `http://en.wikipedia.org/wiki/Fast_chess`

ID	graph pattern (GP)	inferred axioms
gp_1	e owl:sameAs x && x domain:aliasOf y && y owl:sameAs z && z rdf:type C	e rdf:type C
gp_2	e rdf:type x && x owl:sameAs y && y domain:aliasOf z && w owl:sameAs z && w rdf:type C	e rdf:type C
gp_3	e owl:sameAs x && x [r] y && y rdf:type C	e rdf:type C
gp_4	e owl:sameAs x && x rdf:type C	e rdf:type C
gp_5	e dul:associatedWith x && x rdf:type C	e rdf:type C
gp_6	(e owl:sameAs x && x anyP y && y rdf:type C) || (e anyP x && x rdf:type C)	e rdf:type C

Table 6.1: Graph patterns and their associated type inferred triples for individual entities. Order reflects priority of detection. [r] $\in R =$ {wt:speciesOf, wt:nameOf, wt:kindOf, wt:varietyOf, w:typeOf, wt:qtyOf, wt:genreOf, wt:seriesOf}); [anyP] $\in \{*\} - R$.

can derive that "Fast chess" is a *type* (meta-level type) as well as a *chess game* (domain-level type). This situation makes FRED output include a GP detecting "type" as a type i.e., gp_8, as well as a GP detecting "chess game" as a type i.e., gp_7, as depicted in Figure 6.7. In this use case our goal is to type DBpedia entities only from a domain-level perspective. Furthermore, this graph pattern matching for this experiment executes only one heuristics: that associated with the first GP that matches in FRED output graph. Given the above rationale, gp_7 is inspected before gp_8. The same rationale applies to GP for individual entities, illustrated in Table 6.1. For the dbp:Fast_chess[20] example, the type selector detects that the entity is

[20]dbp: http://dbpedia.org/resource/

ID	graph pattern (GP)	inferred axioms
gp_7	x `rdf:type` e && x `owl:sameAs` y && y `[r]` z && z `rdf:type` C	e `rdfs:subClassOf` C
gp_8	x `rdf:type` e && x `owl:sameAs` y && y `rdf:type` C	e `rdfs:subClassOf` C
gp_9	x `rdf:type` e && e `dul:associatedWith` y && y `rdf:type` C	e `rdfs:subClassOf` C
gp_{10}	(x `rdf:type` e && x `owl:sameAs` y && y `[anyP]` z && z `rdf:type` C) || (x `rdf:type` e && y `[anyP]` x && y `rdf:type` C)	e `rdfs:subClassOf` C

Table 6.2: Graph patterns and their associated type inferred triples for class entities. `[r]` $\in R = \{$wt:speciesOf, wt:nameOf, wt:kindOf, wt:varietyOf, w:typeOf, wt:qtyOf, wt:genreOf, wt:seriesOf$\}$); [anyP] $\in \{*\} - R$.

a class and the first GP detected is gp_7, hence it produces the additional triples:

```
dbp:Fast_chess rdfs:subClassOf wt:ChessGame

wt:ChessGame rdfs:subClassOf wt:Game
```

The execution of the algorithm on a sample set of randomly selected ∼800 Wikipedia entities has shown that the most frequent GPs are gp_4 and gp_8, which is not surprising, since they are the most common linguistic patterns for definitions. Table 6.3 reports the frequency of each GP on the sample set.

The *graph-pattern matching* algorithm for this use case implements an additional heuristics: it detects if any of the terms referring to a type in the graph can be referenceable as a DBpedia entity. For example, the term "chess" in the definition of "Fast chess" is resolvable to `dbpo:Chess`. In such case, the GP matching step of the algorithm produces the following triple:

```
dbpo:Fast_chess rdfs:subClassOf dbpo:Chess
```

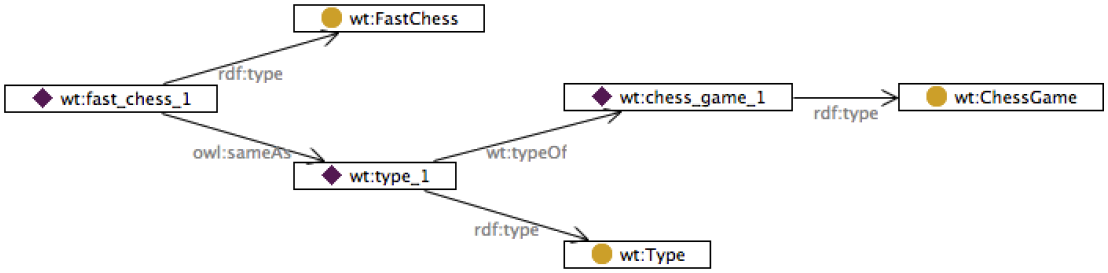


Figure 6.7: FRED result for the definition "Fast chess is a type of chess game in which each side is given less time to make their moves than under the normal tournament time controls of 60 to 180 minutes per player."

GP	**frequency** (%)
gp_1	0
gp_2	0.15
gp_3	3.98
gp_4	79.34
gp_5	0
gp_6	0.31
gp_7	1.11
gp_8	11.46
gp_9	0
gp_{10}	3.65

Table 6.3: Normalized frequency of GPs on a sample set of ∼800 randomly selected Wikipedia entities.

This additional heuristics improves the internal linking within DBpedia, resulting in higher cohesion of the resource graph.

By following the defined heuristics, we are able to select the terms that refer to the types of an entity e, and to create a namespace of Wikipedia types that captures the variety of terms used in Wikipedia definitions[21].

[21] Wikipedia class taxonomy, wt: = `http://www.ontologydesignpatterns.org/ont/wikipedia/type/`

6.2.3 Evaluation

We evaluate our work considering the accuracy of types assigned to the sample set of Wikipedia entities, and the soundness of the induced taxonomy of types for each DBpedia entity. The accuracy of types has been measured in two ways:

- in terms of precision and recall against a gold standard of 100 entities;
- by performing a user study. The soundness of the induced taxonomies has been assessed in a user study.

Building a sample set of Wikipedia pages

We have performed our experiments on a sample set of ∼800 randomly selected Wikipedia pages. From the 800 set, we have removed all pages without an abstract text, e.g. *redirect* pages, categories, and images. The resulting sample includes 627 pages with the following characteristics: (i) each page has a corresponding DBpedia entity, (ii) each DBpedia entity has either a DBpedia type, a YAGO type, or no type, (ii) 67.62% of the corresponding DBpedia entities have a YAGO type, 15.47% have a DBPO type, and 30% of them have no type.

Building a gold standard

We have built a manually annotated gold standard of Wikipedia entity types based on the sample set used for our experiments. To support this process we have developed a web-based tool named *WikipediaGold*[22] that manages argumentation among users in order to support them in discussing and reaching agreement on decisions (agreement was considered reached with at least 70% users giving the same answer). Ten users with expertise in ontology design (four senior researchers and six PhD students in the area of knowledge engineering) have participated in this task, and have reached agreement on 100 entities. We have used such 100 entities as a gold standard for evaluating and tuning our method. The gold standard can be retrieved from the

[22] Available online at `http://wit.istc.cnr.it/WikipediaGold`, demonstrating video at `http://wit.istc.cnr.it/stlab-tools/video/`

cited Wikipedia Ontology page, and it can be useful for future development and for comparing our work with possible other approaches to this same task.

WikipediaGold is based on a simple user task, repeated iteratively: given an entity e e.g., `dbp:Vladimir_Ramnik`, WikipediaGold visualizes its definition e.g., *"Vladimir Borisovich Kramnik is a Russian chess grandmaster."* and asks users to:

- indicate if e refers to a concept/type or to a specific instance. Users can select either "is a" or "is a type of" as possible answers. This value allows us to evaluate if our process is able to distinguish entities which are typical individuals, from those that are typical classes;
- copy and paste the terms in the definition that identifies the types of e, or indicate a custom one, if the definition does not contain any. In our example, a user could copy the term *"Russian chess grandmaster"*. This information is meant to allow us evaluating the performances of the *type selector*;
- select the most appropriate concepts for classifying e from two lists of terms. The first list includes 21 WordNet supersenses, and the second list includes 23 classes from DULplus and Dolce Zero. Each type is accompanied by a describing gloss and some examples to inform the user about its intended meaning. In the example, users can select the type "Person" available in both lists. The two lists of concepts are available online at the Wikipedia ontology page.

For each answer, users can optionally include a comment motivating their choice. When there is disagreement among users about an entity, WikipediaGold submits it again to users who have already analyzed it. In these cases a user can see other users' choices and comments, and decide if either to keep her decision, or to change it. In both cases, a comment motivating own decision must be entered.

Evaluation against the gold standard

Our evaluation is based on measuring precision and recall of the output of the three main steps of the process, against the gold standard: (i) graph-pattern matching

Step	precision	recall	F-measure (F1)
Graph-pattern matching	.93	.90	.92
Word-sense disambiguation (most frequent sense)	.77	.73	.75
Word-sense disambiguation (UKB)	.86	.82	.84
Ontology alignment (Supersense)	.73	.73	.73
Ontology alignment (DUL+/D0)	.80	.80	.80

Table 6.4: Performance evaluation of the individual pipeline step.

(step 3), (ii) word-sense disambiguation (WSD) (step 4), and (iii) ontology alignment (step 5). We also measure precision and recall of the overall process output.

Typing process	precision	recall	F-measure (F1)
WordNet types	.76	.74	.75
Supersenses	.62	.60	.61
Dul+/D0	.68	.66	.67

Table 6.5: Performance evaluation of the overall process.

The results shown in Table 6.4 indicate the performances of the individual components. The graph-pattern matcher stands out as the most reliable step ($F1 = .92$), which confirms our hypothesis that a rich formalization of definitions and a good design of graph patterns are a healthy approach to entity typing. The WSD task has been performed with two approaches: we analyze its performance by executing UKB as well as a most-frequent-sense-based (MFS) approach. The WSD based on UKB shows to perform better ($F1 = .84$) than the approach based on the most frequent sense in WordNet ($F1 = .75$), suggesting that Wikipedia definitions often include polysemous senses, and that the used language tends to be specialized i.e., polysemous terms are used with different senses. The ontology alignment performs better with DULplus/Dolce Zero types than with WordNet supersenses, which shows an improvement with respect to the state of the art considering that WordNet super senses are considered an established and reliable semantic resource when used as a top-level ontology for WordNet.

Table 6.5 illustrates the performance of the overall automatic typing process. As

expected, the steps that map the extracted types to WordNet types, super senses, and top-level ontologies tend to decrease the initial high precision and recall of the type selector. In fact, when put into a pipeline, errors typically reinforce previous ones, producing in this case an overall decrease of $F1$ from .92 of the type selection step to .75 of the combined type selection and WSD, to .67 with the addition of DULplus/Dolce Zero alignment (type matcher). However, the modularity of our process enables to reuse the results that are actually useful to a certain project, e.g. discarding a step that performs worse.

The good performances observed in our evaluation experiments make us claim that using our algorithm (Tìpalo [23]) brings advantages when compared to the most prominent existing approaches i.e., DBpedia project [88] and YAGO [133] to DBpedia entity typing, for the following reasons: (i) Tìpalo potentially ensures complete coverage of Wikipedia domains (intensional coverage) as it is able to capture the reachness of terminology in NL definitions and to reflect it in the resulting ontology, while DBpedia and YAGO depend both on the limited intensional completeness of infobox templates and Wikipedia categories, (ii) Tìpalo is independent from the availability of structured information such as infobox templates and Wikipedia categories, hence ensuring higher extensional completeness as most Wikipedia entities have a definition while many of them lack infoboxes.

A direct comparison of our results with DBpedia and YAGO approaches occurred to be unfeasible in the scope of this paper because the two approaches differ from ours on important aspects: they use different reference type systems; they rely on Wikipedia categories or infobox templates while we rely on the NL descriptions used for defining Wikipedia entities by the crowds, hence it is difficult (if not impossible) to compare the derived vocabularies; finally, the granularity of their type assignments is heterogeneous. These cases make it hard to define criteria for performing a comparison between the accuracy of the automatically assigned types. Hence, we could not consider either DBpedia or YAGO suitable gold standards for this specific task, which motivates the construction of a specific gold standard.

[23] Tìpalo is also the name of the tool that implements such algorithm.

Evaluation by user study

In order to further verify our results, we have conducted a user study. We have implemented a second Web-based tool, named *WikipediaTypeChecker*[24], for supporting users in expressing their judgement on the accuracy of Tìpalo types assigned to the sample set of Wikipedia entities.

WikipediaTypeChecker asks users to evaluate the accuracy of Tìpalo types, the soundness of the induced taxonomies, and the correctness of the selected meaning of types, by expressing a judgement on a three-valued scale: *yes, maybe, no*. Users' task, given an entity with its definition, consists of three evaluation steps. Consider for example the entity `dbp:Fast_chess`: in the first step, users evaluate the accuracy of the assigned types by indicating the level of correctness of proposed types. In this example, for the entity "Fast chess" three types are proposed: "Chess game", "Game", and "Activity"; in the second step users validate the soundness of the induced taxonomy of types for an entity. In this example, the proposed taxonomy is `wt:ChessGame rdfs:subClassOf wt:Game`; in the third step users evaluate the correctness of the meaning of individual types (i.e. WSD). For example, the proposed meaning for "Chess game" is "a board game for two players who move their 16 pieces according to specific rules; the object is to checkmate the opponent's king". Five users with expertise in knowledge engineering have participated in the user study (three PhD students and two senior researchers). For each entity and for each evaluation step, we have computed the average value of judgements normalized to an interval [0,1], which gives us a value for the precision of results. The results are shown in Table 6.6, with a (high) inter-rater agreement (Kendall's W) of .79[25]. These results confirm those observed in the evaluation against a gold standard (cf. Tables 6.4 and 6.5). In this case, we have split the evaluation of the correctness of extracted types between *assigned types* (.84), and *induced taxonomy* (.96): their combination

[24]Available online at `http://wit.istc.cnr.it/WikipediaTypeChecker`, demonstrating video at `http://wit.istc.cnr.it/stlab-tools/video`

[25]Kendall's W is a coefficient of concordance used for assessing agreement among raters. It ranges from 0 (no agreement) to 1 (complete agreement), and is particularly suited in this case as it makes no assumptions regarding the nature of the probability distribution and handles any number of distinct outcomes.

Task	Type extraction	Taxonomy induction	WSD
Correctness	.84	.96	.81

Table 6.6: Results of the user-based evaluation, values are expressed in percentage and indicate precision of results. Inter-rater agreement (Kendall's W) is .79, Kendall's W ranges from 0 (no agreement) to 1 (complete agreement).

is comparable to the precision value observed for the type selector against the gold standard (.93). The performance of the WSD task is a bit lower (.81 against .86 precision).

6.2.4 ORA: towards the Natural Ontology of Wikipedia

Tìpalo has been used for automatically generating a natural ontology of Wikipedia, i.e., an ontology that is extracted by exploiting the natural language definitions of Wikipedia entities. Hence, it reflects the richness of terms used and agreed by the crowds. The extraction of the ontology has been run on a Mac Pro Quad Core Intel Xeon 2.8Ghz with 10Gb RAM and took 15 days (which can be easily reduced by parallelizing the activity by means of grid computing composed by machines with similar or more powerful characteristics). The process resulted in 3,023,890 typed entities and associated taxonomies of types. Most of the missing results are due to the lack of matching Tìpalo heuristics, which means that by improving Tìpalo we will improve coverage (this is part of our current work). The resulting ontology includes 585,474 distinct classes organized in a taxonomy with 396,375 `rdfs:subClassOf` axioms; 25,480 if these classes are aligned through `owl:equivalentClass` axioms to 20,662 OntoWordNet synsets by means of a word-sense disambiguation process. The difference between the number of disambiguated classes (25,480) and the number of identified synsets (20,662) means that there are at least 4,818 synonym classes in the ontology. We expect the number of actual synonyms to be greater. Hence, we are planning to investigate some sense-similarity-based metric in order to reduce the number of distinct classes in the ontology by merging synonyms or at least providing explicit similarity relations with confidence scores between classes.

In order to prevent polysemy deriving from merging classes with same names

but aligned to different synsets, it has been adopted a criterion of uniqueness for the generation of the URIs of these classes. For example, let us consider the entity `dbpedia:The_Marriage_of_Heaven_and_Hell`[26]. For this entity Tìpalo generates the following RDF:

```
dbpedia:The_Marriage_of_Heaven_and_Hell
    a  wt:Book .
wt:Book
    owl:equivalentClass  wn30-instance:synset-book-noun-2 .
```

Similarly, for the entity `dbpedia:Book_of_Revelation`[27] Tìpalo generates the following RDF:

```
dbpedia:Book_of_Revelation
    a  wt:CanonicalBook .
wt:CanonicalBook
    rdfs:subClassOf  wt:Book .
wt:Book
    owl:equivalentClass  wn30-instance:synset-book-noun-10 .
```

The two `wt:Book` classes refers to two distinct concepts. Hence, they cannot be merged during the generation of the ontology. We solve this by appending the ID of the closest synset in the taxonomy to the URI of a new generated class: this approach guarantees to prevent polysemy and to identify synonymity at the same time. Finally, all the classes aligned to OntoWordNet have been also aligned to WordNet supersenses and a subset of DOLCE+DnS Ultra Lite classes by means of `rdfs:subClassOf` axioms. The following example shows a sample of the ontology which has been derived by typing the two entities used as examples previously:

[26]The definition of `dbpedia:The_Marriage_of_Heaven_and_Hell` is: *"The Marriage of Heaven and Hell is one of William Blake's books."*

[27]The definition of `dbpedia:Book_of_Revelation` is: textit"The Book of Revelation is the last canonical book of the New Testament in the Christian Bible."

```
dbpedia:The_Marriage_of_Heaven_and_Hell
    a  wt:Book_102870092 .
dbpedia:Book_of_Revelation
    a  wt:CanonicalBook_106394865 .
wt:CanonicalBook_106394865
    rdfs:subClassOf  wt:Book_106394865 ;
    rdfs:label  "Canonical Book"@en-US .
wt:Book_102870092
    owl:equivalentClass  wn30-instance:synset-book-noun-2 ;
    rdfs:label  "Book"@en-US .
wt:Book_106394865
    owl:equivalentClass  wn30-instance:synset-book-noun-10 ;
    rdfs:subClassOf  wn30-instance:supersense-noun_communication ,
                     d0:InformationEntity ;
    rdfs:label  "Book"@en-US .
```

ORA is available for download [28]. We claim that this ontology provides an important resource that can be used as alternative or complement for YAGO and DBPO, and that it can enable more accurate usage of DBpedia in Semantic Web based applications such as: mash-up tools, recommendation systems, and exploratory search tools (see for example Aemoo [108]), etc. Currently, we are working at refining ORA and to align it to DBPO and YAGO.

6.3 Identifying functions of citations

References are *tools* for *linking* research. Whenever a researcher writes a paper she uses bibliographic references as pointers to related works, to sources of experimental data, to background information, to standards and methods linked to the solution being discussed, and so on. Similarly, citations are tools for disseminating research. Not only on academic conferences and journals. Dissemination channels also include

[28] http://stlab.istc.cnr.it/stlab/ORA

publishing platforms on the Web like blogs, wikis, social networks. More recently, semantic publishing platforms are also gaining relevance [129]: they support users in expressing semantic and machine-readable information. From a different perspective, citations are tools for exploring research. The network of citations is a source of rich information for scholars and can be used to create new and interesting ways of browsing data. A great amount of research is also being carried on sophisticated visualisers of networks of citations and powerful interfaces allowing users to filter, search and aggregate data. Finally, citations are tools for evaluating research. Quantitative metrics on bibliographic references, for instance, are commonly used for measuring the importance of a journal (e.g. the *impact factor*) or the scientific productivity of an author (e.g. the *h-index*). Furthermore, being links citations can be exploited for investigating Citational Knowledge Patterns. This work begins with the basic assumption that all these activities can be radically improved by exploiting the actual nature of citations. Let us consider citations as means for evaluating research. Could a paper that is cited many times with negative reviews be given a high score? Could a paper containing several citations of the same research group be given the same score of a paper with heterogeneous citations? How can a paper cited as plagiarism be ranked? These questions can be answered by looking at the nature of the citations, not only their existence. On top of such characterisation, it will also be possible to automatically analyse the pertinence of documents to some research areas, to discover research trends and the structure of communities, to build sophisticated recommenders and qualitative research indicators, and so on. There are in fact ontologies for describing the nature of citations in scientific research articles and other scholarly works. In the Semantic Web community, the most prominent one is CiTO (Citation Typing Ontology) [29] [112]. CiTO is written in OWL and is connected to other works in the area of semantic publishing. It is then a very good basis for implementing sophisticated services and for integrating citation data with linked data silos. In this section we present CiTalO a tool that implements the method presented in previous section. The contribution about CiTalO is the result

[29]CiTO: http://purl.org/spar/cito.

of works presented in [83, 44]. We refer to chapter 7 for the details about the design and the implementation of CiTalO .

6.3.1 The CiTalO algorithm

In this section, we introduce CiTalO, an algorithm [30] that infers the function of citations by using the method described in Section 6.1.3 plus sentiment-analysis. This method is applied in a pipeline whose input is the textual context containing the citation and the output is a one or more properties of CiTO [112]. Figure 6.8 shows the Citalo algorithm.

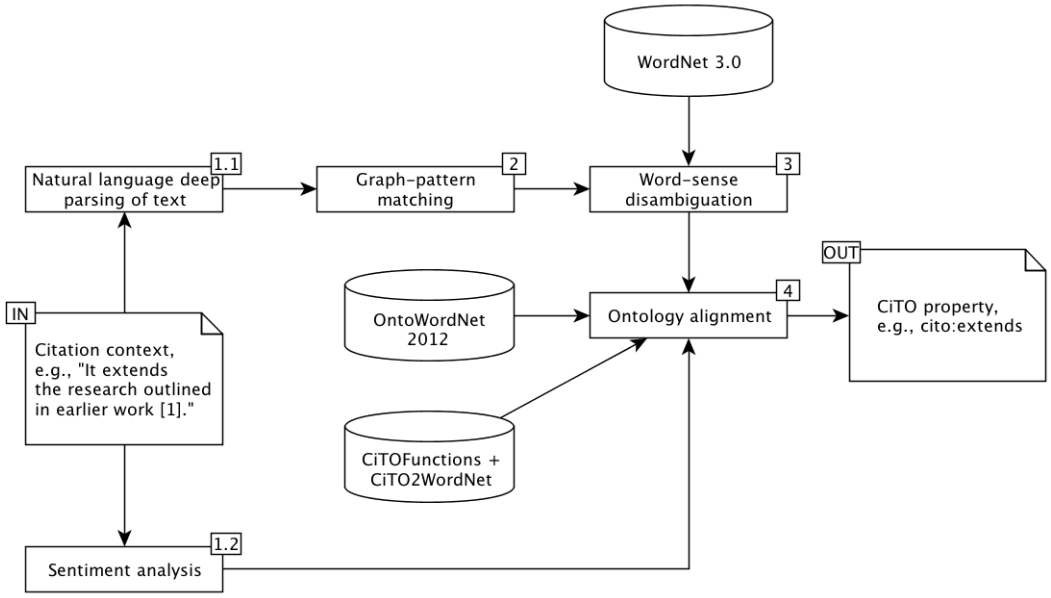


Figure 6.8: Pipeline implemented by CiTalO. The input is the textual context in which the citation appears and the output is a set of properties of the CiTO ontology.

Inasmuch as in previous section we have already explained how steps 1.1 (i.e., natural language deep parsing of text) and 3 (i.e., word-sense disambiguation) are performed, we will give only details about the steps that differ or introduce novelty with respect to the previous approach. Namely these steps are those numbered

[30] Details about the CiTalO tool can be found in Section 7.4.2

1.2 (i.e., sentiment analysis), 2 (i.e., graph-pattern matching) and 4 (i.e., ontology alignment).

Sentiment-analysis to gather the polarity of the citational function

The aim of the sentiment-analysis in our context is to capture the sentiment polarity emerging from the text in which the citation is included. The importance of this step derives from the classification of CiTO properties according to three different polarities, i.e., positive, neuter and negative. This means that being able to recognise the polarity behind the citation would restrict the set of possible target properties of CiTO to match.

Citation type extraction through pattern matching

The second step consists of extracting candidate types for the citation, by looking for patterns in the FRED result. In order to collect these types we have designed 6 graph-based heuristics and we have implemented them as SPARQL queries. These patterns are described in Table 6.7 where the rationale differs from that used in Tìpalo because the order of patterns is irrelevant as they are all evaluated allowing to collect multiple typing inferences.

For example, given the following sentence which contains a citation:

It extends the research outlined in earlier work X.

where X is the cited work, FRED returns the graph shown in Figure 6.9. Based on the graph-patterns in Table 6.7 gp_1, gp_2 and gp_5 match and allow to identify as candidate types the terms *Outline* (gp_1), *Extend* (gp_5), *EarlierWork* (gp_1), *Work* (gp_2), and *Research* (gp_5).

We still have to extract statistics on graph-patterns for citations that allow to identify what are the most frequently matching patterns. Anyway, we are investigating new patterns and we are continuously updating the catalogue.

ID	graph pattern (GP)	inferred axioms
gp_1	e anyP wt:X && e rdf:type t	wt:X rdf:type t
gp_2	e anyP wt:X && e rdf:type x && x rdfs:subClassOf+ t	wt:X rdf:type t
gp_3	(e anyP+ wt:X || wt:X anyP+ e) && e rdf:type dul:Event && e rdf:type t && t != dul:Event	wt:X rdf:type t
gp_4	(e anyP+ wt:X || wt:X anyP+ e)+ && e rdf:type dul:Event && e rdf:type x && x rdfs:subClassOf+ t && t != dul:Event	wt:X rdf:type t
gp_5	(e anyP+ wt:X || wt:X anyP+ e)+ && e rdf:type dul:Event && (e boxer:theme x || e boxer:patient x) && x rdf:type t	wt:X rdf:type t
gp_6	(e anyP+ wt:X || wt:X anyP+ e)+ && e rdf:type dul:Event && (e boxer:theme x || e boxer:patient x) && x rdf:type y && y rdfs:subClassOf+ t	wt:X rdf:type t

Table 6.7: Graph patterns and their associated type inferred triples. Order reflects priority of detection. [anyP] $\in \{*\}$.

Alignment to CiTO

The final step consists of assigning CiTO types to ciations. We use two ontologies for this purpose: *CiTOFunctions* and *CiTO2Wordnet*. The CiTOFunctions ontology [31] classifies each CiTO property according to its factual and positive/neutral/negative rhetorical functions, using the classification proposed by Peroni *et al.* [112]. CiTO2Wordnet [32] maps all the CiTO properties defining citations with the appropriate Wordnet synsets (as expressed in OntoWordNet). This ontology is part of the contribution of this case study ans it was built in three steps:

- *identification step.* We have identified all the Wordnet synsets related to each of the thirty-eight sub-properties of `cites` according to the verbs and nouns used in property labels (i.e. `rdfs:label`) and comments (i.e. `rdfs:comment`)

[31] CiTOFunctions: `http://www.essepuntato.it/2013/03/cito-functions`.
[32] CiTO2Wordnet ontology: `http://www.essepuntato.it/2013/03/cito2wordnet`.

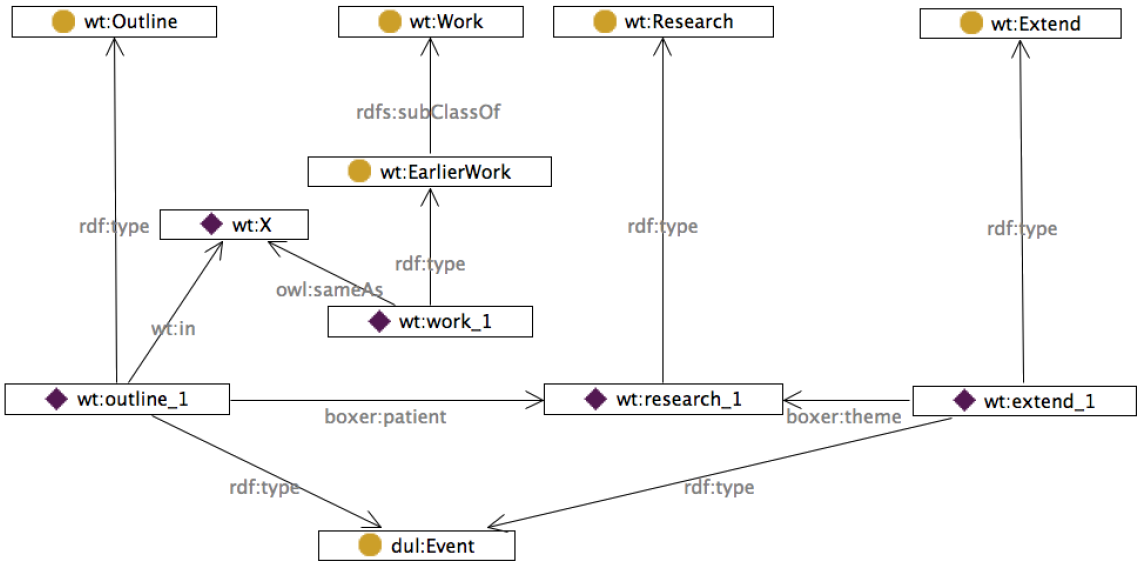


Figure 6.9: RDF graph resulting from FRED for input "It extends the research outlined in earlier work X"

for instance, the synsets `credit#1`, `accredit#3`, `credit#3`, `credit#4` refers to the property `cito:credits`;

- *filtering step.* For each CiTO property, we filtered out those synsets of which the gloss [33] is not aligned with the natural language description of the property in consideration. For instance, the synset `credit#3` was filtered out since the gloss "accounting: enter as credit" means something radically different to the CiTO property description "the citing entity acknowledges contributions made by the cited entity";

- *formalisation step.* Finally, we linked each CiTO property to the related synsets through the property `skos:closeMatch`. An example in Turtle is:

```
cito:credits skos:closeMatch synset:credit-verb-1 .
```

The final alignment to CiTO is performed through a SPARQL CONSTRUCT query that uses the output of the previous steps, the polarity gathered from the sentiment-

[33] In Wordnet, the *gloss* of a synset is its natural language description.

analysis phase, OntoWordNet and the two ontologies just described. In the case of empty alignments, the CiTO property *citesForInformation* is returned as base case. In the example, the property *extends* is assigned to the citation.

6.3.2 Evaluation

The evaluation consisted of comparing the results of CiTalO with a human classification of the citations. The test bed we used for our experiments includes some scientific papers (written in English) encoded in XML DocBook, containing citations of different types. The papers were chosen among those published in the proceedings of the Balisage Conference Series. In particular, we automatically extracted citation sentences, through an XSLT document [34], from all the papers published in the seventh volume of Balisage Proceedings, which are freely available online [35]. For our test, we took into account only those papers for which the XSLT transform retrieved at least one citation, i.e., 18 papers written by different authors. The total number of citations retrieved was 377, for a mean of 20.94 citations per paper. Notice that the XSLT transform was quite simple at that stage. It basically extracted the *citation sentence* around a citation, i.e., the sentence in which that citation is explicitly used, preparing data for the actual CiTalO pipeline. We first filtered all the citation sentences from the selected articles, and then we annotated them manually using the CiTO properties. Since the annotation of citation functions is actually an hard problem to address (it requires an interpretation of author intentions) we mark only the citations that are accompanied by verbs (extends, discusses, etc.) and/or other grammatical structures (uses method in, uses data from, etc.) carrying explicitly a particular citation function. We considered that rule as a strict guideline as also suggested by Teufel *et al.* [135]. We marked 106 citations of out the 377 originally retrieved, obtaining at least one representative citation for each of the 18 paper used (with a mean of 5.89 citations per paper). We used 21 CiTO properties out of 38 to annotate all these citations, as shown in table 6.8. Interesting similarities can be

[34] Available at `http://www.essepuntato.it/2013/sepublica/xslt`.
[35] Proceedings of Balisage 2011: `http://balisage.net/Proceedings/vol7/cover.html`.

found between such a classification and the results of [135]. In this paper, the neutral category *Neut* was used for the majority of annotations by humans; similarly the most neutral CiTO property, `cito:citesForInformation`, was the most prevalent function in our dataset too. The second most used property was *usedMethodIn* in both analyses.

# of Citations	CiTO property
53	citesForInformation
15	usesMethodIn
12	usesConclusionsFrom
11	obtainsBackgroundFrom
8	discusses
4	citesAsRelated, extends, includesQuotationFrom, citesAsDataSource, obtainsSupportFrom
< 4	credits, critiques, useConclusionsFrom, citesAsAuthority, usesDataFrom, supports, updates, includesExcerptFrom, includeQuotationForm, citesAsRecommendedReading, corrects

Table 6.8: The way we marked the citations within the 18 Balisage papers.

We run CiTalO on these data (i.e. 106 citations in total) and compared results with our previous analysis [36] We also tested eight different configuration of CiTalO, corresponding to all possible combinations of three options:

- activating or deactivating the sentiment-analysis module;
- applying or not the proximal synsets [37] to the word-disambiguation output;

The number of *true positives* (TP), *false positives* (FP) and *false negatives* (FN) obtained comparing CiTalO outcomes with our annotations are shown in table 6.9.

We calculated the precision and the recall obtained by using each configuration. As shown in figure 6.10, *Filtered* and *Filtered+Sentiment* have the best precision

[36] All the source materials we used for the test is available online at `http://www.essepuntato.it/2013/sepublica/test`. Note that a comparative evaluation with other approaches, such as Teufel's, was not feasible at this stage since input data and output categories were heterogeneous and were not directly comparable.

[37] We used the same the RDF graph of proximal synsets introduced in [64]

(.348) and the second recall (.443). Instead *All* and *All+Sentiment* have the second precision (.313) and the best recall (.491). There is no configuration that emerges as the absolutely best one from these data. They rather suggest an hybrid approach that also takes into account some of the discarded synsets. It is evident that the worst configuration were those that took into account all the proximal synsets. It looks that the more synsets CiTalO uses, the less the citation functions retrieved conform to humans' annotations.

Configuration	**TP**	**FP**	**FN**
Filtered (with or without Sentiment)	47	88	59
Filtered + Proximity	40	137	66
Filtered + Proximity + Sentiment	41	136	65
All (with or without Sentiment)	52	114	54
All + Proximity (with or without Sentiment)	45	174	64

Table 6.9: The number of true positives, false positives and false negatives returned by running CiTalO with the eight different configurations.

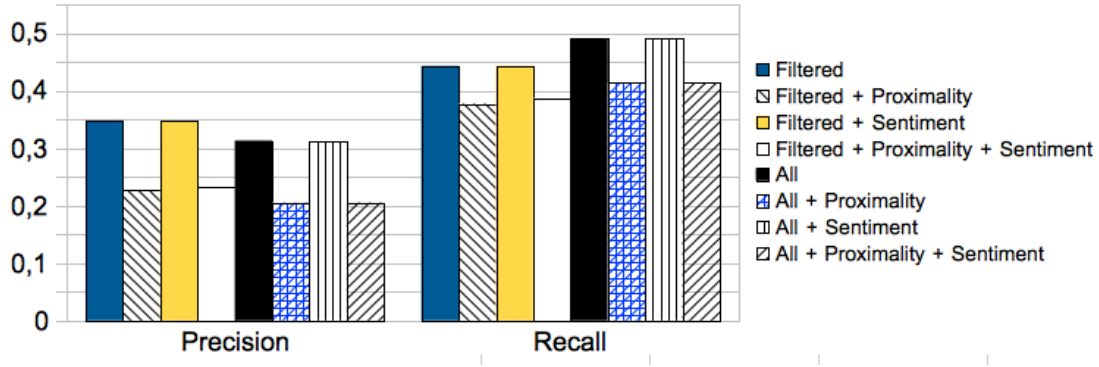


Figure 6.10: Precision and recall according to the different configuration used.

Chapter 7

A software architecture for KP discovery and reuse

In this Chapter we present:

- K∼ore, a software architecture to serve as the basis for the implementation of systems aimed at KP extraction, transformation and reuse. K∼ore design benefits of the combination of the Component-based and REST architectural styles that enable a Service-oriented architecture with high modularity, extensibility and customization (cf. Section 7.2);

- K∼tools, a set of tools implementing K∼ore. K∼tools were used in all the use cases presented in this work (cf. Section 7.4.2).

7.1 Requirements

The methods we have described in previous chapters are the basis for the design and the implementation of a software system aimed at providing tools for KP transformation, KP extraction and source enrichment. The system requirements have been collected from two main perspectives: (i) the perspective of the methods used, i.e., any requirements explicitly emerging from the definition of the methods, (ii) the case study perspective, i.e., any requirements explicitly emerging by analyzing

how to address the concrete case studies presented in previous chapters. These two perspectives led to the requirements elicitation process [130].

The high-level requirements that so far have driven the design and the implementation of the system are divided into the classical dichotomy between functional and non-functional requirements: The functional requirements are the following:

- *HLR-01: Format reengineering* - the system has to provide mechanisms that enable the conversion to RDF of non-RDF sources that provides KP-like artifacts. An example of non-RDF source of KP-like artifacts is FrameNet which is an XML lexical knowledge base, consisting of a set of *frames*;

- *HLR-02: Refactoring* - the system has to implement functions for RDF refactoring. This requirements is highly related to HLR-01. This is because typically the refactoring is applied to data coming from the reengineering of KP-like artifacts. In fact, HLR-01 and HLR-02 identify the Semion methodology [105] that has been explained in chapter 4 and successfully used for gathering KPs from FrameNet;

- *HLR-03: Detecting invariances over data* - given a certain RDF dataset in the Web the system has to provide functions which allow to to empirically identify invariances in the organization of data in the dataset. The method used for the identification of the invariances has to be transparent to the system. This allows, depending on the scenario, to switch from different techniques, e.g., machine learning, statistics, natural language processing, etc., by choosing from time to time the most suitable one;

- *HLR-04: Drawing boundaries around data* - once invariances over data have been gathered the system has to give interpretation to these invariances. In fact, the invariances are expressed by symbols, but according to [66], KPs are not only symbolic patterns: they also have an interpretation, be it formal, or cognitive. Hence, giving an interpretation to emerging invariances means to provide de facto the meaning of a KP. This is compliant to what has been called by Gangemi and Presutti in [66] *the knowledge boundary problem*;

- *HLR-05: KP storage and querying* - the system has to be able to collect KPs in a dedicated storage. The storage has to support query mechanisms in order to facilitate KP reuse;

- *HLR-06: source enrichment*: The system has to provide functionalities in order to overcome problems related to Linked Data that are not necessarily clean, optimized, or extensively structured;

- *HLR-07: Rest services* - the system has to expose its core functions as HTTP REST [50] services. This allows to made available all the core services of the system as CRUD [1] [95] operations over the Web. In fact, it is a generally accepted that REST Web Services should implement CRUD operations in the HTTP protocol for all of their resources, modulo specific constraints for granting access to themIt 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.

We will refer to the functional requirements as to core requirements (CRs) as they basically identify core functions in the system.

The non-functional requirements are the following:

- *HLR-08: adaptivity* - the system should be adaptive to any possible source and format available in the Web both for KP transformation/extraction and source enrichment. This means that it could be easily extended to accept new formats in which external data for KP gathering are expressed.

- *HLR-09: components* - KP transformation, KP extraction, source enrichment and the storage should be implemented as independent components within the system. These components should be compliant with the loose coupling principle, i.e., each component has, or makes use of, little or no knowledge of the definitions of other separate components. On the contrary, all the elements

[1] The acronym CRUD stands for create, read, update and delete.

belonging to a single component should result highly cohesive. High cohesion is typically associated with several desirable traits of software including robustness, reliability, reusability, and understandability.

- *HLR-10: customization of components* - each component of the system should provides interfaces for customization. This enables users to configure the components as better as possible according to a specific use case. For example, a user may want to configure the system in order to extract KPs from natural language. This would require to properly configure the component for the source enrichment in order to select functions for that specific enrichment, e.g., language recognition, named-entity recognition and resolution, relation extraction, event and situation recognition, etc.;

- *HLR-11: scalability* the system should be able both to handle a growing amount of work for KP gathering and to accept and to keep alive different customizations of its components without losing performance.

7.2 The architectural binding

A software architectural style is a specific method of construction, characterized by the features that make it notable. It defines: (i) a family of systems in terms of a pattern of structural organization; (ii) a vocabulary of components and connectors, with constraints on how they can be combined [128]. With respect to architectural patterns, architectural styles are more general. In fact, they do not provide a general schema to be used in order to address a problem, rather they provide the basis for configuring software architectures. Hence, an architectural style, defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined. These can include topological constraints on architectural descriptions. Other constraints-say, having to do with execution semantics-might

also be part of the style definition [67]. the former is more general and does not require a problem to solve for its appearance. The basic styles can be classified in four main categories as illustrated in table 7.1 [111]:

Category	**Style**
Communication	Service-Oriented Architecture (SOA), Message Bus
Deployment	Client/Server, N-Tier, 3-Tier
Domain	Domain Driven Design
Structure	Pipe&Filer, Component-Based, Object-Oriented, Layered Architecture

Table 7.1: Classification of the basic architectural styles.

The architectural styles we have used for modelling our software are the Component-based and REST, as they better meets the requirements described in the previous section.

7.2.1 Background on the Component-based architectural style

The component-based architectural style describes a software engineering approach to system design and development. It focuses on the decomposition of the design into individual functional or logical components that expose well-defined communication interfaces containing methods, events, and properties. This provides high level of abstraction, even higher than object-oriented design principles, and does not focus on issues such as communication protocols and shared states. The key principle of the component-based style is the use of components that are software packages, web services, web resources, or modules that encapsulates a set of related functions and data. In the component-based style, each component addresses the following properties [111]:

- **Reusability**. Components are usually designed to be reused in different scenarios in different applications. However, some components may be designed for a specific task.

- **Replaceability**. Components may be readily substituted with other similar components.
- **Non context specificity**. Components are designed to operate in different environments and contexts. Specific information, such as state data, should be passed to the component instead of being included in or accessed by the component.
- **Extensibility**. A component can be extended from existing components to provide new behavior.
- **Encapsulability**. Components expose interfaces that allow the caller to use its functionality, and do not reveal details of the internal processes or any internal variables or state.
- **Independece**. Components are designed to have minimal dependencies on other components. Therefore components can be deployed into any appropriate environment without affecting other components or systems.

The main benefit of designing software by adopting the Component-based architectural style derives from the principle of *encapsulation*. In fact, each component exposes its functionalities to the rest of the system by providing an interface, which specifies the services available and hides implementation details to other components. Hence, with regard to system-wide co-ordination, components communicate with each other via interfaces. This makes it easy to add new components, to substitute them and to modify the configuration of the communication among components, which means to make customizable the system behaviour.

7.3 K∼ore: design

K∼ore [2], the architecture we have designed, is targeted at KP discovery and reuse, e.g., KP can be used by KP-aware applications. A KP-aware application is a system

[2]The name is the composition of the words Knowledge and cORE, as the system provides core functionalities for experimenting with KPs.

or an agent on the Web which provides knowledge interaction services to humans or to other systems by exploiting KPs for organizing, exchanging knowledge, and reasoning over knowledge. Examples of such KP-aware application are intelligent recommendation systems, more sophisticated question answering systems, exploratory search tools with cognitive grounding, agents based systems, cognitive architectures, etc. We provide an example of such a KP-aware application in Chapter 8, in which we describe Aemoo, a tool for entity summarization and exploratory search that applies KPs as lenses over data.

K∼ore is designed by addressing the requirements (see Section 7.1) and it consists of a set of integrated standalone components according to the architectural style adopted, i.e., the component-based style. The core functionalities that K∼ore addresses are:

- KP transformation from KP-like repositories;
- KP extraction from the Web;
- enrichment of sources of KPs in the Web that do not provide rich data for KP investigation but that for some reason can be elected as potential sources of KPs;
- storage and querying of KPs for realizing a knowledge base of patterns for their reuse in the Web.

In Figure 3.3 (cf. Section 3.3) we anticipated the general schema of the methodology for extracting and tranforming KPs. Following that intuition, Figure 7.1 shows the UML component diagram representing the architecture of the system with its core components. We propose this architecture as a reference solution to be adapted to specific tasks which require to deal with KPs. Our architecture benefits of all the properties which come from the Component-based and REST architectural styles used for its design. Hence, it is particularly suited to be customized depending on a specific scenario by providing implementations to the high-level interfaces exposed by the components or configuring the communication interfaces among components.

We remark how each component in Figure 7.1 is a single and high-level view of a more detailed reality, consisting in sub-components. Furthermore, the Component-based architectural style is combined with the Representational State Transfer (REST) architectural style [50] applied to Web services in order to enable access to the functionalities that are exposed by the system through the HTTP protocol.

7.3.1 Source Enricher

Our hypothesis for KP extraction and transformation mainly relies on the fact that we deal with rich data expressed as RDF. According to our assumptions and observations, even RDF is not rich enough if the subjects and the objects of available triples miss type axioms (extensional incompleteness). Unfortunately many data sets in Linked Data (e.g. DBpedia) are extensionally incomplete. The situation is even more complex is we take into account that most of the Web knowledge is contained either in a variety of structured repositories of different formats and compliant to different schemas or in HTML pages which require to handle with natural language.

The Source Enricher is the component responsible for the enhancement of these kind of sources. As we distinguish between structured data and natural language the design of the Source Enricher follows this distinction. In fact, it is specialized by a two sub-components (cf. Figure 7.1)

- the *Reengineer*;
- the *Natural Language Enhancer*.

The Reengineer

The reengineer allows to transform non-RDF structured data to RDF. For example, if we want to extract KPs from XML we are firstly supposed to ask the reengineer to express XML data as RDF. The aim of the reengineer is to prepare non-RDF data so that they can be used for KP investigation. The transformation is performed without fixing any assumption on the domain semantics of original data, but only

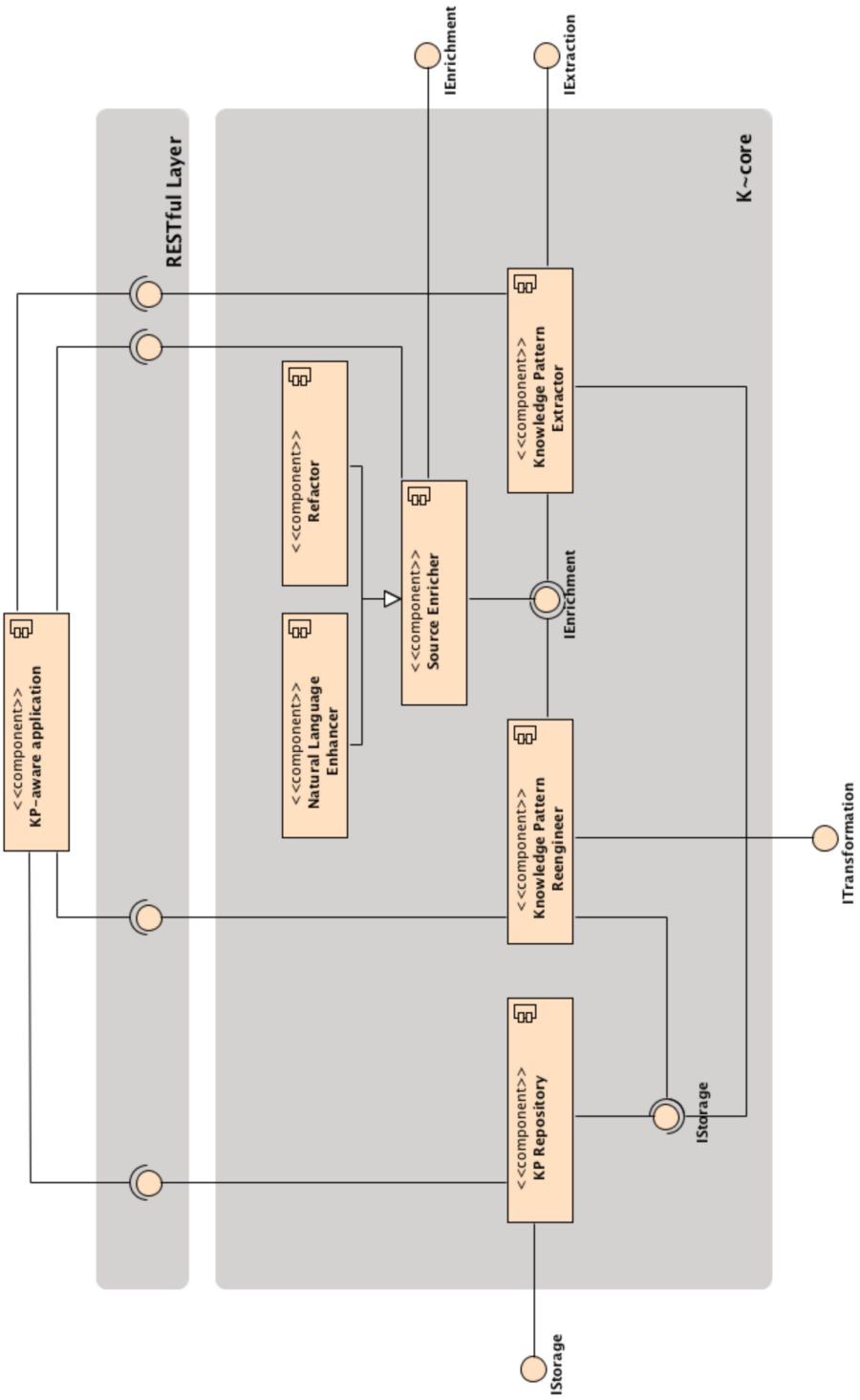


Figure 7.1: UML component diagram of K∼core.

applying a conversion driven by the meta-model of the original source type provided as an OWL ontology. Hence, the reengineer is provided of a list of meta-model expressed as OWL for each source type, e.g., RDBMS, XML, XSLT, etc. This list can be extended and customized according to specific sources that are eventually not originally provided. Each source meta-model is used by the reengineer for tuning the transformation of the elements of the original source to RDF. For example, the OWL meta-model for a relational database would include the classes "table", "column", and "row";

The Natural Language Enhancer

The Natural Language Enhancer is a peculiar specialization of the Source Enricher. It consists of four components aimed at extracting RDF annotations from natural language. These components and the way they communicate through their interfaces are the result of the studies conducted for automatically typing Wikipedia entities [64] and for identifying the nature of citations in scholarly publishing [44] (cf. Chapter 6). Accordingly, each sub-component of the Natural Language Enhancer and their relations derive from the main steps identified in the method for source enrichment described in Section 6.1.3. Namely:

- **the Ontology learner**. It is the architectural counterpart of the step of *deep parsing of text* in our method (see in Section 6.1.3). It basically provides functionalities to obtain a logical form, e.g., an OWL ontology, from a text in natural language;

- the **Graph-pattern Matcher**. It allows to recognize known patterns from a graph being the logical representation of the original text in natural language. The list of graph-patterns is part of the configurable data that are passed to the component. This is completely within the scope of the non-context specificity of the Component-based architectural style and allows to adapt this component basically by properly configuring the graph-patterns. For example, the list of patterns can be configured for recognizing specific lexico-syntactic

patterns (e.g., for identifying definitions, facts, descriptions, etc.) from the logical representation of the natural language. This component corresponds to the step of *graph-pattern matching* in the method for source enrichment;

- the **Word-sense Disambiguator**. This component provides interfaces to the system for addressing word-sense disambiguation (WSD) tasks. The WSD is performed for discriminating about the meaning associated to the labels of the classes and the properties generated by the Ontology Learner. For example, this is useful in order to provide alignments to other ontologies by means of a mediator lexical knowledge base such as WordNet [48]. The Word-sense Disambiguator is aimed at providing support to the step of *word-sense disambiguation* in our method for source enrichment;

- the **Ontology Aligner**. It supports methods for ontology alignments in order to provide mappings between terms of the logical representation of a text and terms into existing ontologies in the Web. This component provides interfaces for building new ontology alignment systems or to encapsulate existing ones, e.g. the Alignment API [46]. The Ontology Aligner corresponds to the step identified by the *ontology alignment*.

Figure 7.2 shows the UML component diagram of the Natural Language Enhancer. The components are organized in a pipe&filter fashion, but the architecture benefits of the flexibility and adaptivity of the component model. The Natural Language Enhancer is externally seen as a single component, namely as an instantiation of the Source Enrichment of the K∼ore architecture. This allows to maximize the freedom dor internally configuring the components, both in terms of what they compute and how they interact among them. Hence, the configuration of single components (e.g., the configuration of the graph-patterns or the configuration of the alignments) combined with the configuration of communication ports enable several implementations for handling with natural language. Additionally, in our opinion the architecture realized by the Natural Language Enhancer could a reference architecture for designing components for natural language interpretation in

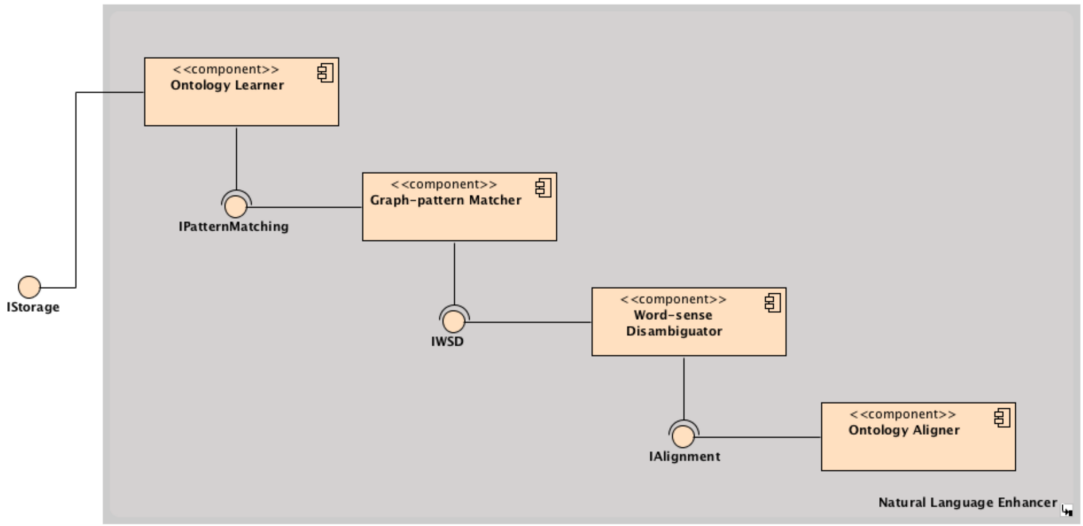


Figure 7.2: UML component diagram of the Natural Language Enhancer.

cognitive architectures [87]. A cognitive architecture proposes artificial computational processes that act like certain cognitive systems, most often, like a person, or acts intelligent under some definition. Cognitive architectures form a subset of general agent architectures, that attempt to model not only behavior, but also structural properties of the modelled system [3].

7.3.2 Knowledge Pattern Extractor

The Knowledge Pattern Extractor is the component aimed at extracting KPs by providing a software architecture based on the method described in Chapter 5 and used in the corresponding case study (cf. Section 5.2), which explains how to extract KPs from Wikipedia links. That method is based on the identification of type paths [4] plus their statistical analysis by means of the measure defined as *pathPopularity* [5]. The notion of path used for the sake of design of this component is wider

[3] cf. `http://en.wikipedia.org/wiki/Cognitive_architecture`

[4] We remark that type paths are paths whose occurrences have (i) the same rdf:type for their subject nodes, and (ii) the same rdf:type for their object nodes.

[5] The ratio of how many distinct resources of a certain type participate as subject in a path to the total number of resources of that type. Intuitively, it indicates the popularity of a path for a certain subject type.

that that of type path. In fact, in this context we adopt property paths instead of type paths for enabling the system to be reusable and customizable according to different methods and statistical measures or algorithms. Similarly, the Knowledge Pattern Extractor defines a top-level interface for the measure to use for the statistical analysis of property paths. This allows to take advantage of the properties of extensibility and encapsulability, typical of Component-base architectural styles. In fact, it is possible to extend the component by defining additional measures or by implementing common interfaces that are in turn the only known signature of the component with respect to other the rest of the system. Internally the Knowledge Pattern Extractor is composed by four sub-components, namely (cf. figure 7.4):

- the *Property path identifier*, which identifies the property paths of a certain length from a given data set;
- the *Property path storage*, which stores the property paths identified by the Property path identifier;
- the *Property path analyzer*, which applies a specific statistical measure defined by a user, e.g., the path popularity, on the property paths that can be retrieved from the store;
- the *KP drawer*, which formalizes KPs by drawing boundaries around sets of property paths. This is performed by configuring the threshold criterion to apply to property paths after they have been analyzed by the Property path analyzer.

7.3.3 Knowledge Pattern Refactor

The transformation of KPs available into KP-like repositories is performed by the component of the system called Knowledge Pattern Refactor. Because most of the repositories of KP-like artifacts provides them in heterogeneous formats different from OWL, this component is designed to work in cooperation with the Reengineer,

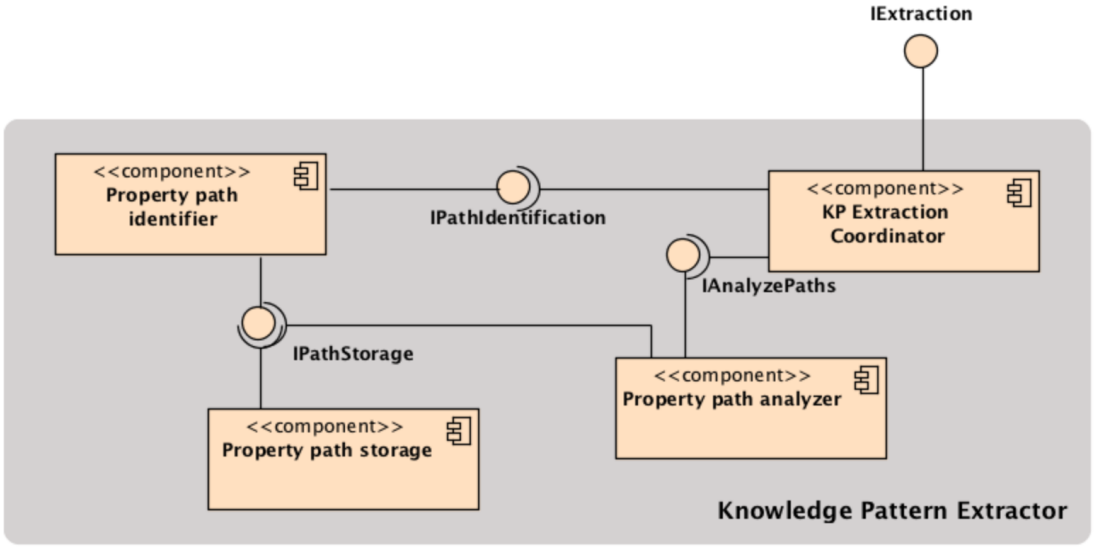


Figure 7.3: Sub-components of the Knowledge pattern extractor.

which is the specialization of the Source enricher for transforming structured data to RDF. Once data are expressed as RDF, the Knowledge Pattern Refactor is designed to apply methods for the transformations of the RDF data, as they converted by the reengineer, into a form able to capture the semantics of the original KPs. This can be perfomed by actual implementations of this components that provide functionalities for reconizing invariances and drawing boundaries over RDF data. In the next section we are going to give an example of implementation of the Refactor based on the method we used for transforming FrameNet [11] to KPs [104] (see Chapter 4). The transformation methods are conceptually organized by the Refactor into *transformation recipes*, that are basically transformation patterns [65]. Recipes are uniquely identified internally by the Recipe manager and can be re-used several times in order to transform similar KPs to OWL. Figure 7.4 shows the two components of the Knowledge Pattern Refactor, i.e., the Refactor and the Recipe manager. The former applies transformation recipes that are managed by the latter.

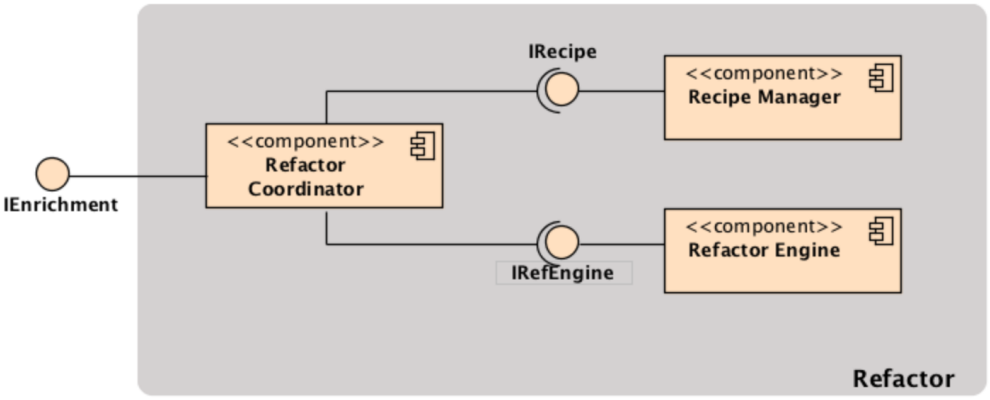


Figure 7.4: Sub-components of the Knowledge pattern extractor.

7.3.4 Knowledge Pattern Repository

The Knowledge Pattern Repository is aimed at providing basic functionalities for indexing, storing and retrieving the Knowledge Patterns that have been discovered by the system. Figure 7.5 shows the UML component diagram of the Knowledge

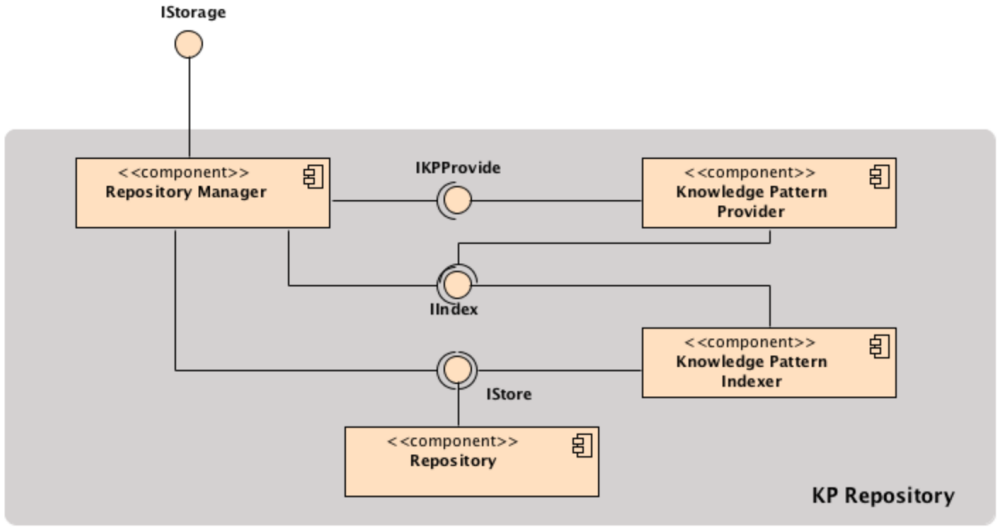


Figure 7.5: UML component diagram of the Natural Knowledge Pattern Repository.

Pattern Repository. The components are the following:

- the **Repository Manager**. It provides to access through the the interface IStorage to the functionalities defined by the other components of K∼ore.
- the **Repository**. It provides classes and methods for abstracting to the system the storage mechanisms of KPs with respect to the physical store, e.g., a RDBMS, a triple store, a file system, etc.
- the **Knowledge Pattern Indexer**. This component provides interfaces that enable the indexing of KPs int the KP store. The index allows to speed up KP fetching in the store and also to design caching mechanisms.
- the **Knowledge Pattern Provider**. The Provider is the responsible for KP fetching and their serialization to specific formats, e.g., OWL/XML, OWL/-Functional, RDF/XML, etc. Therefore, it interacts with the Knowledge Pattern Index component, which in turn is able to retrieve KPs by interacting with the Repository component.

7.4 Implementation

The system is implemented as a modular set of Java [8] components. Each component is accessible via its own RESTful Web interface. From this viewpoint, all the features can be used via RESTful service calls. Components do not depend on each other. However they can be easily combined if needed. All components are implemented as OSGi [5, 6, 7] bundles, components and services.

7.4.1 The OSGi framework

OSGi is 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, 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, 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. Any framework that implements the OSGi standard provides an environment for the modularization of applications into smaller bundles. Each bundle is a tightly coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any) [5]. The architectural stack of OSGi is depicted in Figure 7.6, which is divided into:

- **Bundles**: a group of Java classes and additional resources equipped with a detailed manifest file on all its contents, as well as additional services needed to give the included group of Java classes more sophisticated behaviors, to the extent of deeming the entire aggregate a component;

- **Services**: they connects bundles in a dynamic way by offering a publish-find-bind model for Plain Old Java Interfaces (POJI) or Plain Old Java Objects (POJO);

- **Services Registry**: an API which provides functionalities for the management of services;

- **Life-Cycle**: it provides the API for the run-time management of bundles. The API allow to be dynamically install, uninstall start, stop, and update bundles in the OSGi framework.

- **Modules**: they allow to define policies for defining encapsulation and dependencies.

By default our system uses the Apache Felix OSGi environment [6].

[6]Apache Felix OSGi environment: `http://felix.apache.org/`

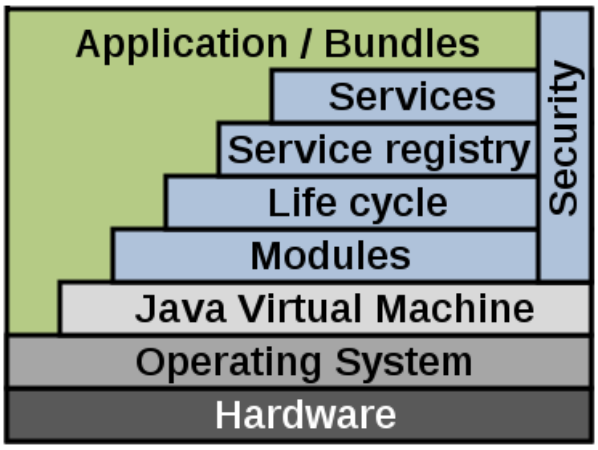


Figure 7.6: OSGi Service Gateway Architecture

7.4.2 The K∼tools

K∼ore provides a reference architecture for developing a system for KP discovery and reuse. It is implemented and realeased as an API [7] which interfaces and methods for developing actual components and group them as Apache Felix OSGi bundles. On top of the API provided by K∼ore we have implemented a set of independent tools, called K∼tools [8], used for performing the experiments described in the use cases presented in Chapters 4,5,6.

Semion

The Reengineer and the Refactor are the reference Component-based architectures that we have used for developing the Semion tool [9] [105]. The Reengineer is implemented in the Semion tool by applying reengineering patterns, namely, OWL ontologies that describe how objects from an non-RDF source have to be converted into RDF by taking into account the meta-description (again an OWL ontology) of the structure of the original source. This allows to obtain pure RDF triples

[7] Refer to `http://stlab.istc.cnr.it/stlab/K~ore` for information about licencing and download.

[8] Refer to `http://stlab.istc.cnr.it/stlab/K~tools` for information about licencing and download.

[9] Semion: `http://stlab.istc.cnr.it/stlab/Semion`

compliant to the meta description of the original source (refer to Section 4.2 for details).

The Semion tool also provides an implementation of the Refactor architecture. Here the refactoring of RDF graphs is implemented through the execution of refactoring patterns. The refactoring patterns are organized into *recipes* and expressed according to the Refactor Rule Language [10]. The Refactor Rule language has no reference rules engine that can directly execute rules expressed in that syntax. Hence, recipes of rules need to be converted into syntaxes that can be executed by available rule engines. For this purpose, the Semion Refactor implements the Adapter design pattern, which allow to translates the Java classes that represent original rules into a compatible classes that implement different rule languages. By default, the Refactor adapts rules expressed in its native language to the following languages:

- SWRL [82] through the OWL API [11] [79] binding. This allows to execute the recipes by means of reasoning with an inference engine for the Semantic Web, such as Pellet [37] or Hermit [81];

- SPARQL CONSTRUCT [119] through the adapter to Apache Jena [12] [97] and to Apache Clerezza [13];

The reason of the definition of a new language for expressing rules derives from the need of having a language that might result simpler than SWRL or SPARQL to non-experts users. Both the Semion Reengineer and Refactor of K∼tools became part of the Apache Stanbol project [14] since its incubation in the Apache Software Foundation in 2011 and are currently part of the main stream project.

[10] Refer to Appendix 9 for the BNF syntax of the rule language and to Section 4.2 for examples of such language.
[11] The OWL API: `http://owlapi.sourceforge.net/`
[12] Apache Jena: `http://jena.apache.org/`
[13] Apache Clerezza: `http://clerezza.apache.org/`
[14] Apache Stanbol: `http://stanbol.apache.org/`

PathPopularity$_{DBpedia}$

PathPopularity$_{DBpedia}$ is the implementation of the Knowledge Pattern Extractor of K∼ore. It allows to:

- create a dataset of type paths from DBpedia;
- calculate *pathPopularity* values (cf. Section 5.1.2) for each type paths;
- formalize KPs from type paths by applying boundary induction based on a configurable threshold.

Tìpalo and CiTalO

So far, we have two different implementation of the Natural Language Enhancer that come from two consequently different configurations and extensions of the its components.

The first tool is Tìpalo [15], used for automatically typing Wikipedia entities based on their natural language definitions [107], that we have described in Section 6.2. Tìpalo extends and configures the components Natural Language Enhancer in the following way:

- it uses FRED [117] as the implementation of the Ontology Learner. FRED performs ontology learning by relying on Boxer [40], which implements *computational semantics*, a deep parsing method that produces a logical representation of NL sentences in DRT;
- it implements a graph-pattern matcher based on SPARQL aimed at identifying and selecting terms from the graph returned by FRED that can be used as candidate types for a Wikipedia entity. The graph-patterns are expressed as SPARQL CONSTUCT queries and they are logically described in Tables 6.1 and 6.2;

[15] Tìpalo: `http://wit.istc.cnr.it/stlab-tools/tipalo`

- it extends the Word-sense Disambiguator by embedding UKB [3], which is a third part software for addressing word-sense disambiguation tasks based on the Personalized Page-Rank algorithm;

- it configures the Ontology Aligner in order to provide alignments to OntoWordNet [62] and a subset of Dolce+DnS classes [60]. The alignments are aimed at providing foundational grounding to the selected types that have been disambiguated.

The second tool is CiTalO [16], used for identifying the type of citations in scholarly publishing [44] (cf. Section 6.3). CiTalO uses the same architecture as Tìpalo except a different configuration of some components, such as:

- it configures the Graph-pattern Matcher with respect to different graph-patterns (see Section 6.3);

- it customize the architecture by enabling a new component for sentiment analyisis based on the Alchmemy API [17];

- it configures the Ontology Aligner in order to return properties of the CiTO ontology [112] that add semantics to the citations in scholarly publications. The alignments are computed thanks to a mapping expressed by means of `skos:closeMatch` axioms between the CiTO ontology and OntoWordNet.

[16] CiTalO: `http://wit.istc.cnr.it:8080/tools/citalo`
[17] Achemy API: `http://www.alchemyapi.com/`

Chapter 8

Aemoo: Exploratory search based on Knowledge Patterns

The contributions presented in the Chapter are the following:

- a software called Aemoo that implements a KP-aware application for supporting knowledge exploration and entity summarization on the Web based on Knowledge Patterns (cf. Sections 8.1 and 8.2);
- a user study aimed at evaluating the effectiveness and the usability of Aemoo in exploratory search task with respect to Google Search and RelFinder [77].

8.1 Approach

The Web is a huge source of knowledge and one of the main research challenges is to make such knowledge easily and effectively accessible to Web users. Applications from the Web of Data, social networks, news services, search engines, etc., attempt to address this requirement, but it is still far from being solved, due to the many challenges arising, e.g. from the heterogeneity of sources, different representations, implicit semantics of links, as well as the sheer scale of data on the Web.

Existing semantic mashup and browsing applications, such as [136, 77, 78], mostly focus on presenting linked data coming from different sources, and visualizing it in interfaces that mirror the linked data structure. Typically, they rely on

semantic relations that are explicitly asserted in the linked datasets, or in explicit annotations, e.g., microdata, without exploiting additional knowledge, e.g. coming from hypertext links, which makes both the data provided and its visualization and navigation quite limited. In practice, the problem of delivering tailored and contextualized knowledge remains unsolved, since retrieved knowledge is returned without tailoring it to any context-based rationale.

Other applications focus on text enrichment by performing identity resolution of named entities. Examples are: Zemanta[1], Stanbol Enhancer[2] and Calais[3]. Such applications are useful for enhancing text with hypertext links to related Web pages and pictures, and sometimes they provide information about the type of the identified entities, which can be useful for designing simple faceted interfaces. However, their approach does not seem inspired by relevance rationales, e.g. their results are provided without any explanation or criterion of why a piece of news, or a set of resources, is to be considered relevant.

To the best of our knowledge no existing approach attempts to organize or filter knowledge before presenting it by drawing a meaningful boundary around the retrieved data in order to limit the visualized results to what is meaningful/useful. Instead, Aemoo [4] is an application that supports knowledge exploration on the Web based on knowledge patterns, and that exploits Semantic Web technologies as well as the structure of hypertext links for enriching query results with relevant related knowledge coming from diverse Web sources. In addition, Aemoo organizes and fiters the retrieved knowledge in order to show only relevant information to users, and providing the motivation of why a certain piece of information is included. We define Aemoo a KP-aware application because KPs are part of its background knowledge and it is able to use and interact with the Web at the knowledge level (in the sense defined by Newell [103]). The approach used by Aemoo is described the following subsections.

[1] http://www.zemanta.com/
[2] http://stanbol.apache.org
[3] http://www.opencalais.com/
[4] Aemoo: `http://aemoo.org`

8.1.1 Identity resolution and entity types

Aemoo exploits DBpedia for identity resolution and to gather Wikipedia knowledge about an entity. Such task is performed in two main situations: (1) when a user types a query; (2) when collecting and filtering relevant knowledge.

User queries are processed and matched against DBpedia entities (1) for identifying the identity of the resource referred to in a query. As a result users are provided with a list of possible options (autocompletion), among which they can perform a selection (the selected entity is called hereon "subject").

Aemoo currently uses three main resources (that can be easily extended): Wikipedia, Google News, and Twitter. All entities that are linked from the Wikipedia page of the subject are used for filling the set of nodes associated with it. Additionally, it processes the current stream of Twitter messages and available articles provided by Google News in order to identify entities that co-occur with mentions of the subject. For example, consider a user that selects "Steve Jobs" as a subject, and Aemoo processing the following tweet: "Steve Jobs leaving his place at Apple to Tim Cook". Aemoo will resolve the identity of "Tim Cook" and "Apple" and would add them to the appropriate set nodes of entities related to Steve Jobs.

Aemoo retrieves the types of the resolved entities, according to the DBpedia taxonomy of types. The type is used for providing users with additional information about the subject (it is indicated on the top-left), and as a criterion for assigning an entity to a certain set node.

8.1.2 Knowledge Patterns

Types are also used as a criterion for filtering the knowledge to be presented. Aemoo approach is based on the application of *Encyclopedic Knowledge Patterns* (EKPs) (cf. Section 5.2) used as a means for designing a meaningful knowledge summary about a subject. Aemoo builds entity summaries by applying EKPs as lenses on data associated with that entity: the concept map of a subject is built by only including the elements i.e., types, of the EKP associated with that subject type. For example,

given the subject "Paris", which has type `dbpo:City`[5], and the EKP associated with `dbpo:City` that includes the types: `dbpo:City`, `dbpo:Country`, `dbpo:University`, and `dbpo:OfficeHolder`, Aemoo summary for Paris shows a concept map including a set node for each of these types, and each set contains entities of that type that are linked from the Paris Wikipedia page.

EKPs are also used for identifying "curiosities" about a subject. Aemoo uses long-tail links (that are normally taken out by the EKP lens) for building a different perspective over the knowledge related to an entity, which includes peculiar facts instead of core knowledge.

8.1.3 Explanations and semantics of links

All entities included in a subject summary are related to it by a hypertext link, or because they co-occur with the subject in a news article or a tweet. The meaning of such relations is implicit but explained by the text surrounding the anchor or the co-occurrence reference. Aemoo exploits this aspect by extracting such pieces of text and showing them in association with each specific link. Additionally, Aemoo takes advantage of the statistics about semantic relations asserted in DBpedia: it shows to users a list of relations that typically hold between the types of two linked entities together with their frequency data.

In summary, Aemoo performs KP-based knowledge exploration, which makes it especially novel. It exploits the structure of linked data, and organizes it by means of EKPs for supporting exploratory search. The use of EKPs allows Aemoo to draw meaningful boundaries around data. In this way, Aemoo performs both *enrichment* and *filtering* of information, based on the structure of EKPs, which reflects the most common way to describe entities of a particular type. Users are guided through their navigation: instead of being presented with a bunch of triples or a big unorganized graph they navigate through *units of knowledge* and move from one to the other without losing the overview of an entity.

[5]dbpo: stands for http:/dbpedia.org/ontology

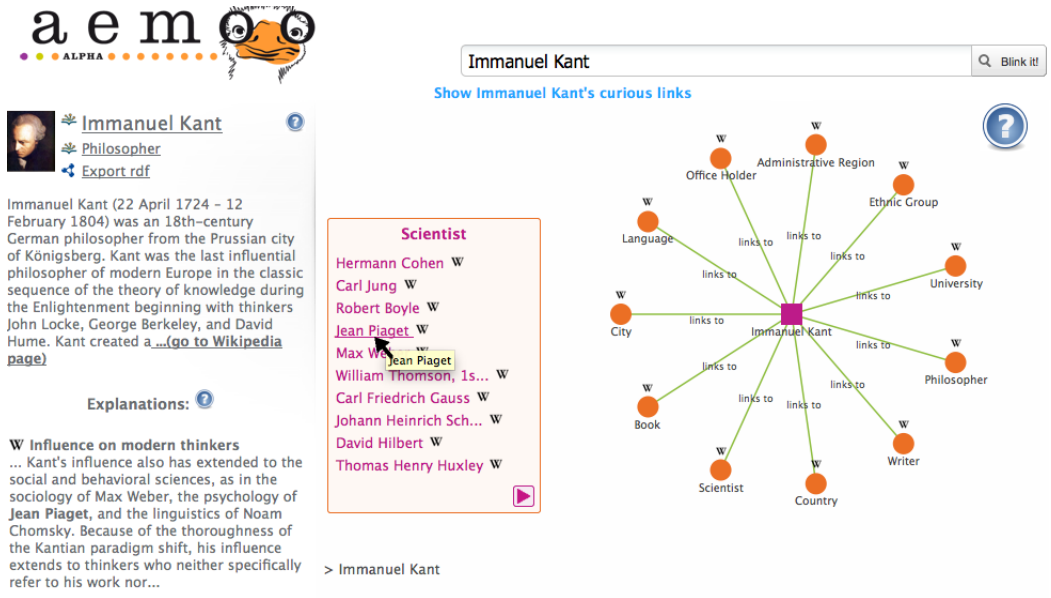


Figure 8.1: Aemoo: initial summary page for query "Immanuel Kant".

8.2 Usage scenarios

In this section we describe Aemoo usage and interface through three simple scenarios.

8.2.1 Scenario 1: Knowledge Aggregation and Explanations.

Pedro is a high school student, his homework today is to write a report about Immanuel Kant (IK). He types "Immanuel Kant" in the search interface, and Aemoo returns a summary page about him (cf. Figure 8.1). On the left side of the page, Pedro can read that IK is a philosopher, together with some general information about him, and a thumbnail image. This information will be enriched as a consequence of, and during, his navigation. At the same time, a concept map built around IK (as a central squared node) has appeared in the center-right of the page. The circular nodes represent sets of resources of a certain type (the type is shown as the label on the node), we refer to them as *set nodes*. Additionally, icons on set nodes indicate the source from which its contained information is taken, e.g. Wikipedia.

The set nodes change depending on the type of the inspected entity (Philosopher in this case), according to the knowledge pattern associated with it (cf. Section 8.1).

Such types are the ones that a user would intuitively expect to see in a summary description of a Philosopher, according to the empirical study described in [106].

An infotip appearing when hovering over a link between IK and a set node, shows a list of possible semantic relations that can explain that specific link type, according to their frequencies in DBpedia (cf. Figure 8.1). Such list is not exhaustive, it only shows existing DBpedia relations (extracted from Wikipedia infoboxes) that hold between two specific types. For example, the relations between IK and cities could be `birthPlace` or `placeOfBirth` if we considered only DBpedia asserted relations. This example shows the limit of the current DBpedia relation coverage, in fact cities can be related to Immanuel Kant also for other reasons than being his place of birth, which can be explained in Aemoo by additional information in the explanation section (left-bottom side of the interface).

IK links to a set of scientists, which is interesting information for our user Pedro, who wants to know more about this relation. By hovering on a set node e.g., *Scientist*, he triggers the visualization of a list of resources contained in the set, meaning that those resources are connected to IK (cf. Figure 8.2. By hovering on a specific entity of the set e.g., Jean Piaget, new information is visualized under the "Explanations" section (left-bottom). Such information explains the meaning of that connection. In the example, Jean Piaget is linked to IK because his work was influenced by Kant's.

Explanations come from different possible sources i.e., Wikipedia, Twitter, and Google news. The sources to be used can be chosen by users through a set of checkbox put in the top-right corner of the interface.

8.2.2 Scenario 2: Exploratory search.

Pedro, however, would like to collect some more information about Jean Piaget, hence, he clicks on that entity in the list. Aemoo changes context from IK to Jean Piaget, showing a new summary page for the scientist. Pedro can perform exploratory search by inspecting set nodes and lists associated with Jean Piaget, and possibly other entities. Figure 8.3 shows the situation after some exploration

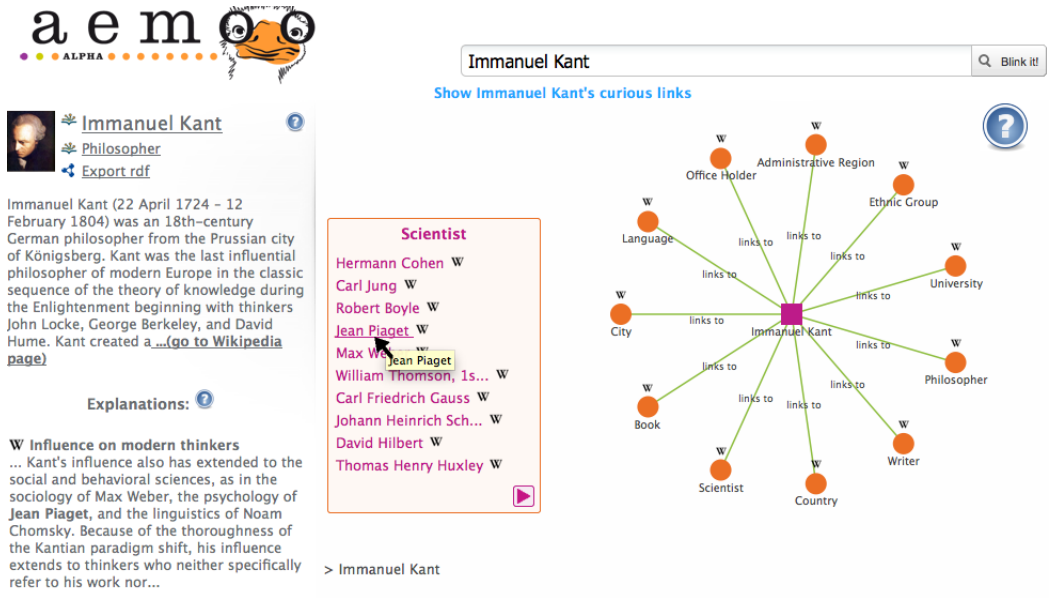


Figure 8.2: Aemoo: browsing relations between "Immanuel Kant" and scientists.

steps. Through the breadcrumb (located at the center-bottom of the interface) Pedro can go back and forth, and revisit his exploration path and its associated knowledge.

8.2.3 Scenario 3: Curiosity.

Eva is an editorial board member of a TV program that dedicates each episode to a different country. Now she has to edit the episode about Italy. She uses Aemoo, as described above, for building a summary about the country that can be useful for the introductory part of the show. However, she wants to find peculiar information that make the episode more interesting to her audience. Aemoo helps on this task through the "curiosity" functionality, that can be triggered by clicking on the link between the search field and the concept map (cf. Figure 8.3). Aemoo will change perspective and will provide a new summary for the same entity. In fact, Eva will be presented with additional knowledge about Italy, which was not previously included in the summary. What is now shown are "special" facts about Italy, things that are not commonly used to describe a country. Knowledge is again visualized as a

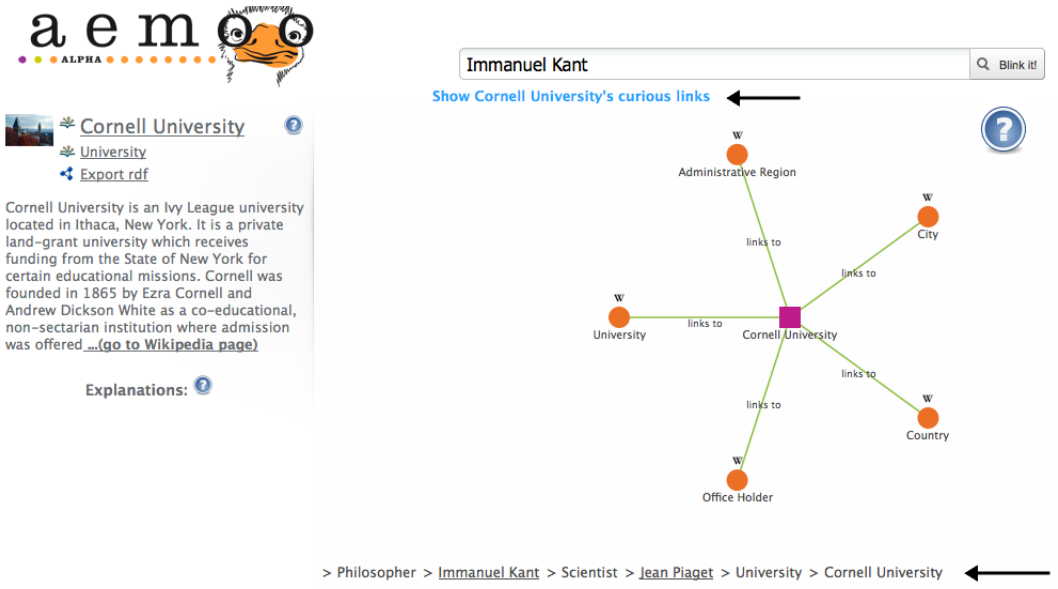


Figure 8.3: Aemoo: breadcrumb and curiosity

concept map, and enriched with news and tweets just as it happens for the previous summary, but this time the set nodes are selected with a different criterion: they are types of resources that are unusual to be included in the description of a country, hence possibly peculiar.

8.3 Under the hood: design and implementation of Aemoo

Aemoo is released as a web application: it consists of a server side component implemented as a Java-based REST service, and a client side component based on HTML and JavaScript. The client side interacts with third party components via REST interfaces through AJAX.

The server side exposes a REST service for retrieving EKP-based graphs as well as "curiosity graphs" about entities. Its input is an entity URI, e.g. dbpedia:Barack_Obama[6]. Its output is an RDF graph corresponding to the summariza-

[6]dbpedia: stands for http://dbpedia.org/resource/

tion based on the EKP associated with the entity type. The RDF graph is obtained by generating a SPARQL CONSTRUCT from the selected EKP.

The client side component handles the graphical visualization of Aemoo through the JavaScript InfoVis Toolkit[7], a Javascript library that supports the creation of interactive data visualizations for the Web. Abstracts and thumbnails are retrieved by querying the DBpedia SPARQL endpoint exposed as a REST service.

Aemoo also detects relations between the inspected entity and other entities from Twitter [8] as well as from Google News[9]. Tweets and news are retrieved using their respective REST services. For performing identity resolution on user queries, tweets, and news we use Apache Stanbol Enhancer[10]. Entities recognized in tweets and news are dynamically added to the graph map. Explanations are extracted from the text surrounding wikilinks in the subject Wikipedia page, the text of tweets and news, and are associated with provenance information.

8.4 Evaluation

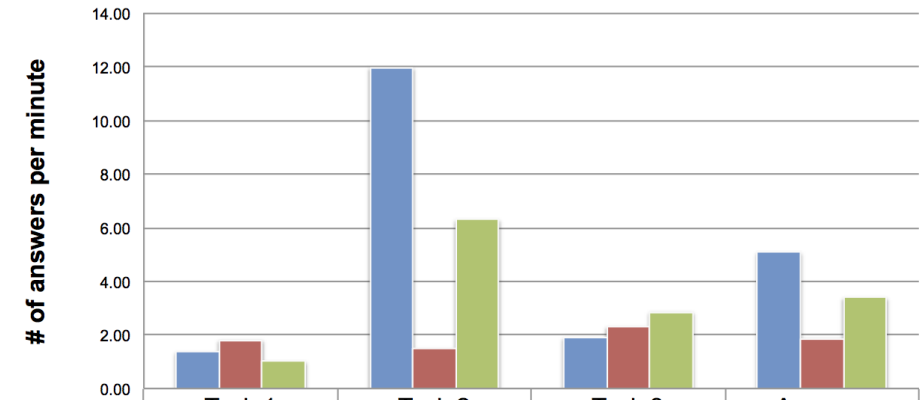
We carried out a series of user-based evaluation tests to assess the efficiency, effectiveness and user satisfaction of Aemoo with respect to some exploratory search tools. We identified two tools that support exploratory search tasks, i.e., Google Search and RefFinder [77]. Google Search is the most-used search engine on the World Wide Web, handling more than three billion searches each day. RelFinder is a tool that shows in a graph-based interface the relations it finds between two or more entities in Linked Data. A user can explore entities or focus on specific relations by interacting with this graph. We asked 5 groups of users (for a total number of 32 users) to perform three different kind of exploratory search tasks. Each group performed the three tasks on two tools, i.e., Aemoo and one between Google and RelFinder. The order in which each user evaluated the two tools was homogeneously

[7]http://thejit.org/
[8]https://search.twitter.com/search.json
[9]https://ajax.googleapis.com/ajax/services/search/news
[10]http://stanbol.apache.org/

alternated among users in order to avoid evaluation biases with respect to the tool used as first. Hence, there is the same number of users who started the evaluation by using Aemoo as the first tools and the same number of users who started the evaluation by using Google or RelFinder as first. The tasks asked to the users are the following:

- to make a summary on a specific topic, i.e., to make a summary on the topic "Alan Turing";
- to find specific relations between an entity and a category of entities, i.e., What are the places related to "Snow White"?
- to provide an explanation about existing relations between two entities, i.e., Why "Snow White" is related to "Charlize Theron"?

The three tasks are thought to cover typical exploratory search scenarios as identified by Marchionini [94] and to balance the comparison taking into account the differences among tools in order to avoid to privilege one tool with respect to the others. Figure 8.4 shows the number of correct answers per minute given by users for each task and tool. On average Aemoo performs better than the other

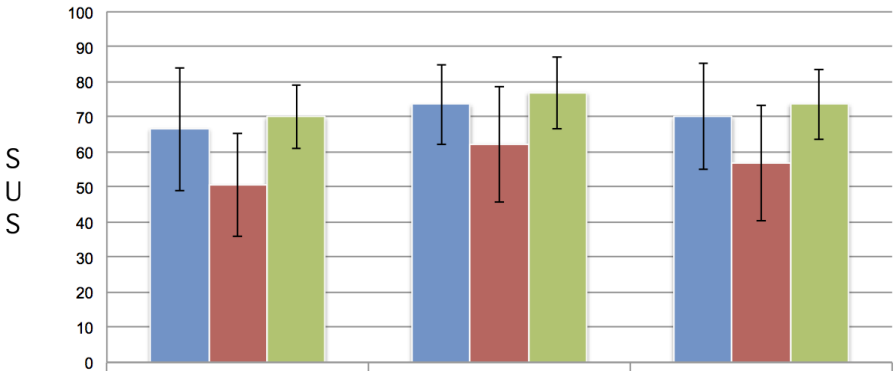


	Task 1	Task 2	Task 3	Average
Aemoo	1.37	11.99	1.90	5.08
RelFinder	1.81	1.49	2.30	1.87
Google	1.01	6.31	2.86	3.39

Figure 8.4: Number of correct answers per minute for each task and tool.

two tools, but it is due to its efficiency in look-up tasks (it outperforms the other tools in the second task). Instead, in the first task, which is about summarization, RelFinder performs slightly better than Aemoo and Google and, finally, in the third task Google performs slightly better than Aemoo and RelFinder.

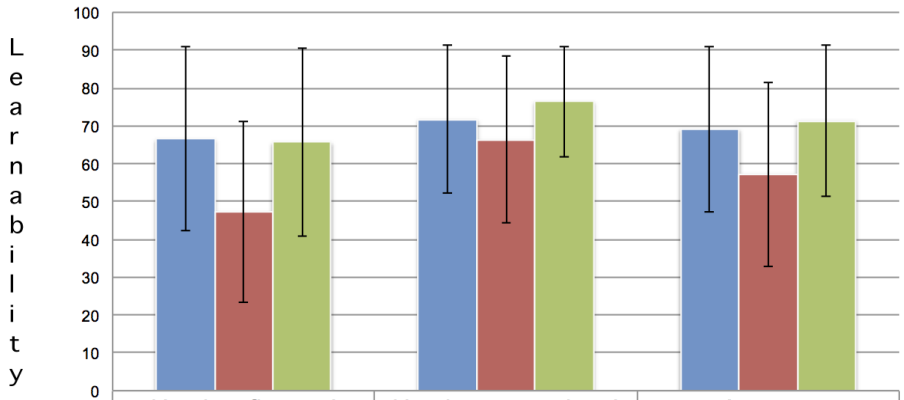
After the completion of the three tasks we asked the users to rate the system from ten perspectives on a five-point Likert scale [90] aimed at capturing System Usability Scale (SUS) [29]. Figure 8.5 depicts the comparison of the SUS results among the three tools in the cases they are used as first tool, as second tool and the overall average. SUS values are weighted on a scale between 0 and 100. Values between



	Used as first tool	Used as second tool	Average
Aemoo	66.58 (17.58)	73.55 (11.53)	70.06 (15.08)
Relfinder	50.55 (14.72)	62.25 (16.56)	56.71 (16.41)
Google	70 (8.96)	76.94 (10.29)	73.67 (10.04)

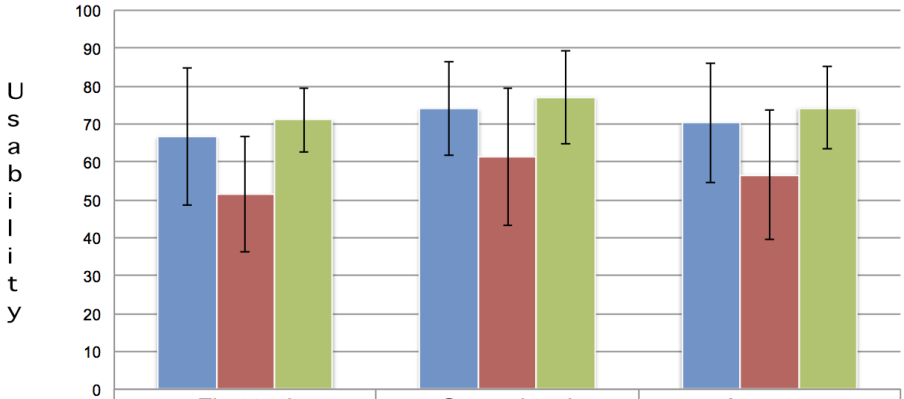
Figure 8.5: SUS scores and standard deviation values for Aemoo, RelFinder and Google. Standard deviation values are expressed between brackets and shown as black vertical lines in the chart.

brackets represent standard deviations and they are also reported into the chart as vertical error bars in black. It emerges that in Aemoo the user experience (70.06) is perceived significantly better than in RelFinder (56.71). Instead, the best user experience is perceived with Google (73.67), but most of the users reported Google as their favorite search engine during a pre-questionnaire aimed at understanding users' skills. Furthermore, the average SUS score reported by Aemoo surpasses the target of 68, which is required to demonstrate a good level of usability [124] and so does the SUS score reported by Google. Contrariwise, RelFinder fails to surpass



	Used as first tool	Used as second tool	Average
Aemoo	66.45 (24.31)	71.71 (19.47)	69.08 (21.89)
Relfinder	47.22 (24.02)	66.25 (22.08)	57.24 (24.4)
Google	65.62 (24.77)	76.39 (14.58)	71.32 (20.13)

(a) Learnability.



	First tool	Second tool	Average
Aemoo	66.61 (18.07)	74.01 (12.41)	70.31 (15.75)
Relfinder	51.39 (15.16)	61.25 (18.11)	56.58 (17.08)
Google	71.09 (8.48)	77.08 (12.3)	74.26 (10.8)

(b) Usability.

Figure 8.6: Learnability and Usability values and standard deviations. Standard deviation values are expressed between brackets and shown as black vertical lines in the chart.

this target. It is reasonable that SUS values of a tool used as second tool are better than those of the same tool used as first because each task performed by a user is repeated twice (one time per tool). Hence, the second time a user is more familiar with a task and this affects the usability. In addition to the main SUS scale, we are

also interested in examining the sub-scales of pure Usability and pure Learnability of the system that have been proposed some years ago by Lewis and Sauro [89]. Figure 8.6 shows values and the standard deviations for the these two orthogonal structures of the SUS. Namely, Figure 8.6(a)) depicts Learnability scores and their related standard deviations and Figure 8.6(b) reports Usability scores and their related standard deviations. Again, Aemoo is significantly perceived easier to learn and more usable than RelFinder, that is actually the reference competitor of Aemoo for this evaluation.

As part of our future work we want to evaluate the final surveys we proposed to the users after the SUS questionnaire for analyzing their satisfaction in using the systems. We are proceeding with the *open coding* (extracting relevant sentences from the text, i.e., codes) and *axial coding* (the rephrasing of the codes so as to have connections emerge from them and generate concepts), and we are analysing their frequency so as to let more relevant concepts emerge. These concepts associated to a satisfaction score will help us in better understanding what are the most prominent or the missing features of the tools.

Chapter 9

Conclusion and future work

The realization of the Web of Data (aka Semantic Web) partly depends on the ability to make meaningful knowledge representation and reasoning. Recently [66] has introduced a vision of a pattern science for the Semantic Web as the means for achieving this goal. Such a science envisions the study of, and experimentation with, Knowledge Patterns (KPs): small, well connected units of meaning which are (i) task-based, (ii) well-grounded, and (iii) cognitively sound. Linked data and social web sources such as Wikipedia give us the chance to empirically study what are the KPs in organizing and representing knowledge. Furthermore, KPs can be used for evaluating existing methods and models that were traditionally developed with a top-down approach, and open new research directions towards new reasoning procedures that better fit the actual Semantic Web applications need.

In this work we have treated the problem of extracting, transforming and reusing KPs in the Web. This means we have provided solutions to two main challenging issues, i.e.,

- the *knowledge soup* problem;
- the *knowledge boundary* problem.

The knowledge soup is the heterogeneity of formats and semantics used in the Web of Data for representing knowledge. For example Linked Data contain datasets with

real world facts (e.g. geo data), conceptual structures (e.g. thesauri, schemes), lexical and linguistic data (e.g. wordnets, triples inferred from NLP algorithms), social data about data (e.g. provenance and trust data), etc. The knowledge boundary problem is about the need of drawing relevant boundaries around data that allow to select the meaningful knowledge that identifies a certain KP. For example, this means to select a set of triples in an RDF graph able to give a unifying view with respect to a certain context. In order to deal with these issues we have proposed an approach that tackles KP discovery from two different perspectives:

- the transformation of top-down modelled KP-like artifacts from existing sources in the Web (e.g. FrameNet [11]);
- the bottom-up extraction of KPs based on the analysis of how the knowledge in Linked Data is typically organized.

We enclose the first perspective in a solution based on the Semion methodology [105] (cf. Chapter 4). This allows to transform KP-like artifacts available in the Web in heterogeneous formats and semantics thanks to a two step-based approach aimed at (i) performing a purely syntactic transformation of the original source to RDF, i.e, the reengineering step, and (ii) adding semantics to RDF data in order to make KPs emerge, i.e., the refactoring step. Based on this solution we have discussed a case study that we presented in [104] about the transformation of FrameNet frames to KPs. The result of this case study is twofold: (i) an RDF dataset available as Linked Data [1] and linked to WordNet and other lexical datasets; (ii) a collection of 1024 KPs [2] formalized as small OWL2 ontologies. From this case study we learn that the customization is key with KP-like artifacts because there are use cases for maintaining the semantics of the original resource, often a purely intensional one (similar to the practice of using SKOS with thesauri), as well as for morphing the original semantics to something closer to the extensional formal semantics of web ontologies. In between these two ends, there are several intermediate cases

[1] The dataset is avaliable at http://ontologydesignpatterns.org/ont/framenet/fndata_v5.rdf.zip
[2] Available at `http://ontologydesignpatterns.org/ont/framenet/kp`.

and exceptions, which make the case for tools that minimize hard-coding of the transformation semantics, and preserve the opportunity to learn and share good practices for transforming KP-like artifacts to linked data and domain knowledge. With regard to this case study our ongoing work concentrates on the refinement of the RDF dataset with the Berkeley FrameNet group, the generation of new links to lexical datasets as well as other relevant LOD datasets (e.g. DBpedia), the creation of the FrameNet valence dataset, which will be a substantial (about 35 million triples) resource for hybridizing Semantic Web and Linked Data, and the refinement of the recipe to produce and automatically publish FrameNet-based KPs on the ODP portal. These KPs implement a large section of the rich KP structure envisaged by [66], with formal axioms, lexically motivated vocabulary, textual corpus grounding, and data grounding.

We enclose the second perspective in a solution based on the analysis of path types (cf. Chapter 5). A type path is a sequence of connected triple patterns whose occurrences have (i) the same `rdf:type` for their subject nodes, and (ii) the same `rdf:type` for their object nodes. Type paths allow to analyze data and the linking structure among data by looking for recurrent structures from an intensional point of view. We have defined a measure for drawing boundaries around data based on the notion of *pathPopularity*. Informally, the *pathPopularity* is a contextualized indicator that allows to determine how popular, i.e., frequent, is a certain path in a dataset. We have shown in a case study how it is possible to extract KPs by applyping this solution to Wikipedia links [3]. Such case study was presented in [106] and allowed us to collect 184 KPs, called Encyclopedic as they are able to capture encyclopedic knowledge having been extracted from Wikipedia, the largest collaboratively built encyclopedia. There are many directions that the kind of research we did with EKPs opens up. For example, investigating how to make relevant "long tail" features (c.f. Section 5.2.3) emerge for specific resources and requirements is one of the research directions we want to explore. This is useful for evolving Aemoo to meaningfully take into account the peculiar knowledge for building entity

[3] represented as Linked Data in the `dbpedia_page_links_en` dataset of DBpedia

summaries. Another obvious elaboration of EKP discovery is to infer the object properties that are implicit in a wikilink. This task is called *relation discovery*. An approach we want to investigate is the hybridization between the EKPs and the Statistical Knowledge Patterns (SKPs) [146] that provide patterns of object relations among DBpedia classes. Other approaches we want to investigate are the induction of relations from infobox properties, from top superclasses, or by punning of the object type, i.e., treating the object type as an object property.

Inasmuch as the limited usage of ontologies and controlled vocabularies in Linked Data restricts the KP extraction method based on type paths, we have proposed a solution for the enrichment of Linked Data with additional metadata, e.g., `rdf:type` axioms, based on the exploitation of natural language annotations (cf. Chapter 6). Based on this method we have proposed two case studies, i.e.:

- the *Tìpalo* algorithm, which allows to infer an entity type by analyzing the natural language definition available in its abstract. *Tìpalo* allowed us to extract a initial version of a natural ontology of Wikipedia, i.e., ORA, that we want to use for further refining the extraction of KPs from Wikipedia. Currently, we are working at refining ORA for limiting synonymy among classes and to align it to DBpedia Ontology [4] and YAGO [133].

- the *CiTalO* algorithm, which allows to infer the type of citations in scholarly articles. In this context, citation are interpreted as links among articles. We are currently working on the evaluation of the CiTO ontology, which is used for labeling citations with a property able to capture the citational meaning, and on the investigation of lexico-syntactical patterns for citations that can be converted to graph-patterns to use in our method (cf. Section 6.1.3)

An important research direction we want to carry on concerns the validation of top-down defined KPs (e.g., KPs from FrameNet) with respect to bottom up emerging KPs. In fact, the nature of KPs is mainly empirical [52, 66], hence the validity of top-down KPs should be empirically proved. So far, evidence about this is

[4] `http://dbpedia.org/ontology`

only episodic, even if [53] has recently shown correlations between frame elements as defined in FrameNet frames and frame elements defined by users in a crowdsourced experiment.

The three solutions proposed have been implemented in a software architecture, i.e., K∼ore, based on the hybridization of the Component-based and REST software architectural styles. K∼ore provides API for experimenting with KP transformation, extraction and reuse. These API composes the framework of tools, i.e., K∼tools, that we used for implementing the case studies illustrated so far. We believe that K∼ore can be a valid solution for supporting the development of cognitive architectures [87] and our ongoing work is oriented in this direction.

Finally, we have presented Aemoo, a tool that uses KPs for providing entity summaries in the context of exploratory search. We have defined Aemoo as a KP-aware application as it exploits KPs at the knowledge level, as defined by Newell [103]. An initial evaluation shows promising results that we want to investigate farther. Hence, we are planning a user-based evaluation aimed at comparing the effectiveness and the usability of the summary proposed by Aemoo with respect to that proposed by the Google Knowledge Graph.

Appendix A

Refactor Rule Language

```
<DEFAULT> SKIP : {
" "
}

<DEFAULT> SKIP : {
"\r"
| "\t"
| "\n"
}


<DEFAULT> TOKEN : {
<LARROW: "->">
| <COLON: ":">
| <EQUAL: "=">
| <AND: ".">
| <COMMA: ",">
| <REFLEXIVE: "+">
| <SAME: "same">
| <DIFFERENT: "different">
| <LESSTHAN: "lt">
| <GREATERTHAN: "gt">
| <IS: "is">
| <NEW_NODE: "newNode">
| <LENGTH: "length">
| <SUBSTRING: "substring">
| <UPPERCASE: "upperCase">
| <LOWERCASE: "lowerCase">
| <STARTS_WITH: "startsWith">
| <ENDS_WITH: "endsWith">
| <LET: "let">
```

```
| <CONCAT: "concat">
| <HAS: "has">
| <VALUES: "values">
| <NOTEX: "notex">
| <PLUS: "sum">
| <MINUS: "sub">
| <NOT: "not">
| <NAMESPACE: "namespace">
| <LOCALNAME: "localname">
| <STR: "str">
| <APOX: "^">
| <UNION: "union">
| <CREATE_LABEL: "createLabel">
| <SPARQL_C: "sparql-c">
| <SPARQL_D: "sparql-d">
| <SPARQL_DD: "sparql-dd">
| <PROP: "prop">
| <IS_BLANK: "isBlank">
| <FORWARD_CHAIN: "!">
}


<DEFAULT> TOKEN : {
<LPAR: "(">
| <RPAR: ")">
| <DQUOT: "\"">
| <LQUAD: "[">
| <RQUAD: "]">
}


<DEFAULT> TOKEN : {
<NUM: (["0"-"9"])+>
| <VAR: (["0"-"9","a"-"z","A"-"Z","-","_","."])+>
| <VARIABLE: "?" (["0"-"9","a"-"z","A"-"Z","-","_"])+>
| <URI: "<" (["0"-"9","a"-"z","A"-"Z","-","_",".","#",":","/","(",")"])+ ">">
| <STRING: "\"" (["0"-"9","a"-"z","A"-"Z","-","_",".",":","/","#","\\","?"," ","!","$","%"])+ "\"">
| <SPARQL_STRING: "%" (["0"-"9","a"-"z","A"-"Z","-","_",".",":","/","#","\\","?"," ","!","$","%",
  "{","}","(",")","\"","<",">","=","+","\n","\t","&","|",","])+ "%">
| <BNODE: "_:" (["0"-"9","a"-"z","A"-"Z","-","_","."])+>
}

NON-TERMINALS

start ::= expression expressionCont
expressionCont ::= ( <AND> expression )
```

```
|
expression ::= prefix expressionCont
prefix ::= getVariable ( equality | rule )
| <FORWARD_CHAIN> getVariable rule
| <REFLEXIVE> getVariable rule
equality ::= <EQUAL> ( getURI )
rule ::= <LQUAD> ruleDefinition <RQUAD>
ruleDefinition ::= atomList <LARROW> atomList
| <SPARQL_C> <LPAR> <SPARQL_STRING> <RPAR>
| <SPARQL_D> <LPAR> <SPARQL_STRING> <RPAR>
| <SPARQL_DD> <LPAR> <SPARQL_STRING> <RPAR>
atomList ::= atom atomListRest
|
atomListRest ::= <AND> atomList
|
atom ::= classAtom
| individualPropertyAtom
| datavaluedPropertyAtom
| letAtom
| newNodeAtom
| comparisonAtom
| unionAtom
unionAtom ::= <UNION> <LPAR> atomList <COMMA> atomList <RPAR>
createLabelAtom ::= <CREATE_LABEL> <LPAR> stringFunctionAtom <RPAR>
propStringAtom ::= <PROP> <LPAR> stringFunctionAtom <COMMA> stringFunctionAtom <RPAR>
endsWithAtom ::= <ENDS_WITH> <LPAR> stringFunctionAtom <COMMA> stringFunctionAtom <RPAR>
startsWithAtom ::= <STARTS_WITH> <LPAR> stringFunctionAtom <COMMA> stringFunctionAtom <RPAR>
stringFunctionAtom ::= ( concatAtom | upperCaseAtom | lowerCaseAtom | substringAtom | namespaceAtom |
    localnameAtom | strAtom | stringAtom | propStringAtom | createLabelAtom )
strAtom ::= <STR> <LPAR> iObject <RPAR>
namespaceAtom ::= <NAMESPACE> <LPAR> iObject <RPAR>
localnameAtom ::= <LOCALNAME> <LPAR> iObject <RPAR>
stringAtom ::= uObject
concatAtom ::= <CONCAT> <LPAR> stringFunctionAtom <COMMA> stringFunctionAtom <RPAR>
upperCaseAtom ::= <UPPERCASE> <LPAR> stringFunctionAtom <RPAR>
lowerCaseAtom ::= <LOWERCASE> <LPAR> stringFunctionAtom <RPAR>
substringAtom ::= <SUBSTRING> <LPAR> stringFunctionAtom <COMMA> numericFunctionAtom
    <COMMA> numericFunctionAtom <RPAR>
numericFunctionAtom ::= ( sumAtom | subtractionAtom | lengthAtom | numberAtom )
lengthAtom ::= <LENGTH> <LPAR> stringFunctionAtom <RPAR>
sumAtom ::= <PLUS> <LPAR> numericFunctionAtom <COMMA> numericFunctionAtom <RPAR>
subtractionAtom ::= <MINUS> <LPAR> numericFunctionAtom <COMMA> numericFunctionAtom <RPAR>
numberAtom ::= ( <NUM> | <VARIABLE> )
classAtom ::= <IS> <LPAR> iObject <COMMA> iObject <RPAR>
newNodeAtom ::= <NEW_NODE> <LPAR> iObject <COMMA> dObject <RPAR>
letAtom ::= <LET> <LPAR> iObject <COMMA> stringFunctionAtom <RPAR>
```

```
individualPropertyAtom ::= <HAS> <LPAR> iObject <COMMA> iObject <COMMA> iObject <RPAR>
datavaluedPropertyAtom ::= <VALUES> <LPAR> iObject <COMMA> iObject <COMMA> dObject <RPAR>
sameAsAtom ::= <SAME> <LPAR> stringFunctionAtom <COMMA> stringFunctionAtom <RPAR>
lessThanAtom ::= <LESSTHAN> <LPAR> iObject <COMMA> iObject <RPAR>
greaterThanAtom ::= <GREATERTHAN> <LPAR> iObject <COMMA> iObject <RPAR>
differentFromAtom ::= <DIFFERENT> <LPAR> stringFunctionAtom <COMMA> stringFunctionAtom <RPAR>
reference ::= getURI
    | getVariable <COLON> getVariable
varReference ::= getURI
    | getVariable <COLON> getVariable
getURI ::= <URI>
getVariable ::= <VAR>
getString ::= <STRING>
getInt ::= <NUM>
uObject ::= ( variable | reference | getString | getInt )
iObject ::= variable
    | reference
dObject ::= ( literal | variable )
literal ::= ( getString typedLiteral | getInt typedLiteral )
typedLiteral ::= ( <APOX> <APOX> reference | )
variable ::= <NOTEX> <LPAR> <VARIABLE> <RPAR>
    | <VARIABLE>
    | <BNODE>
notAtom ::= <NOT> <LPAR> comparisonAtom <RPAR>
isBlankAtom ::= <IS_BLANK> <LPAR> iObject <RPAR>
comparisonAtom ::= ( sameAsAtom | lessThanAtom | greaterThanAtom | differentFromAtom | notAtom
    |   startsWithAtom | endsWithAtom | isBlankAtom )
```

References

[1] RDFa in XHTML: Syntax and Processing, W3C recommendation, 2008.

[2] OWL 2 Web Ontology Language Document Overview. W3C Recommendation, 10 2009.

[3] E. Agirre and A. Soroa. Personalizing PageRank for Word Sense Disambiguation. In *Proceedings of the 12th conference of the European chapter of the Association for Computational Linguistics (EACL-2009)*, Athens, Greece, 2009. The Association for Computer Linguistics.

[4] C. Alexander. The timeless way of building. 1979.

[5] T. O. Alliance. OSGi Service Platform Release 4 Version 4.2, Compendium Specification. Committee specification, Open Services Gateway initiative (OSGi), September 2009.

[6] T. O. Alliance. OSGi Service Platform Release 4 Version 4.2, Core Specification. Committee specification, Open Services Gateway initiative (OSGi), September 2009.

[7] T. O. Alliance. OSGi Service Platform Release 4 Version 4.2, Enterprise Specification. Committee specification, Open Services Gateway initiative (OSGi), March 2010.

[8] K. Arnold and J. Gosling. *The Java Programming Language*. Addison Wesley, 1996.

[9] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using formal concept analysis. In *Proceeding of the International Joint Conferences on Artificial Intelligence*, pages 230–235, 2007.

[10] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[11] C. F. Baker, C. J. Fillmore, and J. B. Lowe. The Berkeley FrameNet Project. In *Proc. of the 17th international conference on Computational linguistics*, pages 86–90, Morristown, NJ, USA, 1998.

[12] K. Barker, T. Copeck, S. Szpakowicz, and S. Delisle. Systematic construction of a versatile case system. *Natural Language Engineering*, 3(4):279–315, 1997.

[13] K. Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *Proceedings of the 1st international conference on Knowledge capture*, pages 14–21. ACM, 2001.

[14] K. Barker, B. Porter, and P. Clark. A Library of Generic Concepts for Composing Knowledge Bases. In *Proceedings of the International Conference on Knowledge Capture*, pages 14–21, Victoria, British columbia, 2001. ACM Press, New York.

[15] L. W. Barsalou. Perceptual symbol systems. *Behavioral and brain sciences*, 22(04):577–660, 1999.

[16] T. Berners-Lee. Design issues: Linked Data. Technical report, World Wide Web Consortium (W3C), July 2006.

[17] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), 2005. Available at `http://www.ietf.org/rfc/rfc3986.txt`.

[18] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[19] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data-The story so far. *International Journal on Semantic Web and Information Systems*, 4(2):1–22, 2009.

[20] C. Bizer, A. Jentzsch, and R. Cyganiak. State of the LOD cloud. Technical report, Freie Universität Berlin, September 2011.

[21] S. Bloehdorn and Y. Sure. Kernel Methods for Mining Instance Data in Ontologies. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, G. Schreiber, and P. Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 58–71, Busan, Korea, November 2007. Springer Verlag.

[22] E. Blomqvist, V. Presutti, E. Daga, and A. Gangemi. Experimenting with extreme design. In P. Cimiano and H. S. Pinto, editors, *EKAW*, volume 6317 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2010.

[23] E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svátek. *Proc. 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.

[24] H. Bohring and S. Auer. Mapping xml to owl ontologies. In *Proceedings of 13. Leipziger Informatik-Tage (LIT 2005), Sep. 21-23*, Lecture Notes in Informatics (LNI), September 2005.

[25] M. Bramer and V. 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.

[26] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.

[27] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[28] D. Brickley and L. Miller. FOAF vocabulary specification. Technical report, FOAF project, May 2007. Published online on May 24th, 2007 at `http://xmlns.com/foaf/spec/20070524.html`.

[29] J. Brooke. SUS: A quick and dirty usability scale. *Usability evaluation in industry*, pages 189–194, 1996.

[30] J. Cardoso, M. Hepp, and M. D. Lytras, editors. *The Semantic Web: Real-World Applications from Industry*, volume 6 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2007.

[31] N. Christianini and J. Shawe-Taylor. *Support Vector Machines and Other kernel-based Learning Methods*. Cambridge University Press, 2000.

[32] P. Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer, 2006.

[33] P. Cimiano, A. Hotho, and S. Staab. Learning Concept Hierarchies from Text Corpora using Formal Concept Analysis. *J. Artif. Intell. Res.(JAIR)*, 24:305–339, 2005.

[34] P. Cimiano and J. Völker. Text2Onto. In *Natural language processing and information systems*, pages 227–238. Springer, 2005.

[35] P. Clark and B. Porter. *KM - The Knowledge Machine 2.0: Users Manual*. Boeing Phantom Works/University of Texas at Austin, 1999.

[36] P. Clark, J. Thompson, and B. Porter. Knowledge Patterns. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge*

Representation and Reasoning, pages 591–600, San Francisco, 2000. Morgan Kaufmann.

[37] Clark & Parsia, LLC. Pellet: OWL 2 Reasoner for Java, 2011.

[38] A. Collins and M. Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8:240–248–, 1969.

[39] A. M. Collins and E. F. Loftus. A spreading activation theory of semantic processing. *Psychological Review*, 82:407–428, 1975.

[40] J. R. Curran, S. Clark, and J. Bos. Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the ACL 2007 Demo and Poster Sessions*, pages 33–36, Prague, Czech Republic, 2007.

[41] C. d'Amato, N. Fanizzi, and F. Esposito. Query Answering and Ontology Population: an Inductive Approach. In M. Hauswirth, M. Koubarakis, and S. Bechhofer, editors, *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, Tenerife, Spain, June 2008. Springer Verlag.

[42] C. d'Amato, N. Fanizzi, and F. Esposito. Inductive Learning for the Semantic Web: What does it buy? *Semantic Web*, 1(1):53–59, 2010.

[43] G. De Chalendar and B. Grau. SVETLAN'or how to Classify Words using their Context. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, pages 203–216. Springer, 2000.

[44] A. Di Iorio, A. G. Nuzzolese, and S. Peroni. Identifying Functions of Citations with CiTalO. In P. Cimiano, M. Fernández, V. Lopez, S. Schlobach, and J. Völker, editors, *The Semantic Web: ESWC 2013 Satellite Events*, volume 7955 of *Lecture Notes in Computer Science*, pages 231–235. Springer Berlin Heidelberg, 2013.

[45] M. Egaña, R. Stevens, and E. Antezana. Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In *Proceedigns of OWLED 2008 DC OWL: Experiences and Directions*, Washington, DC, USA, 2008.

[46] J. Euzenat. An API for Ontology Alignment. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 698–712, Berlin, Heidelberg, November 2004. Springer.

[47] N. Fanizzi, C. d'Amato, and F. Esposito. Statistical Learning for Inductive Query Answering on OWL Ontologies. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 195–212, Karlsruhe, Germany, October 2008. Springer.

[48] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

[49] D. Fensel, C. Bussler, Y. Ding, V. Kartseva, M. Klein, M. Korotkiy, B. Omelayenko, and R. Siebes. Semantic Web application areas. In *Proc. 7th Int. Workshop on Applications of Natural Language to Information Systems (NLDB 2002)*, Stockholm, Sweden, 2002.

[50] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[51] C. Fillmore. The case for the case. In E. Bach and R. Harms, editors, *Universals in Linguistic Theory*. Rinehart and Winston, New York, 1968.

[52] C. J. Fillmore. Frame semantics and the nature of language*. *Annals of the New York Academy of Sciences*, 280(1):20–32, 1976.

[53] M. Fossati, C. Giuliano, and S. Tonelli. Outsourcing framenet to the crowd. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 742–747.

[54] K. T. Frantzi, S. Ananiadou, and J. Tsujii. The c-value/nc-value method of automatic recognition for multi-word terms. In *Research and Advanced Technology for Digital Libraries*, pages 585–604. Springer, 1998.

[55] V. Gallese and T. Metzinger. Motor ontology: the representational reality of goals, actions and selves. *Philosophical Psychology*, 16(3):365–388, 2003.

[56] P. Gamallo, M. Gonzalez, A. Agustini, G. Lopes, and V. S. De Lima. Mapping syntactic dependencies onto semantic relations. In *Proceedings of the ECAI Workshop on Machine Learning and Natural Language Processing for Ontology Engineering*, pages 15–22, 2002.

[57] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: Abstraction and reuse of object-oriented design*. Springer, 1993.

[58] A. Gangemi. Ontology Design Patterns for Semantic Web Content. In *The Semantic Web–ISWC 2005*, pages 262–276. Springer, 2005.

[59] A. Gangemi. Norms and plans as unification criteria for social collectives. *Autonomous Agents and Multi-Agent Systems*, 17(1):70–112, 2008.

[60] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In *Knowledge engineering and knowledge management: Ontologies and the semantic Web*, pages 166–181. Springer, 2002.

[61] A. Gangemi, J. Lehmann, V. Presutti, M. Nissim, and C. Catenacci. C-ODO: an OWL Meta-model for Collaborative Ontology Design. In N. F. Noy, H. Alani, G. Stumme, P. Mika, Y. Sure, and D. Vrandecic, editors, *CKC*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[62] A. Gangemi, R. Navigli, and P. Velardi. The ontowordnet project: Extension and axiomatization of conceptual relations in WordNet. In R. Meersman and Z. Tari, editors, *Proc. of On the Move to Meaningful Internet Systems (OTM2003) (Catania, Italy)*, pages 820–838. Springer-Verlag, 2003.

[63] A. Gangemi, R. Navigli, and P. Velardi. The OntoWordNet Project: extension and axiomatization of conceptual relations in WordNet. In *in WordNet, Meersman*, pages 3–7. Springer, 2003.

[64] A. Gangemi, A. G. Nuzzolese, V. Presutti, F. Draicchio, A. Musetti, and P. Ciancarini. Automatic Typing of DBpedia Entities. In *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 65–81. Springer, 2012.

[65] A. Gangemi and V. Presutti. Ontology Design Patterns. In S. Staab and R. Studer, editors, *Handbook on Ontologies, 2nd Edition*. Springer Verlag, 2009.

[66] A. Gangemi and V. Presutti. Towards a Pattern Science for the Semantic Web. *Semantic Web*, 1(1-2):61–68, 2010.

[67] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume I. River Edge, NJ: World Scientific Publishing Company, 1993.

[68] T. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[69] T. R. Gruber. Ontology. In *Encyclopedia of Database Systems*, pages 1963–1965. Springer-Verlag, 2009.

[70] M. Gruninger and M. S. Fox. The role of competency questions in enterprise engineering. In *Proc. of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, pages 83–95, Trondheim, Norway, 1994.

[71] N. Guarino. *Formal Onthology in Information Systems: Proceedings of the First International Conference (FIOS'98), June 6-8, Trento, Italy*, volume 46. IOS press, 1998.

[72] B. J. Hansen, J. Halvorsen, S. I. Kristiansen, R. Rasmussen, M. Rustad, and G. Sletten. Recommended application areas for semantic technologies. Technical report, Norwegian Defence Research Establishment (FFI), February 2010.

[73] P. Hayes. RDF Semantics. W3C recommendation, W3C, Feb. 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.

[74] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[75] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, pages 539–545, 1992.

[76] T. Heath, J. Domingue, and P. 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.

[77] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing relationships in RDF knowledge bases. In *Proceedings of the 3rd International Conference on Semantic and Media Technologies (SAMT)*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer, 2009.

[78] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A browser for the web of data. In S. Auer, S. Dietzold, S. Lohmann, and J. Ziegler, editors, *Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08)*, pages 49–58. CEUR-WS, 2008.

[79] M. Horridge and S. Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[80] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The manchester owl syntax. In *OWLED2006 Second Workshop on OWL Experiences and Directions*, Athens, GA, USA, 2006.

[81] I. Horrocks, B. Motik, and Z. Wang. The hermit owl reasoner. In I. Horrocks, M. Yatskevich, and E. Jiménez-Ruiz, editors, *ORE*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[82] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium (W3C), May 2004.

[83] A. D. Iorio, A. G. Nuzzolese, and S. Peroni. Towards the automatic identification of the nature of citations. In *SePublica*, pages 63–74, 2013.

[84] I. Jacobson, M. Griss, and P. Jonsson. *Software reuse: architecture, process and organization for business success*. ACM Press/Addison-Wesley Publishing Co., 1997.

[85] M. Kifer. Rule interchange format: The framework. In *RR*, pages 1–11, 2008.

[86] C. W. Krueger. Software Reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[87] P. Langley, J. E. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.

[88] J. Lehmann, C. Bizer, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[89] J. R. Lewis and J. Sauro. The Factor Structure of the System Usability Scale. In M. Kurosu, editor, *HCI (10)*, volume 5619 of *Lecture Notes in Computer Science*, pages 94–103. Springer, 2009.

[90] R. Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[91] W. Maass and S. Janzen. A Pattern-based Ontology Building Method for Ambient Environments . In E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. 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.

[92] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16:pp. 72–79, March-April 2001.

[93] F. Manola and E. Miller. RDF primer. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[94] G. Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, Apr. 2006.

[95] J. Martin. *Managing the data-base environment.* (The James Martin books on computer systems and telecommunications). Prentice-Hall, 1983.

[96] R. C. Martin. *Agile software development: principles, patterns, and practices.* Prentice Hall PTR, 2003.

[97] B. McBride. Jena: a semantic web toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.

[98] M. Migliore, G. Novara, and D. Tegolo. Single neuron binding properties and the magical number 7. *Hippocampus*, 18(11):1122–1130, 2008.

[99] A. Miles and S. Bechhofer. Skos simple knowledge organization system reference, Aug. 2009.

[100] G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[101] M. Minsky. A Framework for Representing Knowledge. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.

[102] R. Navigli, P. Velardi, and A. Gangemi. Ontology learning and its application to automated terminology translation. *Intelligent Systems, IEEE*, 18(1):22–31, 2003.

[103] A. Newell. The Knowledge Level. *AI Magazine*, 2(2):1–20, 33, Summer 1981.

[104] A. G. Nuzzolese, A. Gangemi, and V. Presutti. Gathering Lexical Linked Data and Knowledge Patterns from FrameNet. In *Proc. of the 6th International Conference on Knowledge Capture (K-CAP)*, pages 41–48, Banff, Alberta, Canada, 2011.

[105] A. G. Nuzzolese, A. Gangemi, V. Presutti, and P. Ciancarini. Fine-tuning triplification with Semion. In V. Presutti, V. Svatek, and F. Sharffe, editors, *Wks. on Knowledge Injection into and Extraction from Linked Data (KIELD2010)*, pages 2–14, Lisbon, Portugal, October 2010.

[106] A. G. Nuzzolese, A. Gangemi, V. Presutti, and P. Ciancarini. Encyclopedic Knowledge Patterns from Wikipedia Links. In L. Aroyo, N. Noy, and C. Welty, editors, *Proceedings fo the 10th International Semantic Web Conference (ISWC 2011)*, pages 520–536. Springer, 2011.

[107] A. G. Nuzzolese, A. Gangemi, V. Presutti, F. Draicchio, A. Musetti, and P. Ciancarini. Tìpalo: A tool for automatic typing of dbpedia entities. In P. Cimiano, M. Fernández, V. Lopez, S. Schlobach, and J. Völker, editors,

ESWC (Satellite Events), volume 7955 of *Lecture Notes in Computer Science*, pages 253–257. Springer, 2013.

[108] A. G. Nuzzolese, V. Presutti, A. Gangemi, A. Musetti, and P. Ciancarini. Aemoo: Exploring knowledge on the web. In *Proceedings of the 5th Annual ACM Web Science Conference*, pages 272–275. ACM, 2013.

[109] R. Pal. Secure Semantic Web ontology sharing. Master's thesis, University of Southampton, January 2011.

[110] G. Papamargaritis and A. Sutcliffe. Applying the domain theory to design for reuse. *BT technology journal*, 22(2):104–115, 2004.

[111] Patterns&Practices. *Microsoft Application Architecture Guide*. Microsoft Corporation, 2nd edition, 2009.

[112] S. Peroni and D. Shotton. Fabio and cito: ontologies for describing bibliographic resources and citations. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2012.

[113] V. Presutti, L. Aroyo, A. Gangemi, A. Adamou, B. A. C. Schopman, and G. Schreiber. A knowledge pattern-based method for linked data analysis. In M. A. Musen and O. Corcho, editors, *K-CAP*, pages 173–174. ACM, 2011.

[114] V. Presutti, V. K. Chaudhri, E. Blomqvist, O. Corcho, and K. Sandkuhl. *Proc. of the Workshop on Ontology Patterns (WOP 2010) at ISWC-2010 Shangai, China, November 8th, 2010*. CEUR Workshop Proceedings, 2010.

[115] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. eXtreme Design with Content Ontology Design Patterns. In E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svátek, editors, *WOP*, volume 516 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[116] V. Presutti, E. Daga, A. Gangemi, and A. Salvati. http: //ontologydesignpatterns.org [ODP]. In C. Bizer and A. Joshi, editors, *International Semantic*

Web Conference (Posters & Demos), volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[117] V. Presutti, F. Draicchio, and A. Gangemi. Knowledge extraction based on Discourse Representation Theory and linguistic frames. In *Knowledge Engineering and Knowledge Management (EKAW 2012)*, pages 114–129. Springer, 2012.

[118] V. Presutti, A. Gangemi, S. David, G. A. de Cea, M. Surez-Figueroa, E. Montiel-Ponsoda, and M. Poveda. NeOn Deliverable D2. 5.1. A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies. *NeOn Project. http://www. neon-project. org*, 2008.

[119] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, World Wide Web Consortium (W3C), January 2008.

[120] M. R. Quillian. Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities. *Behavioral Science*, 12:410–430, 1967.

[121] Rhizomik. ReDeFer. http://rhizomik.net/html/redefer, 2011. (accessed 15-02-2011).

[122] S. Rudolph. Acquiring generalized domain-range restrictions. *Formal Concept Analysis*, pages 32–45, 2008.

[123] J. Ruppenhofer, M. Ellsworth, M. R. L. Petruck, C. R. Johnson, and J. Scheffczyk. FrameNet II: Extended Theory and Practice. http://framenet.icsi.berkeley.edu/book/book.html, 2006.

[124] J. Sauro. *A practical guide to the system usability scale: Background, benchmarks & best practices*. Measuring Usability LCC, 2011.

[125] F. Scharffe and D. Fensel. Correspondence patterns for ontology alignment. In *Knowledge Engineering: Practice and Patterns*, pages 83–92. Springer, 2008.

[126] G. Schreiber, M. van Assem, and A. Gangemi. RDF/OWL Representation of WordNet. W3C Working Draft, W3C, June 2006. http://www.w3.org/TR/2006/WD-wordnet-rdf-20060619/.

[127] K. K. Schuler. *VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon*. PhD thesis, University of Pennsylvania, 2006.

[128] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[129] D. Shotton. Semantic publishing: the coming revolution in scientific journal publishing. *Learned Publishing*, 22(2):85–94, 2009.

[130] I. Sommerville and P. Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[131] J. F. Sowa. Conceptual structures: information processing in mind and machine. 1983.

[132] F. Suchanek, G. Kasneci, and G. Weikum. Yago - A Large Ontology from Wikipedia and WordNet. *Elsevier Journal of Web Semantics*, 6(3):203–217, 2008.

[133] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, pages 697–706, New York, NY, USA, 2007. ACM Press.

[134] O. Sváb-Zamazal, V. Svátek, and F. Scharffe. Pattern-based ontology transformation service. In *KEOD*, pages 42–47, 2009.

[135] S. Teufel, A. Siddharthan, and D. Tidhar. Automatic classification of citation function. In *EMNLP '06: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 103–110, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[136] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker. Sig.ma: Live views on the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):355 – 364, 2010. Semantic Web Challenge 2009, User Interaction in Semantic Web research.

[137] K. Viljanen, J. Tuominen, E. Mäkelä, and E. Hyvönen. Normalized Access to Ontology Repositories. In *ICSC*, pages 109–116, 2012.

[138] J. Völker. *Learning expressive ontologies*, volume 2. IOS Press, 2009.

[139] J. Völker and M. Niepert. Statistical Schema Induction. In *Proc. of the Eighth Extended Semantic Web Conference (ESWC2011), Part I*, pages 124–138. Springer, 2011.

[140] J. Völker and S. Rudolph. Lexico-logical acquisition of OWL DL axioms. In *Formal Concept Analysis*, pages 62–77. Springer, 2008.

[141] D. Vrandečić. *Ontology evaluation*. Springer, 2009.

[142] Y. Wang, D. J. DeWitt, and J.-Y. Cai. X-Diff: An effective change detection algorithm for XML documents. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 519–530. IEEE, 2003.

[143] P. Wolfgang. *Design patterns for object-oriented software development*. Reading, Mass.: Addison-Wesley, 1994.

[144] D. Wood. The state of RDF and JSON. `http://www.w3.org/blog/SW/2011/09/13/the-state-of-rdf-and-json/`.

[145] S.-H. Wu and W.-L. Hsu. Soat: a semi-automatic domain ontology acquisition tool from chinese corpus. In *Proceedings of the 19th international conference on Computational linguistics-Volume 2*, pages 1–5. Association for Computational Linguistics, 2002.

[146] Z. Zhang, A. L. Gentile, E. Blomqvist, I. Augenstein, and F. Ciravegna. Statistical knowledge patterns: Identifying synonymous relations in large linked

datasets. In H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 703–719. Springer, 2013.