FRAME-BASED
ONTOLOGY POPULATION FROM TEXT:
MODELS, SYSTEMS, AND APPLICATIONS

Francesco Corcoglioniti

Adviser
Dr. Luciano Serafini
Fondazione Bruno Kessler-IRST

Committee
Prof. Dr. Paolo Bouquet
Università degli Studi di Trento

Prof. Dr. Philipp Cimiano
Universität Bielefeld

Co-Advisor
Dr. Marco Rospocher
Fondazione Bruno Kessler-IRST

Prof. Dr. Piek Th. J. M. Vossen
Vrije Universiteit Amsterdam

April 2016
Abstract

Ontology Population from text is an interdisciplinary task that integrates Natural Language Processing (NLP), Knowledge Representation, and Semantic Web techniques to extract assertional knowledge from texts according to specific ontologies. As most information on the Web is available as unstructured text, Ontology Population plays an important role in bridging the gap between structured and unstructured data, thus helping realizing the vision of a (Semantic) Web where contents are equally consumable by humans and machines.

In this thesis we move beyond Ontology Population of instances and binary relations, and focus on (what we call) Frame-based Ontology Population, whose target is the extraction of semantic frames from text. Semantic frames are defined by RDFS/OWL ontologies, such as FrameBase and the Event Situation Ontology derived from FrameNet, and consist in events, situations and other structured entities reified as ontological instances (e.g., a sell event) and connected to related instances via properties specifying their semantic roles in the frame (e.g., seller, buyer). This representation (called neo-Davidsonian) supports expressing n-ary and arbitrarily qualified relations, and permits leveraging complex NLP tasks such as Semantic Role Labeling (SRL), which annotates frame-like structures in text consisting of predicates and their semantic arguments as defined by domain-general predicate models.

We contribute to the task of Frame-based Ontology Population from multiple directions. We start with developing an extension of the Lemon lexicon model for ontologies (PreMOn) to represent predicate models—PropBank, NombBank, VerbNet, and FrameNet—and their mappings to FrameBase. Based on this, our core contribution is a Frame-based Ontology Population approach (PIKES) where processing is decoupled in two phases: first, an English text is processed by a SRL-based NLP pipeline to extract mentions, i.e., snippets of text denoting entities or facts; then, mentions are processed by mapping rules to extract ontological instances aligned to DBpedia and Yago, and semantic frames aligned to FrameBase. We represent all the contents involved in this process in RDF with named graphs, according to an ontological model (KEM) built on top of the semiotic notions of meaning and reference, aligned to DOLCE and the NLP Interchange Format (NIF) ontologies. The model allows navigating from any piece of extracted knowledge to its mentions and back, and allows representing all the generated intermediate information (e.g., NLP annotations) and associated metadata (e.g., confidence, provenance). Based on this model, we propose a scalable system (KnowledgeStore) for storing and querying all the text, mentions, and RDF data involved in the population process, together with relevant RDF background knowledge, so that they can be jointly accessed by applications. Finally, to support the necessary RDF processing tasks, such as rule evaluation, RDFS and owl:sameAs inference, and data filtering and integration, we propose a tool (RDFpro) implementing a simple, non-distributed processing model combining streaming and sorting techniques in complex pipelines, capable of processing billions of RDF triples on a commodity machine.
We describe the application of these solutions for processing differently scoped-sized datasets within and outside the NewsReader EU Project, and for improving search performances in Information Retrieval, through an approach (KE4IR) that enriches the term vectors of documents and queries with semantic terms obtained from extracted knowledge.

All the proposed solutions were implemented and released open-source with demonstrators, and ontological models were published online according to Linked Data best practices. The results obtained were validated via empirical performance evaluations and case studies.

**Keywords:**
Ontology Population, Semantic Frames, Ontologies, Predicate Models, Semantic Web
Acknowledgments

This thesis was made possible by its framing in two research projects providing guidance and usage scenarios—LiveMemories and NewsReader—and by the support of many people.

First and foremost, I want to thank my PhD advisor and co-advisor. Dr. Luciano Serafini, my advisor, has shown and helped me to appreciate and adopt the mindset, rigor, and commitment qualities required for conducting quality research. Dr. Marco Rospocher has been a very patient and supportive co-advisor. We had countless meetings to brainstorm and deeply analyze the research problems and solutions of this thesis, where his pragmatism and clarity of thought were crucial in identifying the most effective ways to proceed. By working with him every day, in NewsReader and as papers co-authors, I had the opportunity to learn by example many of the skills expected from a researcher.

Many outcomes of this thesis result from collaborative work and the expertise, intuitions, and contributions of many other colleagues. I thank Dr. Alessio Palmero Aprosio for his work on PIKES and PreMOn, and for having made available his expertise in Natural Language Processing. I thank Dr. Mauro Dragoni for having sparked the idea of applying PIKES in Information Retrieval, one of his fields of expertise, and for his help in developing further the idea. I thank Prof. Dr. Bernardo Magnini and Dr. Roldano Cattoni for their initial vision of the KnowledgeStore, which I helped building in LiveMemories first and NewsReader later, also supported by Drs. Renato Marroquín Mogrovejo and Mohammed Qwaider. I thank Drs. Marco Amadori and Michele Mostarda for helping me with RDFpro, and Dr. Sara Tonelli for her investigation of several linguistic issues related to PreMOn.

During my PhD, I also had interactions with other people that positively impacted my work. I thank (in alphabetical order) Drs. Loris Bozzato, Chiara Di Francescomarino, Christian Girardi, Martin Homola, Chiara Ghidini, Octavian Popescu, and Roberto Zanoli, who involved me in their activities and in the co-authoring of several papers, allowing me to learn a number of “tricks” on how to conduct and publish research. For their feedback, interesting discussions, technical advice, and support in any form, I want to thank (in alphabetical order) Drs. Volha Bryl, Gaetano Calabrese, Irene Celino, Claudia d’Amato, Emanuele Della Valle, Riccardo De Masellis, Ivan Donadello, Marco Fossati, Mathew Joseph, Muhammad Tahir Kahn, Massimo Legnani, Radim Nedbal, Giulio Petrucci, Irina Sergiienza, Manuela Speranza, Rachele Sprugnoli, Andrei Tamilin, Stefano Teso, and Roberto Tiella. I want also to thank our internship students Gianluca Apriceno, Enrico Magnago, and Alessio Scussel, for their extensions of the KnowledgeStore and PreMOn that allowed me to better appraise their benefits, limitations, and potential.

I acknowledge the partial funding of my PhD by Fondazione Bruno Kessler, who hosted me in the DKM group for the whole duration of the PhD program. My gratitude to the members of the defense committee for their reviewing work and for coming to Trento: Prof. Dr. Piek Th. J. M. Vossen, Prof. Dr. Philipp Cimiano, and Prof. Dr. Paolo Bouquet.
# Contents

Abstract ................................. i
Acknowledgments .................... iii
Contents ................................ iv
List of Tables ......................... vii
List of Figures ......................... viii
List of Abbreviations ................ x

1 Introduction .......................... 1
   1.1 Context ................................ 1
   1.2 Problem ................................ 4
   1.3 Contributions ....................... 5
   1.4 Structure of the Thesis ........... 8
   1.5 Publications ....................... 9
   1.6 Artifacts ............................ 10

2 Background ........................... 11
   2.1 Semantic Web ....................... 11
      2.1.1 RDF ................................ 11
      2.1.2 SPARQL ............................ 12
      2.1.3 RDFS and OWL Ontologies ....... 12
      2.1.4 Linked Open Data ................ 14
   2.2 Linguistic Resources .............. 15
      2.2.1 WordNet ......................... 15
      2.2.2 Predicate Models ............... 16
   2.3 Natural Language Processing .... 18
   2.4 Linguistic Ontologies ............. 19
   2.5 Frame-based Ontologies .......... 20

3 PreMOn: Predicate Models and FrameBase Mappings in RDF/OWL .............. 23
   3.1 The PreMOn Ontology – Core Module .......... 24
      3.1.1 Semantic Classes and Roles ........ 25
      3.1.2 Mappings .......................... 25
      3.1.3 Annotations ...................... 26
   3.2 The PreMOn Ontology – Specialized Modules .... 27
      3.2.1 PropBank Module ................. 27
      3.2.2 NomBank Module .................. 28
      3.2.3 VerbNet Module ................... 29
      3.2.4 FrameNet Module ................. 30
   3.3 The PreMOn Dataset .................. 31
   3.4 FrameBase Mappings ............... 34
      3.4.1 Mapping FrameNet to FrameBase .... 34
      3.4.2 Mapping PropBank and NomBank to FrameBase 35
   3.5 Related Work ...................... 37
List of Tables

2.1 Comparison of predicate models. ......................................................... 16
3.1 PreMOn Dataset: predicate model statistics. ........................................ 32
3.2 PreMOn Dataset: mapping statistics. ....................................................... 32
3.3 FrameBase mapping statistics ............................................................. 35
5.1 Association between NLP tasks and semantic annotations of mentions. ........ 57
5.2 Sentences of the gold standard .......................................................... 65
5.3 Precision/recall of FrameBase ontology population with different SRL setups. .. 68
5.4 Precision/recall of ontology population with respect to predicate models. ........ 69
5.5 Precision/recall of ontology population with respect to predicate models, against FRED. 70
5.6 PIKES precision on Simple English Wikipedia knowledge graph. ................ 71
5.7 PIKES processing time grows linearly with the number of tokens. .............. 71
6.1 KnowledgeStore population time and rate based on number of mentions per news. ... 87
6.2 KnowledgeStore population rate based on amount of data already stored. ........ 88
6.3 KnowledgeStore retrieval evaluation: test datasets. ................................... 89
6.4 KnowledgeStore retrieval evaluation: parametric requests. ......................... 90
7.1 RDFpro dataset analysis scenario: results. ............................................ 106
7.2 RDFpro dataset filtering scenario: results. .......................................... 108
7.3 RDFpro dataset merging scenario: results. .......................................... 109
8.1 Terms extracted by KE4IR from query “astronomers influenced by Gauss.” ......... 120
8.2 Comparison of KE4IR against the baselines. ......................................... 126
8.3 KE4IR results using different layer combinations. ..................................... 126
8.4 NDCG@10 and MAP of textual plus semantic layer against textual, query-by-query. 127
8.5 KE4IR results for selected queries using all the semantic layers. .................. 128
9.1 Overview of KnowledgeStore instances deployed in NewsReader .................. 138
9.2 DBpedia background knowledge datasets: numbers of triples. ....................... 147
9.3 DBpedia background knowledge datasets: numbers of instances. .................... 148
9.4 DBpedia background knowledge: processing statistics. ............................. 148
A.1 KnowledgeStore retrieval evaluation: request parameters (with examples). ......... 170
List of Figures

1.1 Semantic frame structure. ................................................................. 3
1.2 Example of extracting frame-based ontological knowledge from a sentence. .... 5
1.3 Thesis contributions. ................................................................. 8

3.1 PreMOn Ontology: Core module. .................................................. 24
3.2 Example of predicate model data represented using the PreMOn Ontology. .... 24
3.3 PreMOn Ontology: NomBank module. ................................................ 28
3.4 PreMOn Ontology: PropBank module. ................................................ 28
3.5 PreMOn Ontology: FrameNet module. ................................................ 28
3.6 PreMOn Ontology: VerbNet module. ................................................ 29
3.7 PreMOn Ontology: VerbNet module (restrictions). .................................. 29
3.8 Example of SPARQL query on PreMOn Dataset. .................................. 33
3.9 Graphical depiction of PreMOn Dataset including FrameBase mappings. ....... 35

4.1 KEM Core with its three representation layers: Resource, Mention, and Instance .... 40
4.2 Representing triple-level metadata using named graphs. ............................ 45
4.3 KEM Text. ........................................................................................ 46
4.4 Using KEM to represent a sentence, its SRL annotations, and extracted knowledge .... 51

5.1 Knowledge extracted from the running example. ........................................ 55
5.2 Knowledge distillation via mapping rules. ................................................ 58
5.3 PIKES architecture. ............................................................................ 62
5.4 PIKES web user interface. ...................................................................... 64
5.5 Bar plot for Table 5.3 (FrameBase ontology population). ................................ 68
5.6 Bar plot for Table 5.4 (ontology population with respect to predicate models). ... 69
5.7 Bar plot for Table 5.5 (comparison with FRED, graph $G'$). ....................... 70

6.1 KnowledgeStore interactions with external modules. .................................. 78
6.2 KnowledgeStore data model, highlighting its three layers and the configurable part .... 79
6.3 Invocation of a CRUD retrieve operation via the KnowledgeStore ReST API ... 81
6.4 KnowledgeStore UI: (a) resource lookup; (b) mention lookup; (c) SPARQL query. 82
6.5 KnowledgeStore UI reports. .............................................................. 84
6.6 KnowledgeStore Architecture. ........................................................... 85
6.7 KnowledgeStore retrieval evaluation: (a) throughput; (b) average evaluation time ... 91
6.8 KnowledgeStore retrieval evaluation: average time of each parametric request. .... 92

7.1 RDFpro processing model. .................................................................... 99
7.2 Using RDFpro: (a) command line; (b) web interface. ............................... 104
7.3 RDFpro dataset analysis scenario: pipeline. ............................................ 106
7.4 RDFpro dataset analysis scenario: TBox and statistics visualization in Protégé. ... 107
7.5 RDFpro dataset filtering scenario: pipeline. ............................................ 108
7.6 RDFpro dataset merging scenario: pipeline. ........................................... 109

8.1 KE4IR implementation: (a) term extraction; (b) indexing, searching and scoring. 123
8.2 Trends of (a) NDCG@10 and (b) MAP based on the weight of semantic information. 130
9.1 KnowledgeStore interaction with other NewsReader components. .................. 134
9.2 KnowledgeStore data model configured for the NewsReader scenarios. ............. 135
9.3 Example of SPARQL query (a) with and (b) without smushing and RDFS inference. 145
9.4 DBpedia background knowledge datasets: browsing statistics in Protégé. ............. 150

10.1 Thesis contributions and future works. .................................................. 153
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL</td>
<td>Entity Linking</td>
</tr>
<tr>
<td>ESO</td>
<td>Event Situation Ontology (Segers et al., 2015)</td>
</tr>
<tr>
<td>FN</td>
<td>FrameNet (Baker et al., 1998)</td>
</tr>
<tr>
<td>GAF</td>
<td>Ground Annotation Framework (Fokkens et al., 2013)</td>
</tr>
<tr>
<td>IE</td>
<td>Information Extraction</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>ITS</td>
<td>Internationalization Tag Set (Filip et al., 2013)</td>
</tr>
<tr>
<td>KE</td>
<td>Knowledge Extraction</td>
</tr>
<tr>
<td>KEM</td>
<td>Knowledge Extraction Model (Chapter 4)</td>
</tr>
<tr>
<td>KR</td>
<td>Knowledge Representation</td>
</tr>
<tr>
<td>lemon</td>
<td>lexical model for ontologies (McCrae et al., 2012)</td>
</tr>
<tr>
<td>LMM</td>
<td>Linguistic Meta-Model (Picca et al., 2008)</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>MAP</td>
<td>Mean Average Precision</td>
</tr>
<tr>
<td>NAF</td>
<td>NLP Annotation Format (Fokkens et al., 2014)</td>
</tr>
<tr>
<td>NB</td>
<td>NomBank (Meyers et al., 2004)</td>
</tr>
<tr>
<td>NDCG</td>
<td>Normalized Discounted Cumulated Gain (Järvelin and Kekäläinen, 2002)</td>
</tr>
<tr>
<td>NERC</td>
<td>Named Entity Recognition and Classification</td>
</tr>
<tr>
<td>NIF</td>
<td>NLP Interchange Format (Hellmann et al., 2013)</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>OLiA</td>
<td>Ontologies of Linguistic Annotation (Chiarcos and Sukhareva, 2015)</td>
</tr>
<tr>
<td>PB</td>
<td>PropBank (Palmer et al., 2005)</td>
</tr>
<tr>
<td>PIKES</td>
<td>PIKES is a Knowledge Extraction Suite (Chapter 5)</td>
</tr>
<tr>
<td>KE4IR</td>
<td>Knowledge Extraction for Information Retrieval (Chapter 8)</td>
</tr>
<tr>
<td>PreMOn</td>
<td>Predicate Model for Ontologies (Chapter 3)</td>
</tr>
<tr>
<td>RDFpro</td>
<td>RDF Processor (Chapter 7)</td>
</tr>
<tr>
<td>SKOS</td>
<td>Simple Knowledge Organization System (Miles and Bechhofer, 2009)</td>
</tr>
<tr>
<td>SRL</td>
<td>Semantic Role Labeling</td>
</tr>
<tr>
<td>SW</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>TERN</td>
<td>Temporal Expression Recognition and Normalization</td>
</tr>
<tr>
<td>VN</td>
<td>VerbNet (Kipper Schuler, 2005)</td>
</tr>
<tr>
<td>VOID</td>
<td>Vocabulary of Interlinked Datasets (Alexander et al., 2009)</td>
</tr>
<tr>
<td>VSM</td>
<td>Vector Space Model (Salton et al., 1975)</td>
</tr>
<tr>
<td>WSD</td>
<td>Word Sense Disambiguation</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

This thesis contributes to the Knowledge Extraction research area, and tackles the extraction, representation, and storage of assertional knowledge from English texts expressed using semantic frames. In this chapter, we introduce the context of the thesis (Section 1.1), delimit the addressed problem (Section 1.2), and summarize the proposed solution and the thesis contributions (Section 1.3). We also describe the thesis structure (Section 1.4), and list the publications supporting the thesis (Section 1.5) and the artifacts (software, ontologies, datasets) developed as part of it (Section 1.6).

Acknowledgments  The thesis includes material resulting from collaborative work or described in previous publications, which is acknowledged at the beginning of each chapter.

1.1 Context

With Semantic Web (SW) technologies coming of age and the public acclaim of the Linked Open Data (LOD) initiative, the last years have seen a massive proliferation of structured data,\footnote{For instance, see the fast growing rate of the LOD cloud (http://youtu.be/TXFYSWuE00w), or the number of datasets currently available on DataHub (http://datahub.io/).} both on the Web and within organizations. Nonetheless, the majority of information remains available only in unstructured form and, as observed by Gantz and Reinsel (2011), unstructured data still accounted for more than 90% of the digital universe in 2011. While different in form, both unstructured and structured information sources provide information about entities of the world (e.g., persons, organizations, locations, events), their properties, and relations among them. Indeed, coinciding, contradictory, and complementary facts about these entities could be available in structured form, unstructured form, or both, and being able to leverage both kinds of content is a requirement for applications needing access to “complete” knowledge (e.g., for supporting potentially critical user decisions).

Information Extraction (IE) (Grishman, 2010; Weikum and Theobald, 2010) aims at bridging the gap between structured and unstructured contents, by developing techniques for the automatic extraction of structured data out of unstructured or semi-structured documents. Knowledge Extraction (KE)\footnote{http://en.wikipedia.org/wiki/Knowledge_extraction} builds on IE and goes one step further, targeting the extraction of assertional (ABox, i.e., instances and facts) and/or terminological (TBox, i.e., classes and properties) formal knowledge from unstructured contents, anchoring it to existing ontologies and vocabularies. KE from natural language text, a kind of unstructured content, is also referred to as Ontology Learning and Population (Cimiano, 2006), with
Ontology Learning targeting the extraction of TBox knowledge, and Ontology Population targeting the extraction of ABox knowledge according to specific ontologies, and typically disambiguating extracted instances with respect to well-known LOD knowledge bases such as DBpedia (Lehmann et al., 2015). KE from text is a challenging interdisciplinary task standing at the crossroad of Natural Language Processing (NLP), Knowledge Representation (KR), and SW. This task has gained momentum in the last ten years thanks to the growing maturity of the SW field: being able to efficiently and effectively extract knowledge from text contributes to address the vision of a (Semantic) Web that could be equally consumed both by humans and machines.

State-of-the-art approaches for KE, such as FRED (Presutti et al., 2012), LODifier (Augenstein et al., 2012), Graphia (Freitas et al., 2013), and NewsReader (Rospocher et al., 2016), typically exploit a mix of NLP, SW, and LOD resources and techniques. NLP is often performed by reusing existing NLP tools as they are, whose outputs, generally consisting of annotated pieces of text, are then reinterpreted under a more “semantic” perspective by applying SW resources (e.g., mappings from linguistic categories to ontological concepts) and techniques (e.g., reasoning), taking care of representation problems that are generally irrelevant in the NLP field (e.g., the proper ontological modeling of certain relations, or the choice of the most appropriate vocabulary for a certain kind of extracted information).

KE approaches in the literature differ under different points of view:

- The target is either Ontology Learning, Ontology Population, or both. Approaches of the latter type are similar in spirit to Open IE (Etzioni et al., 2008) that extract facts without a predefined schema, which in this case can be seen as built incrementally as (the lexicalizations of) new ontological types and properties are found in the text, whereas in a pure Ontology Population system the TBox is known in advance.

- The structures extracted are different, ranging from plain binary relations to semantic frames (Figure 1.1). The latter consist in n-ary relations, events, situations, and other structured entities reified as ontological instances (e.g., a sell event) and connected to related instances via properties specifying their semantic roles in the frame (e.g., seller or buyer). This kind of representation is called neo-Davidsonian (Parsons, 1990) and we refer to approaches extracting semantic frames as Frame-based approaches. The schema of semantic frames can be formalized by RDFS/OWL ontologies, such as FrameBase (Rouces et al., 2015) and ESO (Event Situation Ontology, Segers et al., 2015) recently derived from FrameNet (Baker et al., 1998).

- The core NLP tasks approaches for KE rely on are different, including Relation Extraction, the extraction of Discourse Representation Structures (DRS, Kamp and Reyle, 1993), and SRL (Semantic Role Labeling). DRS provides a logical-like representation of text, with verb and noun predicates mapped to logical predicates, and variables standing for entity and event instances. This representation is closer to knowledge than other NLP outputs, thus simplifying the job of KE approaches
operating on it; on the other hand, such approaches end up being closely tied to the specific tools providing such representation. SRL allows annotating frame-like structures in text consisting of predicates and their semantic arguments, as defined by domain-general predicate models such as FrameNet. While these predicate models are far from being considered formal ontologies (Ovchinnikova et al., 2010), they have nevertheless provided the basis for some frame-based ontologies (e.g., FrameBase), and a natural correspondence can be drawn between their predicates and roles, and the ontological classes and properties of frame-based ontologies.

Several ontologies related to KE have been proposed in recent years. LMM (Linguistic Meta-Model, Picca et al., 2008) extends the upper ontology DOLCE (Gangemi et al., 2002) to model fragments of text, the ontological instances they refer to, and a meaning concept linking them, this way providing an ontological realization of the semiotic triangle of meaning. While LMM is rather abstract (being close to DOLCE), NIF (NLP Interchange Format, Hellmann et al., 2013) provides a specific vocabulary and URI design for identifying and annotating arbitrary substrings of text within a larger text (e.g., a sentence, a document). Finally, the lemon ontology by McCrae et al. (2012), recently standardized by the W3C Ontology Lexicon Community Group (Cimiano et al., 2015), allows mapping lexical entries to the ontological concepts they denote, operating at a type level rather than the token (i.e., occurrence) level of NIF. All these models, together with the vocabularies further specializing them, play a role in KE: LMM provides an abstract framework for tracking the provenance of extracted knowledge down to the pieces text it derives from, often an important requirement of KE applications; NIF provides a concrete way to identify theses pieces of text; and lemon supports the encoding of the mappings from text to ontological concepts possibly used in the extraction process.

Being capable of relating text to knowledge, KE opens the way for seamlessly integrating text and structured knowledge in a single repository, so to provide a unified, complete view over all the available structured and unstructured knowledge to applications requiring it, which is particularly important as source texts will always contain more information than what can be automatically extracted. To that respect, state-of-the-art approaches such as KIM (Popov et al., 2003), Stanbol (Gönül and Sinaci, 2012), and LMF (Linked Media Framework, Kurz et al., 2014) typically adopt a ‘two-layer’ representation and storage model, covering both text with its metadata and the instances and facts (RDF triples) extracted from it. While storing NLP annotations and other intermediate results can be useful, e.g., to debug a KE systems, it is not a goal of these frameworks.
1.2 Problem

Within the KE landscape, in this thesis we focus on the task of Frame-based Ontology Population from English text. We deem this task of practical interest for several reasons:

- by focusing on Ontology Population only (without introducing new TBox concepts as needed) it simplifies the use of extracted knowledge as its TBox is known in advance;
- by considering semantic frames (rather than plain binary relations), it allows representing complex structures and relations for which domain-general ontologies, such as FrameBase and ESO, are available; and
- by targeting the extraction of more complex structures—again, semantic frames—it acknowledges and allows leveraging the maturity level recently reached in complex NLP tasks such as SRL, for which there are now practically usable tools, comprehensive predicate models, and annotated training data (also for languages different from the English one considered here).

We address this task from a point of view closer to SW and KR, focusing on the extraction of quality knowledge and its proper ontological representation more than on the NLP machinery involved in the process, for which we target the reuse of standard NLP components, relying on their continuous evolution and improvement as a result of ongoing NLP research. Specifically, we tackle three aspects of the considered task that we deem equally important for applications (the latter two are actually generalizable to any KE task and not just Frame-based Ontology Population):

1. Extraction approach. The problem here is mapping the output of NLP tools (generally expressed in terms of strings of text) to ontological instances and assertions. Using a single NLP tool is not sufficient and, depending on the core NLP technique chosen, additional NLP tasks have to be integrated, e.g., in order to: locate instances in the text; disambiguate them with respect to external knowledge resources; identify coreferring expressions denoting the same instance; type instances according to well-known ontologies; and so on. An example of this process is shown in Figure 1.2, where SRL is complemented with other NLP analyses and the obtained linguistic categories and relations are mapped to ontological classes and properties. Integrating many NLP tasks in a coherent process, keeping it manageable and extensible (e.g., for replacing the NLP tool for a given task, or integrating new tasks) is not straightforward.

2. Data representation. The problem here is defining an ontological representation model, possibly building on prior works such as LMM and NIF, for the joint representation of the input text, its metadata, the associated NLP annotations, the knowledge extracted from it (instances and triples), and the relations linking all of them. Such a model is a requirement for KE applications needing to keep track of the provenance of each piece of extracted knowledge, and may also help better structuring the extraction process.
"G. W. Bush and Bono are very strong supporters of the fight of HIV in Africa."

Figure 1.2. Example of extracting frame-based ontological knowledge from a sentence, combining SRL with other NLP analyses.

3. **Data storage and access.** The problem here is supporting applications in jointly storing and accessing all the data involved in KE, including NLP annotations and not “just” text and knowledge, and providing the necessary scalability (e.g., up to millions of documents and billions of RDF triples) and data access and manipulation methods. Such a solution is required by applications needing complete access to all the structured and unstructured knowledge in KE, with the explicit storage of NLP annotations supporting the debugging and investigation of adopted KE techniques.

1.3 **Contributions**

The core contribution of the thesis is PIKES (PIKES is a Knowledge Extraction Suite), an approach for Frame-based Ontology Population where an English text is first processed.
by an NLP pipeline, implemented on top of SRL tools such as Semafor (Das et al., 2014) and Mate-tools (Björkelund et al., 2009), to extract an RDF graph of mentions, which is then processed via mapping rules to generate instances and semantic frames. Instances are aligned to DBpedia, YAGO (Hoffart et al., 2013), and SUMO (Niles and Pease, 2001), while semantic frames populate the FrameBase ontology, chosen because it provides a domain-general, broad-coverage, SW-ready inventory of semantic frames.

To realize PIKES and address the data representation problem, we realized two ontological models. The first model, KEM (Knowledge Extraction Model), builds on the semiotic notions of reference and meaning and their modeling in DOLCE (as well as LMM and NIF) to support a single, layered RDF/OWL representation of all the contents manipulated in PIKES and more generally involved in KE: source texts with their metadata in the Resource layer; extracted instances and triples in the Instance layer; and mentions of instances and triples with associated NLP annotations in the Mention layer, which enables navigating from each piece of knowledge to the pieces of texts it originates from and back. The second model, PreMOn (Predicate Model for Ontologies), extends lemon for representing predicate models and their mappings, covering FrameNet, PropBank (Palmer et al., 2005), NomBank (Meyers et al., 2004), VerbNet (Kipper Schuler, 2005), and SemLink (Palmer, 2009), and providing the mappings used in PIKES from the first three models (the ones used by Semafor and Mate-tools) to FrameBase.

To address the data storage problem, we built the KnowledgeStore, a dedicated, SW-grounded scalable system for storing and querying all the data involved in a KE process (text resources, mentions, instances and relevant background knowledge), represented and interlinked according to KEM. The KnowledgeStore integrates a triplestore for managing Instance layer data, and a scalable, distributed HBase\(^3\) and Hadoop\(^4\) HDFS infrastructure for storing Resource and Mention layer contents, providing ReST APIs for accessing all this data as well as a simple user interface.

Both PIKES and the KnowledgeStore—and more generally other KE systems operating on RDF data—require performing several RDF processing tasks possibly on large datasets, such as inference materialization, rule evaluation, and owl:sameAs smushing\(^5\) (i.e., merging together coreferring URIs). Urged by the practical need of solving these tasks, we also investigated a simple model for non-distributed RDF processing based on streaming and sorting techniques and the arbitrary (parallel, sequential) composition of processing tasks in complex pipelines, leading us to the realization of RDFpro (RDF Processor), a tool capable of processing billions of RDF triples on a single commodity machine.

Finally, among the possible applications of the approaches developed in this thesis, we investigated the use of Frame-based Ontology Population to enhance search performances in Information Retrieval (IR). This led us to propose KE4IR (Knowledge Extraction for Information Retrieval), an IR approach where documents and queries are processed with

---

\(^3\)http://hbase.apache.org/
\(^4\)http://hadoop.apache.org/
\(^5\)http://patterns.dataincubator.org/book/smushing.html
PIKES and augmented with URI, TYPE, FRAME, and TIME semantic terms, which are then exploited together with TEXTUAL terms to compute the similarity between a document and a query using an adaptation of VSM (Vector Space Model, Salton et al., 1975).

The activities of this thesis were conducted in the same period and in collaboration with the NewsReader\(^6\) EU project, whose aim was to extract events from large news corpora, in order to help decision makers and other users in making sense of news contents. NewsReader has been a valuable source of ideas, real world requirements, and large datasets for this thesis. At the same time, many ideas and results developed in the thesis (in particular, the KnowledgeStore and RDFpro systems) found application in NewsReader, this way contributing to the project and allowing the acquisition of important feedback about the practical applicability of developed techniques.

All the proposed solutions have been implemented and released open-source with demonstrators, and ontological models have been published online according to LOD best practices. The results obtained have been validated via empirical performance evaluations and case studies, within and outside the scope of NewsReader.

Summing up, this thesis advances the state of the art with six contributions reported in Figure 1.3 and listed below (in parenthesis, chapters and relevant publications):

**Contribution C1 – PreMOn** An ontological model for representing predicate models and their mappings, based on lemon and covering PropBank, NomBank, VerbNet, FrameNet, SemLink, and FrameBase (Corcoglioniti et al., 2016b, Chapter 3).

**Contribution C2 – KEM** An ontological model for representing all the contents involved in KE along three layers: Resource, Mention, and Instance (Chapter 4).

**Contribution C3 – PIKES** A 2-phase, SRL-based Frame-Based Ontology Population approach for English, extracting FrameBase semantic frames and instances aligned to DBpedia, YAGO, and SUMO (Corcoglioniti et al., 2016c, 2015e, Chapter 5).

**Contribution C4 – KnowledgeStore** A SW-grounded scalable system for storing all the Resource, Mention, and Instance data described and interlinked according to KEM (Corcoglioniti et al., 2015c, 2013; Rospocher et al., 2014a,b, Chapter 6).

**Contribution C5 – RDFpro** A tool for non-distributed processing of large amounts of RDF data, based on streaming, sorting, and the arbitrary composition of processing tasks in complex pipelines (Corcoglioniti et al., 2015d,a, 2014, Chapter 7).

**Contribution C6 – KE4IR** An IR approach using PIKES to augment documents and queries with URI, TYPE, FRAME, and TIME semantic terms, which are indexed and matched using an adaptation of VSM (Corcoglioniti et al., 2016a, Chapter 8).

\(^6\)[http://www.newsreader-project.eu/](http://www.newsreader-project.eu/)
1.4 Structure of the Thesis

The remainder of the thesis is structured as follows:

**Chapter 2** provides some background on the concepts, systems, and resources referenced throughout the thesis. Note that related work specific to each contribution of the thesis is discussed at the end of the respective chapter.

The next two chapters present the two ontological resources built to support Ontology Population from text, covering Contributions C1 and C2 in the list of Section 1.3:

**Chapter 3** describes PreMOn, presenting its core module, the specializations for the various predicate models, the PreMOn Dataset and its detailed statistics, and an account of how mappings to FrameBase were generated.

**Chapter 4** describes KEM and its representation layers: Resource, Mention, and Instance.

The following three chapters describe the three systems developed and evaluated as part of this thesis, which represent Contributions C3, C4, and C5 in the list of Section 1.3:

**Chapter 5** describes PIKES, presenting its 2-phase approach based on KEM and the PreMOn FrameBase mappings, its implementation, and the three evaluations conducted to assess precision/recall on a small gold standard, and throughput and sampled precision on a larger Wikipedia-like corpus.

**Chapter 6** describes the KnowledgeStore, presenting its data model based on KEM, its architecture, ReST and user interfaces, and the experiments conducted to evaluate the data population and retrieval performances of the system on real data.
Chapter 7 describes RDFpro and its evaluation in four scenarios to assess whether, using the proposed processing, the processing tasks considered by the tool can be feasibly performed on a single commodity machine on large RDF datasets.

The next two chapters present applications of the previous results, the first of them, KE4IR, representing Contribution C6 of the list of Section 1.3:

Chapter 8 describes KE4IR, building on PIKES and presenting the approach, implementation, and evaluation performed on a recent dataset that shows how using semantic information coming from a KE system like PIKES can improve IR performances.

Chapter 9 describes the use in NewsReader of the KnowledgeStore, deployed in multiple instances, and RDFpro, used for post-processing extracted RDF data and preparing the necessary DBpedia background knowledge datasets.

Finally, and based on the findings reported in previous chapters:

Chapter 10 summarizes the results of the thesis, discussing limitations, possible research directions, and some engineering consideration concerning the work presented.

1.5 Publications

The core publications supporting the contents presented in this thesis are listed below:


Preliminary results and system demonstrations were reported in the next publications:


### 1.6 Artifacts

**Software:**


**Ontologies:**


**Datasets:**

- PreMOn dataset – [https://premon.fbk.eu/download.html](https://premon.fbk.eu/download.html)
Chapter 2

Background

The work presented in this thesis falls in the broad area of Knowledge Extraction (KE) from text, also known as Ontology Learning and Population (Cimiano, 2006). This area deals with the extraction of structured knowledge from unstructured, natural language texts, and thus combines techniques and resources from the fields of Natural Language Processing, Semantic Web, and Knowledge Representation. In this chapter, we briefly recap the main notions of these fields that are referenced throughout the thesis. We start presenting relevant Semantic Web (SW) standards and concepts (Section 2.1), such as RDF, SPARQL, RDFS and OWL ontologies, and Linked Open Data. We then present the linguistic resources (Section 2.2), Natural Language Processing (NLP) tasks (Section 2.3), and linguistic ontologies (Section 2.4) leveraged in the thesis. We conclude discussing frame-based ontologies and their linguistic grounding (Section 2.5).

Acknowledgments  This chapter reuses material previously published in the state-of-the-art sections of (Corcoglioniti et al., 2016b,c).

2.1 Semantic Web

The term SW refers to a set of standards and best practices, promoted by W3C, for realizing a Web of interlinked, machine-processable data. In the following, we present the main SW concepts and standards used in this thesis.

2.1.1 RDF

The Resource Description Framework (RDF, Wood et al., 2014), first standardized in 2004 (RDF 1.0) and then revised in 2014 (RDF 1.1) by W3C, is the SW data model for describing arbitrary instances (resources) via graphs of ⟨subject, predicate, object⟩ triples.

An RDF node is either a URI,\(^1\) a literal, or a blank node. URIs are global instance identifiers such as <http://dbpedia.org/page/Semantic_Web>, which are often abbreviated in RDF syntaxes using qualified names (QNames) such as dbpedia:Semantic_Web, where dbpedia is the prefix corresponding to namespace http://dbpedia.org/resource/. Literals consist of a string label, a datatype, and an optional language specifier; examples are “42”^^xsd:int, which is an integer literal, and “graph”@en, which is an English string literal. Blank nodes, or bnodes, are locally-scoped identifiers (e.g., _:bnodes123) acting

\(^1\)Recently generalized to IRI, which is the internationalized version of URI whose syntax allows the use of a broader set of (unescape) characters, e.g., for non Western languages.
like existential variables. Within an RDF triple, the subject is a URI or blank node, the predicate is a property URI, and the object is anything: URI, blank node, or literal.

The named graph extension of RDF, introduced by Carroll et al. (2005) and standardized in RDF 1.1, allows placing triples in different graphs named with URIs (some implementations also permit blank nodes). Apart enabling a better organization of triples, named graphs allow giving a name to one or more triple(s), and then using that name to encode triple-level metadata (something not possible with plain triples alone). A triple placed in a named graph can be seen also as a ⟨subject, predicate, object, graph⟩ quad.

Different syntaxes have been defined to serialize RDF data, such as RDF/XML, Turtle, and NTriples for plain RDF, and TriG, NQuad, and TQL for RDF with named graphs.

2.1.2 SPARQL

The SPARQL Protocol and RDF Query Language (SPARQL, Harris and Seaborne, 2013), first standardized by W3C in 2008 (SPARQL 1.0) and then revised in 2013 (SPARQL 1.1), is both a language and an HTTP-based protocol offered by SPARQL endpoints for querying and manipulating RDF data organized in named graphs. Similarly to SQL in relational databases, SPARQL is the primary mechanism for data access and manipulation in RDF triplestores, which are software systems for storage and querying of RDF data.

Data querying in SPARQL is based on subgraph matching: the triple patterns in the WHERE clause of a SPARQL query are matched against RDF triples in the queried dataset, taking into account the named graphs where these triples occur in. Each match provides a tuple of bindings for the variables in the query, which are then combined using standard algebraic operations (e.g., inner and outer join, selection, projection, union, intersection, aggregation) to obtain the tuples representing the solutions of the query. SPARQL SELECT queries return these solutions as they are, while SPARQL CONSTRUCT queries use them to instantiate other triple patterns and thus produce RDF triples; the additional ASK and DESCRIBE query forms are also available.

Data manipulation in SPARQL is based on several operations for adding, modifying or removing RDF triples in a dataset (known overall as SPARQL Update operations), such as the flexible DELETE ... INSERT ... WHERE construct that allows deleting and/or inserting triples in a dataset based on previously matched data.

Both SPARQL CONSTRUCT queries and the DELETE ... INSERT ... WHERE operation provide a general mechanism for deriving RDF triples starting from other RDF triples, and thus for expressing head-body inference rules.

2.1.3 RDFS and OWL Ontologies

In Knowledge Representation (KR), an ontology is defined as a formal explicit specification of a shared conceptualization of a domain (Staab and Studer, 2013). Formal means based on formal languages (e.g., logic-based). Explicit means that the assumptions of the conceptualization are made explicit, enabling reasoning. Shared means that the
conceptualization is adopted by some community, enabling interoperability. The *domain* can be very specific or consist of very general concepts, the latter being described by so-called *upper* or *foundation* ontologies, examples being DOLCE (Gangemi et al., 2002) and SUMO (Niles and Pease, 2001).

An ontology describes instances, classes of instances, and properties of these instances, both from a terminological level or *TBox* (i.e., which classes and properties exist) and from an assertional level or *ABox* (i.e., which classes a specific instance belong to, or which pairs of instances are related by a property). These definitions are expressed as axioms of some formal *ontology language*, which is typically based on first-order logic (e.g., KIF\(^2\)) or description logic. The SW provides two ontology languages for defining SW ontologies that describe the schema and meaning of RDF data: RDFS and OWL.

**RDFS**  *RDF Schema* (RDFS, Guha and Brickley, 2014) is a lightweight ontology language for RDF, first standardized in 2004 and then revised in 2014 by W3C. RDFS allows defining subsumption hierarchies of properties and classes via *rdfs:subClassOf* and *rdfs:subPropertyOf* axioms (encoded themselves as triples), and *rdfs:domain* and *rdfs:range* axioms specify the classes associated to subjects and objects of a certain property. Efficient reasoning in RDFS can be performed via rules, and allows inferring implicit RDF triples based on explicit data (e.g., that an individual of a class is also member of its superclasses).

**OWL** The *Web Ontology Language* (OWL, Motik et al., 2009) extends RDFS with more expressive constructs rooted in Description Logics: class union, intersection, and complement; (qualified) property cardinality restrictions; inverse, symmetric, transitive, reflexive, irreflexive, and functional properties; property chains; class and property disjointness; and equality of ontological individuals. The latter feature, supported via *owl:sameAs* axioms, provides a way for stating that two instances actually are the same (e.g., as result of coreference resolution), and allows merging them, and the assertions about them, into a unique instance with a unique description, a process also known as *owl:sameAs* smushing\(^3\). Two versions of OWL were standardized: OWL 1 dates back to 2004, while OWL 2 was introduced in 2009. OWL 2, in its OWL 2 DL fragment, is based on the description logic SROIQ (Horrocks et al., 2006) extended with datatypes and punning.\(^4\) Even if reasoning is decidable in OWL 2 DL, the complexity of most reasoning tasks is exponential in the number of axioms. For this reason, three sub-fragments of OWL 2 DL were introduced—OWL 2 EL, OWL 2 QL, and OWL 2 RL (Motik et al., 2012)—with the goal of making the complexity of certain reasoning tasks polynomial. Among these fragments, OWL 2 RL represents a good compromise between expressiveness and complexity, enabling the use of forward-chaining rule engines on large ABoxes to materialize all the triples that can be

\[^2\]http://en.wikipedia.org/wiki/Knowledge_Interchange_Format

\[^3\]http://patterns.dataincubator.org/book/smushing.html

\[^4\]I.e., the restricted capability of using the same symbol both as a class and as individual, with the two interpreted as distinct, independent entities, however.
inferred based on the ontology axioms, after which query answering can be performed on a fully materialized knowledge base without the need of considering inference anymore.

2.1.4 Linked Open Data

Linked Open Data (LOD) refers to “a set of best practices for publishing and interlinking structured data on the Web” (Heath and Bizer, 2011) that basically consist in: (i) using resolvable HTTP URIs for identifying instances; (ii) return machine processable (RDF) descriptions of these instances when their URIs are dereferenced; and (iii) provide links to the URIs of related entities in returned instance descriptions, similarly to how a web page provides links to related web pages. Other access mechanisms, such as dump files and SPARQL endpoints, are often employed although not central to Linked Data principles.

LOD best practices were advocated with the goal to foster the publication of data on the SW and thus its bootstrapping, leading to a community effort aiming at making available as many public datasets as possible as LOD, forming what has become to be known as the LOD cloud. Based on recent statistics, the LOD cloud is a collection of RDF data about entities in different domains consisting of over 85 billions of triples in ~3.4K interlinked datasets. Although this data presents shallow structure and semantics, the wealth of information conveyed, and the fact that data is constantly updated (mainly through community efforts) make this kind of data particularly useful for use as background knowledge in knowledge-intensive applications.

In the following, we briefly describe some relevant LOD resources used in this thesis.

DBpedia DBpedia (Lehmann et al., 2015) is a cross-domain dataset extracted automatically from Wikipedia in different languages and representing the hub of the LOD cloud to which many other LOD datasets are interlinked. DBpedia aims at providing as much of factual knowledge in Wikipedia as possible. Each entity having a Wikipedia page becomes a DBpedia instance, described with properties from Wikipedia infobox data. Two families of properties are used: the first, referred as raw infobox data, closely mirror the structure and names used in infoboxes with their inconsistencies; the second, referred as mapped types and properties, are defined in a DBpedia OWL ontology and instantiated starting from raw infobox properties via mappings. DBpedia data is available via RDF dumps, SPARQL, and URI dereferencing.

Freebase Freebase (Bollacker et al., 2008) is a discontinued cross-domain dataset containing community-contributed interlinked data, linked to DBpedia and structured according

---


6See statistics from http://stats.lod2.eu/ as of December 2014. Note that LOD statistics are highly time-depending and slightly inaccurate as influenced by the occasional inaccessibility of some datasets.

7http://wiki.dbpedia.org/services-resources/ontology
to a schema generated and edited by users. Freebase was acquired by Google in July 2010 and used as a source for the Google Knowledge Graph launched in May 2012.

**GeoNames** GeoNames\(^8\) is a geographic database containing the most significant geographical features of Earth (e.g., countries, populated places) with georeferencing and containment relationships. It is linked to DBpedia and used as a hub for geographical data, within and outside the SW.

**YAGO** YAGO (Hoffart et al., 2013) is a cross-domain dataset automatically extracted from Wikipedia, WordNet (Fellbaum, 1998), and GeoNames, with a rich type taxonomy (350K classes) and facts annotated with confidence, time and space validity; 95% accuracy was manually measured. YAGO is linked to DBpedia and its data is available via RDF dumps and URI dereferencing. In this thesis we leverage the YAGO taxonomy, whose upper levels consist of classes derived from WordNet synsets (and thus aligned to them, which is useful for KE), while leaf classes consist of selected Wikipedia categories.

### 2.2 Linguistic Resources

Many NLP tasks and KE techniques rely on the availability of comprehensive linguistic resources. This thesis, and the ontologies here referenced, build on WordNet and on various predicate models, which we introduce in the following.

#### 2.2.1 WordNet

WordNet (Fellbaum, 1998) is a *lexical database* that provides the possible *senses* of words and other lexical entries, grouping them into sets of synonymous senses called *synsets*. Synsets are specific to a part-of-speech (noun, verb, adjective, adverb) and are associated to a short definition (gloss). Different synset relations are represented, including hyponymy/hypernymy, meronymy/holonymy, and linguistic derivation (e.g., between related synsets of different parts-of-speech). WordNet was originally introduced for (and refers to) the English language, although wordnets for other languages are now available, although not used in this thesis. The latest English version is WordNet 3.1, for which an official RDF version exists,\(^9\) although version 3.0 is still used in many applications and resources derived from WordNet; mappings between WordNet versions are provided by third parties.\(^10\) While WordNet cannot be seen as a proper ontology, mappings from WordNet to several ontologies exist (see previously mentioned SUMO and YAGO).

---

\(^8\) [http://www.geonames.org/](http://www.geonames.org/)


Table 2.1. Comparison of predicate models.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>NB</th>
<th>PB</th>
<th>VN</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parts-of-speech</td>
<td>noun</td>
<td>verb</td>
<td>verb</td>
<td>any (9 total)</td>
</tr>
<tr>
<td>Term for semantic classes</td>
<td>roleset</td>
<td>roleset</td>
<td>verb (sub-)class</td>
<td>frame</td>
</tr>
<tr>
<td>Lexical entries per class</td>
<td>exactly one</td>
<td>exactly one</td>
<td>zero or more</td>
<td>zero or more</td>
</tr>
<tr>
<td>Scope of semantic roles</td>
<td>local to semantic class</td>
<td>local to semantic class</td>
<td>global</td>
<td>local to semantic class (core roles)</td>
</tr>
<tr>
<td>Types of semantic roles</td>
<td>numbered, modifier</td>
<td>numbered, modifier, secondary agent</td>
<td>thematic role (hierarchy)</td>
<td>frame elements (core + other 3 types)</td>
</tr>
<tr>
<td>Semantic class relations</td>
<td>–</td>
<td>–</td>
<td>subclass</td>
<td>inheritance + 8 relations</td>
</tr>
<tr>
<td>Semantic role relations</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>inheritance + 9 relations</td>
</tr>
<tr>
<td>Additional features</td>
<td>mappings to PropBank, VerbNet</td>
<td>mappings to VerbNet</td>
<td>selectional restrictions, syntactic frames</td>
<td>semantic types on classes, roles, lexical units</td>
</tr>
</tbody>
</table>

2.2.2 Predicate Models

Predicate models provide rich descriptions of predicates abstracting from many linguistic phenomena related to their realization in text. In these models, lexical entries acting as predicates (e.g., the “sell” verb) are associated to semantic classes\(^{11}\) (e.g., “Commerce Sell”), each one conveying a possible predicate sense characterized by a set of possible semantic roles (e.g., “Seller,” “Buyer”) that can be played by predicate arguments in the text. Thanks to the availability of mappings between different predicate models, their integration in SRL (Semantic Role Labeling) tools, and, more generally, their laying at the syntactic-semantics interface, predicate models have become central to a number of tasks such as Information Extraction (IE), Question Answering, and Natural Language Generation, and are increasingly used in the SW community for KE and as starting point for deriving general-domain ontologies grounded in natural language. In Table 2.1 and the following, we briefly present the main predicate model resources considered in the thesis.

PropBank  PropBank is a predicate model for verbs, later extended in OntoNotes (Hovy et al., 2006) to other parts-of-speech. PropBank associates a lexical entry to one or more semantic classes called rolesets that are not shared in general with other entries. Semantic roles are defined locally to each semantic class and categorized as numbered arguments (e.g., Arg0 and Arg1, usually the proto-agent and proto-patient), modifiers (ArgM plus a function tag, such as LOC for location), and secondary agent (ArgA in OntoNotes). Annotated examples are provided for each semantic class.

\(^{11}\)Called with different names in each model, such as roleset, verb class, and frame.
NomBank  NomBank (NB, Meyers et al., 2004) is a model for noun predicates that closely mirrors PropBank, in that it associates nouns to noun-specific semantic classes also called rolesets, and defines semantic roles locally to semantic classes, categorizing them as numbered arguments and modifiers. Examples are provided for each semantic class.

VerbNet  VerbNet is a predicate model for verbs inspired by Levin classes (Levin, 1993). Semantic classes in VerbNet are organized in a hierarchy of verb classes. A verb class is associated to multiple verb lexical entries and to multiple globally defined thematic roles (e.g., agent, patient), which describe the semantic roles for that class (with possible selectional restrictions) and form themselves a role hierarchy. Verb classes are also associated to syntactic frames. They define how semantic roles can be realized syntactically (e.g., “Agent V Theme”) and are associated to a specification of the conveyed semantics based on logical-like predicates applied to thematic roles and event variables (e.g., “cause(Agent, E”) ). Both thematic roles and syntactic frames are inherited by subclasses. An example sentence is provided for each frame of a verb class.

FrameNet  FrameNet (FN, Baker et al., 1998) builds on the linguistic theory of Frame Semantics introduced by Fillmore (1982), and provides semantic classes called frames that define prototypical situations evoked by lexical entries of different parts-of-speech. The pair ⟨frame, lexical entry⟩ is called lexical unit. The semantic roles of a frame are called frame elements (FEs): core and core unexpressed FEs classify mandatory arguments and characterize a frame, differently from peripheral FEs, while extra-thematic FEs (e.g., “Iteration”) situate the frame in a larger context. A set of typically co-expressed FEs form a CoreSet. Several frame and FE relations are defined, starting from frame inheritance along which core FEs (but not unexpressed ones) are propagated. A small subset of frames and FEs is annotated with semantic types, which may express selectional constraints when used with FEs. Annotated examples are provided for each lexical unit.

SemLink  SemLink (Palmer, 2009) is a resource providing mappings between the ⟨lexical entry, semantic class⟩ pairs of (i) VerbNet and PropBank and (ii) VerbNet and FrameNet as well as between the semantic roles of these resources.

Predicate Matrix  The Predicate Matrix (Lacalle et al., 2014) is a mapping resource relating semantic classes and semantic roles of VerbNet, PropBank, and FrameNet. The Predicate Matrix includes the mappings of SemLink and augments them based on the use of automatic techniques leveraging WSD (Word Sense Disambiguation). Mappings to WordNet senses, to concepts of the Multilingual Central Repository (MCR)^12, and to additional resources are also provided.

^12http://adimen.si.ehu.es/MCR
2.3 Natural Language Processing

Ontology Population (including our approach presented in Chapter 5) involves several NLP tasks. Next, we briefly recall the tasks related to the contents presented in this thesis.

**Tokenization, POS-Tagging, Lemmatization** These basic tasks split a text into tokens, each annotated with its lemma and part-of-speech (POS), such as proper noun. The Penn Treebank POS tags\(^{13}\) (NN, NNP, VB, . . . ) are commonly used for the English language. Almost every NLP toolkit, such as Stanford CoreNLP (Manning et al., 2014) used in this thesis, supports these tasks, and very high performances have been achieved.

**WSD** The WSD (Word Sense Disambiguation) task disambiguates tokens with respect to WordNet, assigning each token to the word sense (and thus WordNet synset) most likely identifying the meaning conveyed by that specific word occurrence in the text. Several WSD tools are available using different disambiguation techniques (e.g., unsupervised, knowledge-based), like the UKB (Agirre et al., 2014) tool used in this thesis.

**NERC** The NERC (Named Entity Recognition and Classification) task aims at identifying spans of text denoting named entities of predefined categories having a proper name (e.g., persons, organizations, locations) or some equivalent identifying information (e.g., time entities, numerical quantities, amounts of money). Different catalogs of NERC categories (with training sets for supervised NERC systems) exist, such as the BBN one\(^{14}\). Most NLP toolkits support NERC, as Stanford CoreNLP used in this thesis.

**EL** The EL (Entity Linking) task aims at linking named entities to well-known instances in external knowledge bases such as DBpedia. When the scope of linking is expanded to common nouns and other, non-named entities, the task is also known as Wikification. A popular EL system, specific to DBpedia, is DBpedia Spotlight (Mendes et al., 2011). Knowledge base-agnostic EL systems also exist, such as AGDISTIS (Usbeck et al., 2014).

**TERN** The TERN (Temporal Expression Recognition and Normalization) task identifies dates, times and temporal durations in a text. A normalized time value (e.g., the date components) is extracted, usually based on the TimeML (Pustejovsky et al., 2010) standard. Relative temporal expressions such as ‘yesterday’ are normalized based on a reference date associated to the text (the so-called document creation time). TERN is provided by many NLP toolkits, such as Stanford CoreNLP, and by specialized tools such as HeidelTime.\(^{15}\)

\(^{13}\)http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

\(^{14}\)http://catalog.ldc.upenn.edu/docs/LDC2005T33/BBN-Types-Subtypes.html

\(^{15}\)http://dbs.ifi.uni-heidelberg.de/index.php?id=129
Parsing  This task aims at extracting and representing the syntactic structure of a sentence. In this thesis we rely mainly on Dependency Parsing (DP), which organizes the tokens of a sentence in a tree, with directed edges connecting a governing token (e.g., a verb) to its dependent tokens (e.g., its subject), labeled with a syntactic function (e.g., subj);\(^{16}\) given a text span, its head is the span token that is ancestor in the tree of all the span tokens. Differently, Constituency Parsing identifies the constituent phrases of a sentence and organizes them in a tree. As for NERC, different tag sets exist for annotating syntactic dependencies and constituent types.

SRL  The SRL (Semantic Role Labeling) task annotates occurrences of semantic predicates in the text, identifying predicate arguments with their roles as defined in a reference predicate model. SRL tools for different predicate models are available, such as Mate-tools (Björkelund et al., 2009) for PropBank and NomBank, and Semafor (Das et al., 2014) for FrameNet (both used in this thesis). SRL tools may be integrated or replaced one with another by leveraging available mappings between predicate models.

Coreference Resolution  This task identifies sets of text spans, called coreference sets, that denote the same referent, such as a proper noun in a text followed by a pronoun referring to it (anaphora). Differently from the previous tasks, that typically operate on a per-sentence basis, Coreference Resolution is defined either at the intra-document level, working on all (or a window of) the sentences of a document, and at the cross-document level, spanning multiple documents. In this thesis we use the state-of-the-art coreference resolution system included in Stanford CoreNLP.

2.4 Linguistic Ontologies

The use of SW technologies and the adoption of the LOD paradigm have already been recognized as particularly beneficial to linguistic resources (see, e.g., Chiarcos et al., 2013), leading to the creation of the Linguistic LOD cloud curated by the Open Linguistic subgroup of the Open Knowledge Foundation.\(^{17}\) In that context, several linguistic ontologies have been introduced. In the following, we briefly describe the ones most relevant to this thesis.

Lemon  The lemon (lexical model for ontologies) ontology by McCrae et al. (2012), recently revised by the W3C Ontology Lexicon Community Group (Cimiano et al., 2015), has been introduced to allow mapping linguistic expressions to the ontological concepts they denote. This is achieved by linking a ontolex:LexicalEntry to one or more ontological concepts via corresponding ontolex:LexicalSense instances. The OntoLex version of lemon also supports linking lexical entries to intensional, non-formal concepts (e.g., WordNet synsets, skos:Concepts), and comes with additional modules for further describing lexical

---

\(^{16}\)http://en.wikipedia.org/wiki/Dependency_grammar

\(^{17}\)http://linguistics.okfn.org/
entries in terms of syntax and semantics (synsem module), decomposition (decomp), variation and translation (vartrans), and for providing linguistic metadata (lime).

**LexInfo** The LexInfo (Cimiano et al., 2011) ontology provides data categories (e.g., possible values of morphosyntactic properties, structure of syntactic frames) for use with lemon, and is currently being updated to the OntoLex version of lemon.

**NIF** NIF (NLP Interchange Format, Hellmann et al., 2013) is an RDF/OWL ontology for representing NLP annotations in a common way and foster interoperability between NLP tools, language resources and annotations (which typically come with their own annotation formats), developed with the support of the LOD2 EU Project (Auer et al., 2014). The core of NIF consists of a vocabulary and a URI design that permit identifying and describing strings and substrings, to which arbitrary annotations can be attached using vocabularies external to NIF. Currently available annotation vocabularies (see below) can be used for NLP tasks such as Tokenization, POS-Tagging, and NERC.

**OLiA** OLiA (Ontologies of Linguistic Annotation, Chiarcos and Sukhareva, 2015) is a suite of ontology modules providing an RDF/OWL version of several NLP annotation tag sets covering, e.g., morphology, morphosyntax, constituent types, and dependency relations. Each tag (e.g., the NNP part-of-speech tag of Penn Treebank) is mapped to an ontological instance, and classes of tags are defined to group tags (also in different models) of the same type (e.g., proper noun). These tag individuals and classes can be used together with NIF, via properties nif:oliaLink and nif:oliaClass, respectively.

**ITS** ITS (Internationalization Tag Set, Filip et al., 2013) is a W3C specification defining document metadata for guiding the translation of structured texts. ITS defines a few ontological properties that can be used with NIF for specifying, e.g., the class of a named entity (property its:taClassRef) and the URI it has been disambiguated to (its:taIdentRef).

**NERD** NERD (Named Entity Recognition and Disambiguation, Rizzo and Troncy, 2012) defines a hierarchy of ontological classes for modeling NERC categories, covering the categories of different NERC tools and making their outputs interoperable.

### 2.5 Frame-based Ontologies

By **frame-based ontology** we intend an ontology where knowledge is primarily organized around semantic frames, in what is also known as neo-Davidsonian representation (Parsons, 1990). Semantic frames are events, situations, n-ary relations, and other structured entities *reified* (i.e., made explicit) as ontological instances, which are then linked to related instances via object properties specifying their semantic roles in the frame, according to a star-shaped structure. While here defined independently, semantic frames can be seen as
akin to frames in *frame languages* for KR (Minsky, 1974) and to frames of the linguistic theory of *Frame Semantics* (Fillmore, 1982). From a pragmatic point of view, semantic frames provide a possible solution to the problem of expressing n-ary relations in RDF, a solution that is argued to be more compact (in terms of number of triples) and “clean” (as it does not privilege a specific pair of instances) than other solutions such as reification, annotated subproperties, and role objects (Rouces et al., 2015).

Many non frame-based ontologies naturally recur to a representation based on semantic frames for what concerns the modeling of events, including upper ontologies such as DOLCE and SUMO, using however other approaches (e.g., binary relations) for non-event knowledge. In the following, we focus on frame-based ontologies, and introduce the main ontologies of this kind that are relevant to the contents of this thesis.

**SEM and LODE** SEM (Simple Event Model, van Hage et al., 2011) and LODE (Linking Open Descriptions of Events, Shaw et al., 2009) are ontologies that allow modeling events as semantic frames. Both ontologies define only very general classes and properties for events, event actors, event time and event place, leaving to users the task of further specifying these concepts for a specific domain.

**ESO** ESO (Event Situation Ontology, Segers et al., 2015) is an OWL 2 ontology derived from FrameNet and SUMO that models events, the roles of their participants, and their pre-, post-, and during- situations. An event in ESO is reified as a semantic frame connected to its participant instances via object properties. A hierarchy of event classes is proposed, featuring a major division among *dynamic event*, i.e., things that happen in particular points of time (e.g., *eso:Hiring* and *eso:Firing*), and *static events*, i.e., state of affairs that hold during a period of time (e.g., *eso:InEmployment* for being employed at). Pre-, post-, and during- situations in ESO are sets of ABox triples linking event participants (e.g., $X$ *eso:employs* $Y$) that hold, respectively, before a dynamic event, after a dynamic event, and during a static event. When instantiating the ESO in the ABox, the triples of a situation are grouped in a named graph annotated with a temporal validity and linked to the event associated to the pre-, post-, or during-situation. ESO Ver. 1 consists of 59 event classes (9 static events) derived from and mapped to SUMO classes and FrameNet frames and frame elements. The latter mappings provide a linguistic grounding that allow instantiating the ESO ontology based on the output of SRL, as done within the NewsReader18 EU Project.

**FrameBase** FrameBase (Rouces et al., 2015) is an RDFS ontology of semantic frames derived from FrameNet and WordNet in 2005. In FrameBase, two hierarchies of classes modeling semantic frames and object properties modeling roles are derived from FrameNet frame and frame element hierarchies, based on inheritance and perspectivization relations. Each frame class is further specialized by introducing two types of *microframe* classes: *LU-microframes*, associated to lexical units of that frame class; and *cluster-microframes*,

18http://www.newsreader-project.eu/
grouping all the LU-microframes belonging to the same or to derivationally-related synsets in WordNet, based on a specifically-created mapping from lexical units to WordNet synsets. This specialization, based on WordNet, accounts for the fact that lexical units of a frame often express very different meanings (e.g., lexical units “increase” and “decrease” of frame “Change position on a scale”). FrameBase also defines direct binary properties to link specific frame participants. Different heuristics are applied to select and name these properties, based on the analysis of annotated sentences of FrameNet. Reification/dereification (reder) rules are provided to automatically map from semantic frames to binary properties and back. While FrameBase is primarily envisioned to support the seamless representation and querying of n-ary relations differently modeled in knowledge bases, its roots in FrameNet and WordNet, two domain-general, broad-coverage linguistic resources, allow it to provide a large inventory of linguistically-grounded semantic frames of particular interest for KE.
Chapter 3

PreMOn: Predicate Models and FrameBase Mappings in RDF/OWL

Predicate models such as PropBank (PB, Palmer et al., 2005), NomBank (NB, Meyers et al., 2004), VerbNet (VN, Kipper Schuler, 2005), and FrameNet (FN, Baker et al., 1998) provide rich descriptions of predicate semantic classes (e.g., “Commerce Sell”) and semantic roles (e.g., “Seller,” “Buyer”), abstracting from a number of linguistic phenomena related to their realization in text. Semantic classes and semantic roles are annotated in text by tools for SRL (Semantic Role Labeling) and are mappable to classes and properties of frame-based ontologies like FrameBase (Rouces et al., 2015) and ESO (Event Situation Ontology, Segers et al., 2015), thus providing a solid basis for the task of Frame-based Ontology Population from text investigated in this thesis.

However, if we focus on the PB, NB, VN, and FN predicate models for which mature SRL systems exist—e.g., Mate-tools (Björkelund et al., 2009) for PB and NB, Semafor (Das et al., 2014) for FN, and Boxer (Bos, 2008) for VN—two problems arise for their use for Ontology Population: (i) the lack of an homogeneous RDF/OWL formalization of these models, each one coming with its own terminology, structure and proprietary XML format; and (ii) the existence of FrameBase mappings only for FN.

In this chapter, we tackle these problems by building PreMOn (predicate model for ontologies), a linguistic resource for representing predicate models, the mappings between them, such as SemLink (Palmer, 2009) and the Predicate Matrix (Lacalle et al., 2014), and the mappings to frame-based ontologies in RDF/OWL. PreMOn consists of two components: (i) the PreMOn Ontology, an OWL 2 ontology that extends lemon (McCrae et al., 2012; Cimiano et al., 2015) and models the core concepts of semantic class, semantic role, mapping, and annotation common to all predicate models (Section 3.1), as well as their specializations for each model (Section 3.2); and (ii), the PreMOn Dataset, a freely-available, interlinked RDF dataset containing data for PB, NB, VN, and FN, the SemLink mappings (Section 3.3), and specifically created mappings from PB, NB, and FN to FrameBase (Section 3.4), all published online as Linked Open Data (LOD) according to the PreMOn Ontology.

PreMOn is the first resource providing an homogeneous ontological modeling of the major predicate models (Section 3.5), and we believe that its utility is not restricted to ontology population (as we use it for in this thesis), but rather regards all the users of predicate models within and outside the Semantic Web (SW) area (Section 3.6).

1Available at http://premon.fbk.eu/
Acknowledgments  The material presented in this chapter is the result of collaborative work partially published in (Corcoglioniti et al., 2016b), excluded FrameBase mappings.

3.1 The PreMOn Ontology – Core Module

Namespace: http://premon.fbk.eu/ontology/core#
Prefix: pmo

The PreMOn Ontology is an extension of lemon for representing predicate models and their mappings. An overview of the PreMOn Ontology core module, and its relation with lemon, is shown in Figure 3.1 using a UML-like notation. To guide the exposition, we will also refer to Figure 3.2, showing an example of instantiation of semantic classes and roles for the predicate models here considered, as well as an example of mapping between resources from different models.
3.1.1 Semantic Classes and Roles

The lemon ontology represents lexical entries (class `onto:LexicalEntry`) with their associated lexical forms, and allows relating entries to the ontological entities they denote (classes, properties, individuals) using the `onto:LexicalSense` reified relation. Besides mapping to an ontology, which provides the extensional (formal) interpretation of lexical entries, lemon supports mapping entries to `onto:LexicalConcepts` (subclass of `skos:Concept`), each denoting an intensional (~informal) meaning evoked by a set of lexical entries. Example of lexical concepts are the synsets of WordNet (Fellbaum, 1998), whose semantics is not formally encoded in an ontology.

We extend lemon by introducing classes `pmo:SemanticClass` and `pmo:SemanticRole` (filled in light green in Figure 3.1). `pmo:SemanticClass` homogeneously represents the semantic classes from the various predicate models. That is, individuals of this class correspond to rolesets in PB and NB (e.g., `pm:nb10-seller.01` and `pm:pb17-sell.01` in Figure 3.2), verb classes in VN (e.g., `pm:vn32-give-13.1-1`), and frames in FN (e.g., `pm:fn15-commerce_sell`).

An instance of `pmo:SemanticClass` typically has (via property `pmo:semRole`) a number of `pmo:SemanticRoles`, representing, from a semantic point of view, the roles the arguments of that `pmo:SemanticClass` can play. For instance, the triple

```
   pm:pb17–sell.01 pmo:semRole pm:pb17–sell.01–arg1
```

states that `pm:nb10-seller.01` has the semantic role `pm:nb10-seller.01-arg1`. Importantly, semantic roles are defined locally to semantic classes, so VN ‘agent’ is represented as multiple semantic roles, one for each verb class it occurs in, and with each semantic role linked to its specific selectional restrictions (if any). Note that `pmo:SemanticClass` is defined as subclass of `onto:LexicalConcept`, as we see `pmo:SemanticClasses` as essentially informal concepts rather than well defined concepts of a formal ontology (although an ontology can be derived from them, cf. FrameBase and ESO). Being `onto:LexicalConcepts`, `pmo:SemanticClasses` inherit the link to lexical entries as well as the link (via `onto:isConceptOf`) to the ontological entities formalizing them, typically event classes.

Properties `pmo:classRel` and `pmo:roleRel`, and their resource-specific subproperties, are introduced to express the relations between elements at each level, such as subtyping, and predicate and role inheritance (e.g., `pmofn:inheritsFrom` and `pmofn:inheritsFromFER` for FN). Additional resource-specific classes (e.g., `pmovn:ThematicRole`, filled in light blue in Figure 3.1) and properties (e.g., `pmovn:thematicRole`) further characterize important aspects of each predicate model, like commonalities between semantic roles.

3.1.2 Mappings

Mappings between different predicate models cannot be expressed using only the classes above, as they are often defined (e.g., in SemLink and Predicate Matrix) in terms of

---

2 `pm: http://premon.fbk.eu/resource/` is the namespace of PreMOon Dataset (Section 3.3); the fragments nb10, pb17, and so on identify the resource and its version (e.g., NB 1.0, PB 1.7).
pairs. To model these pairs, one could reuse the notion of `ontolex:LexicalSense`. However, its formalization in lemon as reified relation depends on the existence of (exactly) one ontological entity for each `(ontolex:LexicalConcept, ontolex:LexicalEntry)` pair, a strong constraint that we do not necessarily need for our purposes. Therefore, we introduce the `pmo:Conceptualization` class (filled in light red in Figure 3.1, together with other mapping related classes). Structurally, a `pmo:Conceptualization` can be seen as the reification of the `ontolex:evokes` relation between `ontolex:LexicalEntry` and `ontolex:LexicalConcept`. Semantically, it can be seen as a very specific intensional concept (among many, in case of polysemy) evoked by a single `ontolex:LexicalEntry`, which can be generalized to a `ontolex:LexicalConcept` when multiple entries are considered but with a possible loss of information that prevents precise alignments to be represented.

Mappings are explicitly represented as individuals of class `pmo:Mapping`, and can be seen as sets of (or n-ary relations between) either (i) `pmo:Conceptualizations`, (ii) `pmo:SemanticClasses`, and (iii) `pmo:SemanticRoles`, with role mappings anchored to conceptualization or class mappings via property `pmo:semRoleMapping`. Figure 3.2 shows an example of mapping (pm:mapping_1356) between two `pmo:Conceptualizations`, one from NB (pm:co-n-seller-nb10-seller.01) and one from PB (pm:co-v-sell-pb17-sell.01). We rely on this set-like modeling, since mappings are not necessarily represented as binary relations in mapping resources: e.g., in the Predicate Matrix, each row represents the mapping of a semantic role / lexical entry pair over the different resources (e.g., (13.1-1-agent, deal) in VN, (sell.01-arg0, sell) in PB, (Commerce_Sell-seller, sell) in FN) as well as the corresponding WordNet sense. Reifying the n-ary mapping relation also allows us, if needed, to further characterize each mapping, asserting additional information such as confidence and reliability. Moreover, it is possible to further specialize mappings—e.g., to model directional mappings, or to represent different relationships among their members—by subtyping the `pmo:Mapping` class or the property (pmo:item) relating a `pmo:Mapping` to its members.

### 3.1.3 Annotations

Predicate models are typically complemented by examples showing concrete annotations of semantic classes and roles in text. More generally, a text can be annotated with semantic classes and roles as a result of manual or automatic SRL.

The PreMOn Ontology provides some common primitives (filled in light yellow in Figure 3.1) aiming at properly modeling the heterogeneous annotations of a text for different predicate models. These primitives are based on NIF (NLP Interchange Format, Hellmann et al., 2013). NIF introduces the general notion of `nif:String` to represent arbitrary text strings; `nif:Context` is a particular subclass of `nif:String`, representing a whole string of text, and any substring (itself a `nif:String`) has a `nif:referenceContext` relation to the `nif:Context` individual representing the whole text containing it.

To specifically model the aforementioned examples complementing predicate models, we introduce `pmo:Example`, subclass of `nif:Context`, to represent the string associated with
the example. The occurrence of a `ontolex:LexicalEntry`, `pmo:SemanticClass`, or `pmo:SemanticRole` in a `nif:Context` is denoted by an instance of `nif:Annotation`, related to the given `ontolex:LexicalEntry`, `pmo:SemanticClass`, or `pmo:SemanticRole` via property `pmo:valueObj` (the value attached to the annotation), and to the `nif:Context` instance via property `nif:annotation`. If detailed information on the specific span of text denoting the `ontolex:LexicalEntry`, `pmo:SemanticClass`, or `pmo:SemanticRole` is available (e.g., FN provides the specific offsets of lexical units, frames, and frame elements in the example text) an additional instance of `pmo:Markable`, subclass of `nif:String`, is created and linked to the specific `nif:Annotation` and `nif:Context` via properties `nif:annotation` and `nif:referenceContext`, respectively. As the same `nif:Context` may contain multiple `nif:Annotations` referring to one or more semantic classes and their corresponding roles, an additional `pmo:AnnotationSet` instance is created to cluster annotations referring to the same predicate structure.

3.2 The PreMOOn Ontology – Specialized Modules

While the PreMOOn Ontology Core Module provides an homogeneous abstraction over heterogeneous predicate models, additional ontology modules, specializing or extending the PreMOOn Ontology core elements, can be defined to properly capture and model resource-specific aspects (including terminology) in a way compatible with the underlying PreMOOn Ontology assumptions. We developed four ontology modules, one for each predicate model: PropBank, NomBank, VerbNet, and FrameNet. An overview of the main additional classes (filled in light blue) and properties is shown in Figures 3.3, 3.4, 3.5, 3.6, and 3.7.

3.2.1 PropBank Module

Namespace: `http://premon.fbk.eu/ontology/pb#`
Prefix: `pmopb`

We define classes `pmopb:Roleset` and `pmopb:SemanticRole`\(^3\) as subclasses of `pmo:SemanticClass` and `pmo:SemanticRole`, respectively. Each `pmopb:SemanticRole` instance is related (via property `pmopb:argument`) to exactly one `pmopb:Argument`, which is defined as the disjoint union of three subclasses: `pmopb:NumberedArgument`, containing the individuals corresponding to numbered arguments (e.g., Arg0, Arg1); `pmopb:Modifier`, containing the individuals corresponding to modifiers (e.g., ArgM-LOC, ArgM-TMP); and, `pmopb:SecondaryAgent`, containing the single individual annotating secondary agents (ArgA). While PB annotation guidelines define a single modifier (ArgM) with multiple function tags (e.g., LOC, TMP), we opt to specialize the modifier for each tag, similarly to the way these arguments are actually annotated by state-of-the-art SRL tools. Property `pmopb:tag` enables associating possible tags, either a `pmopb:Modifier` or a tag defined in class `pmopb:Tag`, to

\(^3\)While this class does not necessarily specialize `pmo:SemanticRole` with additional properties or restrictions, we add it to ease the retrieval of PB-specific semantic roles, something handy when the same repository contains roles from several predicate models.
3.2.2 NomBank Module

Namespace: http://premon.fbk.eu/ontology/nb#
Prefix: pmonb

Similarly to PB, we define pmonb:Roleset and pmonb:SemanticRole as subclasses of pmo:SemanticClass and pmo:SemanticRole, respectively. Each pmonb:SemanticRole instance is related (via property pmonb:argument) to exactly one pmonb:Argument, which is defined as the disjoint union of two subclasses: pmonb:NumberedArgument, containing the individuals corresponding to numbered argument (e.g., Arg0, Arg1), and pmonb:Modifier, containing...
the individuals corresponding to modifiers (e.g., ArgM-LOC, ArgM-TMP). We also define class pmonb:Tag to capture (via property pmonb:tag) some specific annotations of markables (e.g., PRD, REF, SUPPORT) in the examples.

### 3.2.3 VerbNet Module

**Namespace:** http://premon.fbk.eu/ontology/vn#

**Prefix:** pmovn

We define classes pmovn:VerbClass and pmovn:SemanticRole as subclasses of pmo:SemanticClass and pmo:SemanticRole, respectively. VN class members are modeled as instances of ontolex:LexicalEntry, connected to their class via property ontolex:evokes. The VN class hierarchy is modeled via the pmovn:subclassOf property (subproperty of skos:broader), that relates a verb class (e.g., 13.1-1) with its parent class (e.g., 13.1). Given the propagation of semantic roles along the class hierarchy, we introduce property pmovn:definesSemRole to differentiate the pmovn:SemanticRole instances defined on a class from the ones inherited from its ancestor classes. Each pmovn:SemanticRole instance is related (via property pmovn:thematicRole) to exactly one pmovn:ThematicRole, which contains all the thematic
roles defined in VN. These thematic roles are organized in a hierarchy, which is formalized via the skos:broader property. For instance,

\texttt{pmovn:agent skos:broader pmovn:actor}

states that \texttt{pmovn:agent} is more specific than \texttt{pmovn:actor}. VN selectional restrictions on \texttt{pmovn:SemanticRoles} (e.g., restricting “theme” to something not animate) are formalized using property \texttt{pmovn:restriction} and class \texttt{pmovn:Restriction}.

A verb class may have one or more \texttt{pmovn:VerbNetFrames} (via property \texttt{pmovn:frame}, or its subproperty \texttt{pmovn:definesFrame}, to distinguish frames defined on the class from inherited frames), which have one or more ordered\footnote{We relied on the standard first/next/item pattern for lists.} \texttt{pmo:Preds}, modeling a syntactic construction (e.g., “Agent V Theme [-sentential]”) shared by all class members, and one or more ordered semantic \texttt{pmo:Preds}, modeling the meaning of the event, and its participants, expressed by the verb class for that syntactic construction (e.g., “approve(during(E), Agent, Theme”)\texttt{)}. \texttt{pmo:SynItems} are specialized according to their syntactical function (e.g., \texttt{pmovn:NpSynItem} for noun phrases). A \texttt{pmovn:NpSynItem} can point (via \texttt{pmo:value-Obj}) to a \texttt{pmovn:SemanticRole}, and define, via \texttt{pmovn:restriction}, (i) a selectional restriction holding for the \texttt{pmovn:SemanticRole} in that frame (e.g., “animate”), or (ii) some other syntactic restriction (e.g., “np_to_inf”). Similarly, selectional restrictions can be modeled on \texttt{pmovn:PrepSynItem} (e.g., “spatial”). Predicates in \texttt{pmovn:Pred} have a type (\texttt{pmovn:Pred-Type}, e.g., “approve”) and can be further decomposed in \texttt{pmovn:PredArg} (e.g., “during(E)”) of various types (e.g., \texttt{pmovn:EventPredArg}). Predicate negation is expressed by typing its instance as \texttt{pmovn:NegPred}, while implicit \texttt{pmovn:PredArgs} are typed as \texttt{pmovn:ImplicitArg}.

\texttt{pmovn:Restrictions} are modeled structurally as AND / OR compositions (\texttt{pmovn:CompoundRestriction} and subclasses) of atomic restrictions (\texttt{pmovn:AtomicRestriction} and subclasses) that require the presence or absence of a certain \texttt{pmovn:RestrictionProperty} (Figure 3.7). Semantically, restrictions are divided in \texttt{pmovn:SyntacticRestrictions}, which refer to a \texttt{pmovn:SyntacticRestrictionProperty} and are attached to \texttt{pmovn:NpSynItems}, and \texttt{pmovn:SelectionalRestrictions}. The latter are divided in \texttt{pmovn:PrepositionalSelectionalRestrictions}, which refer to a \texttt{pmovn:PrepositionalRestrictionProperty} and are attached to \texttt{pmovn:PrepSynItems}, and \texttt{pmovn:RoleSelectionalRestrictions}, which refer to a \texttt{pmovn:RoleRestrictionProperty} and are attached both to \texttt{pmovn:SemanticRoles} and \texttt{pmovn:NpSynItems}. Every restriction instance instantiates both a structural and a semantic subclass of \texttt{pmovn:Restriction}.

### 3.2.4 FrameNet Module

**Namespace:** http://premon.fbk.eu/ontology/fn#

**Prefix:** pmofn

We define classes \texttt{pmofn:Frame} and \texttt{pmofn:FrameElement} as subclasses of \texttt{pmo:SemanticClass} and \texttt{pmo:SemanticRole}, respectively. \texttt{pmofn:FrameElement} is further specialized in four subclasses, denoting the four typologies of FN frame elements (e.g., \texttt{pmofn:CoreFrameElement}). Being \texttt{pmo:SemanticRoles}, in PreMOn Ontology frame elements are
always specific to the frame where they are defined, even for extra thematic frame elements that are typically shared across frames in FN (e.g., the “Circumstances” extra thematic frame element corresponds to multiple individuals of type pmofn:ExtraThematicFrameElement, one for each frame where it is defined). Frame element core sets of a pmofn:Frame are represented as reified objects of type pmofn:FECoreSet, having as members some pmofn:FrameElements. Relations between pmofn:Frames are modeled using the subproperties of pmofn:frameRelation (e.g., pmofn:inheritsFrom). Similarly, mappings between pmofn:FrameElements of pmofn:Frames related via some pmofn:frameRelation are represented using frame relation-specific subproperties of pmofn:frameElementRelation (e.g., pmofn:inheritsFromFER). Within a frame, a frame element may exclude/require the presence of another frame element (pmofn:excludesFrameElement/pmofn:requiresFrameElement). pmofn:LexicalUnit, associating a lexical entry with a frame, is defined as subclass of pmofn:Conceptualization. A pmofn:LexicalUnit may have a development status (pmofn:LUStatus) and can incorporate a pmofn:FrameElement (e.g., “microvawe.v,” besides evoking frame “Apply heat,” also incorporates the frame element “Heating instrument”). Finally, pmofn:Frames, pmofn:FrameElements and pmofn:LexicalUnits can be constrained according to some semantic types defined in pmofn:SemType, and organized in a hierarchy according to pmofn:subTypeOf relations between them.

3.3 The PreMOn Dataset

Namespace: http://premon.fbk.eu/resource/
Prefix: pm

To populate PreMOn with content from the various resources (predicate models, mappings), we developed an open-source Java command-line tool available on PreMOn website. The tool applies pluggable, resource-specific converters to the original distribution files of each resource, instantiating the proper individuals and assertions according to the PreMOn Ontology. If available, mappings to additional resources, such as WordNet synsets and groupings of OntoNotes (Hovy et al., 2006), are also extracted. OWL 2 RL inference, statistics extraction and some cross-resource cleanup (e.g., for dropping inconsistent mappings) are applied to extracted triples, leveraging RDFpro (Chapter 7) for RDF processing. The resulting triples are placed in distinct named graphs identified by the resource name and version (e.g., pm:fn15 for FN v1.5), so to track provenance at a coarse-grained level and allow querying only data of specific predicate models using SPARQL clauses FROM and FROM NAMED. Examples, and related triples, are placed in further separated named graphs (e.g., pm:fn15-ex), and their extraction can be enabled/disabled.

Specific conversion strategies had to be implemented for each predicate model. E.g., in VN, semantic roles (with selectional constraints) and frames have to be propagated from a class to its subclasses, unless redefined in the latter. In PB (and NB), the instantiation of pmopb:SemanticRoles requires creating an individual for each \( \langle \text{pmopb:Role-set}, \text{pmopb:Argument} \rangle \) pair, as no information is provided on which arguments a predicate
Table 3.1. PreMOn Dataset: predicate model statistics.

<table>
<thead>
<tr>
<th>Source-Target</th>
<th>pmob:SemanticClass</th>
<th>pmob:SemanticRole</th>
<th>pmob:Conceptualization</th>
<th>ontolex:LexicalEntry</th>
<th>pmob:Example</th>
<th>nif:AnnotationSet</th>
<th>Core Triples</th>
<th>Example Triples</th>
<th>Inferred Triples</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb10</td>
<td>5,576</td>
<td>76,241</td>
<td>5,576</td>
<td>4,704</td>
<td>9,583</td>
<td>9,583</td>
<td>517,495</td>
<td>339,496</td>
<td>252,738</td>
<td>1,109,729</td>
</tr>
<tr>
<td>pb17</td>
<td>6,181</td>
<td>145,586</td>
<td>6,196</td>
<td>5,187</td>
<td>10,221</td>
<td>10,221</td>
<td>629,932</td>
<td>429,953</td>
<td>371,586</td>
<td>1,431,471</td>
</tr>
<tr>
<td>pb215</td>
<td>8,751</td>
<td>206,701</td>
<td>9,208</td>
<td>7,671</td>
<td>15,832</td>
<td>15,832</td>
<td>926,466</td>
<td>653,266</td>
<td>539,087</td>
<td>2,118,819</td>
</tr>
<tr>
<td>vn32</td>
<td>484</td>
<td>1,396</td>
<td>6,338</td>
<td>4,402</td>
<td>1,396</td>
<td>1,396</td>
<td>236,526</td>
<td>22,404</td>
<td>26,202</td>
<td>285,132</td>
</tr>
<tr>
<td>fn15</td>
<td>1,018</td>
<td>9,633</td>
<td>11,887</td>
<td>9,413</td>
<td>153,589</td>
<td>154,573</td>
<td>332,386</td>
<td>548,291</td>
<td>548,291</td>
<td>861,677</td>
</tr>
<tr>
<td>fn16</td>
<td>1,204</td>
<td>11,251</td>
<td>174,425</td>
<td>10,107</td>
<td>174,425</td>
<td>196,610</td>
<td>411,554</td>
<td>7,122,995</td>
<td>632,890</td>
<td>14,320,246</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>23,214</td>
<td>450,808</td>
<td>52,452</td>
<td>17,195</td>
<td>365,046</td>
<td>388,215</td>
<td>3,035,670</td>
<td>14,320,246</td>
<td>2,414,220</td>
<td>19,770,136</td>
</tr>
</tbody>
</table>

Table 3.2. PreMOn Dataset: mapping statistics.

<table>
<thead>
<tr>
<th>Source-Target (Resource)</th>
<th># Good Mappings</th>
<th># Discarded Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Concept.</td>
<td>Class</td>
</tr>
<tr>
<td>pb17-to-vn32 (pb17)</td>
<td>2,754</td>
<td>3,899</td>
</tr>
<tr>
<td>pb215-to-vn32 (pb215)</td>
<td>4,765</td>
<td>6,186</td>
</tr>
<tr>
<td>nb10-to-vn32 (nb10)</td>
<td>1,081</td>
<td>1,664</td>
</tr>
<tr>
<td>vn32-to-vn31 (vn32)</td>
<td>5,248</td>
<td>–</td>
</tr>
<tr>
<td>vn32-to-on5 (vn32)</td>
<td>3,357</td>
<td>–</td>
</tr>
<tr>
<td>vn32-to-fn15 (sl122c)</td>
<td>3,617</td>
<td>1,713</td>
</tr>
<tr>
<td>vn32-to-fn16 (fn16)</td>
<td>5,416</td>
<td>5,563</td>
</tr>
<tr>
<td>fn15-to-fn16 (fn16)</td>
<td>36,253</td>
<td>33,272</td>
</tr>
</tbody>
</table>

may have (besides explicit occurrence in frame files, in which case semantic role attributes pmob:core/pmobb:core are set to “true”).

We applied the conversion suite on a large collection of resources, producing a comprehensive dataset, namely the **PreMOn Dataset**, containing: PB v1.7 (pb17), PB v2.1.5 released with OntoNotes v5 (pb215), NB v1.0 (nb10), VN v3.2 (vn32), FN v1.5 (fn15), FN v1.6 (fn16), and SemLink 1.2.2c (sl122c). The PreMOn Dataset contains the mappings between semantic classes and roles provided by each predicate model and SemLink, as well as the mappings between VN classes and lexical senses in WordNet 3.1 (wn31) and OntoNotes 5 groupings (on5). Table 3.1 reports the number of individuals (e.g., of pmob:SemanticClasses), relations (e.g., pmob:ClassRel), and triples for the considered predicate models and their versions. Table 3.2 reports the numbers of mappings, which may be lower than the ones in the original resources as we drop mappings involving non-existing semantic classes,
roles, or conceptualizations (e.g., because removed/reorganized in the new versions of the resources, released after the mapping was defined).

All the datasets generated (as well as the PreMOn Ontology) are documented and published online as Linked Open Data on PreMOn website. Data can be accessed in three ways: bulk dataset download; URI dereferencing with content negotiation (RDF and HTML formats supported); and SPARQL querying via the PreMOn endpoint. Figure 3.8 shows an example of query, run through the web interface of the SPARQL endpoint, which returns pairs of conceptualizations in PB 2.1.5 and FN 1.5 that insist on the same lexical entry (?lex) and have no existing mappings (FILTER NOT EXIST clauses). Each result is a mapping suggestion and, to help judging the possible mapping, it contains the definitions of the PB role set and FN lexical unit, the number of annotated examples

```
2  GRAPH pm:pb215 { ?pbConc pmo:evokingEntry ?lex; pmo:evokedConcept ?pbRoleset }
3  GRAPH pm:fn15  { ?fnConc pmo:evokingEntry ?lex; pmo:evokedConcept ?fnFrame }
4  FILTER NOT EXISTS { [] a pmo:ConceptualizationMapping; pmo:item ?pbConc,  
5  [ pmo:evokedConcept [ a pmoFrame; ] ] }
6  FILTER NOT EXISTS { [] a pmo:ConceptualizationMapping; pmo:item ?fnConc,  
7  [ pmo:evokedConcept [ a pmo:Roleset ] ] }
8  BIND (EXISTS { [] a pmo:ConceptualizationMapping ; pmo:item ?pbConc , ?conc .  
10  { SELECT ?pbRoleset (GROUP_CONCAT(?def; SEPARATOR=" | ") AS ?pbDef)  
11  WHERE { ?pbRoleset skos:definition ?def } GROUP BY ?pbRoleset }
12  { SELECT ?fnConc (GROUP_CONCAT(?def; SEPARATOR=" | ") AS ?fnDef)  
13  WHERE { ?fnConc skos:definition ?def } GROUP BY ?fnConc }
14  { SELECT ?pbConc (COUNT(*) AS ?pbE)  
15  WHERE { [] a nif:Annotation; pmo:valueObj ?pbConc } GROUP BY ?pbConc }
16  { SELECT ?fnConc (COUNT(*) AS ?fnE)  
17  WHERE { [] a nif:Annotation; pmo:valueObj ?fnConc } GROUP BY ?fnConc }
18  } ORDER BY DESC(?chain) ?pbConc ?fnConc
```

Figure 3.8. Example of SPARQL query on PreMOn Dataset.
for both conceptualizations, and whether a chain of two mappings exists that allows connecting them. This query hints a way to exploit PreMOn to investigate, and possibly extend, mappings between predicate models. As illustrated by the query examples on PreMOn website, other interesting queries can be answered given PreMOn data as, e.g., to map a semantic class or role (for a lexical entry) from a resource to another, possibly navigating chains of mappings via SPARQL property paths, or, together with WordNet RDF data, to find synonymous lexical entries that can extend a resource lexicon.

### 3.4 FrameBase Mappings

We map semantic classes and roles of FN 1.5, PB 2.1.5, and NB 1.0—the most recent versions with trained models available for Semafor and Mate-tools SRL systems—to ontological classes and properties of FrameBase, a frame-based ontology based on FN 1.5 and WordNet. Specifically:

- FrameBase *microframe* classes (8151 total) are mapped to pmo:Conceptualizations of pmol:SemanticClasses (at most one microframe class per conceptualization), representing the mapping by (i) typing the pmo:Conceptualization instance as ontolex:LexicalSense\(^5\) and linking it to the FrameBase microframe class via property ontolex:reference, and (ii) linking the pmol:SemanticClass individual, which is also a ontolex:LexicalConcept, to the microframe class via property ontolex:isConceptOf.

- FrameBase properties (9632 total) are mapped to pmol:SemanticRoles, representing the mapping by linking them via property ontolex:isConceptOf.

While the mappings between FN and FrameBase were readily available, for the mappings between PB/NB and FrameBase we implemented a specialized mapping procedure, which we describe in the following. Table 3.3 reports the numbers of pmol:Conceptualizations and pmol:SemanticRoles mapped to FrameBase concepts. FrameBase mappings are included in the PreMOn Dataset and are available as an independent download on PreMOn website. Figure 3.9 graphically depicts the PreMOn Dataset including the FrameBase mappings.

#### 3.4.1 Mapping FrameNet to FrameBase

The mapping from FN 1.5 to FrameBase is straightforward. Indeed, the class hierarchy of FrameBase is based on FN 1.5 frame hierarchy, augmented with leaf *microframe* classes that group all the lexical units of a frame that express the same ‘meaning’, as encoded by WordNet synsets associated to pmofn:LexicalUnit by FrameBase authors. Each and every

\(^5\)As discussed in Section 3.1.2, pmol:Conceptualizations and ontolex:LexicalSenses differ only because an ontolex:LexicalSense must refer exactly to one ontological concept (and possibly to multiple ontolex:LexicalConcepts), while a pmol:Conceptualization may refer to multiple ontological concepts (and exactly one ontolex:LexicalConcept). As in this case both ontolex:LexicalSense and pmol:Conceptualization refer exactly to one ontological concept (and one ontolex:LexicalConcept), we merge them together in a single individual, something that is not prohibited in the PreMOn Ontology.
Table 3.3. FrameBase mapping statistics (number and percentage of mapped elements, excluded role percentage as not meaningful due to modifiers and non-core roles).

<table>
<thead>
<tr>
<th></th>
<th>nb10</th>
<th>pb215</th>
<th>fn15</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmo:Conceptualization</td>
<td>596 (10.69%)</td>
<td>2,246 (24.39%)</td>
<td>11,887 (100%)</td>
</tr>
<tr>
<td>pmo:SemanticRole</td>
<td>491</td>
<td>3,083</td>
<td>9,632</td>
</tr>
<tr>
<td>FrameBase microframe classes</td>
<td>596 (7.31%)</td>
<td>2,246 (27.55%)</td>
<td>8,151 (100%)</td>
</tr>
<tr>
<td>FrameBase properties</td>
<td>252 (2.61%)</td>
<td>900 (9.34%)</td>
<td>9,632 (100%)</td>
</tr>
</tbody>
</table>

Figure 3.9. Graphical depiction of PreMOn Dataset including FrameBase mappings.

A lexical unit is associated to exactly one frame, and thus it is also mapped to exactly one FrameBase microframe class. As the property hierarchy of FrameBase reflects exactly the hierarchy of frame elements in FN, each `pmofn:FrameElement` can be uniquely mapped to a FrameBase property, realizing a one-to-one mapping.

### 3.4.2 Mapping PropBank and NomBank to FrameBase

In principle one could reuse the manual mappings from PB/NB to FN (from SemLink, PB and NB, part of the PreMOn Dataset), or rely on the automatic mappings in the Predicate Matrix (currently not included in the PreMOn Dataset), and then apply the mappings to FrameBase described in Section 3.4.1. However, existing mappings do not cover nominal predicates and (especially for the Predicate Matrix, due to its automatic construction) may be inconsistent with the association of WordNet synsets to FN lexical units underlying the FrameBase hierarchy. Thus, we directly map PB and NB to FrameBase using a specific procedure starting from the high quality manual mappings of SemLink, PB and NB.
Mapping semantic classes  We first map PB/NB semantic classes to FrameBase ontological classes. For each PB/NB pm:Conceptualization $c$ (e.g., pm:co-v-look-pb17-look.01) having lexical entry $le_c$ (e.g., pm:v-look), we build a set $F_c$ of FN lexical units and a set $S_c$ of WordNet synsets:

- Set $F_c$ contains all the FN lexical units that can be reached from $c$ by navigating chains of pm:ConceptualizationMappings from NB, PB, and SemLink; if this leads to an empty set, we use as $F_c$ the set of all FN lexical units for the lexical entry $le_c$.

- Set $S_c$ contains all the WordNet synsets compatible with the part-of-speech of $le_c$ and reachable from $c$ by navigating chains of pm:ConceptualizationMappings. As we have WordNet mappings only for PB rolesets, for NB rolesets we compute $S_c$ by (i) identifying all the PB rolesets mapped to $c$, (ii) taking the verb synsets mapped to them, and (iii) mapping these verb synsets to noun synsets using WordNet relations ‘nominalization’, ‘participle of’, ‘pertainym’ and ‘derived’. In any case, if $S_c$ is empty we fall back to assign $S_c$ all the synsets defined in WordNet for the lexical entry $le_c$.

Given $c$, $F_c$, and $S_c$, we restrict $F_c$ to $F'_c$ by keeping only the lexical units that are mapped to a synset in $S_p$ in FrameBase, so to enforce mappings that are consistent with the assignment of synsets backing FrameBase. If $F'_c$ consists of a singleton lexical unit $lu$, then we record a mapping from conceptualization $c$ to the FrameBase microframe for $lu$. As a result, we obtained FrameBase mappings for 2,246 PB and 596 NB conceptualizations.

Mapping semantic roles  We map PB/NB semantic roles to FrameBase properties using two techniques that guarantee compatibility with previously found class mappings.

First, for each PB/NB semantic role $r$ we identify a set of corresponding FN pmofn:Frame-Elements $E_r$ by considering all the possible chains of pm:SemanticRoleMappings between NB, PB, VN, and FN semantic roles, taking only the chains where the source is a PB/NB semantic role $r$, the target role is a FN frame element $e$, and both $r$ and $e$ refer to semantic classes for which a chain of pm:ConceptualizationMappings can be found leading from a conceptualization of the PB/NB class (roleset) to a conceptualization of the FN class (frame). In case $E_r$ contains a singleton frame element $e$, we map the role $r$ to the FrameBase property corresponding to $e$ (as of Section 3.4.1). As a result, we obtained FrameBase mappings for 182 PB and 80 NB semantic roles.

Second, we learn mappings by annotated data. We start from the 154,485 FN 1.5 example sentences manually annotated with a gold FN frame and corresponding frame elements. We process these sentences with Mate-tools, obtaining automatic (and thus noisy) annotations of PB/NB rolesets and semantic roles. We keep only 106,777 sentences where the predicate token is annotated with both a PB/NB roleset and a FN frame, with the corresponding conceptualizations previously mapped to the same FrameBase microframe class. For each PB/NB semantic role $r$ we build the set $T_r$ containing the head tokens of markables automatically annotated with role $r$. Similarly, for each FN frame
element $e$ we build the set $T_e$ containing the head tokens of markables gold-annotated with $e$. For each pair $\langle r, e \rangle$ we check two conditions:

$$
(1) \quad |T_r| \geq k' \\
(2) \quad \frac{|T_r \cap T_e|}{|T_r|} \geq k''
$$

where $k'$ and $k''$ are two manually-set thresholds. If the two conditions are met, we map the PB/NB semantic role $r$ to the FrameBase property corresponding to $e$.

In order to tune $k'$ and $k''$, we compare the semantic role mappings computed as above with the ones obtained with the first technique, which we treat as a gold standard since obtained using exclusively manual mappings. Let $S'$ be the set of PB/NB semantic roles mapped with the first technique, $S''$ the set of semantic roles mapped with the second technique, and $S$ the set of true positive semantic roles that both techniques mapped to the same FrameBase property. We compute precision $P = \frac{|S'|}{|S'' \cap S'|}$, recall $R = \frac{|S|}{|S'|}$ and $F_1 = \frac{2PR}{P+R}$ using different values of $k'$ and $k''$, and settle on the pair $k' = 2$ and $k'' = 0.5$ that provides the best $F_1$ ($P = .942$, $R = .551$, $F_1 = .695$). As a result, we obtained FrameBase mappings for additional 2,901 PB and 411 NB semantic roles.

### 3.5 Related Work

The use of SW technologies and the LOD paradigm have already been recognized as particularly beneficial to linguistic resources (see, e.g., Chiarcore et al., 2013), leading to the creation of different linguistic ontologies and datasets populating the Linguistic LOD cloud, such as lemon that we use here. Despite this, and notwithstanding the fact that lemon already sets the basis for modeling the lexical entries referenced in predicate models and their links to ontological concepts, few attempts have been made for representing predicate models in RDF.

To the best of our knowledge, no RDF versions of PB and NB exist. An ontological model of VN, called OntoVerbNet, is presented by Gangemi (2010), but no RDF data is available. An RDF version of FN 1.5 is available\(^7\) (Nuzzolese et al., 2011), but its schema is not aligned to lemon and instead closely mirrors the structure and naming used in FN frame files. Finally, VN and FN have been exposed in RDF/lemon as part of lemonUby (Eckle-Kohler et al., 2015), a lemon version of the Uby resource integrating a number of linguistic datasets, but several modeling decisions\(^8\) are no more aligned with the latest developments of lemon by the W3C Ontology-Lexica Community Group (besides, only RDF data for an old FN version is available).

\(^6\)In the denominator we consider only the mappings by the second technique that can be compared with the ones of the first technique, hence the $S' \cap S''$ intersection.

\(^7\)http://ontologydesignpatterns.org/ont/framenet/html/

\(^8\)E.g., modeling of semantic classes as ontolex:LexicalSenses and owl:sameAs links between syntactic and semantic arguments.
3.6 Summary

PreMOn is both a lemon, NIF, and KEM (Chapter 4) compliant ontology for representing predicate models with their common aspects, specificities, and mappings, and a LOD dataset based on that ontology that contains interlinked predicate data for PropBank, NomBank, VerbNet, FrameNet, and SemLink, as well as the mappings of semantic classes and roles to ontological concepts of FrameBase.

Compared to the current situation where each predicate model has its own terminology, structure and proprietary XML format, we believe that PreMOn contribution of a single RDF/OWL ontological model for heterogeneous resources brings several benefits to users of predicate models, within and outside the SW area:

1. ease of access and reuse of predicate model data, due to the adoption of a common RDF format, stable URIs, and LOD best practices;
2. possibility to abstract and capture the aspects common to different predicate models, while at the same time keeping track of the peculiarities of each model (using RDFS/OWL subclass/subproperty primitives);
3. possibility to apply SW technologies to predicate model data, such as automated reasoning and SPARQL querying. In particular, PreMOn makes possible the joint querying of data from different resources, as shown in Section 3.3;
4. possibility to combine PreMOn with other linguistic ontologies, e.g., for providing the SRL annotations of a text according to NIF;
5. possibility for third parties to publish and interlink their datasets with PreMOn, extending it in a decentralized way (e.g., with new mappings).

In turn, these capabilities opens up new opportunities for analyzing, validating, and possibly cleaning up or extending predicate model data, as we exemplified with the identification and removal of inconsistent mappings (Section 3.3) and the development of the FrameBase mappings (Section 3.4).

By extending lemon, PreMOn endorses the latter’s goal to provide proper linguistic grounding for ontologies. In particular, PreMOn enables to ground event and frame ontologies, such as FrameBase, on a proper representation of the linguistic information for predicates, thus supporting the development of a comprehensive catalog, from the linguistic to the knowledge level, of frame and frame participant types. We believe that this catalog is of particular interest for knowledge extraction tools employing SRL and SW techniques, as it provides a convenient way to represent, query, and convert between predicate models and the SRL annotations they work with, allowing their mapping to ontological concepts. Indeed, in Chapter 5 we leverage these capabilities to build PIKES (Chapter 5), the Frame-based Ontology Population tool presented in this thesis.
Chapter 4

KEM: an Ontological Model for Knowledge Extraction

Ontology Population, and Knowledge Extraction (KE) in general, requires dealing with different types of interlinked information: the unstructured resources knowledge is extracted from; their Natural Language Processing (NLP) annotations; and the extracted knowledge.

In this chapter we present KEM (Knowledge Extraction Model),\(^1\) an ontological model that supports the RDF/OWL representation and alignment of all the aforementioned information, and serves as the basis for the Frame-based Ontology Population approach and storage solution proposed in this thesis. KEM is built around the semiotic notions of reference and meaning, and consists of two modules: KEM Core and KEM Text.

KEM Core (Section 4.1) introduces, in a media-independent way (i.e., not considering just text), the core abstractions of resource, mention, and instance, each one being the root of a corresponding representation layer. Resources and instances (with their triples) are respectively the input and output of KE, while mentions are pieces of a resource referring to instances or triples, and enriched with semantic annotations that describe their meaning. Together, mentions, instances, and semantic annotations realize the so-called triangle of reference\(^2\) used in semiotics to represent the meaning of signs (as mentions are). KEM Core is aligned to DOLCE (Gangemi et al., 2002) and LMM (Linguistic Meta-Model, Picca et al., 2008), and comes with a mechanism based on named graphs for linking mentions to triples and for attaching arbitrary metadata to contents of each representation layer.

KEM Text (Section 4.2) specializes the notions of resource, mention, and annotation from KEM Core for their use in scenarios of KE from text. This is done by reusing several concepts from NIF (NLP Interchange Format, Hellmann et al., 2013), including the notion of (sub-)string and annotation, and by modeling specific mentions and annotations subclasses after relevant NLP formats, such as TimeML (Pustejovsky et al., 2010), and tasks, such as SRL (Semantic Role Labeling) and Coreference Resolution.

Overall, KEM builds on several state-of-the-art vocabularies and ontological distinctions, providing an integrated, ready-to-use solution (differently, e.g., from LMM) for representing all the input, output, and intermediate data involved in KE from text (Section 4.3).

Acknowledgments The material presented in this chapter results from the integration, re-engineering (notably, addition of annotations), and extension (e.g., new mention types) of models originally proposed in (Corcoglioniti et al., 2016c, 2015c).

\(^{1}\)Available at: http://knowledgestore.fbk.eu/ontologies/kem/.

39
Indonesia Hit By Earthquake
A United Nations assessment team was dispatched to the province after two quakes, measuring 7.6 and 7.4, struck west of Manokwari Jan. 4. At

4.1 KEM Core
Namespace: http://knowledgestore.fbk.eu/ontologies/kem/core#
Prefix: kem

KEM Core is the main, media-independent module of KEM. It introduces the interlinked Resource, Mention, and Instance representation layers, which are schematically shown and exemplified in the UML-like diagram of Figure 4.1. We present these layers next (Sections 4.1.1, 4.1.2, and 4.1.3), reporting also their alignments to other ontologies (Section 4.1.4) and the proposed RDF representation using named graphs (Section 4.1.5)

4.1.1 Resource Layer

The Resource layer comprises the unstructured content from which knowledge is extracted, and consist of kem:Resources. A kem:Resource is defined as a self-contained, globally identified, and immutable information object:

- **information object** has to be understood in an abstract sense, not considering the way information is physically realized (i.e., the actual sequence of bytes encoding the information), in line with a commonly made ontological distinction;
- **self-contained** means that a kem:Resource represents a complete and meaningful unit of information for humans (e.g., a whole news article instead of an arbitrary fragment of text in that article); however, it does not imply that the unit is atomic and, indeed, a kem:Resource can be a kem:CompositeResource having other resources as components (e.g., a Web page containing a text and several images);
- **identifiable** means that every resource is assigned a unique identifier (e.g., URI, DOI, ISBN) that permits to unambiguously refer to it and provides an identity criterion;
• **Immutable** means that the abstract encoding of a resource (i.e., the words in a text, regardless of their physical representation) does not change in time. This assumption allows not adding a time index to properties involving `kem:Resource` and their fragments (which are themselves immutable), thus simplifying the resulting model.³

A `kem:Resource` can be described in terms of its content (e.g., the text of a textual resource) and the associated metadata, possibly exploited for KE, which can be encoded using properties of popular metadata vocabularies such as Dublin Core⁴ (e.g., `dcterms:title`, `dcterms:created`, `dcterms:creator`, `dcterms:publisher`, `dcterms:rights`). The core module of KEM does not prescribe a specific way to model content and metadata, as both depend on the specific kind of `kem:Resource` considered; in particular, for several media (e.g., image, audio, video), the representation of content cannot be done within an RDF/OWL ontology, but requires associating some external, content-specific file (e.g., image file) to the `kem:Resource` ontological individual, as done in the KnowledgeStore (Chapter 6).

### 4.1.2 Instance Layer

The Instance layer is the home of structured content, such as the content resulting from knowledge extraction. Differently from the Resource layer, this layer aims at providing a formal and concise representation of the world, abstracting from the many, redundant, and ambiguous ways it can be encoded in unstructured content.

The Instance layer contains instances (class `kem:Instance`) of persons, organizations, locations, frames, dates and other entities of the domain of discourse. These instances are described and related one to another by means of RDF triples, grouped in named graphs (class `kem:Graph`) treated as ontological individuals to which (metadata) assertions can be attached. KEM does not impose a specific set of vocabularies for describing instances, and does not specify how triples should be formed.⁵ Rather, KEM provides a way to ground instances and their triples to the (fragments of) resources they were extracted from.

### 4.1.3 Mention Layer

The Mention layer comprises annotated fragments of resources (class `kem:Fragment`) and, among them, annotated fragments called mentions (class `kem:Mention`) that are about some instances or graphs of triples in the Instance layer, overall realizing the connection between the Resource and the Instance layers.

**Fragments and annotations** The most basic concepts introduced in the Mention layer are `kem:Fragment` and `kem:Annotation`, which together provide a general-purpose mechanism for marking and annotating arbitrary fragments of resources.

³In scenarios where mutations are of interest (e.g., evolving documents), one can introduce/reuse a concept of mutable resource and treat `kem:Resource` as a specific snapshot/version of it in time.


⁵This aspect is however addressed in the ontology population approach described in Chapter 5.
A `kem:Fragment` is a piece of a `kem:Resource`, and its identity is defined by that resource (property `kem:fragmentOf`) and a set of media-specific location attributes, specified in `kem:Fragment` subclasses, that allow precisely locating the fragment within the resource (e.g., character offsets for a text resource). This identity criterion implies that two fragments cannot be identical if they refer to different resources, or even to the same resource but to different locations of it, independently from other properties being equal, such as the textual (or other media) surface form of the fragment. This is consistent with the fact that fragments having the same surface form but appearing in different contexts may be interpreted and processed differently one to another. A consequence of the dependency of fragment identity on location attributes is that a resource and its maximal fragment (covering the whole resource) are distinct ontological individuals, i.e., `kem:Fragment` and `kem:Resource` are disjoint classes. A part-of relationship can be defined among fragments by requiring that they belong to the same resource and one fragment contains the other. Disjoint fragments of the same resource can be composed in a `kem:CompositeFragment`, a feature useful, e.g., to select non-contiguous pieces of text within a text resource.

A general annotation mechanism is provided by class `kem:Annotation`. A `kem:Annotation` is an arbitrary piece of information that annotates a `kem:Fragment` via property `kem:hasAnnotation` (e.g., for marking it as a temporal expression) or, if the fragment cannot be identified for that annotation, a `kem:Resource` via property `kem:hasResourceAnnotation` (e.g., for associating the resource to the implicit temporal expression corresponding to its creation time). No identity criteria are specified for `kem:Annotations`.

**Mentions and semantic annotations** Differently from `kem:Fragment` and `kem:Annotation`, which refer to arbitrary resource fragments and annotations, their specializations `kem:Mention` and `kem:SemanticAnnotation` are given a precise semantics by KEM.

A `kem:Mention` is a resource fragment that is about (property `kem:isAbout`) some `kem:Instances`, and/or that conveys (property `kem:conveys`) some graph of triples. While `kem:conveys` links a mention to any graphs of triples formalizing facts that are represented in the mention, `kem:isAbout` links a mention to any instance that is represented, directly or indirectly, by that mention. This primarily includes the semiotic notion of reference (specialized property `kem:refersTo`) between a mention acting as a sign (e.g., a proper name in a text) and the instance indicated by that sign (e.g., the person carrying that name). There are however other cases where a mention, besides having a referent, also implies or otherwise points to other instances that are not their referent. For instance, the word “American” in “An American in Paris” refers to some unspecified person, but

---

6Annotations can be also expressed by directly predicating over the fragment or resource being annotated (a solution used in previous versions of the model). The introduction of an annotation individual is mandatory when an anchor is needed to refer to the annotation as a whole, and also for differentiating between multiple annotations of the same type attached to the same resource or fragment, a case arising, e.g., when a single text fragment is annotated with multiple FrameNet (Baker et al., 1998) frame elements.

7In KEM we do not consider the philosophical distinction about whether the referent actually exists, it is an imaginary entity, or cannot exist at all (as for the mention “the current king of France”).
also is about America as a geopolitical entity, as well as the people-by-location semantic frame expressing the provenance of that person from America. Clearly, a resource may contain many mentions, each one associated to some instance or graph of triples, and an instance or graph of triples may have multiple associated mentions within the same resource, denoting the fact that it occurs multiple times in the resource.

The specialization of \texttt{kem:Fragment} into \texttt{kem:Mention} is mirrored by the specialization of \texttt{kem:Annotation} into \texttt{kem:SemanticAnnotation}. A \texttt{kem:SemanticAnnotation} is an annotation that describes the meaning of a resource or of a mention of it, and that may substantiate (property \texttt{kem:substantiates}) some triples extracted from text. Often, the description of a \texttt{kem:SemanticAnnotation} concerns a specific subject instance (property \texttt{kem:subject}) of the Instance layer, as in case of annotations of named entities, temporal expressions, or events in a text. The description encoded in the semantic annotation may also involve (property \texttt{kem:involves}, super-property of \texttt{kem:subject}) additional instances beyond the subject one, as in case of the source and target instances of some relation annotation (whose reification is the subject of the annotation). While \texttt{kem:involves} allows directly relating a \texttt{kem:SemanticAnnotation} to the \texttt{kem:Instances} it involves, during KE these instances may be initially unknown, and the only information available is that they are the subject of another semantic annotation, or the referent of another mention. Properties \texttt{kem:involvesSubjectOf} and \texttt{kem:involvesReferentOf} are thus introduced to represent these cases. To relate all these properties, the following OWL 2 property chain axioms are defined (DL notation):

\[
\begin{align*}
  & \texttt{kem:involvesSubjectOf} \circ \texttt{kem:subject} \sqsubseteq \texttt{kem:involves} \\
  & \texttt{kem:involvesReferentOf} \circ \texttt{kem:refersTo} \sqsubseteq \texttt{kem:involves} \\
  & \texttt{kem:hasAnnotation} \circ \texttt{kem:subject} \sqsubseteq \texttt{kem:isAbout} \\
  & \texttt{kem:hasAnnotation} \circ \texttt{kem:substantiates} \sqsubseteq \texttt{kem:conveys}
\end{align*}
\]

Different subclasses and attributes of \texttt{kem:Mention} and \texttt{kem:SemanticAnnotation} can be defined to represent the results of the content analyses of a resource (e.g., its NLP processing). These subclasses and attributes are not part of KEM Core, but are instead provided by its specializations for different media (e.g., text).\footnote{Some relevant types of mentions or semantic annotations might be possibly defined in a media-independent way in KEM Core, but since we currently focus on text we leave this task as future work.}

4.1.4 Alignments

We provide here the alignments between concepts of KEM Core and corresponding concepts of LMM and DOLCE. These alignment are not hardcoded in KEM Core but are provided in additional ontology files that can be imported on-demand where necessary.

\textbf{LMM} Classes \texttt{kem:Mention}, \texttt{kem:SemanticAnnotation}, and \texttt{kem:Instance}, together with their interlinking properties \texttt{kem:hasAnnotation}, \texttt{kem:subject}, and \texttt{kem:refersTo} realize the semiotic triangle of reference and as such can be aligned to the concepts of LMM that
provide an abstract formalization of such triangle.\(^9\) Classes can be aligned using the following OWL 2 axioms:

\[
\begin{align*}
\text{kem: Mention} & \sqsubseteq \text{lmm: Expression} \\
\text{kem: SemanticAnnotation} & \sqsubseteq \text{lmm: Meaning} \\
\text{kem: Instance} & \sqsubseteq \text{lmm: Reference}
\end{align*}
\]

Only two properties can be aligned using OWL 2 constructs:

\[
\begin{align*}
\text{kem: refersTo} & \sqsubseteq \text{lmm: denotes} \\
\text{kem: subject} & \sqsubseteq \text{lmm: isInterpretationOf}
\end{align*}
\]

The remaining \text{kem:hasAnnotation} requires the use of a rule:

\[
\text{kem:hasAnnotation}(x, y) \land \text{kem:SemanticAnnotation}(y) \rightarrow \text{lmm:hasInterpretant}(x, y)
\]

\text{DOLCE}  \text{ kem:Resources} and \text{kem: Fragments} are information objects, in DOLCE terminology, while the \text{description} and \text{situations} pattern included in DOLCE can be used to characterize \text{kem: SemanticAnnotations} and \text{kem: Graphs}: the first are descriptions, the latter are the situations (i.e., real-world state of affairs) described by the first. These class alignments can be formalized in OWL 2 as follows:

\[
\begin{align*}
\text{kem: Resource} & \sqsubseteq \text{dul: InformationObject} \\
\text{kem: Fragment} & \sqsubseteq \text{dul: InformationObject} \\
\text{kem: SemanticAnnotation} & \sqsubseteq \text{dul: Description} \\
\text{kem: Graph} & \sqsubseteq \text{dul: Situation}
\end{align*}
\]

Consequently, the following property alignments can be established using OWL 2 axioms:

\[
\begin{align*}
\text{kem: refersTo} & \sqsubseteq \text{dul: isAbout} \\
\text{kem: subject} & \sqsubseteq \text{dul: describes} \\
\text{kem: involvesSubjectOf} & \sqsubseteq \text{dul: isRelatedToDescription} \\
\text{kem: substantiates} & \sqsubseteq \text{dul: isSatisfiedBy}
\end{align*}
\]

As for LMM, the alignment of \text{kem:hasAnnotation} requires the use of a rule:

\[
\text{kem:hasAnnotation}(x, y) \land \text{kem:SemanticAnnotation}(y) \rightarrow \text{dul:expresses}(x, y)
\]

\(^9\)A parallel can also be drawn with the \text{sense} vs. \text{reference} distinction in semiotics and philosophy of language—see, e.g., \url{http://en.wikipedia.org/wiki/Sense_and_reference} and (Frege, 2000). While \text{kem:refersTo} and its \text{kem:Instance} are associated to the notion of reference, a \text{kem:SemanticAnnotation} with its attributes can be related to the notion of sense, as it describes the meaning of a specific occurrence of an instance in a resource. E.g., mentions “Barack Obama,” “US president,” and “Nobel peace prize winner” have the same referent (the latter two under certain circumstances) but different senses as they highlight different roles and aspects of the referent, which can be captured by corresponding semantic annotations.
4.1.5 Named Graphs

As previously shown in Figure 4.1, the linking of mentions and semantic annotations to triples of the Instance layer is realized by grouping triples in *named graphs*, and then using the identifier of the graph as target of these links. Since named graphs can be used for any triple, including triples expressing contents at the Resource and Mention layers, the named-graph mechanism proposed in KEM can be generalized to allow attaching arbitrary metadata to triples at any representation layer. In detail, metadata triples are attached to a named graph containing KEM ABox triples (either using the graph URI as subject or object of these triples) and apply, by convention, to all the triples in the graph.

Two types of metadata are especially useful in a KE scenario: *confidence* of information at each layer (e.g., some NLP annotation); and *provenance* of a piece of information, in terms of agent producing it (e.g., a specific NLP tool), and duration, location (e.g., system or machine), and other metadata characterizing the activity producing that information. To express these kinds of metadata, we propose reusing terms from PROV (McGuinness et al., 2013), Dublin Core Terms, and NIF, as shown in Figure 4.2.

4.2 KEM Text

**Namespace:** http://knowledgestore.fbk.eu/ontologies/kem/text#

**Prefix:** kemt

KEM Text is the module of KEM covering KE from text media. As shown in Figure 4.3, it builds on KEM Core and NIF, to which it is compatible. NIF provides terms for describing text annotations (nif:Annotation) and fragments (nif:String), as well as the structure of text in terms of paragraphs, sentences, words; it also provides URI schemes for naming text fragments, such as the ⟨document_uri#char=begin,end⟩ scheme based on RFC 5147.10

KEM Text introduces several subclasses of kem:SemanticAnnotation for modeling the entities and the semantic relations among them that can be extracted from text via NLP. These semantic annotations can be instantiated based on NLP outputs, and aim at: (i) generalizing the heterogeneous annotations produced by tools for typical NLP tasks, usually coming in different formats, thus enabling to uniformly handle the different types

---

Figure 4.3. KEM Text (bottom) derivation from KEM Core (top): (1) KEM Text – Main concepts; (2) Syntax; (3) Named entities, coreference and coordination; (4) Time expressions, events and factuality; (5) Semantic Role Labeling.
of linguistic annotations produced; and (ii) representing, in a structured form, all the
linguistic information needed to extract the knowledge conveyed by the text, such that
access to the original text is not needed for further processing.

In the following, we detail KEM Text starting from the main concepts it introduces
extending KEM Core and NIF, from which other KEM Text concepts derive from (Sec-
tion 4.2.1). We then present the modeling of syntax, based on NIF (Section 4.2.2), followed
by an account of different subclasses of semantic annotations (Sections 4.2.3, 4.2.4, 4.2.5).

4.2.1 Main Concepts

KEM Text builds on some main concepts (Figure 4.3, box 1) that specialize KEM Core
for the text media, reusing where appropriate terms from NIF:

- **kemt:TextResource**, refinement of **kem:Resource**, represents a textual resource con-
sisting of plain text without any structure or markup, described with Dublin Core
attributes such as **dct:title**, **dct:creator**, and **dct:created** (the document creation time).

- **nif:String** (from NIF) is declared as a specialization of **kem:Fragment**, as it represents
a string within a larger textual context. NIF attributes **nif:beginIndex** and **nif:endIndex**
are used to locate the string within the containing **kemt:TextResource**, with data
property **nif:anchorOf** providing the substring associated to the fragment; these
properties are specified both for consecutive strings (the only ones considered by
NIF) and for non consecutive strings, whose structure is modeled by typing them
(also) as **kem:CompositeFragment** and linking them to their component **nif:Strings**.
We also add a property **kemt:lexicalEntry** for linking a **nif:String** to the corresponding
**ontolex:LexicalEntry** individual in some external lexicon (type/token relation).

- **nif:Context** (from NIF) is the maximal fragment associated to a **kemt:TextResource**. To achieve
full compatibility with NIF, three requirements must be satisfied: (i) a
**nif:Context** must be always present for a **kemt:TextResource**; (ii) property **nif:sourceURL**, subproperty of **kem:fragmentOf**, must link a **nif:Context** to its **kemt:TextResource**; and
(iii) all the **nif:String** fragments of a **kemt:TextResource** (including the context itself)
must be also associated to the **nif:Context** via property **nif:referenceContext**. Two
OWL 2 property chain axioms help satisfying these constraints, by exploiting the
redundancies among **nif:sourceURL**, **nif:referenceContext**, and **kem:fragmentOf**:\(^{11}\)

\[
\text{km:fragmentOf} \circ \text{nif:sourceURL} \sqsubseteq \text{nif:referenceContext} \\
\text{nif:referenceContext} \circ \text{km:fragmentOf} \sqsubseteq \text{nif:sourceURL}
\]

- **nif:Annotation** (from NIF) is declared as a specialization of **kem:Annotation**, and
property **nif:annotation** is declared as a subproperty of **kem:hasAnnotation**. We also

\(^{11}\)A property chain axiom **nif:referenceContext** \circ **nif:sourceURL** \sqsubseteq **km:fragmentOf** can be also defined, but
this would require property **nif:sourceURL** to be exclusively used to link a **nif:Context** to a **kemt:TextResource**.
add two properties `kem:rawString` and `kem:signalString` to link annotations to strings: the first permits keeping track of the string originally annotated by NLP tools before any normalization is applied (e.g., to align the string to tokens, or to discard irrelevant tokens such as determiners and prepositions); the second models a generic notion of *signal*, inspired by TimeML, that links an annotation to some fragment of text supporting it (e.g., an “after” token supporting a temporal relation annotation).

- `kem:EntityAnnotation` and `kem:RelationAnnotation` are introduced as subclasses of both `kem:SemanticAnnotation` and (being them annotation of strings) `nif:Annotation`. A `kem:EntityAnnotation` describes an entity lexicalized in the text (e.g., a named entity), while `kem:RelationAnnotation` relates two or more `kem:EntityAnnotations` (via property `kem:relates`) and describes a binary or n-ary semantic relation between the entities associated to these annotations (e.g., coreference). Together, these classes are the roots from which several other semantic annotation subclasses are derived.

### 4.2.2 Syntax

NIF provides several terms for modeling fragments of text with specific syntactic roles, which we reuse in KEM Text (Figure 4.3, box 2): `nif:Paragraph`, `nif:Sentence`, `nif:Word` (a token) and its properties, which include `nif:oliaLink` for attaching arbitrary data categories from the OLiA (Ontologies of Linguistic Annotation, Chiarcos and Sukhareva, 2015) ontology, such as part-of-speech, and the pair `nif:dependency` and `nif:dependencyRelationType` supporting the representation of the dependency parse tree. We complement these concepts by: (i) introducing a notion of `kem:ConstituentString`, to which `kem:ConstituentNodes` can be attached to in order to represent also the constituency parse tree; and (ii) supporting the representation of word decomposition into components (classes `kem:CompoundWord` and `kem:WordComponent`). These extensions, while not central to modeling the semantic relations between text and extracted knowledge, provide KEM Text with the capability of representing also syntactic information captured in annotation formats such as NAF (NLP Annotation Format, Fokkens et al., 2014).

### 4.2.3 Named Entities, Coreference, Coordination

A general mechanism for annotating arbitrary entities in the text is provided by semantic annotation `kem:EntityAnnotation`, with properties `itsrdf:taClassRef` and `itsrdf:taIdentRef`, from ITS (Internationalization Tag Set, Filip et al., 2013), supporting respectively the representation of category and entity linking information. The semantic annotation of named entities, resulting from the NERC (Named Entity Recognition and Classification) NLP task, is supported via `kem:NamedEntity` (Figure 4.3, box 3). This class provides additional properties for representing the name or value associated to the entity: `kem:properName` contains the proper name as mentioned in the text, while the pair `kem:literalValue` and
\texttt{kem:unit} can be used for numeric named entities that consist of a number (e.g., amount of money) and a unit of measure or similar qualifier (e.g., a currency).

The coreference relation between lexicalized entities, resulting from the Coreference Resolution NLP task, is modeled with class \texttt{kem:Coreference}, whose individuals are related to the \texttt{kem:EntityAnnotations} they corefer via \texttt{kem:coreferring} links. If only an indication of coreferring text spans is available, property \texttt{kem:coreferringString} can be used to link those spans to the \texttt{kem:Coreference} individual.

Linguistic relations, such as the aforementioned coreference, often involve spans of text consisting in the \textit{coordination} of several conjuncts (e.g., “X, Y, and Z”), each one possibly annotated with a \texttt{kem:EntityAnnotation}. These coordinated conjuncts, as a whole, represent a group entity that can participate to other relations, and thus it may be useful to annotate the whole text fragment containing these conjuncts with a \texttt{kem:EntityAnnotation}. To relate this annotation, standing for a group of entities, to the annotations of the member entities (one for each conjunct), the semantic annotation class \texttt{kem:Coordination} with properties \texttt{kem:group} and \texttt{kem:conjunct} is defined. As for \texttt{kem:Coreference}, a string variant is offered (property \texttt{kem:conjunctString}) in case only the textual extents of conjuncts are available.

### 4.2.4 Time Expressions, Events, Factuality and Causality

KEM Text provides semantic annotations for temporal elements based on (and fully covering) the TimeML standard\textsuperscript{12}, plus its extensions FactBank (Saurí and Pustejovsky, 2009) and the causality model by Mirza and Tonelli (2014) (Figure 4.3, box 4).

Temporal entities (perdurants, in ontological terms) can be annotated via \texttt{kem:TemporalElement} and its subclasses: \texttt{kem:Timex} for time expressions (e.g., “yesterday,” “April, 26th,” “two months”), and \texttt{kem:EventInstance} for events expressed by verbs, nouns, and adjectives in the text (e.g., the noun “earthquake”). Their properties, among which the normalized time value represented using both the TimeML format (property \texttt{kem:literalValue}) and OWL Time\textsuperscript{13} (property \texttt{kem:objectValue}), are closely based on TimeML, and can be populated based on the output of tools for the automatic annotation of text according to TimeML, such as tools for TERN (Temporal Expression Recognition and Normalization).

TimeML and its extensions define several binary semantic relations (\textit{links}) involving a \texttt{kem:source} and \texttt{kem:target} temporal entities. We model these relations with class \texttt{kem:Link}, specifying the exact relation with property \texttt{kem:relation} and annotating with a \texttt{kem:Link} individual the text fragment embracing the fragments of the source and target arguments. The following subclasses of \texttt{kem:Link} are defined:

- \texttt{kem:TemporalLinks} (or TLink) express temporal relations among temporal entities (e.g., before, simultaneous—Allen relations in general), including also the \textit{beginPoint}, \textit{endPoint}, and \textit{anchorTime} relations between time expressions that are modeled differently (i.e., not as links) in TimeML;

\textsuperscript{12}To the best of our knowledge, KEM Text is the first ontology fully covering TimeML.

\textsuperscript{13}http://www.w3.org/TR/owl-time/
• kemt:SubordinateLinks (or SLink) express subordination relations among kemt:Event-Instances, such as the relation among the event referred by “refused” and the one referred by ”talk” in the text “he refused to talk”;

• kemt:AspectualLinks (or ALink) relate a (source) aspectual kemt:EventInstance to its argument (target) kemt:EventInstance, such as event “started” linked to “talk” in the text “he started to talk”;

• kemt:CausalLinks (or CLink), introduced by Mirza and Tonelli (2014) and used in causal relation extraction, express causal relations between two kemt:EventInstances.

• kemt:FactualityLinks, from FactBank,14 link a kemt:EventInstance to the annotation of a kemt:FactualitySource agent in text, and either express: (i) a factuality judgment by that agent regarding the event (encoded by attribute kemt:factValue); and (ii) the fact that an agent (expressing some factuality judgment) is introduced by a particular event (e.g., in “X believes that Y will talk,” the event “believes” introduces the source “X”); and (iii) the fact that a source-introducing event originates from another source, which allows deriving the nesting relation among sources considered in FactBank;

It is worth noting that TimeML makes use of a particular kemt:Timex temporal expression—the Document Creation Time (DCT)—which is associated to the whole textual resource and not to a specific fragment of it. As this annotation is used in kemt:TemporalLinks, it is useful to explicitly represent it, which can be achieved by linking the DCT kemt:Timex annotation to the kem:Resource individual via kem:hasResourceAnnotation.

4.2.5 Semantic Role Labeling

SRL consists in marking predicates and their arguments in text, which in KEM Text are modeled as semantic kemt:EntityAnnotations using, respectively, classes kemt:Predicate and kemt:Argument (Figure 4.3, box 5): the first describes the event, relation or other structured entity (i.e., semantic frame) the predicate is about, while the latter describes any argument entity (of any kind) associated to the predicate. The connection between a predicate and its arguments is given by the kemt:Participation relation annotation, which annotates a textual fragment embracing the fragments of the predicate and its arguments, to which it is related using properties kemt:predicate and kemt:argument. The semantic class of the predicate and the semantic role of the argument, coming from SRL and identified using individuals of PreMOn (Chapter 3), are attached to the kemt:Predicate and kemt:Argument individuals using properties itsrdf:taClassRef and itsrdf:taPropRef.

An example of using KEM (Core and Text modules) for representing a simple sentence, its SRL annotations, and the instances and RDF graphs extracted from it is shown in Figure 4.4. The whole sentence is represented as kemt:TextResource <r1>, while its fragment “supporters” is modeled as kemt:Mention <r1#char=36,46>. The mention carries three semantic annotations: (i) a kemt:Predicate annotation, referring to NomBank (Meyers et al.,

14 We model factuality relations as binary links for consistency with other binary relations modeled this way, although in FactBank they are not called ‘link’.
“G. W. Bush and Bono are very strong supporters of the fight of HIV in Africa.”

Figure 4.4. Using KEM to represent a sentence, its SRL annotations, and extracted knowledge.

2004) semantic class support.01; (ii) a kem:Argument annotation, referring to the Arg0 (agent) argument of support.01; and (iii) a kem:Participation relation annotation, linking the previous two annotations. These annotations have two kem:Instances as kem:subjects, and kem:substantiate several RDF kem:Graphs; correspondingly, instances and graphs are linked to the mention via properties kem:refersTo, kem:isAbout, and kem:conveys.

4.3 Related Work

The ontologies most close to KEM are LMM (Linguistic Meta-Model, Picca et al., 2008) and GAF (Ground Annotation Framework, Fokkens et al., 2013).

LMM models the semiotic triangle of meaning on top of DOLCE (see Sections 4.1.4 and 2.4). KEM also provides concepts realizing this triangle but, differently from LMM, it provides concrete ways to identify signs (kem:Mentions) in the text based on NIF, and further expands the modeling of meaning, that in KEM is equated to (NLP) annotations of resources conveying semantic information about referred instances.

GAF provides a mechanism—the gaf:denotedBy property—that, combined with NIF and the use of named graphs, allows linking a fragment of text to the ontological instance it refers to and to the RDF triples extracted from it.15 Differently from LMM and KEM, however, GAF does not provide an ontological model of the semantic descriptions (the meaning) of referred instances coming from NLP annotations. To account for these annotations, in general and not specifically from a semantic point of view, GAF has been complemented in practice with NAF, an XML-based format supporting the representation of different NLP annotation layers originated in the NewsReader16 project and aligned to the project’s annotation guidelines (Tonelli et al., 2014).

15This mechanism is proposed especially for triples expressing the participation of instances to events.
16http://www.newsreader-project.eu/
Different proposals allow modeling and annotating specific fragments of resources. NIF (NLP Interchange Format, Hellmann et al., 2013) is an ontology increasingly used for identifying fragments of textual resources and associating them to (some types of) NLP annotations, in combination with other vocabularies such as OLiA (Ontologies of Linguistic Annotation, Chiarcore and Sukhareva, 2015).\footnote{See, e.g., the guidelines at http://bpmlod.github.io/report/nif-corpus/index.html.} EARMARK (Iorio et al., 2011) is another ontology, mainly targeting markup languages such as XML, that can be used to model specific fragments in a text resource. Going beyond textual resources, the Annotation Ontology (Ciccarese et al., 2011) and the Open Annotation Model (Sanderson et al., 2013) provide general-purpose mechanisms for arbitrarily annotating pieces of (multimedia) resources, and their integration is being promoted by the Web Annotation Working Group of W3C.\footnote{http://www.w3.org/annotation/} Common to these proposals, and differently from KEM and LMM, is the fact that they do not cover the semiotic relations between resource fragments and the instances they refer to, and between semantic annotations and the instances they describe.

### 4.4 Summary

KEM is an RDF/OWL ontological model for representing all the contents involved in KE in three interlinked layers: Resource layer, containing the unstructured (e.g., textual) resources that are the input of KE; Mention layer, containing fragments of resources referring to instances or conveying facts about them; and Instance layer, containing the ontological instances and the graphs of RDF triples about them that are the output of KE. KEM is organized in two modules: KEM Core introduces the three layers and their main concepts in a media-independent way, while KEM Text specializes these concept for the scenario of KE from text, building also on NIF to which KEM Text is fully compatible. Alignments of KEM Core concepts to DOLCE and LMM are provided, and a general-purpose mechanism based on named graphs is introduced for representing triple-level metadata at each representation layer. Overall, KEM supports navigating from any piece of extracted knowledge to its mentions in unstructured resources and back, and allows representing all the generated intermediate information (e.g., NLP annotations) and associated metadata (e.g., confidence, provenance).

We reference and leverage KEM in several tasks in the scope of this thesis, in this way demonstrating the practical usability of the model. In Chapter 5, we propose a Frame-based Ontology Population approach built on top of KEM, where the Mention layer is used as an intermediate representation for splitting the population process in two decoupled phases. In Chapter 6, we describe a scalable storage framework for contents involved in KE from text, whose data model is aligned to KEM. In Chapter 9, we reuse this storage framework in the scope of the NewsReader EU project, customizing its KEM-based data model to represent the mention annotations considered in the project.
Chapter 5

PIKES: an Approach for Frame-based Ontology Population from Text

We call Frame-based Ontology Population the task of extracting semantic frames from natural language text to populate an ontology ABox. Semantic frames are defined by RDFS/OWL ontologies, such as FrameBase (Rouces et al., 2015) and ESO (Event Situation Ontology, Segers et al., 2015), and consist in events, situations and other structured entities reified as ontological instances (e.g., a sell event) and connected to related instances via properties specifying their semantic roles in the frame (e.g., seller, buyer). This kind of representation (called neo-Davidsonian, Parsons, 1990) supports expressing n-ary and arbitrarily qualified relations, and permits leveraging Natural Language Processing (NLP) tasks such as SRL (Semantic Role Labeling), which annotates frame-like structures in text consisting of predicates and their semantic arguments as defined by predicate models.

Leveraging KEM (Chapter 4) and PreMOn (Chapter 3), in this chapter we present PIKES, an approach for frame-based ontology population from English text that extracts ontological instances aligned to DBpedia (Lehmann et al., 2015), YAGO (Hoffart et al., 2013), and SUMO (Niles and Pease, 2001), and semantic frames aligned to FrameBase.

Processing in PIKES is decoupled in two independently-tunable phases (Section 5.1). First (phase 1: linguistic feature extraction, Section 5.2), text is processed by several NLP tasks, including SRL, to produce a structured representation of the input text where all the produced NLP annotations are organized in an RDF graph of mentions—spans of text denoting some entities or facts—according to KEM. Then (phase 2: knowledge distillation, Section 5.3), the mention graph is processed using SPARQL-like rules and the PreMOn FrameBase mappings to distill an RDF knowledge graph whose nodes are ontological instances uniquely identifying entities of the world, events or situations, possibly disambiguated against DBpedia, and arcs represent relations between them (e.g., the participation and role of an entity in a frame) modeled after FrameBase semantic frames. This knowledge graph represents the knowledge conveyed by the text, in a way that abstracts from the specific occurrences of entities and relations in it.

We implemented PIKES as an open source (GPL) Java tool that includes and combines the outputs of several NLP tools and two complementary SRL systems—Mate-tools (Björkelund et al., 2009) and Semafor (Das et al., 2014)—to improve the detection of semantic frames; a working online demo is also provided on PIKES website (Section 5.4).

We performed three complementary evaluations of PIKES (Section 5.5). First, we evaluated PIKES quality performances in FrameBase ontology population using a manually

---

1 Abbreviation for PIKES Is a Knowledge Extraction Suite. Website: [http://pikes.fbk.eu/](http://pikes.fbk.eu/).
annotated gold standard: PIKES scores 0.599 for precision, and 0.528 for recall ($F_1 = 0.561$). Second, we evaluated PIKES quality performances on the specific task of detecting and representing semantic frames modeled after traditional predicate models, for comparison with state-of-the-art tools and against the same gold standard: PIKES scores 0.711 for precision and 0.562 for recall ($F_1 = 0.628$), outperforming FRED (Presutti et al., 2012), a state-of-the-art Knowledge Extraction (KE) tool. Finally, we assessed the capability of PIKES in processing large document corpora, by processing a collection of 110K Wikipedia-like texts in ~507 core hours with a sampled precision of 0.854.

Altogether, the reported evaluations demonstrate that PIKES can extract quality knowledge from large amounts of text. Differently from related state-of-the-art approaches (Section 5.6), however, PIKES adopts a 2-phase approach that provides additional flexibility by enabling the independent fine-tuning of the two processing phases (Section 5.7).

Acknowledgments The material here presented is the result of collaborative work published in (Corcoglioniti et al., 2016c) and demonstrated in (Corcoglioniti et al., 2015e). The evaluation on the Simple English Wikipedia was also supported by Giulio Petrucci.

5.1 Approach

PIKES performs Frame-based Ontology Population in two main phases, exemplified in Figure 5.1 for a simple running example document comprising two sentences:

"G. W. Bush and Bono are very strong supporters of the fight of HIV in Africa. Their March 2002 meeting resulted in a 5 billion dollar aid."

The first phase, called linguistic feature extraction, consists in the execution of several standard NLP tasks to build a linguistic-oriented structured representation of the text consisting in a graph of mentions, represented according to KEM. A mention is a piece of text denoting something of interest (potential knowledge to be extracted), such as an entity, event, temporal expression, or even fact, and is characterized by different kinds of semantic annotations that describe entities and relations expressed in the text. Mentions and their annotations are extracted from NLP annotations and aim at: (i) generalizing the outputs of tools for typical NLP tasks usually coming in different formats, thus enabling handling uniformly the different types of linguistic annotations produced; and (ii) representing, in a structured form, all the linguistic information needed to extract the knowledge conveyed by the text, such that access to the original text is not needed for further processing.

The second phase, called knowledge distillation, consists in the use of knowledge-oriented techniques to combine structured information spread over different mentions, and build an RDF knowledge graph made of instances of semantic frames and entities, as well as facts about them, abstracting from their actual occurrences in text. PIKES uses

2The complete output of PIKES for this example can be obtained with the online demo of the tool described in Section 5.4 and is directly accessible at http://bit.ly/pikes_example.
G. W. Bush and Bono are very strong supporters of the fight of HIV in Africa. Their March 2002 meeting resulted in a 5 billion dollar aid.
FrameBase, YAGO, and SUMO as the target ontologies populated from text, and reuses DBpedia identifiers where possible to identify extracted instances. With this setup, for instance, the :support frame instance of Figure 5.1 can be described as follows:

```
:support a frb:frame−Taking_sides−back.v ;
  frb:fe−taking_sides−cognizer dbpedia:Bush , dbpedia:Bono ;
  frb:fe−taking_sides−issue :fight ;
  frb:fe−taking_sides−degree attr:very−1r_strong−1a .
```

It is worth noting that knowledge distillation is not a straightforward graph transformation, but has instead to account for a number of linguistic phenomena that require knowing how different linguistic elements (e.g., predicates) behave and how they map to ontological concepts (e.g., FrameBase classes and properties). For instance, in the running example of Figure 5.1, knowledge distillation has to determine that mention ‘supporters’ (a noun predicate) stands for two distinct instances—an unspecified group of persons and a ‘support’ event, linked by an event participation triple. This is not always the case for this kind of mention, as the following mention ‘meeting’ (also a noun predicate) denotes only one instance—a meeting event. Then, from the coreference mention linking ‘supporters’ to the two mentions ‘G. W. Bush’ and ‘Bono’, knowledge distillation has to determine that the group of persons denoted by ‘supporters’ (and not the support event) is owl:sameAs the pair of persons denoted by ‘G. W. Bush’ and ‘Bono’, and possibly link the disambiguated DBpedia URIs of these persons to the support event instance, in place of an anonymous, uninformative instance for the unspecified group of persons.

The effective decoupling of the linguistic feature extraction and the knowledge distillation phases, enabled by the mention graph, leads to several benefits, such as the possibility to use different NLP tools while keeping the same knowledge distillation logic, or to apply / experiment with alternative knowledge distillation techniques without having to reprocess the full text, saving computation time especially when dealing with large corpora. Moreover, both the mention graph and the final knowledge graph are homogeneously expressed in RDF, based on KEM and using named graphs to associate RDF triples of each KEM layer—Resource, Mention, and Instance—to relevant metadata such as the source(s) it has been derived from (prov:wasDerivedFrom—e.g., a mention that kem:conveys an Instance-layer triple) and the agent it is attributed to (prov:isAttributedTo—e.g., the specific version of an NLP module that generated a Mention-layer triple). In details, annotations are attached to a named graph (either using the graph URI as subject or object of the annotations) and apply, by convention, to all the triples in the graph. E.g.:

```
:g1 { <.../res1#char=36,46> nif:annotation [ kemt:predicate pm:fn15−taking_sides ] }
:g2 { <.../res1#char=36,46> nif:annotation [ kemt:predicate pm:nb10−supporter.01 ] }
:g3 { :support a frb:frame−Taking_sides−back.v , kem:Instance . }
pk:meta { :g1 prov:wasAttributedTo pk:semafor−v2.1 ;
  nif:confidence 58.635""xsd:decimal .
:g2 prov:wasAttributedTo pk:mate−v4.31 .
  <.../res1#char=36,46> kem:conveys :g1 . }
```
Table 5.1. Association between NLP tasks and semantic annotations of mentions.

<table>
<thead>
<tr>
<th>Mention’s semantic annotation class</th>
<th>POS</th>
<th>NERC</th>
<th>TERN</th>
<th>EL</th>
<th>WSD</th>
<th>SRL</th>
<th>COREF</th>
<th>DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>kemt:EntityAnnotation (plain)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>kemt:NameEntity</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>kemt:TemporalElement</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kemt:TemporalLink</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kemt:Predicate</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kemt:Argument</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kemt:Participation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kemt:Coreference</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kemt:Coordination</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where the Mention-layer triples in graphs :g1 and :g2 are attributed respectively to Semafor (with confidence 58.635) and to Mate-tools, and the Instance-layer triple in :g3 derives from/is conveyed by mention <../res1#char=36,46>. As shown in the example, a named graph may contain multiple triples and may be the subject (or object) of multiple annotations, which are placed in a separate pk:meta graph for ease of access.3

5.2 Phase 1: Linguistic Feature Extraction

Given an input text, several NLP tasks are applied to produce the necessary NLP annotations from which a set of mentions, enriched with kem:SemanticAnnotations, is extracted. Table 5.1 reports the NLP tasks (columns) and the semantic annotations (rows) considered, showing which tasks contribute to each annotation class.4 Mapping of NLP annotations to KEM semantic annotations is performed as follows:

- **kemt:NameEntity** annotations are mainly derived from NERC (Named Entity Recognition and Classification), except for time and date categories, but also from tokens not marked by NERC with proper noun as part-of-speech (POS); DBpedia links are added via EL (Entity Linking) where possible.

- **kemt:TemporalElement** annotations are derived from TERN (Temporal Expression Recognition and Normalization) and NERC (for time and date entity categories). For the normalized time value we map from the standard TimeML (Pustejovsky et al., 2010) representation to the OWL Time ontology.5

- **kemt:Predicate** annotations are created for verb and noun predicates recognized by SRL, representing the predicate semantic class using attribute its:taClassRef.

- Plain **kemt:EntityAnnotations** are created for each pronoun or common noun token not covered by other mentions, and are enriched with DBpedia links from EL and WordNet (Fellbaum, 1998) synsets from WSD (Word Sense Disambiguation).

3Prefix pk is used for PIKES-specific data and functions.
4Not all the semantic annotations in KEM Text are currently populated. We plan to integrate additional NLP tools in PIKES in the future, e.g., by Mirza and Minard (2015) and Ghosh et al. (2011).
5http://www.w3.org/TR/owl-time/
• kemt:Argument annotations, and the kemt:Participation annotations linking them to predicates, are extracted from text spans marked as predicate arguments by SRL (e.g., ‘of HIV’ for predicate ‘fight’). Dependency data is used (via regular expressions matching) to locate the mention of the argument entity marked as kemt:Argument.
• kemt:Coreference annotations are derived from coreference resolution, where each coreferring text span (e.g., ‘G. W. Bush and Bono’, ‘supporters’, ‘they’) is mapped to a kemt:EntityAnnotation standing for an individual instance (e.g., ‘supporters’) or a group instance (e.g., ‘G. W. Bush’ and ‘Bono’).
• kemt:Coodination annotations are extracted from the dependency parse tree.

5.3 Phase 2: Knowledge Distillation

Our approach for distilling a knowledge graph of instances consists primarily in the evaluation of a set of mapping rules that match certain patterns in the Mention layer and create consequent facts in the Instance layer (Figure 5.2). Mapping rules are formulated as SPARQL Update INSERT… WHERE… statements that are repeatedly executed until a fixed-point is reached. Rules can create new individuals, can invoke external code by means of custom SPARQL functions and can access and match also data in auxiliary background knowledge resources (encoding mappings, e.g., from PreMOn, and further characterizing the behavior of predicates) as well as the instance data created so far. Based on their function, mapping rules can be organized in the six categories described next (Sections 5.3.1–5.3.6) The RDF produced by mapping rules is finally post-processed to obtain the resulting knowledge graph (Section 5.3.7).

5.3.1 Instance Creation Rules

These rules map kemt:EntityAnnotations of kemt:Mentions to instances, using a fresh URI as the instance identifier (we avoid blank nodes), and linking it to the mention via a kem:refersTo or kem:isAbout triple; we rely on extracted owl:sameAs assertions to possibly
ground generated URIs to externally well-known DBpedia identifiers. Exactly one instance is derived from an annotation, except for a predicate mention corresponding to an argument nominalization, which both kem:refersTo a first instance (e.g., the supporter persons) and kem:isAbout the existence of a second instance (the support event). The associated rule is:

```sparql
  ?m kem:refersTo ?i; kem:isAbout ?if.
GRAPH ?g { ?i a kem:Instance. ?if a kem:Instance, frb:Frame }
  ?a a kem:Predicate; itsrdf:taClassRef [ a pk:ArgumentNominalization ]
  BIND (pk:mint(?a) AS ?g) BIND (pk:mint(?s, ?m) AS ?i)
  BIND (pk:mint(concat(?s, "pred"), ?m) AS ?if) }
```

Argument nominalization is detected based on the SRL semantic class (attribute itsrdf:taClassRef), marked in background knowledge as being of type pk:ArgumentNominalization. The custom function pk:mint(args...) maps its arguments to a human-readable URI whose local name is based on the first argument plus a disambiguating hash of all the arguments; pk:mint is used to mint instance URIs as well as the URI of graph ?g linking annotation ?a to the triples it kem:substantiates. The particular choice of graph URIs is irrelevant, as these URIs are replaced and reduced in their number during post-processing. Note that the rule does not assert triple ?m kem:conveys ?g, as it is derived by a property chain axiom in the KEM ontology starting from ?m kem:hasAnnotation ?a and ?a kem:substantiates ?g.

### 5.3.2 Typing Rules

These rules enrich instances with rdf:type triples obtained from several mention and semantic annotation attributes, including WordNet synset, NERC class, lemma, POS, and semantic classes of different predicate models: PropBank (Palmer et al., 2005), NomBank (Meyers et al., 2004), VerbNet (Kipper Schuler, 2005), and FrameNet (Baker et al., 1998). Emitted ontology classes are selected by matching mapping triples loaded from customizable background knowledge, which in our implementation we populate with mappings towards SUMO, YAGO and FrameBase classes (see Section 3.4.2). The main typing rule is shown below, and allows deriving classes (variable ?t) starting from: (i) synsets or lexical senses attached to a mention via itsrdf:termInfoRef; (ii) NERC categories, semantic classes, and other lexical concepts attached to an annotation via itsrdf:taClassRef, using either a direct mapping (via ontolex:isConceptOf) or a pmo:Conceptualization mapping of PreMOn.

```sparql
INSERT { ?a kem:substantiates ?g. GRAPH ?g { ?i a ?t } }
  ?a a kem:Predicate; itsrdf:taClassRef ?c.
  ?c ontolex:isConceptOf|ontolex:reference ?t } UNION
{ ?a itsrdf:termInfoRef ?c.
  ?c ontolex:isConceptOf !ontolex:reference ?t } UNION
{ ?c ontolex:isConceptOf ?t } UNION
{ a pmo:Conceptualization; pmo:evokedConcept ?c;
  pmo:evokingEntry/kem:lexicalEntry ?m; ontolex:reference ?t } }
```
5.3.3 Naming Rules

These rules assert rdfs:label triples based on the textual extent of kem:Mentions. For proper name mentions (i.e., mentions with a kem:NamedEntity annotation) a foaf:name assertion is also generated, as shown by the next rule:

```sql
INSERT { ?a kem:substantiates ?g. GRAPH ?g { ?i foaf:name ?s } }
    BIND (pk:mint(?a) AS ?g) }
```

5.3.4 Linking Rules

These rules link instances to well-known resources in DBpedia and other Linked Open Data (LOD) datasets, based on EL attribute itsrdf:taIdentRef of a kem:EntityAnnotation. Property owl:sameAs is used for mentions annotated as kem:NamedEntity whereas rdfs:seeAlso is used for other mentions (e.g., ‘supporters’), as it is often unclear whether the DBpedia resources linked in these cases (e.g., dbpedia:Supporter) can be treated as individuals or classes for our purposes; moreover, linking of common nouns to DBpedia is in general too noisy to be used with owl:sameAs assertions (e.g., dbpedia:Supporter is a kind of shield decoration in heraldry). The main linking rule implementing this strategy is shown below:

```sql
INSERT { ?a kem:substantiates ?g. GRAPH ?g { ?i ?p ?u } }
WHERE { ?a a kem:EntityAnnotation; kem:subject ?i; itsrdf:taIdentRef ?u. 
    BIND (EXISTS { ?a `nif:annotation/nif:annotation [ a kem:NameMention ] } AS ?named) 
    BIND (IF(?named, owl:sameAs, rdfs:seeAlso) AS ?p) BIND (pk:mint(?a) AS ?g) }
```

5.3.5 Frame-Argument Linking Rules

These rules link frame to argument instances starting from kem:Participation annotations, mapping the semantic roles of arguments (property itsrdf:taPropRef) recognized by SRL to linking properties based on external customizable mapping triples. The main frame-argument linking rule is reported below, and is used together with the mappings of Section 3.3 to derive FrameBase frame-argument triples:

```sql
INSERT { ?a kem:substantiates ?g. GRAPH ?g { ?i ?p ?ia } }
    BIND (pk:mint(?a) AS ?g) }
```

5.3.6 Coreference Assertion Rules

These rules create owl:sameAs links between instances denoted by mentions coreferred via a kem:Coreference annotation. The main rule is reported below:

```sql
INSERT { ?a kem:substantiates ?g. GRAPH ?g { ?i1 owl:sameAs ?i2 } }
    FILTER (?i1 != ?i2) BIND (pk:mint(?a) AS ?g) }
```
If a mention (e.g., ‘supporters’ in Figure 5.1) corefers with a conjunction of coordinated mentions (e.g., ‘G.W. Bush’ and ‘Bono’), the group instance referred by the first and each member instance for the latter are linked by pk:includes triples derived from kemt:Coordination annotations:

```
INSERT { ?a kem:substantiates ?g. GRAPH ?g { ?ig pk:includes ?im } }
WHERE { ?a a kemt:Coordination; kemt:group [ kem:subject ?ig ]; kemt:member [ kem:subject ?im ]
BIND (pk:mint(?a) AS ?g) }
```

Coreference of subject and complement mentions implied by copular verbs (e.g., between ‘Bush’ and ‘president of US’ in ‘Bush became president of US’) is also handled.

### 5.3.7 Post-Processing

The knowledge graph produced by mapping rules is post-processed in four steps:

1. **Inference.** We materialize the inferences computed applying OWL 2 RL (Motik et al., 2012) rules and a custom rule establishing that whenever an instance participates to a frame, also the instances it pk:includes participate to the frame:

   ```
   WHERE { ?a1 kem:substantiates ?g1. GRAPH ?g1 { ?ig ?p ?i FILTER (?p != pk:include) }
          ?a2 kem:substantiates ?g2. GRAPH ?g2 { ?ig pk:include ?im }
          BIND (pk:mint(?g1, ?g2) as ?g) }
   }
   ```

As exemplified above, inference rules are augmented so to assert kem:substantiates links between inferred triples and all the annotations related to antecedent triples (and thus kem:conveys links between inferred triples and mentions of antecedents).

2. **owl:sameAs smushing.** We keep only a canonical URI for each instance, possibly from well-known LOD resources (e.g., dbpedia:HIV), and discard the other owl:sameAs aliases (:HIV) generated by mapping rules.

3. **Redundancy elimination.** We discard every unnamed instance that pk:includes some member instance (e.g., the instance derivable from mention ‘supporters’ of Figure 5.1), as the custom inference rule already propagated participation triples to its members.

4. **Compaction.** We optimize the RDF encoding of triple-level annotations (such as kem:conveys links) by ensuring that whenever a maximal set of triples \{\(s_i, p_i, o_i\)\}, \(i = 1..n\) is associated to the same set \{\(a_j\)\}, \(j = 1..m\) of annotations, then all the triples are placed in a unique graph \(g\) associated to all the annotations. The result is a maximal reduction of the total number of triples from \(2 \cdot n \cdot m\) (worst-case before compaction) to \(n + m\).

### 5.4 Implementation

We implemented PIKES as an open-source (GPL) Java server application. In the following, we describe its internal architecture (Section 5.4.1) and user interface (Section 5.4.2). A

instance of PIKES is publicly available online for demo purposes (reachable from PIKES website) and allows users to freely test our approach on sentences of their choice.

5.4.1 Architecture

PIKES architecture is shown in Figure 5.3. A frontend component exposes an HTTP ReST API and a web user interface that allow clients and end users to extract knowledge graphs from arbitrary texts, providing access also to intermediate results such as the mention graph and the raw linguistic annotations (mainly for debugging purposes).

The linguistic feature extraction phase is implemented using a number of state-of-the-art NLP modules, integrated using the Stanford CoreNLP (Manning et al., 2014) pipeline and Java object model for linguistic annotations. From Stanford CoreNLP we directly use the Tokenization, POS-Tagging, Lemmatization, NERC, TERN, Dependency Parsing (DP), and Coreference Resolution components with their trained models for the English language. For WSD we integrate the unsupervised UKB (Agirre et al., 2014) tool, which is executed as a background server process. Entity linking with respect to DBpedia is performed by DBpedia Spotlight (Mendes et al., 2011), which relies on a local installation of the DBpedia Spotlight server. SRL is performed by two complementary tools: Semafor, against FrameNet 1.5, and Mate-tools, against PropBank 2.1.5 and NomBank 1.0. Semafor
is used as a standalone server and is trained on FrameNet 1.5 full text annotations, while Mate-tools is embedded as a library. The linguistic annotations resulting from all these modules are mapped to an RDF mention graph using a specifically developed Mention Extractor module. In case multiple tools are used for the same NLP task, as happens with SRL, the mentions extracted from each tool are automatically combined and mentions (with associated semantic annotations) corresponding to the same text span (as identified by properties nif:beginIndex and nif:endIndex) are merged. For instance, in case a token is annotated as predicate by both Semafor and Mate-tools, a single mention semantically annotated as kem:Predicate is obtained and described with triples specifying both the FrameNet frame and the PropBank/NomBank predicate, keeping track of provenance (Semafor vs. Mate-tools) by annotating these triples as discussed in Section 5.1); a similar behavior occurs for other types of mentions and semantic annotations.

The knowledge distillation phase is implemented with a mapping rule evaluator module based on RDFpro (Chapter 7). Mapping rules and mapping triples are loaded from customizable files. At runtime, the mention graph is (conceptually) augmented with mapping triples linking specific NLP annotation categories, as well as semantic classes and semantic roles of predicates, to ontological classes and properties, and mapping rules are evaluated on the augmented graph. The results are post-processed (OWL 2 RL inference, owl:sameAs smushing, redundancy elimination and compaction) by a dedicated module built on top of RDFpro and the Sesame (Broekstra et al., 2002) libraries.

5.4.2 User Interface

As shown in Figure 5.4a, the web user interface of PIKES provides an input form where users can enter a text with its relevant metadata (e.g., URI and document creation time) and can select different processing options (e.g., enabled postprocessing steps) and output format (HTML, RDF, raw linguistic annotations). When the HTML output format is selected, several tabs become available once PIKES finished processing the input text, some of which are exemplified in Figure 5.4:

- the Graph tab (Figure 5.4b) shows a graphical rendering of the extracted knowledge graph at the Instance layer, where nodes are instances and arcs are triples between them; additional triples (e.g., types, labels, etc) are shown by tooltips when hovering over a graph element with the mouse;
- the Hybrid tab (Figure 5.4c) highlights, for each sentence, the mentions where they occur in the text, the fragment of knowledge graph extracted from that sentence, and its most relevant linguistic annotations (DP arrows above the text, SRL annotations below it, and token-level annotations such as POS-Tagging and WSD when hovering with the mouse over a token);

Actually, the mention graph is not augmented but instead rules are rewritten by RDFpro taking into considerations all the possible groundings of triple patterns of rule bodies in supplied mapping triples. This results in an equivalent (larger) ruleset that can be efficiently evaluated without feeding it with (a large number of) mapping triples.
Figure 5.4. PIKES web user interface: (a) input form filled with the text of the running example; (b) graph tab with obtained knowledge graph; (c) hybrid tab showing NLP annotations and knowledge extracted for each sentence; (d) mentions tab with mentions, the instances they denote/imply and the triples they express.
Table 5.2. Sentences of the gold standard (from Gangemi, 2013).

<table>
<thead>
<tr>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 The lone Syrian rebel group with an explicit stamp of approval from Al Qaeda has become one of the uprising most effective fighting forces, posing a stark challenge to the United States and other countries that want to support the rebels but not Islamic extremists.</td>
</tr>
<tr>
<td>S2 Money flows to the group, the Nusra Front, from like-minded donors abroad.</td>
</tr>
<tr>
<td>S3 Its fighters, a small minority of the rebels, have the boldness and skill to storm fortified positions and lead other battalions to capture military bases and oil fields.</td>
</tr>
<tr>
<td>S4 As their successes mount, they gather more weapons and attract more fighters.</td>
</tr>
<tr>
<td>S5 The group is a direct offshoot of Al Qaeda in Iraq, Iraqi officials and former Iraqi insurgents say, which has contributed veteran fighters and weapons.</td>
</tr>
<tr>
<td>S6 “This is just a simple way of returning the favor to our Syrian brothers that fought with us on the lands of Iraq,” said a veteran of Al Qaeda in Iraq, who said he helped lead the Nusra Front’s efforts in Syria.</td>
</tr>
<tr>
<td>S7 The United States, sensing that time may be running out for Syria president Bashar al-Assad, hopes to isolate the group to prevent it from inheriting Syria.</td>
</tr>
<tr>
<td>S8 As the United States pushes the Syrian opposition to organize a viable alternative government, it plans to blacklist the Nusra Front as a terrorist organization, making it illegal for Americans to have financial dealings with the group and prompting similar sanctions from Europe.</td>
</tr>
</tbody>
</table>

- the *Mentions* tab (Figure 5.4d) provides a sortable and filterable table with all the mentions identified in the text, their types and properties, the instances they refer to, and the Instance layer triples they convey;
- the *Instances* tab provides a sortable and filterable table with all the extracted Instance layer triples, including kem:conveys links to mentions.
- the *Metadata* tab summarizes the document metadata and lists all the NLP modules applied to extract knowledge together with their versions;
- the *Annotations* tab shows the raw output of NLP modules using NAF (NLP Annotation Format, Fokkens et al., 2014).

### 5.5 Evaluation

To assess the performances of PIKES we performed three complementary evaluations, creating a manually annotated gold standard (Section 5.5.1) that allows both precision and recall to be assessed. First (Section 5.5.2), we evaluated the capabilities of PIKES as a frame-based ontology population approach for the FrameBase ontology, computing precision and recall against the gold standard and investigating the use and combination of different SRL tools by leveraging PIKES decoupled 2-phase approach. Then (Section 5.5.3), in order to compare PIKES performances with the state of the art, we moved away from FrameBase and considered the extraction of frame structures described using traditional
predicate models, as done tools by such as FRED, LODifier (Augenstein et al., 2012), and NewsReader\(^8\) (Rospocher et al., 2016). Specifically, we computed PIKES precision and recall in extracting entities, semantic frames, and their relations from the gold standard text, and we also compared PIKES and FRED performances on the same text. The comparison with FRED is motivated by a recent survey of KE tools for the Semantic Web (SW) (Gangemi, 2013), which showed that FRED stands out on various tasks, including frame and frame-role detection (note that FRED also aims at extracting TBox content from text, something not considered here). Finally, in the third evaluation (Section 5.5.4), we studied the scalability of PIKES, assessing its capability of extracting accurate knowledge from a large text corpus. All data for the three evaluations are available on PIKES website.

5.5.1 Gold Standard

For the first two evaluations we manually created a gold standard consisting of a text \(T\) and a corresponding RDF knowledge graph \(G\). The text \(T\), shown in Figure 5.2, contains the same 8 sentences used in the paper by Gangemi (2013), with the minor exception of sentence S7 being slightly shortened due to its unprocessability with FRED online demo (tested Sep. 2015). Graph \(G\), collaboratively built by two annotators, consists of the relevant RDF triples that should be included in the output of a frame-based ontology population system, which may include additional (irrelevant) triples provided that they do not conflict with the meaning of \(T\). The nodes of \(G\) are the instances mentioned in \(T\). Each instance is anchored to exactly one mention, with coreferring mentions giving rise to distinct instances. Instances are linked to matching entities in DBpedia via owl:sameAs triples, and typed with respect to classes of FrameBase—for the first evaluation—and classes encoding VerbNet (VN), FrameNet (FN), PropBank (PB), and NomBank (NB) predicates—for the second evaluation (with only the most specific types represented). The edges of \(G\) are given by triples connecting different instances. They express owl:sameAs equivalence relations, explicitly representing coreference, and frame-argument participation relations whose RDF properties encode FrameBase and VN, FN, PB, and NB arguments.

The evaluation of a system \(S\) against the gold standard is performed as follows. First, the vocabulary properties and types (excluded FrameBase terms) and the instance identifiers in \(S\) output are replaced with the corresponding ones in the gold graph \(G\) (where defined), obtaining a graph \(\hat{G}_S\) that is now comparable with \(G\). Corresponding instances are identified by leveraging their grounding to mentions in both graphs and, when multiple mappings are possible, by selecting the mapping that maximizes the number of triples shared by \(\hat{G}_S\) and \(G\). Then, elements of different types (instances, edges in the knowledge graph, triples) in \(\hat{G}_S\) are compared with corresponding gold elements in \(G\), computing precision (P), recall (R), and \(F_1\)-measure (\(F_1\)). While true positives and false negatives are computed as usual, for false positives a distinction is made between wrong and irrelevant elements in \(\hat{G}_S\): the first are marked as false positives, the latter are ignored and do not

\(^8\)http://www.newsreader-project.eu/
affect the evaluation results. Irrelevant instances are classified manually, while edges and triples involving irrelevant instances are marked automatically as irrelevant; the remaining edges have to be judged manually.

5.5.2 Evaluating Precision/Recall with respect to FrameBase

We evaluated PIKES as an ontology population system for FrameBase by applying it to the gold standard text, using the FrameBase mappings described in Section 3.4.

For this evaluation we experimented with three different configurations of the linguistic feature extraction phase that differ for the SRL tool(s) used: (i) Semafor only for FN SRL; (ii) Mate-tools only for PB/NB SRL; and, (iii) both Semafor and Mate-tools, relying on the automatic combinations of the respective annotations in the mention graph.

Table 5.3 and Figure 5.5 report the results of the evaluation for the three configurations, both against FrameBase type triples for frame instances (first three columns), FrameBase frame-role triples (middle three columns), and all triples undifferentiated (last three columns). As one could expect, F1 scores of Mate-tools are lower than the ones of Semafor, reflecting the fact that the latter, being specifically designed for FN SRL, is more suitable for use with FrameBase. However, when both tools are combined, we get an increase of recall for frame-argument triples with respect to Semafor or Mate-tools alone, confirming that the combination of these two (in principle) complementary SRL tools, made easier by PIKES approach, is indeed beneficial. Moreover, it is worth noting that precision scores for Mate-tools are on par with the ones of Semafor, with a gap in terms of recall that could be potentially addressed with further work on PB/NB mapping resources.

5.5.3 Evaluating Precision/Recall with respect to Predicate Models

In this evaluation, we computed precision, recall and F1-measure of PIKES (as previously defined) in extracting the following knowledge graph components from the gold standard text: instances, edges (i.e., instance-instance unlabeled relations), and triples, considered both globally and divided by category: VN/FN/PB/NB types and participation relations, linking to DBpedia, owl:sameAs coreference triples.

We first evaluated PIKES alone on the full gold standard ⟨T,G⟩, so to assess to the fullest possible degree the performances of PIKES when extracting knowledge with respect to traditional predicate resources. Table 5.4 and Figure 5.6 report the results obtained, with the number NG of gold elements for each evaluated component. Precision is generally higher than recall, which is common in KE systems where the lack of recall can be compensated by information redundancy in the text. Instances and edges of the knowledge graph are extracted with high precision (> 90%), while for extraction of triples the performances depend on their category. In particular, precision of frame type and role triples is good for PB and low for FN, which reflects the fact that SRL against FN is more difficult due to the greater richness of this predicate resource.
Table 5.3. Precision/recall of FrameBase ontology population with different SRL setups.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Types</th>
<th>Roles</th>
<th>All triples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>F₁</td>
</tr>
<tr>
<td>SEMAFORE only</td>
<td>.617</td>
<td>.698</td>
<td>.655</td>
</tr>
<tr>
<td>Mate-tools only</td>
<td>.792</td>
<td>.358</td>
<td>.494</td>
</tr>
<tr>
<td>SEMAFORE + Mate</td>
<td>.603</td>
<td>.717</td>
<td>.655</td>
</tr>
</tbody>
</table>

We then compared PIKES with FRED on the same text. A fair comparison of the two systems was possible only on a simpler gold graph $G'$ against which both tools are comparable. $G'$ was derived automatically from $G$ by considering two characteristics of FRED: (i) FRED does not return PB/NB frame types and PB/NB/FN frame roles (for FN it uses a generic predicate :fe) so we ignored them; and (ii) FRED does not support nominal predicates and argument nominalization, although it often represents the associated participation relations with arbitrary triples. To exemplify, the span ‘Its fighters’ in sentence S3 is represented by FRED using triple

```
:fighter_1 :fighterOf :neuter_1 .
```

where :fighter_1 and :neuter_1 are the instances denoted by mentions ‘fighters’ and ‘Its’, whereas the gold standard and PIKES use the nominal frame (disambiguated against FN)

```
:fighter_frame a fn:Irregular_combatants ;
    fn:combatant :fighter_1 ;
    fn:side1 :neuter_1 .
```

where :fighter_frame is a frame instance also denoted by ‘fighters’. Thus, we automatically transformed the latter representation (both in $G'$ and in PIKES output) into FRED one. In line with the approach of Gangemi (2013), we also compared PIKES and FRED against an additional gold graph $G''$ obtained by merging the outputs of both tools, cleaned up of

---

9We refer to FRED on-line version as available in Sep. 2015. The RDF output obtained from this version of FRED is included in the evaluation material on PIKES website.
Table 5.4. Precision/recall of ontology population with respect to predicate models on gold graph $G$.

<table>
<thead>
<tr>
<th>Component</th>
<th>$N_G$</th>
<th>P</th>
<th>R</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances</td>
<td>153</td>
<td>.919</td>
<td>.961</td>
<td>.939</td>
</tr>
<tr>
<td>Edges</td>
<td>171</td>
<td>.865</td>
<td>.784</td>
<td>.822</td>
</tr>
<tr>
<td>Triples</td>
<td>596</td>
<td>.711</td>
<td>.562</td>
<td>.628</td>
</tr>
<tr>
<td>types (VN)</td>
<td>44</td>
<td>.706</td>
<td>.545</td>
<td>.615</td>
</tr>
<tr>
<td>types (FN)</td>
<td>53</td>
<td>.603</td>
<td>.717</td>
<td>.655</td>
</tr>
<tr>
<td>types (PB)</td>
<td>53</td>
<td>.841</td>
<td>.698</td>
<td>.763</td>
</tr>
<tr>
<td>types (NB)</td>
<td>37</td>
<td>.774</td>
<td>.649</td>
<td>.706</td>
</tr>
<tr>
<td>roles (VN)</td>
<td>94</td>
<td>.758</td>
<td>.500</td>
<td>.603</td>
</tr>
<tr>
<td>roles (FN)</td>
<td>108</td>
<td>.595</td>
<td>.435</td>
<td>.503</td>
</tr>
<tr>
<td>roles (PB)</td>
<td>119</td>
<td>.817</td>
<td>.563</td>
<td>.667</td>
</tr>
<tr>
<td>roles (NB)</td>
<td>55</td>
<td>.633</td>
<td>.564</td>
<td>.596</td>
</tr>
<tr>
<td>linking</td>
<td>18</td>
<td>.700</td>
<td>.778</td>
<td>.737</td>
</tr>
<tr>
<td>coreference</td>
<td>15</td>
<td>.857</td>
<td>.400</td>
<td>.545</td>
</tr>
</tbody>
</table>

Figure 5.6. Bar plot for Table 5.4 (ontology population with respect to predicate models).

incorrect triples. By definition, $G'' \subseteq G'$. The goal of this additional evaluation, as noted by Gangemi (2013), is to comparatively evaluate each tool within the KE tool space (i.e., considering only correct triples that can be extracted by at least one tool). The results of the comparison against $G'$ and $G''$ are reported in Table 5.5, together with the numbers $N_G$ and $N_G''$ of gold elements for each component (which result lower than corresponding numbers in Table 5.4 due to the performed simplification). PIKES exhibits better precision and recall than FRED for all the considered components and gold graphs: the difference in terms of $F_1$ between PIKES and FRED ranges, for the various components, from .042 to .221 on $G'$, and from .042 to .381 on $G''$. Overall, the results show that splitting the ontology population process in two phases, decoupled by a mention graph, allows reaching
Table 5.5. Precision/recall of Ontology Population with respect to predicate models: comparative evaluation with FRED on simplified gold graph $G'$ and union of correct tools answers $G''$.

<table>
<thead>
<tr>
<th>Component</th>
<th>$N_{G'}$</th>
<th>FRED vs. $G'$</th>
<th>PIKES vs. $G'$</th>
<th>$N_{G''}$</th>
<th>FRED vs. $G''$</th>
<th>PIKES vs. $G''$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>F₁</td>
<td>P</td>
<td>R</td>
<td>F₁</td>
</tr>
<tr>
<td>Instances</td>
<td>137</td>
<td>.930</td>
<td>.869</td>
<td>.898</td>
<td>.911</td>
<td>.971</td>
</tr>
<tr>
<td>Edges</td>
<td>155</td>
<td>.869</td>
<td>.555</td>
<td>.677</td>
<td>.910</td>
<td>.787</td>
</tr>
<tr>
<td>Triples</td>
<td>166</td>
<td>.543</td>
<td>.416</td>
<td>.471</td>
<td>.698</td>
<td>.584</td>
</tr>
<tr>
<td>types (VN)</td>
<td>31</td>
<td>.593</td>
<td>.516</td>
<td>.552</td>
<td>.667</td>
<td>.581</td>
</tr>
<tr>
<td>roles (VN)</td>
<td>76</td>
<td>.547</td>
<td>.382</td>
<td>.450</td>
<td>.741</td>
<td>.526</td>
</tr>
<tr>
<td>linking</td>
<td>18</td>
<td>.615</td>
<td>.444</td>
<td>.516</td>
<td>.700</td>
<td>.778</td>
</tr>
<tr>
<td>coreference</td>
<td>15</td>
<td>.357</td>
<td>.333</td>
<td>.345</td>
<td>.857</td>
<td>.400</td>
</tr>
</tbody>
</table>

Figure 5.7. Bar plot for Table 5.5 (comparison with FRED, graph $G'$).

performances that are competitive with the state of the art while providing at the same time the benefits listed in Section 5.1.

5.5.4 Evaluating Precision/Throughput on Large Corpus

To assess its capability of processing large corpora, we applied PIKES to a general domain corpus, namely a dump of the Simple English Wikipedia (SEW).\footnote{http://simple.wikipedia.org/, dump date: April 6, 2015. The extracted RDF dataset is available at http://pikes.fbk.eu/eval-sew.html.} SEW is a variant of Wikipedia, where each page is written using basic English words. The corpus, which consists of 109,242 text documents for a total of 1,584,406 sentences and 23,877,597 tokens, was chosen for its relatively large size and public availability.
Table 5.6. PIKES precision on Simple English Wikipedia knowledge graph.

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Eval.</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Triples</td>
<td>Triples</td>
<td>Ev 1</td>
</tr>
<tr>
<td>Annotation</td>
<td>1,617</td>
<td>35</td>
<td>.900</td>
</tr>
<tr>
<td>Type</td>
<td>1,699</td>
<td>35</td>
<td>.943</td>
</tr>
<tr>
<td>Participation</td>
<td>10,805</td>
<td>130</td>
<td>.904</td>
</tr>
<tr>
<td>Total</td>
<td>14,121</td>
<td>200</td>
<td>.910</td>
</tr>
</tbody>
</table>

Table 5.7. PIKES processing time grows linearly with the number of tokens.

PIKES processed the whole SEW corpus in ~507 core hours, with an average of 1.2s per sentence and 16.7s per document.\(^\text{11}\) We used 16 parallel instances of PIKES on the same machine,\(^\text{12}\) ending the whole processing in less than 32 hours. Processing time grows linearly with the number of tokens in the input text, as shown in Figure 5.7.

A total of 357,853,792 triples were produced (~2M triples Text layer, ~283M Mention layer, ~72M Instance layer). 27M mentions and more than 4M frame instances were created (most frequent: use.01, play.01, know.01), involving over 72K DBpedia persons (most frequent: Pope, Jesus, Napoleon), 19K DBpedia organizations, and 49K DBpedia places. Additional 470K persons, 173K organizations, and 18K locations not linked to DBpedia URIs were created. In total, we extracted over 26M triples about DBpedia

\(^\text{11}\) A proper baseline (i.e., same hardware and test conditions) to put the reported figures into perspective is not available, as the closest tool to PIKES, FRED, is not available for download (only a demo service is available and cannot be used on the whole SEW). As a purely indicative comparison, note that the online demo of FRED takes ~26s to process sentence by sentence the text in Figure 5.2, while PIKES online demo takes ~11s (times taken invoking the tools’ ReST APIs, avg. on 10 runs).

\(^\text{12}\) Dell PowerEdge M520 Server, 2CPUs Intel Xeon E5-2430 2.50GHz, 192GB RAM, 480GB SSD HD. Note that each PIKES instance was allocated 1 core and 7GB of RAM.
entities (1.7M annotations, 2.6M types, 21M participations in different frame vocabularies for 7M frame-argument pairs), which might be used to extend DBpedia.

To evaluate the quality of the produced knowledge graph, given the absence of a gold standard and the impossibility to properly appraise recall due to the large size of the corpus, we opted to assess the precision of a subset of the extracted triples involving DBpedia entities. A similar strategy was applied to evaluate a large knowledge graph such as YAGO. We built the evaluation dataset as follows. We randomly sampled 200 triples from the graph involving DBpedia entities, focusing in particular on annotation and name assertions (35 triples), type assertions (35), and PropBank/NomBank frame participations (130). For each triple, we randomly selected one of the mentions from where the triple was extracted, and a context of the three sentences in the text centered around it. The evaluation dataset was provided to three evaluators, with skills in knowledge engineering, which were asked to judge if each triple produced by PIKES is correct for the given mention (i.e., if the knowledge encoded in the triple is compatible with the knowledge conveyed by the mention). Evaluators were allowed to use three values: 1 – correct, 0.5 – partly correct, and 0 – not correct; 0.5 was used when the tool correctly identified subject and object of a participation triple, but failed identifying the predicate. Table 5.6 summarizes the results obtained. Precision was computed for each assertion type, and for each evaluator. The average precision over the whole evaluation dataset is slightly above 0.85 and the computed Fleiss’ kappa coefficient ($\kappa = 0.372$) shows a fair agreement between the evaluators. Considering the partly correct cases as not correct, the average precision settles at 0.823 and the evaluators agreement at $\kappa = 0.407$.

5.6 Related Work

Ontology Population has become quite popular in the last decade, thanks to the spreading of LOD and SW technologies (for a review of state of the art up to 2011, see Petasis et al., 2011). In particular, in the last few years (2012 onward) several contributions were presented. We briefly report the most relevant ones for our work.

LODifier (Augenstein et al., 2012) extracts Discourse Representation Structures (DRS) from a text using the statistical parser C&C and the semantics construction toolkit Boxer, complementing them with NLP tasks such as NERC, EL, and WSD. The output of these linguistic analyses are mapped to RDF triples using transformation rules, with Boxer unary relations being translated to rdf:type statements whose class is taken from disambiguated WordNet synsets, and Boxer binary relations becoming RDF property assertions.

FRED (Presutti et al., 2012) is a tool that also builds on DRSs, but differently from LODifier, DRSs are mapped to linguistic frames in VerbNet and FrameNet, which in turn are transformed in RDF/OWL via Ontology Design Patterns (ODP); both ABox and

\[\text{http://svn.ask.it.usyd.edu.au/trac/candc/wiki}\]
\[\text{http://ontologydesignpatterns.org/}\]
TBox triples are emitted, qualifying FRED as both an Ontology Population and Ontology Learning tool. Similarly to LODifier, results are enriched with NERC, EL, and WSD.

Graphia (Freitas et al., 2013) is an Open Information Extraction pipeline producing an entity-centric RDF representation of text in the form of Structured Discourse Graphs (SDGs): nodes represent entities, while edges indicate relations between the nodes. It performs NLP tasks such as Parsing, NERC, Coreference Resolution, and the construction of the graph is based on the matching of syntactic patterns.

In NewsReader (Rospocher et al., 2016), a comprehensive processing pipeline was developed for extracting and coreferring events and entities from large (cross-lingual) news corpora. Similarly to PIKES, the NewsReader pipeline combines several NLP tasks including SRL (Mate-tools is used), and also cover cross-document concerns such as Cross-document Entity and Event Coreference. NLP annotations are collected in a single, layered annotation file based on the NAF format, and the conversion from NAF to RDF is performed according to a rule-based approach.

To the best of our knowledge, none of these tools adopt a 2-phase KE approach where all extracted content, including the intermediate linguistic information, is exposed in RDF according to a comprehensive data model.

5.7 Summary

In this chapter we presented PIKES, an approach for frame-based ontology population from natural language text, aligned to FrameBase, YAGO, SUMO, and DBpedia. The approach, implemented in an publicly-available open source tool, neatly separates processing into two phases. In the linguistic feature extraction phase, several NLP tasks are performed, exploiting state-of-the-art tools such as DBpedia Spotlight, Mate-tools, and Semafor, to produce an RDF graph of mentions, integrating and exposing the heterogeneous NLP annotations produced by the different tools in an homogeneous way according to KEM. In the knowledge distillation phase, the mention graph is transformed via SPARQL-like rules and the PreMOn FrameBase mappings into a knowledge graph, representing the knowledge conveyed by the original text in a way that abstracts from the specific occurrences of entities and relations in it. To the best of our knowledge, PIKES is the first KE tool automatically populating FrameBase from text.

We believe that PIKES decoupling allows: (i) an easy replacement and combination of different NLP tools, even for the same NLP task, as shown by complementing SRL annotations of Mate-tools and Semafor; and (ii) an easy (rule-based) combination of the outputs of different NLP tasks during knowledge distillation, to support non-trivial linguistic phenomena (e.g., argument nominalization). These advantages are augmented by the use of a single RDF model—KEM—to encode all the produced outputs, and by the consistent use of named graphs to represent metadata and fine-grained provenance information at each representation layer of KEM (Resource, Mention, Instance).
We reported PIKES performances on three evaluations conducted on: (i) FrameBase ontology population; (ii) frame and frame-role detection according to PropBank, NomBank, VerbNet, and FrameNet, comparing our results against FRED’s output (where applicable), a state-of-the-art SW tool for KE; and, (iii) knowledge graph extraction from a large text corpus. The results show that PIKES approach is competitive compared to state-of-the-art tools for the SW, being capable of: (i) processing large text corpora, and (ii) extracting quality knowledge from text. In Chapter 6, we build on this first capability (scalability) and propose an architecture for storing all the content generated by ontology population on large corpora. In Chapter 8, we leverage the second capability (results quality) by using PIKES in an Information Retrieval (IR) scenario, to extract semantic information from text that help improving search performances.
Chapter 6

KnowledgeStore: a Storage System for Interlinked Text and Knowledge

Knowledge Extraction (KE) techniques, such as the Frame-based Ontology Population approach of PIKES (Chapter 5) and other state-of-the-art approaches (Weikum and Theobald, 2010; Grishman, 2010), have enabled the large scale extraction of structured knowledge from text and its linking to the text sources it comes from. Although this sets the basis for the development of frameworks seamlessly integrating text and structured knowledge in a single repository, this research direction has been only partially investigated.

In this chapter we describe the KnowledgeStore, a scalable, fault-tolerant, and Semantic Web (SW) grounded open-source (Apache License v2.0) storage system to jointly store, manage, retrieve, and query interlinked text and RDF knowledge extracted from it (e.g., using PIKES) or coming from Linked Open Data (LOD) resources. Conceptually, the KnowledgeStore acts as a data hub populated by KE systems and queried by end users and applications (Section 6.1), whose contents are organized according to the three representation layers of KEM (Chapter 4): Resource, Mention, and Instance. To illustrate the interplay of these layers in the KnowledgeStore, and the capabilities it offers, let us consider the following scenario. Among a collection of news articles, a user is interested in retrieving all 2014 news reporting statements of a 20th century US president where he is positively mentioned as “commander-in-chief.” On one side, the KnowledgeStore supports storing resources (e.g., news articles) and their relevant metadata (e.g., the publishing date of a news article). On the other side, it enables storing structured knowledge about instances of the world (e.g., the fact of being a US president, the event of making a statement), either extracted from text or available in LOD/RDF datasets such as DBpedia (Lehmann et al., 2015) and YAGO (Hoffart et al., 2013). And last, through the notion of mention, it enables linking an instance or fact of the world to each of its occurrences in documents, allowing also to store additional mention attributes, typically extracted while processing the text, such as the position of the instance/fact in the text (e.g., between characters 1022 and 1040), the explicit way it occurs (e.g., “commander-in-chief”), and the sentiment of the article writer on that instance (e.g., positively mentioned). Besides supporting the scalable storage and management of this content, through an architecture compliant with the deployment in distributed hardware settings like clusters and cloud computing, the KnowledgeStore provides a ReST API and a user interface providing query

1http://knowledgestore.fbk.eu/
and retrieval mechanisms that enable accessing all its contents, and thus answering the example query presented above (Section 6.2).

The performances of the KnowledgeStore were evaluated through a number of experiments covering both data population and data retrieval, using different dataset sizes and numbers of concurrent clients (Section 6.3). Moreover, the KnowledgeStore was concretely used in the NewsReader\(^2\) EU project (see Chapter 9), where several KnowledgeStore instances were populated with millions of news articles and billions of RDF triples extracted from them, and a number of applications (mainly for decision support) were successfully built on top of these instances, thus demonstrating the capabilities of the KnowledgeStore.

Compared to the few state-of-the-art proposals for integrating text and structured knowledge (e.g., Popov et al., 2003; Gönül and Sinaci, 2012; Kurz et al., 2014), the KnowledgeStore distinguishes itself for its explicit representation of mentions (Section 6.4), thus allowing to exploit them in applications and enhanced processing tasks. In particular, the possibility to store Natural Language Processing (NLP) annotations and other intermediate KE results in the Mention layer provides an ideal setting for developing, debugging, training, and evaluating tools for a number of NLP and KE tasks (Section 6.5).

**Acknowledgments** The material here presented was published in (Corcoglioniti et al., 2015c) and, previously, in (Corcoglioniti et al., 2013); two system demonstrations are described in (Rospocher et al., 2014a,b). The idea behind the KnowledgeStore was preliminarily investigated in (Cattoni et al., 2013, 2012a,b), and tested in the scope of the LiveMemories\(^3\) project supported by the Province of Trento (Italy). The author also would like to thank all those who contributed to the implementation and use of the various versions of the KnowledgeStore, within and outside the scope of the NewsReader project supported by the European Union (FP7 ICT-316404), including Renato Marroquín Mogrovejo, Alessio Palmero Aprosio, Mohammad Qwaider, Marco Amadori, Michele Mostarda, Enrico Magnago, and Gianluca Apriceno.

### 6.1 Conceptual Overview

To support the storage and alignment of knowledge of unstructured and structured information sources, the KnowledgeStore internally adopts the three-layer content organization of KEM (see also Figure 4.1):

- **the Resource layer**, similarly to a file system, stores the unstructured content in the form of resources (e.g., news articles), each having a representation (e.g., a text file) and some descriptive metadata (e.g., title, actor, document creation date);

- **the Instance layer** is the home of structured content, which, based on SW best practices, consists of (subject, predicate, object) RDF triples describing the instances

\(^2\)http://www.newsreader-project.eu/
\(^3\)http://www.livememories.org/
of the world (e.g., persons, locations, events), and for which additional metadata (e.g., the provenance and confidence attributes produced by PIKES) can be stored using the named graphs mechanism;

- the Mention layer sits between the aforementioned layers and consists of mentions, i.e., snippets of resources (e.g., fragments of text) that denote something of interest, such as an instance or a triple of the Instance layer; clearly, a resource may contain many mentions, and an instance or triple may be mentioned multiple times. The explicit representation of mentions, inherited from KEM, provides an anchor where to attach attributes specific to the particular realization of an instance or triple in the text, including the NLP annotations produced by a KE pipeline and the attributes related to the particular way an instance is mentioned, i.e., to a sign-specific sense\(^4\) (e.g., writer attitude or sentiment, role or category the instance is described with).

From an architectural point of view, the KnowledgeStore can be seen as a centralized service accessed by external clients for storing and retrieving the content they process and produce (Figure 6.1). Clients can be classified in two types according to their function:

- Knowledge Extraction Processors: they produce the structured knowledge stored in the KnowledgeStore, extracting it from unstructured resources. A number of NLP tools and suites can be used for this purpose, like PIKES (Chapter 5) or the KE pipelines used in NewsReader, publicly available online (see Section 9.1). Knowledge Extraction Processors can be classified as either single-resource or cross-resource:
  
  - Single-resource processors perform tasks defined at the level of a resource or of a portion of it (e.g., a sentence), such as SRL (Semantic Role Labeling); for these tasks, the processing of a resource is independent of the processing of other resources and thus multiple resources can be processed in parallel.
  
  - Cross-resource processors, on the other hand, perform tasks defined on whole collections of resources, such as Cross-document Coreference Resolution; these tasks typically combine information from multiple resources, cannot be easily parallelized, and their cost may increase more than linearly with dataset size.

- Applications: they mainly read data from the KnowledgeStore offering services on top of its content, such as decision support systems or enhanced web-based applications.

Note that the KnowledgeStore does not enforce a particular client interaction paradigm for what concerns content access and population. Knowledge Extraction Processors and Applications may interact directly with the KnowledgeStore, possibly using it as a data hub and exchanging data through it. Alternatively, the interaction can be mediated by content populators and exporter tools: populators load the KnowledgeStore with files of

---

unstructured or structured data, either coming from Knowledge Extraction Processors or containing static data such as textual documents, web pages, or RDF/OWL background knowledge; exporters query the KnowledgeStore and generate dump files with the data requested by clients, in the formats they support. Moreover, content can be (i) injected in one shot and then accessed by applications in a sort of “read-only” mode (*write once, read many*), or (ii) continually incrementally added (as in case of a daily feed of news), where clients work more in a sort of “stream-oriented” mode (*write many, read many*).

6.2 System Description

Next, we introduce the main features and components of the KnowledgeStore. Additional documentation, a demo video showcasing the navigation through the KnowledgeStore content, as well as binaries and source code of the KnowledgeStore, are available on the KnowledgeStore website. A running KnowledgeStore instance is also publicly accessible.\(^5\)

6.2.1 Data Model

The KnowledgeStore data model defines what information can be stored in the KnowledgeStore. As pointed out in Section 6.1, it is centered on the Resource, Mention, and Instance layers of KEM. Resources, mentions, and semantic annotations of mentions are described using a configurable set of types, attributes and relations. Instances are described with an open set of triples enriched with metadata attributes (e.g., for confidence and provenance). The KnowledgeStore data model is formalized as an OWL 2 ontology. The UML class diagram of Figure 6.2 summarizes the main aspects of the KnowledgeStore data model.

\(^5\)http://knowledgestore2.fbk.eu/nwr/wikinews/
Flexibility is a key requirement for the data model, as (i) different kinds of unstructured and structured content can be stored in different KnowledgeStore instances; and (ii) the kind of information stored in a KnowledgeStore instance may evolve in time. For this reason, the data model is divided in a fixed part, embodied in the implementation, and a configurable part that is specific to each KnowledgeStore instance and is used to organize and fine tune its storage layout. The fixed part consists of KEM Core and the ks:Representations of resources, which include their files and the associated metadata attributes (reusing terms from the Nepomuk Information Element vocabulary and the Nepomuk File Ontology) that are automatically updated by the system every time a resource is uploaded. The configurable part is specified by a user-supplied OWL 2 ontology, specific to that KnowledgeStore instance, which refines the TBox of KEM Core. Typically, this consists in defining:

- the subclass hierarchy of kem:Resource, kem:Mention, and kem:SemanticAnnotation;
- additional attributes of kem:Resource, kem:Mention, kem:SemanticAnnotation and their subclasses;
- additional relations among resources, mentions, and semantic annotations;
- enumerations and classes used as attribute types (similarly to ks:Representation);
- restrictions on fixed part relations (not shown in figure).

This modular approach enables accommodating very different configurations: from KnowledgeStore instances where mentions are just pointers from instances to the characters in

---

6http://www.semanticdesktop.org/ontologies/nie/
7http://www.semanticdesktop.org/ontologies/nfo/
8As the KnowledgeStore supports storing arbitrary RDF triples describing instances, there is no need to list specific instance subclasses, attributes, or relations at configuration time.
the resources referencing them, with no mention attributes and annotations besides the mention URI (in this special case, the KnowledgeStore basically downgrades to a standard 2-layer resources/instances framework), to more enhanced instances where a very rich set of linguistic attributes is stored for each mention. A concrete example of application-specific customization of the KnowledgeStore data model is presented in Section 9.1.

It is worth noting that the choice of rooting the data model in OWL 2, on top of KEM, using an ontology for its configuration, provides a number of benefits. First, it allows both the model definition and the instance data to be encoded in RDF, enabling the use of SW technologies for manipulating them and their publication on the Web according to LOD best practices. Second, data validation can be performed using SW techniques, such as OWL 2 reasoning (e.g., for consistency checking), or the more recent SHACL (Shapes Constraint Language) W3C proposal that allows specifying the expected ‘shape’ of valid RDF data. Concerning data validation, it must be noted that resource and mention data form a huge ABox that, as a whole, can be hardly managed by a reasoner or (unrestricted) shape validator, thus posing a limit on the scalability of the system. This problem can be tackled by performing validation on a per-resource (and its mentions) basis, exploiting the fact that resource descriptions are largely independent one to another. Of course, this solution sacrifices completeness for scalability, but at the same it enables the use of expressive techniques, such as OWL 2 reasoning extensions (e.g., Patel-Schneider and Franconi, 2012; Tao et al., 2010) realizing a restricted closed world assumption useful for validation purposes.

6.2.2 ReST API

The KnowledgeStore presents a number of interfaces, offered as part of the KnowledgeStore API, through which external clients may access and manipulate stored data. These interfaces are available through two HTTP ReST endpoints: CRUD and SPARQL.

The CRUD endpoint provides the basic operations to access and manipulate any object stored in any of the layers of the KnowledgeStore; for instance, Figure 6.3 shows the HTTP invocation of a retrieve operation returning all the resources with dct:publisher being equal to dbpedia:TechCrunch. Several aspects (e.g., operation granularity, transactional properties, access control) have been considered in defining the operations provided by the CRUD endpoint. For efficiency reasons, the KnowledgeStore offers coarse-grained streaming operations that operate on multiple objects at once (e.g., the simultaneous update of all the mentions of a certain resource). As having fully transactional operations is unfeasible (as an operation can potentially affect all the KnowledgeStore content) and

9[http://www.w3.org/TR/shacl](http://www.w3.org/TR/shacl)

10For example, the RDFox (Motik et al., 2014) OWL 2 RL reasoner exhibits a RAM consumption of ∼30–60 bytes/triple depending on the dataset. We measured an average of ∼2000 triples for a news article and its mentions on the NewsReader datasets (Section 9.1), which leads to 60–120 KB of RAM usage per news article, meaning that a powerful machine with hundreds of GB of RAM would only be able to handle reasoning for few millions of news articles.

11Create, Retrieve, Update, and Delete.
perhaps unwanted (e.g., on an update operation on 1 million objects, failing on a particular object should not cause the rollback of the operation for the other objects), a coarse-grained API call behaves in a transactional way and satisfies ACID\textsuperscript{12} properties only on each single object handled in the call (e.g., a single element in a set of mentions).

The \textit{SPARQL endpoint} allows querying of triples in the Instance layer using the SPARQL query language. This endpoint provides a flexible and SW-compliant way to query for instance data, and leverages the grounding of the KnowledgeStore data model in Knowledge Representation (KR) and SW best practices. Resource and Mention data are (currently) not accessible via SPARQL for technical reasons, since a different backend is used for storing them due to their larger volume (see Section 6.2.4).

For both endpoints, access control is employed to restrict the usage of the KnowledgeStore API only to authorized clients. Authentication is based on separate username/password credentials for each authorized client, while access may be limited to specific layers of the KnowledgeStore. While the two endpoints are language- and platform-neutral and thus allow the integration of the KnowledgeStore in any computing environment, for clients developed in Java a specific client library is also offered to ease the interaction with the KnowledgeStore and take care of the optimal use of the two endpoints.

### 6.2.3 User Interface

While the KnowledgeStore can be programmatically accessed by clients through its ReST API, human users can easily interact with the KnowledgeStore through the \textit{KnowledgeStore User Interface} (UI),\textsuperscript{13} a simple web-based application enabling users to inspect and navigate stored contents without having to develop applications accessing the ReST API. The UI offers two core operations:

\textsuperscript{12}Atomicity, Consistency, Isolation and Durability—see \url{http://en.wikipedia.org/wiki/ACID}

\textsuperscript{13}The KnowledgeStore UI can be seen in action in the demo video at \url{http://youtu.be/YVOQaljLta4}.
Figure 6.4. KnowledgeStore UI: (a) resource lookup; (b) mention lookup; (c) SPARQL query.
• the lookup operation, which, given the URI of an object (i.e., resource, mention, instance), retrieves all the KnowledgeStore content about it. Figures 6.4a and 6.4b show the output of a lookup operation for a resource and for a mention;
• the SPARQL query operation, with which arbitrary SPARQL queries can be run against the KnowledgeStore SPARQL endpoint, obtaining the results directly in the browser or as a downloadable file in various formats. Figure 6.4c shows an excerpt of the results obtained by running a query in the SPARQL tab of the UI.

These two operations are seamlessly integrated in the UI to offer a smooth browsing experience to users. For example, it is possible to directly invoke the lookup operation on any instance returned in the result set of a SPARQL query. Similarly, when performing the lookup operation on a resource, all mentions occurring in the resource are highlighted (see the Resource text box in Figure 6.4a) with a different color for the various mention types (e.g., person, organization, location, event), and by clicking on any of them the user can access all the details for that mention (see Figure 6.4b). Finally, the lookup of a mention (see Figure 6.4b) returns the attributes of the selected mention (box Mention data) as well as its occurrence in the containing resource (box Mention resource) and the structured description of the instance it refers to (box Mention referent), capturing in a single page the three representation layers of the KnowledgeStore as well as the role of mentions as a bridge between unstructured and structured content.

In addition to the lookup and SPARQL operations, and integrated with them, the UI also allows generating informative reports that aggregate information from different layers of the KnowledgeStore. For instance, given an instance URI, the entity mentions (aggregate) report (exemplified in Figure 6.5a for instance dbpedia:General_Motors) produces a sortable and filterable table with all the distinct ⟨RDF property, RDF value⟩ attribute pairs describing the mentions of that instance, with the number of mentions each pair occurs in. This report makes easy spotting wrong attribute pairs (e.g., “Genetically Modified” being a mention of dbpedia:General_Motors), which can be investigated by listing the corresponding mentions in another entity mentions report (Figure 6.5b). Additional reports are available and, altogether, provide concrete tools for spotting KE errors.

6.2.4 Architecture

The internal KnowledgeStore architecture is centered around the KnowledgeStore Server (Figure 6.6), a specifically developed software component that implements the operations of the CRUD and SPARQL endpoints, handling global issues such as access control, data validation and operation transactionality; it also provides the KnowledgeStore UI. Data storage is delegated by the KnowledgeStore Server to three software components distributed on a cluster of machines: Hadoop14, HBase15, and Virtuoso16.

14 http://hadoop.apache.org/
15 http://hbase.apache.org/
16 http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/
The **Hadoop HDFS** distributed filesystem provides a reliable and scalable storage for the files holding the representations of resources (e.g., texts and linguistic annotations of news articles). HDFS provides transparent data distribution and replication, as well as fault tolerance with respect to single node failures.

The **HBase** column-oriented store provides database services for storing and retrieving semi-structured information about resources and mentions. HBase builds on Hadoop HDFS and inherits its reliability and scalability characteristics, being particularly suited for random, real time read/write access to huge quantities of data, when the nature of data does not require a relational model (like in the case of resource and mention data). In the current setup, each resource and mention is stored in HBase as a row indexed by its URI, using dictionary encoding techniques to build a compressed, binary representation of resource and mention attributes. This solution allows for optimal lookup performances, and is insensitive to the number of mentions per resource, i.e., it works equally well with
very small and very large resources—what matters it the total number of objects stored.\textsuperscript{17} On the other hand, retrieval by filter condition on one or more attributes often requires full table scans, a situation that is mitigated by the possibility to distribute and parallelize such scans over all the nodes forming the HBase cluster.

The Virtuoso triplestore indexes triples to provide services supporting reasoning and online SPARQL query answering. Virtuoso has been selected motivated by its competitive performances in recent benchmarks (e.g., the April 2013 BSBM benchmark\textsuperscript{18}), further improved in the latest releases, as well as for its availability for both a single-machine and a cluster deployment configurations. Triples are stored in Virtuoso within named graphs, which can themselves be the subjects of metadata triples that specify properties applying to all the triples in a graph, such as the confidence and provenance metadata generated by PIKES (Section 5.1). Virtuoso supports a limited form of query-time RDFS reasoning but we do not use it. Instead, we perform forward-chaining RDFS reasoning with RDFpro (Chapter 7) to materialize inferrable triples before loading data into Virtuoso. For the time being, we limit reasoning to background knowledge only (i.e., triples coming from LOD resources), as we are still working towards a principled and scalable solution for dealing with the noise and inconsistencies in instance data coming from NLP processing.

Concerning the choice of the storage backend(s), it is worth noting that none of Hadoop HDFS, HBase, and Virtuoso can store all the data alone, and hence their combination

\textsuperscript{17}Of course, the distribution of mentions across resources affects the selectivities of certain relations (e.g., \texttt{kem:refersTo}) and thus query performances; moreover, managing huge resources is supported in the ReST API but not in the current implementation of the UI.

\textsuperscript{18}\url{http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V7/}

---

Figure 6.6. KnowledgeStore Architecture. Hadoop and HBase comprise multiple, distributed processes, while the KnowledgeStore Server and Virtuoso are single processes.
is crucial to realize a hybrid storage system like the KnowledgeStore. In fact, the use of a triplestore for Instance layer triples currently represents the state-of-the-art choice for providing efficient SPARQL access to this kind of data. At the same time, storing mention and resource data in a triplestore is problematic for large datasets, hence an additional storage backend is needed (HBase). Finally, large textual content is poorly supported in triplestores and databases (both relational and NoSQL), and is best stored in a filesystem or similar structure (Hadoop HDFS).

Finally, not shown in Figure 6.6 are additional tools and scripts used to: (i) enforce the transactional guarantees of the KnowledgeStore API operations; (ii) synchronize access and management of HBase nodes; and (iii) manage the complexity of software deployment in a cluster environment (e.g., the management scripts for infrastructure deployment, start-up and shut-down, data backup and restoration, and gathering of statistics).

6.3 Evaluation

In this section we report the results of a number of experiments aimed at assessing the performances of the KnowledgeStore. We focus on two core operations that are relevant for the practical adoption of the system: data population, analyzed in Section 6.3.1, and data retrieval, analyzed in Section 6.3.2. All the experiments:

- used real world data taken from the Cars (Ver. 2) KnowledgeStore instance populated within the NewsReader project (Section 9.1.3), comprising ~1.25M news articles about the global automotive sector, the corresponding 1.25M NLP annotation files (~2.5M resources total), and 205M mentions (163 mentions per news on average, standard deviation = 58), provided by LexisNexis for internal use in NewsReader.
- were conducted on a cluster of five servers connected by a Gigabit LAN: one running the KnowledgeStore Server and Virtuoso, having 1 TB disk, 32 GB RAM and two Intel Xeon E5-2440 CPUs (24 logical cores, 12 physical); the others running Hadoop and HBase, each having 1 TB disk, 32 GB RAM (except a server with 8 GB only) and two Intel Xeon E6545 CPUs (24 logical cores, 12 physical).

6.3.1 Data Population Performances

As previously discussed, the KnowledgeStore offers several data upload operations that support different population workflows. Here we consider the workflow adopted in NewsReader (see Section 9.1), as we deem it representative of a broad range of usage scenarios.

---

19 As confirmed in our usage experience, several hundreds of mentions can be extracted from a resource, each of them described with tens of attributes. Transformed to RDF triples, this means a few thousands triples for each resource. As a triplestore can hardly scale beyond a few billions of triples, storing all the data in a single triplestore would limit the overall system scalability to a few millions of documents. While distribution techniques can be used to divide triples and query load among multiple triplestores, this is still a subject of research that we leave for future investigation.

20 http://www.lexisnexis.nl/
Table 6.1. KnowledgeStore population time and rate based on number of mentions per news.

<table>
<thead>
<tr>
<th>Batch</th>
<th>Avg. mentions per news</th>
<th>Population time [h]</th>
<th>Population rate [news/h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>44</td>
<td>0.51</td>
<td>11,651</td>
</tr>
<tr>
<td>large</td>
<td>300</td>
<td>1.32</td>
<td>4,537</td>
</tr>
</tbody>
</table>

In this workflow, both the Resource and Mention layers are populated first with the results of the single-resource Knowledge Extraction Processors; then, the Instance layer is populated with the instances and triples extracted by the cross-resource Knowledge Extraction Processors, as well as with background knowledge. These two steps have very different performances: while populating the Instance layer is fast—400K triple/s, corresponding to 4M news articles per hour given an average of 350 triples per news article in the Cars (Ver. 2) dataset\(^\text{21}\)—the population of the Resource and Mention layers is around three order of magnitude slower—\(\sim 8K\) news articles per hour on average—and thus dominates and determines the overall population performances.

We now concentrate on the performances of the population of the Resource and Mention layers alone, and investigate whether and how they are impacted by the average resource size (number of mentions per news article) and overall dataset size (number of resources).

**Impact of resource size** The size of a news article can be expressed in many ways: e.g., file size, number of metadata attributes, number of words, number of mentions it contains. Among them, the number of mentions is the property most impacting on the performances of the population process, as each mention must be inserted in HBase and linked to the corresponding news article and instance, and in the KnowledgeStore implementation these operations are more expensive than storing the news article file.

To show the impact of the number of mentions per news article, we conducted an experiment where we populated the Resource and Mention layers of the KnowledgeStore under test with two batches of 6000 news articles from the Cars (Ver. 2) dataset, called ‘small’ and ‘large’, containing respectively news articles with 44 and 300 mentions (these are the smallest and largest numbers in the dataset). We started with an empty KnowledgeStore and measured the population time and rate for each batch.

Table 6.1 reports the results obtained. The difference in performances between the two batches is significant and confirms that the population rate inversely correlates with the average numbers of mentions per news article.

**Impact of dataset size** A degradation of the population rate can be reasonably expected as the amount of data stored in the KnowledgeStore increases and storing and indexing new data become more expensive. We thus conducted another experiment to assess the

---

\(^{21}\)We ignored the time needed for populating the background knowledge, whose size does not depend on the number of news articles populated. In fact, with a rate of 400K triple/s, storing a representative background knowledge dataset of 100M DBpedia triples takes only 4 minutes, which is negligible if compared to the total population time measured in the order of hours.
Table 6.2. KnowledgeStore population rate based on amount of data already stored (the number of resources is twice the number of news articles as it includes also the NLP annotation files).

<table>
<thead>
<tr>
<th>Population stage</th>
<th>Data already stored</th>
<th>Population rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Resources</td>
<td>Mentions</td>
</tr>
<tr>
<td>begin</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>middle</td>
<td>1,239,270</td>
<td>105,674,491</td>
</tr>
<tr>
<td>end</td>
<td>2,296,874</td>
<td>189,360,192</td>
</tr>
</tbody>
</table>

The extent of this phenomenon and, consequently, the scalability of the population process with large dataset sizes. We populated the Resource and Mention layers of the KnowledgeStore under test with the Cars (Ver. 2) dataset. We measured the (instantaneous) population rate at the beginning, in the middle and around the end of the population process, and took note of the state of the system when the three measurements were made, which consists of the number of resources and mentions already stored and the disk space used.

Table 6.2 reports the results obtained. The three population rates measured are similar and do not exhibit any clear trend, thus suggesting that the population rate can be considered roughly constant during the whole population process. This finding is consistent with the performance characteristics of the technologies used—especially HBase and Hadoop HDFS—with small differences in rate imputable to minor differences in the populated news articles or in the occasional triggering of background maintenance processes in HDFS and HBase (e.g., HBase table compaction). Although the results of the experiment cannot be generalized to datasets bigger than the one considered (we expect major degradation and eventual failure when approaching the storage capacity of the Hadoop cluster), they show nevertheless that consistent population performances can be achieved given the software infrastructure the KnowledgeStore builds on.

While population rates in the order of few thousands of news articles per hour may seem inadequate, it must be noted that these rates comprise the indexing of the news text, its NLP annotations and several hundreds of mentions on average, for a total of several MBs of data. For comparison, the processing required to produce the NLP annotations and to extract mentions is at least one order of magnitude slower—tens of seconds vs. ~0.5s for a Cars (Ver. 2) news or similar text, both considering PIKES (Chapter 5) and the NewsReader KE pipeline (Agerri et al., 2015)—making the population cost negligible.

6.3.2 Data Retrieval Performances

In this section we assess the performances of the data retrieval operations offered by the KnowledgeStore (SPARQL queries and resource, mention, and file retrieval) with different dataset sizes and numbers of concurrent clients.

To this purpose, we populated the KnowledgeStore under test with different test datasets of increasing size but similar schema and characteristics, all obtained from the Cars (Ver. 2) dataset. We then selected a set of parametric retrieval requests that are
Table 6.3. KnowledgeStore retrieval evaluation: test datasets (smallest to largest).

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale factor</td>
<td>1.0</td>
<td>2.1</td>
<td>3.5</td>
<td>7.6</td>
<td>15.4</td>
</tr>
<tr>
<td>Resources</td>
<td>163,752</td>
<td>341,464</td>
<td>572,690</td>
<td>1,239,270</td>
<td>2,519,496</td>
</tr>
<tr>
<td>News Articles</td>
<td>81,876</td>
<td>170,732</td>
<td>286,345</td>
<td>619,635</td>
<td>1,259,748</td>
</tr>
<tr>
<td>NAF Documents</td>
<td>81,876</td>
<td>170,732</td>
<td>286,345</td>
<td>619,635</td>
<td>1,259,748</td>
</tr>
<tr>
<td>Mentions</td>
<td>14,031,629</td>
<td>28,836,259</td>
<td>48,329,826</td>
<td>105,674,491</td>
<td>205,114,711</td>
</tr>
<tr>
<td>Instances</td>
<td>2,052,664</td>
<td>4,159,978</td>
<td>6,920,586</td>
<td>14,572,870</td>
<td>27,123,724</td>
</tr>
<tr>
<td>Events</td>
<td>1,829,866</td>
<td>3,752,010</td>
<td>6,285,449</td>
<td>13,398,806</td>
<td>25,156,574</td>
</tr>
<tr>
<td>Persons</td>
<td>81,265</td>
<td>152,951</td>
<td>241,774</td>
<td>446,356</td>
<td>729,797</td>
</tr>
<tr>
<td>Organizations</td>
<td>94,693</td>
<td>177,171</td>
<td>281,526</td>
<td>540,077</td>
<td>947,262</td>
</tr>
<tr>
<td>Locations</td>
<td>46,840</td>
<td>77,846</td>
<td>111,837</td>
<td>187,631</td>
<td>290,091</td>
</tr>
<tr>
<td>Triples</td>
<td>128,035,674</td>
<td>159,180,410</td>
<td>201,090,830</td>
<td>322,981,854</td>
<td>535,035,576</td>
</tr>
<tr>
<td>from Mentions</td>
<td>32,060,740</td>
<td>63,205,476</td>
<td>105,115,896</td>
<td>227,006,920</td>
<td>439,060,642</td>
</tr>
<tr>
<td>from DBpedia</td>
<td>95,974,934</td>
<td>95,974,934</td>
<td>95,974,934</td>
<td>95,974,934</td>
<td>95,974,934</td>
</tr>
<tr>
<td>Total Disk Space (GB)</td>
<td>30</td>
<td>50</td>
<td>76</td>
<td>152</td>
<td>260</td>
</tr>
</tbody>
</table>

representative of possible interactions of the user with the system for these datasets. We call request mix the instantiation of these requests for a specific set of parameter values and define the evaluation of a request mix as the sequential evaluation of the requests of the mix. For each test dataset and number of clients (independent variables) we simulated the concurrent evaluation by these clients of a large number of request mixes, and measured the overall request throughput and the average request evaluation time (dependent variables).

In the following, we describe the test datasets, the parametric requests, and the test procedure we adopted, and we report and discuss the obtained results. While the test datasets are copyrighted and cannot be made publicly available, the test tools used and their configurations are both available on the KnowledgeStore website and can be used to perform similar evaluations with different datasets or parametric requests.23

**Test datasets** Starting from the Cars (Ver. 2) dataset, we built five test datasets of increasing size (D1 to D5) by selecting only specific subsets of the source dataset. Table 6.3 describes the obtained datasets. The scale factor, computed as the ratio of the numbers of resources in a dataset and the number of resources in D1, provides an indication of the relative dataset sizes. They have been chosen to roughly follow a logarithmic scale, with deviations caused by the practical need to base the selection on the available news article batches forming the Cars (Ver. 2) dataset.

**Parametric requests** Table 6.4 reports the name and informal description of the 14 parametric requests selected for the test (parameters are emphasized in the description)
starting from the methods of the NewsReader Simple API (Hopkinson et al., 2014), an
API built on top of the KnowledgeStore to serve practical data analysis needs and used by
data analysts and journalists in several hackathon events (Section 9.1). More details on the
selection and the full specification of the parametric requests are contained in Appendix A.

Overall, the parametric requests and their sequential evaluation within a request mix simulate the typical activities of a user exploring the dataset:

- the user searches for events based on certain properties, such as event year, type,
term, involved actor URIs, and actor types (requests \texttt{sparql4} to \texttt{sparql8}); in order
to constrain these properties, the user may have first to search for a specific actor
(\texttt{sparql1}, \texttt{sparql2}) or get an idea of what event terms are in the dataset (\texttt{sparql3});
- the user then selects an event and retrieves all the information about it (\texttt{sparql9});
- the user selects an event actor and gets the corresponding description (\texttt{sparql10}),
including other persons related to the actor (\texttt{sparql11}) and all the events the actor
participates in (\texttt{sparql12});
- the user chooses a resource mentioning the selected event and retrieves its text
(\texttt{crud2}), metadata and mentions (\texttt{crud1}), i.e., the information needed to build a
visualization such as the one of Figure 6.4a.

\textbf{Test procedure} A single test consists in the evaluation of randomly selected request
mixes by one or more clients for a fixed period of time, after which performance metrics
are produced; clients operate concurrently but each client submits its requests sequentially
(as a user would do). For each dataset, multiple tests were performed according to the
following test procedure:

1. the dataset is loaded into the KnowledgeStore;
2. 1M randomly chosen request mixes are generated, for later use in the test;
3. the KnowledgeStore is restarted, so to begin from a clean state;
4. a warm-up test with 24 clients is run for 45 minutes, discarding its results; the warm-up allows for the initialization of the system and its caches, leading to a steady, optimal performance state;
5. the test with one client is run for 90 minutes, a time large enough to perform a number of request mixes comparable to the one of the other tests;
6. the tests with 2, 3, 4, 6, 8, 12, 16, 24, 32, 48 and 64 clients are run sequentially, 30 minutes each.

To support this procedure two specific tools have been developed and made available on the KnowledgeStore website: (i) the query test generator tool produces an arbitrary number of request mixes by sampling and joining the results of auxiliary queries that extract the admissible parameter values; and, (ii) the query test driver takes the produced request mixes and performs a single test according to the procedure described above, recording several performance figures for later analysis.

**Test results** The two line charts of Figure 6.7 show respectively the throughput measured in request mixes per hour (a), and the average request evaluation time (b) as functions of the number of concurrent clients, with a line for each test dataset.

As one could expect, adding new clients determines an increase of throughput with minor changes of the evaluation time up to a certain threshold, after which all the physical resources of the system (mainly CPU cores) are saturated, the throughput remains (almost) constant, and the evaluation time increases linearly as requests are queued for later evaluation. In the system under test the threshold is located around 12 clients.
(vertical lines in the charts), a quantity that matches the number of physical CPU cores available to the Virtuoso triplestore. This correspondence is explained by the fact that the majority of parametric requests are SPARQL queries that end up hitting Virtuoso. Nevertheless, the request mixes also include CRUD requests that ultimately hit the HBase and Hadoop HDFS clusters and may scale well beyond 12 clients: this fact likely explains the slight increase in throughput after the 12 clients threshold for the smallest datasets, where the performances of Virtuoso impact less.

While quantifying precisely the effect of the dataset size on retrieval performances is difficult, as there are many notions of ‘size’ to account for (number of news articles, resources, triples, instances), it is interesting to note that a ~15 times increase in the number of news articles, from $D1$ (81K news articles) to $D5$ (1.3M news articles), caused ‘only’ a ~2 times decrease in the throughput, from 21,126 to 10,212 requests/h for 64 clients. As the evaluation is made on real-world data, this finding is particularly significant for the practical adoption of the system. The line charts also feature local throughput maxima as well as global evaluation time minima around 4/6 clients, two features whose explanation requires further experimental investigation.\footnote{We suspect that possible causes include (but may not be limited to) the better/worse use of caches at various levels of the system and increased synchronization overhead beyond a certain level of concurrency.}

The bar chart of Figure 6.8 shows, for each parametric request, the average evaluation times for the different test datasets. From the graph it is clear that some parametric requests are much more expensive than others and the performances of some requests (especially the slowest SPARQL queries) degrade more markedly with an increase in dataset size. An analysis of the four most expensive parametric requests ($\text{sparql \text{6}}, \text{sparql \text{3}}, \text{sparql \text{8}}, \text{sparql \text{7}}$).
sparql 8, sparql 7) shows that even if they return few results (due to the use of the SPARQL LIMIT clause) they all present characteristics typical of analytical queries: they are not selective and they match, join, sort and aggregate large amounts of instance data. For instance, sparql 6 has to consider all the events of a certain year (in the order of millions) and all the instances of a given actor type (hundreds of thousands), join them based on the participation relation, and sort the results (possibly millions of tuples) by date to return only the first 20. On the other end, the least expensive queries sparql 4, sparql 12, sparql 9, sparql 11, and sparql 5 are much more selective: sparql 9 is the lookup of a single event, while the other queries consist essentially in the lookup of the events related to a specific actor, whose number is limited and largely independent of the dataset size. In terms of evaluation time, CRUD retrieval operations are situated halfway; while crud 1 (resource and mention retrieval hitting HBase) is slower with larger datasets, the performances of crud 2 (file download hitting HDFS) are largely independent of the dataset size.

6.4 Related Work

The development of frameworks able to store integrated and interlinked unstructured and structured contents has not been deeply explored in literature, although some relevant works closely related to the problems here addressed do exist: KIM, Stanbol, and LMF. KIM (Popov et al., 2003), now evolved into the Ontotext Semantic Platform,\(^\text{25}\) aims at providing a platform for semantic annotations of documents, focusing on named entity recognition and linking to a knowledge base of known instances. The platform’s main components are a document index, a knowledge base and an annotation pipeline. The document index, based on Lucene\(^\text{26}\), stores documents with their metadata and the instances recognized within them. The knowledge base contains the RDFS description of ∼80K instances of international relevance (background knowledge) as well as instances extracted from documents, based on a specifically-designed ontology (KIMO) defining ∼150 top-level instance classes and associated properties. The annotation pipeline is based on the GATE (Cunningham, 2002) NLP suite extended to leverage information in the knowledge base, and allows the automatic annotation of documents with the instances they contain, typed with respect to KIMO and linked to known instances in the knowledge base. Several APIs and UIs are provided for document storage and annotation, as well as for retrieving instances and documents using queries combining keywords and instances, and allowing the navigation from documents to referenced instances and back. KIM has been used in production at several news providers such as BBC, more recently adopting the PROTON upper ontology\(^\text{27}\) in place of KIMO and selected LOD data as background knowledge. The methodology and the software architecture for these applications are described by Georgiev et al. (2013). Compared to our work, the KE pipeline in KIM is

\(^{25}\)http://www.ontotext.com/products/ontotext-semantic/
\(^{26}\)http://lucene.apache.org/
\(^{27}\)http://en.wikipedia.org/wiki/PROTON
fixed and closely tied to a specific ontology for instances (KIMO, then PROTON), whereas the KnowledgeStore is agnostic with respect to which pipeline, ontologies and background knowledge are used.

Stanbol (Gönül and Sinaci, 2012) is a modular server exposing a configurable set of ReST services for the enhancement of unstructured textual contents, originated in the IKS Project.\textsuperscript{28} The main goal of Stanbol is to complement existing CMSs with semantic annotation, indexing, and retrieval functionalities. CMS documents and their metadata are fed to the Stanbol server, where a pipeline of content enhancers is applied to extract instances and additional metadata (e.g., language, topics). Extracted data are augmented with LOD data, and the result is indexed inside Stanbol in a triplestore (similar to the KnowledgeStore) as well as a SOLR\textsuperscript{29} full-text index, supporting respectively SPARQL queries and keyword search. While the KnowledgeStore provides a scalable and reliable primary storage for resources, Stanbol is mainly focused on their indexing for search purposes, and thus their main storage remains in external CMSs.

LMF (Linked Media Framework, Kurz et al., 2014) offers storage and retrieval functionalities for multimedia contents annotated with LOD data. Annotations are provided by external content enhancers such as Stanbol, while LMF focuses on storage and retrieval services as the KnowledgeStore. Similarly to Stanbol, the LMF data server is based on a Sesame (Broekstra et al., 2002) triplestore storing annotations as RDF triples, and on a SOLR full-text index storing document texts as well as selected metadata and annotation values chosen via XPath-like LDPath\textsuperscript{30} expressions; the two storages enable respectively SPARQL queries and keyword-based search. As in the KnowledgeStore, a ReST API extending the Linked Data HTTP publishing scheme allows read/write access to contents.

Compared to the KnowledgeStore, KIM, Stanbol, and LMF all adopt a 2-layer model consisting only of resources (text and metadata indexed in a full-text index) and instances (triples indexed in a triplestore). Indeed, storing and querying mention attributes is not a goal of these frameworks. Although mention data could be stored as additional attributes of resources and/or instances, this is not the intended use of these layers and this expedient may lead to inefficiencies or may be not feasible at all.\textsuperscript{31} On the other hand, using the KnowledgeStore as a two-layer system is possible too, but with a small overhead imposed by the unused Mention layer. Therefore, a fair quantitative comparison between the KnowledgeStore and these frameworks is not possible, as they provide different feature sets and they target different usage scenarios. Beyond the different number of layers, another distinctive feature of the KnowledgeStore compared to KIM, Stanbol, and LMF is its use of named graph to track the provenance of instances and triples.

Apart the mentioned works, some investigations were carried out on document repositories based on semantics (e.g., Bang and Eriksson, 2006; Eriksson, 2007). In these

\textsuperscript{28}http://www.iks-project.eu/
\textsuperscript{29}http://lucene.apache.org/solr/
\textsuperscript{30}http://code.google.com/p/ldpath/
\textsuperscript{31}For instance, storing mentions as instance data in the triplestore may lead to an ‘explosion’ of its size.
approaches, ontologies encoding the domain vocabulary and the document structure are used for annotating documents and document parts. However, the repositories adopting these approaches: (i) emphasize the document structure (e.g., tables, title) rather than content, (ii) do not foresee an integrated framework for storing semantic content and unstructured documents together, and (iii) are not meant to be used in big data contexts.

Relevant to the KnowledgeStore contribution is also the work by Croset et al. (2010). The authors present a framework, based on a RDF triplestore, that enables querying the bioinformatics scientific literature and structured resources at the same time, for evidence of genetic causes, such as drug targets and disease involvement. Differently from our approach, this work does not support storing unstructured content (triplestores currently provide only a limited support for integrating knowledge with unstructured resources, often consisting in simple full text search capabilities on RDF literals), and the framework is focused only on specific types of named entities appearing in the unstructured content, whereas a rich, unconstrained set of instances and mentions can be managed in the KnowledgeStore. Another relevant work, in the biomedical domain, is Semantic Medline,\(^{32}\) a web application that summarizes MEDLINE citations returned by a PubMed search. NLP is performed to extract semantic predications from titles and abstracts (the equivalent of triples about instances in the KnowledgeStore Instance layer). However, differently from the KnowledgeStore, Semantic Medline has a fixed domain-specific data model, built tailored on that application, and predications can be effectively navigated only on a reasonably small selection of citations (max 500 on the website) with no possibility to perform structured queries on the whole corpus (to this respect, a global index of predications seems missing). Furthermore, while capable of handling large quantities of resources (21M MEDLINE citations, see Jonnalagadda et al., 2012), the semantic content extracted is proportionally rather small (∼57.6M predications of 26 types; cf. Instance layer statistics of KnowledgeStore instances populated in NewsReader, Table 9.1).

Although exploited in a different context, dealing with much smaller quantities of content, also Semantic Desktop applications such as MOSE (Xiao and Cruz, 2006) and Nepomuk\(^{33}\) are partly related to the KnowledgeStore contribution. Semantic Desktop applications enrich documents archived on the personal PC of a user with annotations coming from ontologies. However, annotations are attached to the object associated to the document, and not to its content, thus not fully supporting the interlinking between unstructured and structured content.

6.5 Summary

In this chapter we described the KnowledgeStore: a scalable, fault-tolerant, and SW grounded open-source storage system for interlinking text and structured knowledge. Besides presenting its design, functionalities, and implementation, we reported on a number

\(^{33}\)http://nepomuk.semanticdesktop.org/
of experiments measuring the data population and data retrieval performances of the system. These experiments, together with the concrete experience of using the KnowledgeStore in the NewsReader project (see Chapter 9), demonstrate the appropriateness and adequateness of the KnowledgeStore to cope with the goals it was designed for.

Thanks to the explicit representation and alignment of information at different levels, from unstructured to structured knowledge, the KnowledgeStore enables the development of enhanced applications, and favors the design and empirical investigation of information processing tasks otherwise difficult to experiment with. On the one hand, the possibility to semantically query the content of the KnowledgeStore, with requests combining knowledge from structured sources and unstructured sources, allows a deeper exploration and analysis of stored data, a capability particularly useful in applications such as decision support. On the other hand, the joint storage of structured knowledge (both background and extracted knowledge), the resources it derives from, and mention information—all effectively accessible through a single API—provides an ideal scenario for developing, debugging, training, and evaluating tools for a number of NLP and knowledge processing tasks. NLP tasks can benefit from the availability of background knowledge and the textual grounding of mentions, exploiting them to improve their performance: an example is Coreference Resolution (i.e., identifying that two mentions refer to the same instance of the world), especially in cross-document settings. Similarly, Knowledge Extraction Processors can exploit the linking of structured knowledge to mentions, and the linguistic features attached to them, to perform tasks such as knowledge fusion (i.e., the merging of possibly contradicting information extracted from different sources).
Chapter 7

RDFpro: Local RDF Processing using Streaming and Sorting

Several RDF processing tasks are common in practice. We already encountered some of them in previous chapters: OWL 2 RL inference and VOID (Vocabulary of Interlinked Datasets, Alexander et al., 2009) statistics extraction in PreMOn (Chapter 3); rule evaluation and owl:sameAs smushing—i.e., replacing coreferring URI aliases with a “canonical” URI—in PIKES (Chapter 5); and RDFS inference in the KnowledgeStore (Chapter 6). Other tasks, such as triple-level filtering and/or transformation, data deduplication, and various format conversion and data packaging tasks are common when preparing existing RDF datasets—e.g., from Linked Open Data (LOD) sources—for application consumption, as will be described next concerning the preparation of DBpedia (Lehmann et al., 2015) background knowledge (Chapter 9).

Although tools do exist for these tasks, a Semantic Web (SW) practitioner typically faces two challenges. First, tool support is fragmented, often forcing a user to integrate many heterogeneous tools even for simple processing workflows. Second, tools coping with LOD dataset sizes in the range of millions to billions of triples often require distributed infrastructures such as Hadoop\(^1\) that are complex to set up and cannot be used efficiently (if at all) on a single commodity machine, due to their inherent complexity and overhead. Conversely, tools targeting local computation are often based on some form of data indexing (e.g., a triplestore) and typically scale as long as the index can be kept in main memory, incurring a severe performance hit when data has to be accessed on disk.

In this chapter we present RDFpro (RDF processor)\(^2\) a general-purpose, extensible, open source (public domain) Java library and command line tool that addresses these shortcomings (Section 7.1). On the one hand, RDFpro reduces integration efforts by providing out-of-the-box implementations—called processors—of common RDF processing tasks, as well as easy ways to add new processors and compose processors in complex processing pipelines. On the other hand, RDFpro targets local processing of large RDF datasets without requiring clusters and complex computing infrastructures, achieving vertical scalability through multi-threading and a processing model based on streaming and sorting, two scalable techniques well-known in the literature (O’Connell, 2009). Streaming consists in processing one triple at a time, and translates to efficient sequential I/O and low memory footprint. Sorting overcomes many of the limitations of a pure streaming

\(^{1}\text{http://hadoop.apache.org/}\)
\(^{2}\text{http://rdfpro.fbk.eu/}\)
model, supporting tasks such as duplicate removal, set operations, and grouping of data that must be processed together (e.g., all the data about an instance).

We evaluated RDFpro (Section 7.2) in four application scenarios—dataset analysis, filtering, merging, and massaging\footnote{Data massaging informally denotes all the ad-hoc transformation tasks necessary to make data better suited to a particular use (e.g., format conversion, value normalization).}—using a commodity workstation and processing billions of triples of popular datasets—DBpedia, Freebase\cite{Bollacker2008}, GeoNames\footnote{http://www.geonames.org/}—whose contents and sizes are representative of the ones typically faced by applications dealing with LOD data. The results obtained allow giving a positive answer to the question of whether the common RDF processing tasks here considered are feasible on large (LOD-sized) datasets by using streaming and sorting techniques on a single commodity machine. Compared to state-of-the-art tools for RDF processing (Section 7.3), RDFpro requires neither large amounts of memory (for data indexing) nor distribution on a cluster of machines for coping with large dataset sizes, replacing these elements with multiple streaming / sort passes on input data that, while possibly slower than other approaches (although not slower than other practically relevant operations, such as loading processed RDF data into a triplestore), are compatible with the use of a single commodity machine.

Summing up the results presented in this chapter (Section 7.4), and in the light of the practical experience of using RDFpro in different contexts, we deem RDFpro a tool that usable by casual users, and not just by developers, for addressing a variety of processing needs, making it a sort of “Swiss army knife” for exploring and manipulating RDF datasets.

Acknowledgments The material presented in this chapter was partially published (except @mapreduce and @rules processors) in \cite{Corcoglioniti2015d} and, previously, in \cite{Corcoglioniti2014}; a system demonstration is described in \cite{Corcoglioniti2015a}. RDFpro originated from the practical need of processing RDF background knowledge in NewsReader\footnote{http://www.newsreader-project.eu/}, a EU-supported project (FP7 ICT-316404).

\section{7.1 RDFpro Tool}

This section describes how streaming and sorting are combined in the RDFpro tool. We present RDFpro processing model in Section 7.1.1, its processors in Section 7.1.2, its implementation in Section 7.1.3, and its command-line and web-based usage in Section 7.1.4.

\subsection{7.1.1 Processing Model}

The processing model of RDFpro is centered around the concept of RDF processor. A processor \@P\footnote{Processors are denoted by \@ in RDFpro syntax.} (Figure 7.1a) is a software component that consumes an input stream of...
RDF quads—i.e., RDF triples with an optional fourth named graph component\(^7\)—in one or more passes, produces an output stream of quads and may have an internal state as well as side effects like writing or uploading RDF data.

Streaming characterizes the way quads are processed: one at a time, with no possibility for the processor to “go back” in the input stream and recover previously seen quads. Specifically, a processor declares the minimum number \(n \geq 1\) of passes it requires on its input. In the first \(n - 1\) passes (if any), the processor reads the input quads, which may arrive in any order, and updates its state. At pass \(n\) the input is read again and output quads are emitted for the first time. In the next passes, if required to accommodate the requirements of downstream processors, the processor is given again the same input quads and must emit the same output quads of pass \(n\), although their order may change.

Sorting is offered to processors as a primitive to arbitrarily sort selected data—possibly (a subset of) input quads—during a pass. Sorting is often combined to streaming in the literature as it overcomes many of the limitations of a pure streaming model (Aggarwal et al., 2004; O’Connell, 2009). In particular, it enables duplicates removal and set operations and provides the capability to group together information that may be scattered in the stream but must be processed together (e.g., all the quads about an instance when computing statistics). At the same time, most platforms provide an highly-optimized sorting utility that fully exploits available hardware resources: multiple CPU cores to parallelize the sorting algorithm, disk space to manage large datasets via (disk-based) external sorting\(^8\), and memory space to speed up processing.

---

\(^7\)The graph component is unspecified for triples in the default graph of the RDF dataset—see specifications of RDF 1.1 (Wood et al., 2014) and SPARQL (Harris and Seaborne, 2013); this allows using RDFpro on plain triple data.

\(^8\)http://en.wikipedia.org/wiki/External_sorting
Starting from the processors supplied with RDFpro (Section 7.1.2) or implemented by users, new pipeline processors can be derived by (recursively) applying sequential and parallel compositions. In a sequential composition (Figure 7.1b), two or more processors $\text{@P}_i$ are chained so that the output stream of $\text{@P}_i$ becomes the input stream of $\text{@P}_{i+1}$. In a parallel composition (Figure 7.1c), the input stream is sent concurrently to several processors $\text{@P}_i$, whose output streams are merged into a resulting stream using one of several possible set or bag operators (specified with a flag $f$ in figure and syntax), such as union with duplicates (flag $a$), union without duplicates ($u$), intersection ($i$), and difference ($d$) of quads from different branches. The number and orchestration of passes resulting from composition are automatically managed.

An example of composition is shown in Figure 7.1d, where a Turtle+gzip RDF file (file.ttl.gz) is read, TBox and VOID statistics are extracted in parallel and their union is written to an RDF/XML file (onto.rdf). Notably, I/O in the example do not use the input and output streams of the pipeline processor (dotted box in the figure), but rely on specific $\text{@write}$ and $\text{@read}$ processors whose side effects are dumping and augmenting the stream with the contents of external files (similar $\text{@upload}$ and $\text{@download}$ processors allow interfacing with SPARQL endpoints). These I/O processors provide a lot of flexibility in how data is read and written, as they can be placed at any point of a pipeline removing the limit of single input and output streams (indeed, the RDFpro tool relies on these processors for all the I/O, ignoring global input and output streams that are instead accessible when using RDFpro as a library).

### 7.1.2 Available Processors

Out-of-the-box, RDFpro includes the following processors for addressing many common RDF processing tasks:

- **$\text{@read}$**: Reads RDF file(s), emitting their quads together with the input stream. Files are read in parallel and, where possible, split in chunks that are parsed concurrently. May rewrite blank nodes on a per-file basis to avoid clashes.

- **$\text{@write}$**: Writes quads to a single RDF file or splits them to multiple files evenly, so to allow splitting large datasets; quads are also propagated in the output stream for downstream use. Parallel, chunk-based writing is supported as for $\text{@read}$.

- **$\text{@download}$**: Sends a query to a SPARQL endpoint to download quads that are emitted by augmenting the input stream. Both CONSTRUCT and SELECT queries are supported: the first can only return triples in the default graph; the latter produces bindings for specific variables $s$, $p$, $o$, $c$ that are used to build output quads.

- **$\text{@upload}$**: Uploads quads from the input stream to an RDF store using INSERT DATA calls of SPARQL, in chunks of a specified size; quads are also propagated in output.
@smush  Performs *smushing*, replacing the members of each `owl:sameAs` equivalence class with a canonical URI selected based on a ranked namespace list. Replaced URIs (aliases) are preserved and linked to the canonical URI via `owl:sameAs` quads.

@transform  Discards or rewrites input quads one at a time, either based on simple matching criteria or evaluating an arbitrarily complex JavaScript or Groovy\textsuperscript{9} script that can map each input quad to zero (i.e., discard) or more output quads. Scripts can also implement stateful computations, with hooks called to initialize the script and to emit quads with aggregate results after all the input has been processed.

@rdfs  Computes the RDFS deductive closure of an input stream consisting only of ABox quads. A fast, hard-coded implementation loads the TBox from a file and computes its closure first, using the resulting domain, range, sub-class, and sub-property axioms to perform inference on quads of the input stream one at a time, placing inferences in the same graph of the input quad.\textsuperscript{10} Specific RDFS rules may be optionally disabled to avoid unwanted inferences. OWL axioms may be (partially) reduced to corresponding RDFS axioms (e.g., `owl:equivalentClass` → `rdfs:subClassOf`).

@tbox  Filters the input stream by emitting only quads of TBox axioms, identified by searching for triples having certain terms from the RDF, RDFS, and OWL vocabularies in the predicate and object positions. Both RDFS and OWL axioms are extracted, even if the latter are not used by @rdfs.

@stats  Emits VOID structural statistics\textsuperscript{11} for its input. A VOID dataset is associated to the whole input and to each source whose URI is linked to named graphs in the data by a configurable property; class and property partitions are produced for each dataset. Additional terms extend VOID to express the number of TBox, ABox, `rdf:type`, and `owl:sameAs` quads, the average number of properties per instance, and informative labels and examples for TBox terms, viewable in tools such as Protégé\textsuperscript{12}.

@unique  Discards duplicates in the input stream. Optionally, it merges quads with the same subject, predicate and object but different graphs in a unique quad. To track provenance, this quad is placed in a graph representing the “fusion” of the source graphs and inheriting their descriptions (i.e., the quads having them as subject).

\textsuperscript{9}Groovy is a scripting language reusing Java syntax and libraries. Scripts are compiled to JVM byte-code with performances close to Java. See \url{http://groovy.codehaus.org/}.

\textsuperscript{10}This scheme avoids expensive join operations and works with arbitrarily large datasets whose TBox fits into memory. Inference is complete if: (i) \rdfs:domain, \rdfs:range, \rdfs:subClassOf or \rdfs:subPropertyOf axioms in the input stream are also in the TBox; and (ii) the TBox has no quad matching patterns:

\begin{itemize}
  \item X \rdfs:subPropertyOf \{\rdfs:subClassOf | \rdfs:domain | \rdfs:range | \rdfs:subPropertyOf\}
  \item X \{\rdftype|\rdfs:domain|\rdfs:range|\rdfs:subClassOf\} \{\rdfs:Datatype|\rdfs:ContainerMembershipProperty\}
\end{itemize}

\textsuperscript{11}Supports \void:classes, \void:properties, \void:entities, \void:triples, \void:distinctSubjects, \void:distinctObjects.

\textsuperscript{12}\url{http://protege.stanford.edu/}
Applies a custom map script (JavaScript or Groovy) to label and group input quads into partitions, each one reduced with a reduce script. A multi-threaded, non-distributed MapReduce\textsuperscript{13} implementation based on the sort primitive is used.

Emits the closure of input quads using a customizable set of rules, with support for the OWL Horst (ter Horst, 2005) and OWL 2 RL (Motik et al., 2012) rulesets. Rules heads and bodies are SPARQL graph patterns, with all the SPARQL constructs being allowed in rule bodies. A map script or specification can be specified to partition input quads (e.g., by graph) and apply rules on a per-partition basis. Optionally, the TBox and other static data used in the rules can be specified in an external file.

### 7.1.3 Implementation

RDFpro is implemented in Java on top of the open source Sesame (Broekstra et al., 2002) RDF library. It consists of a runtime where multiple processor plugins can be instantiated, assembled using sequential and parallel composition, and executed.

**Runtime implementation** The runtime defines the API of RDF processors and manages their lifecycle. Processors are Java classes extending RDFProcessor and declaring the number of passes they need. Each processor is attached to an input quad queue and an output quad sink. Input quads from the queue are “pushed” to the processor by invoking a callback method, using multiple threads from a common pool to process quads in parallel; other callbacks are invoked at the beginning and end of each pass to allow for initialization and completion of stateful computations. Output quads are emitted to the quad sink (a Sesame RDFHandler), with the runtime taking care of their downstream processing (if any). The design is inspired to SEDA (Staged Event Driven Architecture, Welsh et al., 2001), with processors playing a passive role and all the queue and thread management handled by the runtime with the goal of maximizing CPU usage.

Within the runtime, streaming is embodied in the RDFProcessor API and in the management of input and output streams. Sorting, instead, is realized as a reusable primitive that can be invoked by the runtime and by processor implementations. This primitive is realized on top of the native, highly-optimized sort Unix utility, using dictionary encoding techniques\textsuperscript{14} to compactly encode frequent RDF terms in sorted data, reducing its size (we measured ~40 bytes per quad on real-world data) and improving execution times at the price of some memory consumption for the dictionary and the loss of alphanumeric order of output quads (generally not required).

Processor composition is also managed by the runtime. Sequential composition and union with duplicates are computationally cheap, while the other forms of parallel com-

\textsuperscript{13}http://en.wikipedia.org/wiki/MapReduce

\textsuperscript{14}We encode TBox URIs of known vocabularies (from prefix.cc) with integers. Namespaces and local names of other URLs are separately encoded until the encoding tables are full, after which they are emitted unchanged. For literals we encode the language and datatype tags, but not their (arbitrarily long) labels.
position are more expensive due to their use of sorting. In particular, intersection and difference are implemented by appending a label identifying the branch to each quad, and then gathering and checking all the labels of a sorted quad to decide if to emit it.

**Processor implementation**  Due to their central role, the @read and @write processors feature multi-threaded implementations aiming at transferring data as fast as possible to avoid I/O bottlenecks. Multiple RDF files can be parsed and written in parallel and, for line-oriented RDF formats, a single file can be split in newline-terminated chunks that are processed concurrently to increase the data throughput.

The @smush processor performs two passes: the first to extract the owl:sameAs graph which is kept in memory; the second to replace URIs based on detected equivalence classes. Efficient memory consumption is achieved with a specialized data structure that uses a custom hash table with an open addressing scheme to index URIs; table entries contain also a next pointer that organizes URIs of an owl:sameAs equivalence class in a circular linked list, which expands as new owl:sameAs quads are encountered and allows the structure to grow linearly with the number of URI aliases.

The @stats processor is implemented by sorting the input stream twice (simultaneously within a single pass): based on the subject, to group quads about the same instance and compute instance-based and distinct subjects statistics; and based on the object, to compute distinct objects statistics. Partial statistics are kept in memory during processing.

The @rdfs processor performs TBox inference using an in-memory, semi-naive forward-chaining algorithm (Ceri et al., 1989). ABox inference is done one quad at a time, using multiple threads and special deduplication logic for removing as many duplicate inferred quads as possible, so to avoid an artificial “explosion” of the number of output quads.

The @rules processor adopts a streaming, quad-by-quad evaluation of rules with a single statement pattern (i.e., no joins) in their body, recurring to a semi-naive forward chaining approach, based on an in-memory index, only for rules whose body requires joining multiple input quads. If the (optional) TBox file is supplied, each rule is preprocessed by (i) identifying the statement patterns in its body that refer to TBox axioms, (ii) determining all the possible bindings for these patterns in the supplied TBox data, and (iii) for each binding, generating a simpler (one pattern less) rule. While this technique increases the number of rules, it also simplifies them and generally speeds up their evaluation by increasing the number of single-pattern rules that can be evaluated in streaming.\(^\text{16}\)

\(^\text{15}\)An alternative, more scalable (although slower) sorting-based implementation is realizable based on the algorithm for detecting connected components of a graph by O’Connell (2009).

\(^\text{16}\)We also experimented an alternative implementation based on Drools, a rule engine implementing the RETE algorithm—[http://www.drools.org/](http://www.drools.org/)—obtaining however worse performances and very high memory requirements. This is the consequence of the use of non-specialized data structures (Drools handles generic objects and not just RDF quads) and of the high number of tuples accumulated in the $\beta$-nodes of the RETE network for non-selective body patterns (e.g., $?s ?p ?p \land ?p rdfs:subPropertyOf ?q \rightarrow ?s ?q ?o$) for which a SPARQL evaluation is more efficient.
The \texttt{mapreduce} processor sorts input quads based on the label computed by the \textit{map} script, so that quads with the same label are close each other in the sorted stream, allowing the easy extraction of the partitions to process with the \textit{reduce} script.

### 7.1.4 Using RDFpro

RDFpro binaries and public domain sources are available on its website. RDFpro can be used in three ways: (i) as a command line tool capable of processing large datasets; (ii) as a web tool suited to smaller amounts of data uploaded/downloaded with the browser; and (iii) as a Java library\footnote{Available on Maven Central: \url{http://repo1.maven.org/maven2/eu/fbk/rdfpro/}.} embedded in applications. Users can extend RDFpro via custom scripts and rulesets, while developers can create new processors by implementing a simple
Java API and focusing on the specific task at hand, as efficient streaming, sorting, I/O, thread management, scripting, and composition facilities are already provided.

Examples of using RDFpro as a command line and web tool are shown in Figures 7.2a and 7.2b, where a pipeline is executed to compute the RDFS closure of some DBpedia data (70M triples) and return only rdfs:label triples of instances of type dbo:Company. The pipeline performs 6 tasks: (i) read data; (ii) compute RDFS closure using DBpedia TBox; (iii) keep rdf:type and rdfs:label quads; (iv) partition quads by subject, keeping partitions with object dbo:Company; (v) retain rdfs:label quads; (vi) write results. Instructions and reference documentation for using the library are provided on the website.

A video showing the usage of RDFpro is available on the website, together with a fully-working installation of the RDFpro web interface, where users can try arbitrary commands and processing tasks on data of their choice.

7.2 Evaluation

We evaluated RDFpro and its approach based on the use of streaming and sorting techniques by trying to answer the following research question:

Are relevant RDF processing tasks practically feasible on large datasets by using streaming and sorting techniques on a single commodity machine?

Here, for “relevant” we intend the RDF processing tasks commonly associated to the processing of LOD data (Heath and Bizer, 2011; Heitmann et al., 2013): data filtering and transformation, RDFS inference, owl:sameAs smushing, and statistics extraction. For “large” we intend typical LOD dataset sizes, i.e., millions to billions of triples. For “feasible” we mean the successful completion of tasks with an execution time that is “reasonable” if compared to competing single-machine approaches (where available), and to the time needed for consuming resulting data, which often consists in loading it in a production triplestore where data can be queried by applications and users.

To answer this research question, we performed an empirical evaluation of RDFpro in four broad, relevant usage scenarios that exemplify the considered RDF processing tasks. In the first three scenarios—dataset analysis (Section 7.2.1), filtering (Section 7.2.2) and merging (Section 7.2.3)—we conducted practical experiments using a commodity workstation\(^\text{18}\) and popular datasets (Freebase, DBpedia, GeoNames) whose contents and sizes are representative of the ones typically encountered by LOD applications. In the fourth scenario—dataset massaging (Section 7.2.4)—we categorize miscellaneous data massaging tasks that can be addressed with our approach and show its larger applicability; due to the simple processing involved we do not conduct experiments here, but just elaborate on how these tasks can be addressed using the processing model and processors described in Section 7.1. An extended description of the scenarios, including scripts for reproducing the experiments, is reported on RDFpro website.

\(^{18}\)Intel Core I7 860 CPU (4 physical cores, 8 logical), 16 GB RAM, 500 GB 7200 RPM HD, Linux 2.6.
Table 7.1. RDFpro dataset analysis scenario: results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. TBox</td>
<td>2,863</td>
<td>28,339</td>
<td>0.23</td>
<td>3.01</td>
<td>1.43</td>
<td>14.12</td>
<td>2,006</td>
</tr>
<tr>
<td>2. Statistics</td>
<td>2,863</td>
<td>28,339</td>
<td>0.13</td>
<td>1.36</td>
<td>0.34</td>
<td>3.36</td>
<td>8,443</td>
</tr>
<tr>
<td>1-2. Aggregated</td>
<td>2,863</td>
<td>28,339</td>
<td>0.36</td>
<td>4.35</td>
<td>0.34</td>
<td>3.36</td>
<td>8,426</td>
</tr>
<tr>
<td>3. Comparison</td>
<td>5,486</td>
<td>55,093</td>
<td>260</td>
<td>1,894</td>
<td>0.42</td>
<td>4.25</td>
<td>12,955</td>
</tr>
</tbody>
</table>

Figure 7.3. RDFpro dataset analysis scenario: pipeline.

7.2.1 Dataset Analysis

Dataset analysis comprises all the tasks aimed at providing a qualitative and quantitative characterization of the contents of an RDF dataset, such as the extraction of TBox and VOID statistics from the data. When processing RDF, dataset analysis can be applied both to input and output data. In the first case, it helps identifying relevant data and required preprocessing tasks, especially when the dataset scope is broad (as occurs with many LOD datasets) or its documentation is poor. In the second case, it provides a characterization of output data that is useful for validation and documentation purposes.

Experiment As a representative example of large-scale dataset analysis, we considered the tasks of extracting TBox and VOID statistics from Freebase data (2014/09/10 dump, 2863 MQ – millions of quads), whose schema and statistics are not available online, and the task of comparing this Freebase release with a previous release (2014/07/10 dump, 2623 MQ) in order to identify newly added triples.\(^{19}\)

We used the @tbox and @stats processors to extract TBox and VOID statistics, invoked both separately and aggregated in a pipeline processor as shown in the left part (marked as 1) of Figure 7.3. To extract new triples, we read both dataset releases and used parallel composition with the difference set operator to combine quads, as shown in right part (marked as 2) of Figure 7.3.

\(^{19}\)From this delta, TBox and VOID statistics can be extracted to get a concise summary of what was added. This analysis is analogous to (and computationally cheaper than) the one done on the whole Freebase and was thus omitted.
Table 7.1 reports the tasks execution times, throughputs, input and output sizes both in quads and compressed (gzip) bytes, as measured on our test machine. Additionally, when running the comparison task, we measured a disk usage of 92.8 GB for the temporary files produced by the sorting-based difference set operator (~18 bytes per input triple).

**Comment** Comparing the two Freebase releases resulted the most expensive task due to sorting and involved input size. When performed jointly, TBox and statistics extraction present performance figures close to statistics extraction alone, as data parsing is performed once and the cost of TBox extraction (excluded parsing) is negligible. This is an example of how the aggregation of multiple processing tasks in a single computation, enabled by RDFpro streaming model and composition facilities, can generally lead to better performances due to a reduction of I/O overheads for writing/reading intermediate files.

To provide an idea of how analysis results can be used to explore the dataset, Figure 7.4 shows the joint browsing of TBox and statistics in the Protégé tool, exploiting the specific concept annotations emitted by `@stats`. The class and property hierarchies are augmented with the number of instances and property triples (marked as 1 in the figure), as well as with the detected property usage (2), e.g., 0 for object property, 1 for inverse functional; each concept is annotated with an example instance and a VOID partition individual (3), which provides numeric statistics about the concept (4).

### 7.2.2 Dataset Filtering

When dealing with large RDF datasets, dataset filtering or *slicing* (Marx et al., 2013) is often required to extract a small subset of interesting data, identified, e.g., based on a previous dataset analysis (Section 7.2.1). Dataset filtering typically consists in (i) identifying the instances of interest in the dataset, based on selection conditions on
Table 7.2. RDFpro dataset filtering scenario: results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select instances</td>
<td>2,863</td>
<td>28,339</td>
<td>0.20</td>
<td>0.73</td>
<td>1.36</td>
<td>13.4</td>
<td>2,111</td>
</tr>
<tr>
<td>2. Extract quads</td>
<td>2,863</td>
<td>28,339</td>
<td>0.42</td>
<td>5.17</td>
<td>1.15</td>
<td>11.4</td>
<td>2,481</td>
</tr>
</tbody>
</table>

Figure 7.5. RDFpro dataset filtering scenario: pipeline.

their URIs, rdf:type or other properties; and (ii) extracting all the quads about these instances expressing selected RDF properties. These two operations can be implemented using multiple streaming passes.

**Experiment** We considered a concrete dataset filtering example where the dataset is Freebase (2014/09/10 dump, 2863 MQ), the instances of interest are musical groups (i.e., their rdf:type is fb:music.musical_group) that are still active (i.e., there is no associated property fb:music.artist.active_end), and the properties to extract are the group name, genre and place of origin (respectively, rdfs:label, fb:music.artist.genre and fb:music.artist.origin).

We implemented the task with two invocations of RDFpro as shown in Figure 7.5. The first invocation (marked as 1) generates an RDF file listing as subjects the URIs of the instances of interest; this is done with two parallel @transform processors, extracting respectively musical groups and no more active musical instances, whose outputs are combined using the difference set operator. The second invocation (marked as 2) uses another @transform processor to extract the desired quads, filtering them by predicate and requiring their subjects to be contained in the previously extracted file (whose URIs are indexed in memory by a specific function in the @transform expression).

Table 7.2 reports the execution times, throughputs, input and output sizes of the two invocations on the test machine.

**Comment** Although simple, the experiment shows how practical, large-scale filtering tasks are feasible using the streaming and sorting approach of RDFpro. Performances are worse than the ones obtainable using SPARQL in a triplestore, but better if one considers also the time needed for indexing data in the triplestore (see Section 7.2.5).
Table 7.3. RDFpro dataset merging scenario: results.

<table>
<thead>
<tr>
<th>Step</th>
<th>Input</th>
<th>Output</th>
<th>Throughput</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[MQ]</td>
<td>[MB]</td>
<td>[MQ]</td>
<td>[MB]</td>
</tr>
<tr>
<td>1. Transform</td>
<td>3,394</td>
<td>33,524</td>
<td>3,394</td>
<td>36,903</td>
</tr>
<tr>
<td>2. TBox extraction</td>
<td>3,394</td>
<td>36,903</td>
<td>&lt;1</td>
<td>4</td>
</tr>
<tr>
<td>3. Smushing</td>
<td>3,394</td>
<td>36,903</td>
<td>3,424</td>
<td>38,823</td>
</tr>
<tr>
<td>4. Inference</td>
<td>3,424</td>
<td>38,823</td>
<td>5,615</td>
<td>51,927</td>
</tr>
<tr>
<td>5. Deduplication</td>
<td>5,615</td>
<td>51,927</td>
<td>4,085</td>
<td>31,297</td>
</tr>
<tr>
<td>1-2 Aggregated</td>
<td>3,394</td>
<td>33,524</td>
<td>3,394</td>
<td>36,903</td>
</tr>
<tr>
<td>3-5 Aggregated</td>
<td>3,394</td>
<td>36,903</td>
<td>4,085</td>
<td>31,446</td>
</tr>
</tbody>
</table>

Figure 7.6. RDFpro dataset merging scenario: pipeline.

More complex filtering scenarios can be addressed using set operations for implementing conjunction, disjunction and negation of selection conditions, and with additional invocations of RDFpro that progressively augment the result (e.g., a third invocation can identify albums of selected artists, while a fourth invocation can extract the quads describing them). If keeping a set of previously matched instances in-memory is unfeasible, the `mapreduce` processor can be used to partition the data by instance and then operate on a per-instance basis, with no additional state maintained in-memory (see example of Section 7.1.4). In cases where the RDFpro model is still insufficient (e.g., due to the need for further aggregations or joins), the tool can still be used to perform a first coarse-grained filtering that reduces the number of quads and eases their downstream processing.

7.2.3 Dataset Merging

A common usage scenario is dataset merging, where multiple RDF datasets are integrated and prepared for application consumption. Data preparation typically comprises smushing, inference materialization and data deduplication (possibly with provenance tracking). These tasks make the use of the resulting dataset more easy and efficient, as reasoning and instance aliasing have been already accounted for.

Experiment We considered a concrete dataset merging scenario with data from Freebase (2014/09/10 dump, 2863 MQ), GeoNames (2013/08/27 dump, 125 MQ), and DBpedia in the
four languages EN, ES, IT and NL (version 3.9, 406 MQ without redirects, disambiguation, pages and revisions metadata), for a total of 3394 MQ.

Figure 7.6 shows the required processing steps. A preliminary processing phase (marked as 1) is required to transform input data and extract the TBox axioms required for inference. Data transformation serves (i) to track provenance, by placing quads in different named graphs based on the source dataset; and (ii) to adopt optimal serialization format (Turtle Quads) and compression scheme (gzip) that speed up further processing.\textsuperscript{20} The main processing phase (marked as 2) consists in the cascaded execution of smushing, RDFS inference and deduplication to produce the merged dataset. Smushing identifies owl:sameAs equivalence classes and assigns a canonical URI to each of them. RDFS inference excludes rules rdfs4a, rdfs4b and rdfs8 to avoid materializing uninformative \( \langle X \text{ rdf:type rdfs:Resource} \rangle \) quads. Deduplication takes quads with the same subject, predicate and object (possibly produced by previous steps) and merges them in a single quad inside a graph linked to the involved original sources (e.g., quad from Freebase and DBpedia).

Table 7.3 reports the execution times, throughputs and input and output sizes of each step, covering both the cases where steps are performed separately via intermediate files and multiple invocations of RDFpro (upper part of the table), or aggregated per processing phase using composition capabilities (lower part). Additionally, RDFpro reported the use of \(~2\) GB of memory for smushing an owl:sameAs graph of \(~38\)M URIs and \(~8\)M equivalence classes (\(~56\) bytes/URI on average).

\textbf{Comment} Also in this scenario, the aggregation of multiple processing tasks leads to a marked reduction of the total processing time (33\% reduction from 47803 s to 31981 s) due to the elimination of the I/O overhead for reading/writing intermediate files.

While addressed separately, the three scenarios of dataset analysis, filtering and merging are often combined in practice, e.g., to remove unwanted ABox and TBox quads from input data, merge remaining quads, and analyze the result producing statistics that describe and document it; an example of such combination is reported on RDFpro website.

\subsection*{7.2.4 Dataset Massaging}

We categorize three relevant, broad classes of dataset massaging tasks that are supported by RDFpro processing model: data repackaging, data sanitization and data derivation.

\textit{Data repackaging} comprises all the modifications that preserve data contents, i.e., the quads, and just affect the way data is packaged, i.e., the choices of RDF syntax, compression scheme and number of files. These modifications are often necessary to comply with restrictions of existing tools and systems, or to distribute data in a form that is best suited to the intended use (e.g., machine vs. human consumption). Data

\textsuperscript{20}DBpedia 3.9 data requires costly BZip2 decompression, while GeoNames RDF/XML+zip file format is slow to parse.
repackaging operations are all supported by RDFpro and are best performed in a streaming model, which thus represent the most common choice for this task.

_Data sanitization_ consists in fixing or removing the RDF terms or quads that prevent further processing of data. An example consists in the conversion of literals of a datatype to literals of an alternative datatype, because the former is not (properly) supported by the target system.\(^{21}\) Other tasks supported in a streaming model include the rewriting of URIs (e.g., to change namespace), the normalization of literals (e.g., to ensure that `rdfs:label` strings obey certain capitalization patterns) and the removal of quads whose literal object has an excessive length.\(^{22}\) These and similar tasks are supported by the RDFpro `@transform` processor, and further filtering capabilities are provided by the `@mapreduce` processor.

_Data derivation_ consists in augmenting a dataset with quads computed from original data. Two broad classes of derivations supported in a streaming model are (i) quad-level derivations, and (ii) aggregations of quad-level information with emission of aggregate results at the end. Examples of the first kind include the conversion of a numeric value from a unit of measurement to another, as done in DBpedia “Mapping-based Properties (Specific)” dataset, or the computation of the age of persons starting from their birth dates. Examples of the second kind include counting the occurrences of a certain property for an instance (e.g., the number of person he/she `foaf:knows`). All these derivations are supported by `@transform`, while more complex derivations (e.g., involving joins) may in principle be implemented with `@mapreduce` or new processors exploiting the sorting primitive.

While we do not conduct experiments here, we note that the tasks described can be all implemented in a single pass without sorting. Assuming similar input and output sizes, performances roughly amounts to the ones of reading and writing data in a pass (\(\sim 0.4 – 0.5\) MQ/s on our test machine).

### 7.2.5 Discussion

The experimental results and the applicability of RDFpro in relevant scenarios allow answering our research question and also provide a number of interesting findings on the practical usage of the proposed approach.

**Research question assessment**  Two results emerge from the experiments: (i) RDFpro implementation of the processing tasks here considered succeeds in managing billions of RDF quads on a commodity machine; and (ii) execution times are in the order of hours (1h 16’ for filtering 2.86 BQ, 5h 56’ for analyzing 5.49 BQ, 8h 53’ for merging 3.39 BQ).

The first result, which is trivial for tasks inherently expressible at a quad-level such as TBox extraction and some kinds of filtering, is not obvious for other tasks such as RDFS

\(^{21}\)E.g., the Community Edition of Virtuoso (Ver. 7.1) normalizes `xsd:gDay`, `xsd:gMonth` and `xsd:gYear` values to `xsd:datetime` altering their semantics; changing to `xsd:int` is a partial fix.

\(^{22}\)E.g., the OWLIM Lite 5.4.6486 triplestore cannot store very long literals (e.g., 20M chars of GML geographic data).
inference, smushing, statistics extraction, deduplication and set operations, for which we provide specialized implementations based on a mix of streaming and sorting techniques.

The second result can be put into perspective by comparing it with the time needed to load data in a triplestore. On Virtuoso\textsuperscript{23} (Ver. 7), a state-of-the-art triplestore, the load time for one billion of quads is 9h08’ on our test machine and 27’ on the much more powerful machine used in the latest Berlin SPARQL Benchmark (BSBM) experiment.\textsuperscript{24} Assuming that these rates hold for larger amounts of data, the comparison between these times and our processing times leads to two conclusions. First, given the same hardware, any one-time processing based on the use of a triplestore—a common solution in RDF processing—is not competitive with our approach, as just the loading of input data in the triplestore would take longer than our processing time in the considered scenarios.\textsuperscript{25} Second, our processing times are negligible if compared to load times on the same machine, and have the same order of magnitude of load times in the BSBM machine, overall meaning that RDF processing based on RDFpro approach would not slow down (and is thus compatible with) a typical Extract, Transform, Load (ETL)\textsuperscript{26} procedure where resulting RDF data is put in a production triplestore.

Based on these results, a positive answer can be given to the research question \textit{“Are relevant RDF processing tasks practically feasible on large datasets by using streaming and sorting techniques on a single commodity machine?”}.

**Other findings** The empirical evaluation also highlights the importance of task aggregation and allows us to analyze the factors impacting streaming and sorting performances.

Aggregation of multiple processing tasks in a single RDFpro invocation provides better performances as input data is parsed once and I/O costs for reading/writing intermediate files are eliminated, as shown in Section 7.2.1 and 7.2.3. Task aggregation requires composition primitives and the support for reading and writing data at any point of the pipeline, two features of RDFpro that are relevant to any similar tool.

Streaming performances within a pipeline are highly dependent on the file compression and RDF format used. No compression and best compression (e.g., bzip2 used in DBpedia dumps) are inefficient, with gzip representing a good trade-off; using native compression utilities (and especially their parallel versions, e.g., pigz and pbzip2) is also beneficial. Line-oriented RDF formats such as NTriples and NQuads provide better performances as they allow multi-threaded parsing and serialization (e.g., from 0.61 MQ/s to 1.45 MQ/s for Freebase NTriples+gzip data with multiple threads). We also experimented with the HDT binary format (Fernández et al., 2013), but writing HDT is very expensive while

\textsuperscript{23}http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/
\textsuperscript{24}Two Intel Xeon E5-2650 CPU (16 physical cores total, 32 logical), 256 GB RAM, 3x 1.8 TB 7200 RPM disks in RAID 0, Linux 3.3.4; see http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V7/index.html
\textsuperscript{25}Of course, using a triplestore may pay off in scenarios where data is loaded once and processed many times, or when the triplestore is also the final destination of data.
\textsuperscript{26}http://en.wikipedia.org/wiki/Extract,_transform,_load
reading HDT is not faster than reading other formats (unless lookup of RDF terms in the HDT dictionary is skipped).

Sorting performances depend on a number of factors. In our experience, performances can be improved by allocating a large amount of memory for sorting (we gave 8 GB out of the available 16 GB to the sort utility in our experiments), by using a parallel sort implementation, and by configuring the compression of temporary files. Dictionary encoding of frequent RDF terms also helps improving throughput.

7.3 Related Work

Most RDF data is available as LOD data. An in-depth account of the LOD initiative, including a reference architecture for LOD data consumption, was presented by Heath and Bizer (2011), while Heitmann et al. (2013) reported the findings of a survey on 124 LOD applications submitted to ISWC and ESWC challenges from 2003 to 2009. Our selection of RDF processing tasks, usage scenario and evaluation criteria for processing times are justifiable in the light of these works. Concerning the processing tasks, data filtering, transformation and smushing are at the core of the vocabulary mapping, identity resolution and quality evaluation modules in the reference architecture. Inference—optional task in the architecture—is relevant to 58% of the applications surveyed by Heitmann et al. (2013), while the extraction of VOID statistics is relevant due to VOID being acknowledged as the standard tool to describe LOD datasets (Heath and Bizer, 2011). Concerning the usage scenarios, we already noted that dataset analysis, filtering and merging are part of a larger integration workflow that is the ultimate purpose of the reference architecture and is a required activity in 80% of the surveyed application (69% of which requiring manual analysis of source data). Finally, our comparison of processing times with triplestore loading times is grounded in the use of triplestores in 88% of the surveyed applications.

All the processing tasks we considered can be implemented on a single machine using some kind of data index—typically a triplestore but also an RDF file indexed in the HDT format (Fernández et al., 2013)—and then exploiting the index querying and manipulation primitives—typically SPARQL. Data filtering and transformation are straightforward in this setting and smushing is not complex to implement. Statistics extraction can use queries, as done using SPARQL in make-void to compute VOID statistics, and in RDFStats (Langegger and Wöß, 2009) to compute instance counts and class and property histograms. Finally, inference is possible using forward-chaining algorithms, as done in the OWLIM triplestore (Bishop et al., 2011) for RDFS and various OWL fragments, in RDFox (Motik et al., 2014) for general Datalog-like rules, and in the reasoning approach by Peters et al. (2014) based on an in-memory index and parallel GPUs. While supporting flexible processing and many RDF processing tools, the data index must reside entirely in memory for these tools to operate efficiently (if at all) with performances decreasing quickly if disk access is involved, both for creating the index and accessing it during processing. In

27http://github.com/cygri/make-void
comparison, a streaming approach also hits the disk, but data access is sequential and faster, leading to situations where total processing time with streaming is faster than index creation alone, as reported in Section 7.2.5.

Scalability with respect to data size is generally achieved using distribution. In particular, the MapReduce paradigm and its distributed Hadoop implementation have been used for forward-chaining RDFS and OWL Horst inference in WebPIE (Urbani et al., 2012), for VOID statistics extraction in voidGen (Böhm et al., 2011) and for general RDF processing (with special focus on Freebase data) in Infovore. While MapReduce as a paradigm can be implemented on a single machine (as we did with processor @mapreduce using streaming and sorting), providing vertical scalability if property optimized (Appuswamy et al., 2013), the natural execution environment of an implementation targeting horizontal scalability such as Hadoop is a cluster of many machines, where coordination costs and the overhead associated to distribution are negligible.

A hybrid tool using either a local triplestore or a Hadoop cluster is the Linked Data Integration Framework (LDIF) by Schultz et al. (2012). LDIF implements the architecture of Heath and Bizer (2011) and, in particular, data filtering based on simple rules, data transformation for vocabulary mapping with R2R rules (Bizer and Schultz, 2010) and smushing and entity resolution with Silk (Volz et al., 2009). LDIF partitions data in per-instance graphs processed by distinct threads and machines. This approach scales to hundreds of millions of triples on single machines, and to billions of triples on Hadoop clusters. Its downsides are that computations spanning multiple instances may be infeasible and data partitioning is not for free (especially with a triplestore), which could be the limiting factor to LDIF scalability on single machines.

The streaming computation model has been studied for a long time. O’Connell (2009) provided a good survey of streaming algorithms on graph data such as RDF. Here, different extensions to the base streaming model are analyzed and the one with a sorting primitive, formalized by Aggarwal et al. (2004), results the most expressive. Sorting is an highly-efficient operation on today machines and allows performing set operations and grouping together information scattered in the stream that must be processed together (e.g., quads of an instance), in this way overcoming many of the limitations of a pure streaming model.

Our use of streaming for RDF processing differentiates from the Stream Reasoning paradigm (Della Valle et al., 2009), as we focus on processing large but finite amounts of data using a streaming computation model, whereas a Stream Reasoning engine deals with the processing of possibly time-windowed and infinite streams of temporally-tagged triples; nevertheless, such an engine can be included in our framework as a specialized processor. Tools using a streaming model for RDF processing like us are often limited to syntax conversion, e.g., rapper\(^{29}\), rdfpipe\(^{30}\), Sesame RDFConvert,\(^{31}\) and DotNetRDF

\(^{28}\)http://github.com/paulhoule/infovore
\(^{29}\)http://librdf.org/raptor/rapper.html
\(^{31}\)http://sourceforge.net/projects/rdfconvert
The few exceptions include Jena riot, LODStats and SliceSPARQL. Jena riot\(^{33}\) supports `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf` and `rdfs:subPropertyOf` inference after preliminary indexing of TBox data. This approach is similar to ours, although we cover complete RDFS inference with very loose ABox restrictions. LODStats (Auer et al., 2012) extracts 32 statistics (a superset of VOID) in a single streaming pass, but relies heavily on in-memory data structures whose content is dropped when full, leading to approximate statistics computation. In comparison, our implementation employs sorting and lightweight data structures (size linear in number of classes and properties, rather than instances). SliceSPARQL (Marx et al., 2013) is a SPARQL fragment for data filtering. It includes UNION, FILTER and statement patterns with one join variable, and can be evaluated in (max) three streaming passes with an auxiliary disk index for join candidates. Our @transform processor provides a primitive that can be used for the same filtering, but passes and join candidates must be explicitly managed by users.

We conclude noting that our sequential and parallel composition mechanisms are in line with the workflow primitives for RDF processing studied in the literature, especially in the scope of semantic mash-ups. Here, Semantic Web Pipes by Le-Phuoc et al. (2009) allow composing processing operators in a DAG similarly to us, although their focus is on easy visual composition rather than data streaming and large-scale processing.

### 7.4 Summary

We presented RDFpro, an extensible, general-purpose, open source (public domain) tool for non-distributed RDF processing leveraging streaming and sorting techniques. RDFpro provides out-of-the-box implementations—called processors—of common tasks such as data filtering, rule-based inference, smushing, and statistics extraction, as well as easy ways to add new processors and arbitrarily compose processors in complex pipelines.

With RDFpro, we showed that many RDF processing tasks common in practice can be performed on billions of triples on a single commodity machine using streaming and sorting, with execution times compatible with batch RDF processing and better than the times of approaches using a local data index (e.g., a triplestore). Of course, there are still tasks not supported in RDFpro, such as SPARQL query answering and SPARQL-based data massaging. These tasks involve joining of data and are thus hard to implement and possibly inefficient in a streaming and sorting setting, although specialized algorithms might be devised to implement restricted versions of them.

We consider RDFpro a practically useful tool representing a viable compromise between local RDF processing based on memory-intensive, non-scalable data indexing (e.g., in a triplestore) and distributed RDF processing based on heavyweight infrastructures (e.g., Hadoop). RDFpro processors and composition facilities allow addressing a variety of processing needs with a single tool that can be used by casual users and not just by


\(^{33}\)http://jena.apache.org/documentation/io/
developers, making it a sort of “Swiss army knife” for exploring and manipulating RDF datasets. At the same time, RDFpro can be extended by developers with new processors by just focusing on the specific task at hand, as efficient streaming, sorting, I/O, thread management and composition facilities are already provided by RDFpro runtime.

RDFpro is used in PreMOn (Section 3.3), PIKES (Section 5.4), and the KnowledgeStore (Section 6.2.4), and it was exploited in the NewsReader project to post-process RDF knowledge extracted from text (Section 9.2) and to build background knowledge datasets out of multilingual LOD data sources (Section 9.3).
Chapter 8

KE4IR: Using PIKES for Semantic Information Retrieval

Information Retrieval (IR) is the task of determining, for a user query, the relevant documents in a text collection, ranking them based on their relevance degree for the query.

Inspired by the work of Waitelonis et al. (2015), in this chapter we investigate whether the application of frame-based ontology population systems like PIKES (Chapter 5) to perform the semantic analysis of documents and queries, and the leveraging of background knowledge from Linked Open Data (LOD) resources such as DBpedia (Lehmann et al., 2015) and YAGO (Hoffart et al., 2013), can help overcoming known limitations of traditional term-based IR approaches based on text information only. To explain, let us consider the following query example: “astronomers influenced by Gauss.” Traditional IR approaches match the terms or possible term-based expansions (e.g., synonyms, related terms) of the query and the documents, but relevant documents may not necessarily contain all the query terms. For instance, the term “influenced” or “astronomers” may not be used at all in a relevant document. Similarly, some relevant documents may be ranked lower than other ones containing all the three terms, but in an unrelated way, like documents about some astronomers born centuries before Gauss and influenced by Leonardo Da Vinci.

In our approach, named KE4IR (Knowledge Extraction for Information Retrieval, read kee-fer), both queries and documents are processed to extract semantic content pertaining to the following semantic layers (Section 8.1): (i) entities, e.g., dbpedia:Carl-Friedrich_Gauss from “Gauss” in the previous query example; (ii) types of entities, either explicitly mentioned, such as yago:Astronomer109818343 from “astronomers,” or indirectly obtained from external resources for mentioned entities, such as yago:GermanMathematicians from “Gauss”; (iii) semantic frames and frame roles, such as framebase:Subjective_influence from “influenced”; and, (iv) temporal information, either explicitly mentioned in the text or indirectly obtained from external resources for mentioned entities, e.g., via DBpedia properties such as dbo:dateOfBirth (1777) and dbo:dateOfDeath (1855) for entity dbpedia:Carl_Friedrich_Gauss. We then match queries and documents considering both their textual and semantic content, according to a simple retrieval model extending VSM (Vector Space Model, Salton et al., 1975). This way, we can match documents mentioning someone who is an astronomer (i.e., entities having type yago:Astronomer109818343) even if “astronomers” or one of its term-based variants is not explicitly written in the document text. Similarly, we can exploit the entities and the temporal content to better

1http://pikes.fbk.eu/ke4ir.html
weigh the different relevance of documents mentioning dbpedia:Carl_Friedrich_Gauss and dbpedia:GAUSS_(software), as well as to differently rank documents about Middle Age and 17th/18th centuries astronomers.

We built an evaluation infrastructure (Section 8.2) implementing KE4IR on top of PIKES, RDFpro (Chapter 7), and Lucene\(^2\) that allows applying KE4IR to index documents and evaluate arbitrary IR queries against gold relevance judgments, computing a number of IR metrics. Using this infrastructure, we performed a first assessment of KE4IR (Section 8.3) on a recently released dataset by Waitelonis et al. (2015). Our experiments show that MAP (Mean Average Precision) increases of 3.5 percentage points when combining textual and semantic analysis, thus suggesting that the use of a frame-based ontology population tool like PIKES to extract a large variety of semantic content from texts, richer than what previously considered by state-of-the-art approaches (Section 8.4), can effectively improve the performances of IR systems.

Acknowledgments The material presented in this chapter is the result of a collaborative work published in (Corcoglioniti et al., 2016a).

8.1 Approach

Standard IR systems look at documents and queries as bags of terms (e.g., stemmed tokens). In KE4IR we augment textual terms with additional terms coming from different semantic annotation layers (Section 8.1.1), produced combining ontology population techniques (specifically, PIKES), based on Natural Language Processing (NLP), with LOD background knowledge, and then propose a simple retrieval model that uses this additional semantic information to find and rank the documents matching a query (Section 8.1.2).

8.1.1 Semantic Layers

We consider four semantic layers—URI, TYPE, FRAME, TIME—that complement the base textual layer with semantic terms. These terms can be obtained using frame-based ontology population techniques that identify mentions (i.e., snippets of text) denoting entities, events and relations. From each mention, a set of semantic terms is extracted, and the number of mentions a term derives from can be used to quantify its relevance for a document. Table 8.1 (first three columns) reports an example of the terms and the associated mentions that can be extracted from the example query previously considered: “astronomers influenced by Gauss.”

URI layer The semantic terms of this layer are the URIs of entities mentioned in the text, disambiguated against external knowledge resources such as DBpedia; other

\(^2\)http://lucene.apache.org/
specialized knowledge bases, such as SNOMED CT,\(^3\) can be used depending on the domain of the document collection. Disambiguated URIs are the result of two NLP tasks:\(^4\) NERC (Named Entity Recognition and Classification), which identifies proper names of certain entity classes in a text, such as persons, organizations, and locations; and EL (Entity Linking), which disambiguates those names against the individuals of a knowledge base. The Coreference Resolution NLP task can be also exploited to ‘propagate’ the URI associated to a disambiguated named entity to its coreferring mentions in the text, so to better count entity mentions.

**TYPE layer** This layer contains as terms the URIs of the ontological types (and subtypes) associated to noun phrases in the text. For disambiguated named entities, resulting from NERC and EL, associated types can be retrieved from external background knowledge resources describing those entities, such as DBpedia. For common nouns, disambiguation against WordNet (Fellbaum, 1998) via the WSD (Word Sense Disambiguation) NLP task provides synsets that can be mapped to types of known ontologies leveraging existing mappings. Given these two extraction techniques, an ontology particularly suited to this layer is the YAGO taxonomy, due both to its WordNet origins and the availability of YAGO types for DBpedia entities.

**TIME layer** The terms of this layer are temporal values mentioned in the text, either because explicitly expressed in a time expression, such as “the eighteenth century”, recognized via the TERN (Temporal Expression Recognition and Normalization) NLP task, or because associated via some properties to a disambiguated entity in the background knowledge, such as the birth date associated to dbpedia:Carl_Friedrich_Gauss. We propose to represent time at different granularities—day, month, year, decade, and century—in order to support both precise and fuzzy temporal matching. Therefore, each mentioned date/time value is mapped to (max) five `time` terms, one for each granularity level (e.g., 2015-12-18 is mapped to \texttt{day:2015-12-18}, \texttt{month:2015-12}, \texttt{year:2015}, \texttt{decade:201}, \texttt{century:20}).

**FRAME layer** A semantic frame is a star-shaped structure that represents an event, n-ary relation, or other structured entity, having a certain frame type (e.g., \texttt{framebase:frame-Subjective\_influence}, from “influenced”) and zero or more participants (e.g., \texttt{dbpedia:Carl-Friedrich_Gauss}) that play a specific semantic role in the context of the frame. As done in PIKES, semantic frames can be extracted using tools for the SRL (Semantic Role Labeling) NLP task based on certain predicate models, such as FrameNet (Baker et al., 1998), and then mapped to an ontological representation using an RDF/OWL frame-based ontology aligned to the predicate model, such as FrameBase (Rouces et al., 2015), whose mappings

\(^3\)\texttt{http://b2i.sg/}

\(^4\)We mention in this section the main NLP tasks involved in the extraction of semantic terms. Some of them typically build on additional NLP analyses, such as Tokenization, POS-Tagging, Dependency Parsing (DP) and Constituency Parsing.
Table 8.1. Terms extracted by KE4IR from query “astronomers influenced by Gauss,” with mentions $m_1 = \text{“astronomers”}, m_2 = \text{“influenced”}, m_3 = \text{“Gauss”};$ the textual layer is weighted 0.5; the four semantic layers are weighted 0.125 each.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Term</th>
<th>$M(t_i, q)$</th>
<th>$t_f(q(t_i, q))$</th>
<th>$idf(t_i, q)$</th>
<th>$w(l)$</th>
<th>$q_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXTUAL</td>
<td>astronomer</td>
<td>$m_1$</td>
<td>1.0</td>
<td>2.018</td>
<td>0.5</td>
<td>1.009</td>
</tr>
<tr>
<td>TEXTUAL</td>
<td>influence</td>
<td>$m_2$</td>
<td>1.0</td>
<td>3.404</td>
<td>0.5</td>
<td>1.702</td>
</tr>
<tr>
<td>TEXTUAL</td>
<td>gauss</td>
<td>$m_3$</td>
<td>1.0</td>
<td>1.568</td>
<td>0.5</td>
<td>0.784</td>
</tr>
<tr>
<td>URI</td>
<td>dbpedia:Carl_Friedrich_Gauss</td>
<td>$m_3$</td>
<td>1.0</td>
<td>3.404</td>
<td>0.125</td>
<td>0.426</td>
</tr>
<tr>
<td>TYPE</td>
<td>yago:GermanMathematicians</td>
<td>$m_3$</td>
<td>0.030</td>
<td>2.624</td>
<td>0.125</td>
<td>0.010</td>
</tr>
<tr>
<td>TYPE</td>
<td>yago:NumberTheorists</td>
<td>$m_3$</td>
<td>0.030</td>
<td>2.583</td>
<td>0.125</td>
<td>0.010</td>
</tr>
<tr>
<td>TYPE</td>
<td>yago:FellowsOfTheRoyalSociety</td>
<td>$m_3$</td>
<td>0.030</td>
<td>1.057</td>
<td>0.125</td>
<td>0.004</td>
</tr>
<tr>
<td>TYPE</td>
<td>... other 18 terms ...</td>
<td>$m_3$</td>
<td>0.030</td>
<td>...</td>
<td>0.125</td>
<td>...</td>
</tr>
<tr>
<td>TYPE</td>
<td>yago:Astronomer109818343</td>
<td>$m_1, m_3$</td>
<td>0.114</td>
<td>1.432</td>
<td>0.125</td>
<td>0.020</td>
</tr>
<tr>
<td>TYPE</td>
<td>yago:Physicist110428004</td>
<td>$m_1, m_3$</td>
<td>0.114</td>
<td>0.958</td>
<td>0.125</td>
<td>0.014</td>
</tr>
<tr>
<td>TYPE</td>
<td>yago:Person100007846</td>
<td>$m_1, m_3$</td>
<td>0.114</td>
<td>0.003</td>
<td>0.125</td>
<td>~0</td>
</tr>
<tr>
<td>TYPE</td>
<td>... other 9 terms ...</td>
<td>$m_1, m_3$</td>
<td>0.114</td>
<td>...</td>
<td>0.125</td>
<td>...</td>
</tr>
<tr>
<td>FRAME</td>
<td>(Subjective_influence-influence.v, Carl...Gauss)</td>
<td>$m_2$</td>
<td>0.333</td>
<td>5.802</td>
<td>0.125</td>
<td>0.242</td>
</tr>
<tr>
<td>FRAME</td>
<td>(Subjective_influence, Carl_Friedrich_Gauss)</td>
<td>$m_2$</td>
<td>0.333</td>
<td>5.802</td>
<td>0.125</td>
<td>0.242</td>
</tr>
<tr>
<td>FRAME</td>
<td>(Frame, Carl_Friedrich_Gauss)</td>
<td>$m_2$</td>
<td>0.333</td>
<td>3.499</td>
<td>0.125</td>
<td>0.146</td>
</tr>
<tr>
<td>TIME</td>
<td>day:1777-04-30</td>
<td>$m_3$</td>
<td>0.1</td>
<td>3.404</td>
<td>0.125</td>
<td>0.043</td>
</tr>
<tr>
<td>TIME</td>
<td>day:1855-02-23</td>
<td>$m_3$</td>
<td>0.1</td>
<td>3.404</td>
<td>0.125</td>
<td>0.043</td>
</tr>
<tr>
<td>TIME</td>
<td>century:1700</td>
<td>$m_3$</td>
<td>0.1</td>
<td>0.196</td>
<td>0.125</td>
<td>0.002</td>
</tr>
<tr>
<td>TIME</td>
<td>... other 7 terms</td>
<td>$m_3$</td>
<td>0.1</td>
<td>...</td>
<td>0.125</td>
<td>...</td>
</tr>
</tbody>
</table>

to predicate models are provided by PreMOn (Chapter 3). Semantic frames provide relational information that helps matching queries and documents more precisely. Different approaches can be used to transform a star-shaped semantic frame into a set of terms of the FRAME layer. In this work, we propose to map each (frame type, participant) pair whose participant is a disambiguated entity (e.g., the pair ⟨framebase:frame-Subjective_influence, dbpedia:Carl_Friedrich_Gauss⟩) to a term, including also the terms obtainable by replacing the frame type URI with the URIs of its super-classes in the ontology. We investigated also using non-disambiguated participant entities, obtaining however worse results.

8.1.2 Retrieval Model

We adopt a retrieval model inspired to VSM. Given a document collection $D$, we represent each document $d \in D$ (resp. query $q$) with a vector $\vec{d} = (d_1 \ldots d_n)$ ($\vec{q} = (q_1 \ldots q_n)$) where each element $d_i (q_i)$ is the weight corresponding to a term $t_i$ and $n$ is the number of distinct terms in collection $D$. Differently from text-only approaches, terms of our model come from multiple layers (da Costa Pereira et al., 2012), both textual and semantic, and each document (query) vector can be thought as the concatenation of smaller, layer-specific vectors. Given a term $t$, we denote its layer with $l(t) \in L = \{\text{TEXTUAL}, \text{URI}, \text{TYPE}, \text{FRAME}, \text{TIME}\}$.

To compute the similarity between a document $d \in D$ and a query $q$, we use a similarity function $sim(d, q)$. The documents matching $q$ are the ones with $sim(d, q) > 0$, and they
are ranked according to decreasing similarity values. To derive $sim(d, q)$ we start from the cosine similarity of VSM:

\[
sim_{\text{vsm}}(d, q) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{n} d_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} d_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2}}
\]

and we remove the normalization by $|\vec{d}|$ and $|\vec{q}|$, thus obtaining:

\[
sim(d, q) = d \cdot q = \sum_{i=1}^{n} d_i \cdot q_i
\]

Normalizing by $|\vec{q}|$ does not affect the ranking and only serves to compare scores of different queries, thus we drop it for simplicity. Normalizing by $|\vec{d}|$ has the effect of making the similarity score obtained by matching $m$ terms in a small document higher than the score obtained by matching the same $m$ terms in a longer document. This normalization is known to be problematic in some document collections, is implemented differently and optionally disabled in real systems (e.g., Lucene and derivatives), and we deem it inappropriate in our scenario, where the document vector is expanded with a large amount of semantic terms whose number is generally not proportional to the document length.

We assign the weights of document and query vectors starting from the usual product of Term Frequency (tf) and Inverse Document Frequency (idf):

\[
d_i = tf_d(t_i, d) \cdot \text{idf}(t_i, D)
\]

\[
q_i = tf_q(t_i, q) \cdot \text{idf}(t_i, D) \cdot w(l(t_i))
\]

The values of tf are computed differently for documents ($tf_d$) and queries ($tf_q$), while weights $w(l(t_i))$ quantify the importance of each layer. Given the form of Eq. 8.2, it suffices to apply $w(l(t_i))$ only to one of $\vec{d}$ and $\vec{q}$: we chose $\vec{q}$ so to allow selecting weights on a per-query basis. Table 8.1 reports the $tf_q$, idf, w, and $q_i$ values for the terms of the example query “astronomers influenced by Gauss.”

Several schemes for computing tf and idf have been proposed in the literature. Among them, we adopt the following scheme,\(^5\) where $f(t, x)$ and $f'(t, x)$ are two measures of the frequency of a term $t$ in a text (document or query) $x$:

\[
tf_d(t, d) = 1 + \log(f(t, d))
\]

\[
tf_q(t, q) = f'(t, q)
\]

\[
\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D | f(t, d) > 0\}|}
\]

\(^5\)Given the lack of normalization in $sim(d, q)$, our scheme can be roughly classified as ltn.ntn using the SMART notation; see http://bit.ly/weighting_schemes (Manning et al., 2008).
The raw frequency $f(t, x)$ is defined as usual as the number of occurrences of term $t$ in $x$. To account also for semantic terms, we denote with $M(t, x)$ the set of mentions in text $x$ from where term $t$ has been extracted, valid also for textual terms whose mentions are simply their occurrences in the text, and let $f(t, x) = |M(t, x)|$. The normalized frequency $f'(t, x)$ is newly introduced to account for the fact that multiple terms of a semantic layer can be extracted from a single mention, differently from the textual case. It is defined as:

$$f'(t, x) = \sum_{m \in M(t, x)} \frac{1}{|T(m, l(t))|}$$

where $T(m, l)$ is the set of terms of layer $l$ extracted from mention $m$. As $|T(m, \text{TEXTUAL})|$ is always 1, $f(t, x) = f'(t, x)$ for TEXTUAL terms. Note that Eq. 8.7 can indifferently use $f(t, x)$ or $f'(t, x)$.

The formulation of $f'(t, x)$ and its use in Eq. 8.6 aim at giving each mention the same importance when matching the query against the document collection. To explain, let’s consider a query containing two mentions $m_1$ and $m_2$, with respectively $n_1$ and $n_2$ disjoint terms of a certain semantic layer (e.g., TYPE) extracted from each mention, $n_1 > n_2$; also assume that these terms have equal idf and tf$_d$ values in the document collection. If we give these terms equal tf$_q$ values, then a document matching the $n_1$ terms of $m_1$ (and nothing else) will be scored and ranked higher than a document matching the $n_2$ terms of $m_2$ (and nothing else). However, the fact that $n_1 > n_2$ does not reflect a preference of $m_1$ by the user; rather, it may merely reflect the fact that $m_1$ is described more richly than $m_2$ in the background knowledge. Our definition of normalized frequency corrects for this bias by assigning each mention a total weight of 1, which is equally distributed among the terms extracted from it for each semantic layer (e.g., weight $1/n_1$ assigned to terms of $m_1$, $1/n_2$ to terms of $m_2$).

For similar reasons, the use of $f'(t, x)$ in place of $f(t, x)$ in Eq. 8.5 would be inappropriate. Consider a query whose vector has a single TYPE term $t$ (similar considerations apply to other semantic layers). Everything else being equal (e.g., idf values), two documents mentioning two entities of type $t$ the same number of times should receive the same score. While this happen with $f(t, x)$, using $f'(t, x)$ the document mentioning the entity with fewest TYPE terms (beyond $t$) will be scored higher, although this clearly does not reflect a user preference.

### 8.2 Implementation

We built an evaluation infrastructure that implements the KE4IR approach of Section 8.1 and allows applying it on arbitrary documents and queries, measuring retrieval performances against gold relevance judgments. All the code is available on KE4IR website.

Figure 8.1a shows the pipeline used to extract terms (with raw frequencies) from documents and queries, combining both textual and semantic analysis of input texts.
Textual analysis generates the TEXTUAL layer by employing standard components for text tokenization, stop word filtering, and stemming from Lucene. Semantic analysis makes use of a frame-based Ontology Population (OP) tool for transforming the input text into an RDF knowledge graph where each instance is grounded to one or more mentions. This graph is then enriched with triples about selected URIs (DBpedia entities, YAGO types) retrieved from a key-value store populated with the required LOD background knowledge; the enrichment is done recursively and RDFS reasoning is done at the end to materialize inferences. The enriched graph is finally queried to extract semantic terms.

As frame-based ontology population tool we use PIKES, which integrates a number of state-of-the-art tools in its two processing phases. The linguistic feature extraction phase, extracting a graph of mentions from a text, uses Stanford CoreNLP (Manning et al., 2014) for Tokenization, POS-Tagging, Lemmatization, NERC, TERN, parsing and Coreference Resolution; UKB (Agirre et al., 2014) for WSD; DBpedia Spotlight (Mendes et al., 2011) for EL; Mate-tools (Björkelund et al., 2009) and Semafor (Das et al., 2014) for SRL. The knowledge distillation phase, mapping the mention graph into an RDF knowledge graph, leverages RDFpro for mapping rule evaluation. RDFpro is also used for RDFS reasoning.

Frame-based ontology population as implemented in PIKES is computationally more expensive than standard textual analysis. Using PIKES on a server with 24 cores (12 physical) and 192 GB RAM, we obtained a throughput of ~700K tokens/h (~30K tokens/h core), corresponding to ~1200 documents/h for the document collection of Section 8.3 (570 tokens/document). While inappropriate for a Web-scale deployment, this throughput is however adequate for small to medium-sized document collections (e.g., as encountered in corporate environments). Furthermore, larger collections can also be processed, with some loss in retrieval performances, by disabling the extraction of some layers. To give an idea, the impact of each semantic layer on the whole processing time for the document collection of Section 8.3 is: URI (3.5%), TYPE (16.3%), TIME (2.9%), FRAME (77.3%). Note also that

---

6Background knowledge consists of the subset of FrameBase ontology used in PIKES, and the mapping-based properties with xsd:date, xsd:date-Time, xsd:gYear, and xsd:gYearMonth objects, YAGO types and type hierarchy from DBpedia 2015-04; all data is available on KE4IR website.
substantial improvements of KE4IR indexing throughput can be achieved with further engineering and optimization that we did not consider in this investigation.

Figure 8.1b shows the workflow implemented for indexing extracted terms, executing queries and computing evaluation metrics. Document terms are directly indexed in Lucene together with their raw frequencies. At search time, query terms are OR-ed in an index query that locates the documents containing at least one term, which guarantees that \( \text{sim}(d,q) > 0 \). Matched documents are scored and ranked externally to Lucene (for ease of testing) according to the KE4IR retrieval model of Section 8.1.2, starting from the term vectors of the query and the matched documents, and computing the necessary \( \text{tf}_d \), \( \text{tf}_q \), and \( \text{idf} \) values based on raw and normalized term frequencies and some statistics produced by Lucene (number of documents and document frequencies). The resulting ranking is compared with the gold relevance judgments to compute a comprehensive set of evaluation metrics, which are averaged along queries.

8.3 Evaluation

In this section, we present an evaluation of KE4IR and discuss some insights emerged from it. All the evaluation materials are available on KE4IR website.

8.3.1 Evaluation Setup

KE4IR has been validated on the ad-hoc IR task, consisting in performing a set of queries over a document collection for which the list of relevance judgments is available. For the presented evaluation, we adopted the document collection created by Waitelonis et al. (2015), composed by a set of 331 documents and 35 queries. The relevance of each document is expressed in a multi-value scale with scores going from 5 (the document contains exact information with respect to what the user is looking for) to 1 (the document is of no interest for the query). The peculiarity of this collection is the underlying semantic purpose with which it has been built. Indeed, the set of queries has been selected by varying from queries very close to keyword-base search (i.e., the query “Romanticism”) to queries requiring semantic capabilities for retrieving relevant documents (i.e., “Aviation pioneers’ publications”). In that work, the authors discuss some techniques exploiting manual annotations for semantic IR purposes. Unfortunately, their results and our results described next are not directly comparable, as the semantic techniques described by Waitelonis et al. (2015) are evaluated over annotations manually validated by experts, whereas we rely on totally automatic and thus inevitably noisy annotations.

Thus, we compared our approach against the two baselines introduced below:

- **Google baseline**: we exploited the Google custom search API for indexing pages containing our documents. The rationale behind this choice is to assess the performances of a commercial search engine, having as main challenge the “scalability” of indexing and retrieving documents, when a more custom document analysis is
required. Google can be considered as a black box, as it is not possible to customize the way it analyzes text and computes document scores with respect to queries;

- **Textual baseline**: we indexed the raw text by adopting the standard Lucene library customized with the scoring formula described in Section 8.2. In our experiments, this customization provides the same (actually, slightly better) performances of a standard Lucene configuration,\(^7\) and it also allows properly assessing the impact of semantic layers by excluding any interference related to slight differences in the definition of the scoring formula.

The protocol we used has been inspired by TREC (Voorhees and Harman, 1997); however, due to the small size of the collection, we had to carry out some changes. Instead of drawing the precision/recall curve, we computed the precision values after the first (Prec@1), fifth (Prec@5), and tenth (Prec@10) document, respectively. The rationale behind this decision is the fact that the majority of search result click activity (89.8%) happens on the first page of search results (Spink et al., 2006), which corresponds to a set of 10 to 20 documents. Then, we provided two further metrics: MAP and NDCG (Normalized Discounted Cumulated Gain, Järvelin and Kekäläinen, 2002), computed both on the entire rank and after the first ten documents retrieved (MAP@10 and NDCG@10). Validation on NDCG is necessary in scenarios where multi-value relevance is used.

### 8.3.2 Overall Evaluation Results

We report here an overview of the results obtained, using equal weights for textual and semantic information in KE4IR, i.e., \(w(\text{textual}) = w(\text{semantics}) = 0.5\), with \(w(\text{semantics})\) divided equally among semantic layers. We also provide a first analysis of KE4IR behavior using different layer combinations.

**Comparison with the baselines** Table 8.2 shows the comparison between the results achieved by KE4IR when exploiting all the semantic layers (in addition to the TEXTUAL layer), and the ones obtained by the proposed baselines.

It is possible to see that KE4IR matches or outperforms the baselines for all the considered metrics. With respect to the textual baseline, the higher improvements are registered on the MAP, MAP@10, and Prec@10 values that quantify the capability of the proposed approach of producing an effective documents ranking. While the gains on the MAP and MAP@10 metrics assess only the retrieval of relevant documents without considering their relevance scores, the improvements obtained on the NDCG and NDCG@10 metrics highlight that produced rankings are effective also from a quality point of view. The improvements over the textual baseline are statistically significant for MAP, MAP@10,

\(^7\)For comparison, on KE4IR website we make available for download an instance of SOLR (a popular search engine based on Lucene) indexing the same document collection used in our evaluation, and we report on its performances on the test queries.
Table 8.2. Comparison of KE4IR against the baselines (best/significant values in bold).

<table>
<thead>
<tr>
<th>Approach/System</th>
<th>Prec@1</th>
<th>Prec@5</th>
<th>Prec@10</th>
<th>NDCG</th>
<th>NDCG@10</th>
<th>MAP</th>
<th>MAP@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>0.543</td>
<td>0.411</td>
<td>0.343</td>
<td>0.434</td>
<td>0.405</td>
<td>0.255</td>
<td>0.219</td>
</tr>
<tr>
<td>Textual</td>
<td>0.943</td>
<td>0.669</td>
<td>0.453</td>
<td>0.832</td>
<td>0.782</td>
<td>0.733</td>
<td>0.681</td>
</tr>
<tr>
<td>KE4IR</td>
<td>0.971</td>
<td>0.680</td>
<td>0.474</td>
<td>0.854</td>
<td>0.806</td>
<td>0.758</td>
<td>0.713</td>
</tr>
</tbody>
</table>

KE4IR vs. Textual  
+3.03%  +1.71%  +4.55%  +2.64%  +2.99%  +3.50%  +4.74%

p-value (paired t-test)  
0.324  | 0.160  | 0.070   | 0.003  | 0.015  | 0.024  | 0.029

p-value (approx. random.)  
1.000  | 0.496  | 0.111   | 0.003  | 0.020  | 0.020  | 0.030

Table 8.3. KE4IR results using different layer combinations (best values in bold).

<table>
<thead>
<tr>
<th>Layers</th>
<th>Prec@1</th>
<th>Prec@5</th>
<th>Prec@10</th>
<th>NDCG</th>
<th>NDCG@10</th>
<th>MAP</th>
<th>MAP@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>textual,uri,type,frame,time</td>
<td>0.971</td>
<td>0.680</td>
<td>0.474</td>
<td>0.854</td>
<td>0.806</td>
<td>0.758</td>
<td>0.713</td>
</tr>
<tr>
<td>textual,uri,type,frame</td>
<td>0.971</td>
<td>0.680</td>
<td>0.474</td>
<td>0.853</td>
<td>0.804</td>
<td>0.757</td>
<td>0.712</td>
</tr>
<tr>
<td>textual,uri,type,time</td>
<td>0.971</td>
<td>0.680</td>
<td>0.474</td>
<td>0.851</td>
<td>0.802</td>
<td>0.757</td>
<td>0.712</td>
</tr>
<tr>
<td>textual,uri,type</td>
<td>0.971</td>
<td>0.680</td>
<td>0.474</td>
<td>0.849</td>
<td>0.801</td>
<td>0.755</td>
<td>0.710</td>
</tr>
<tr>
<td>textual,uri,frame,time</td>
<td>0.971</td>
<td>0.674</td>
<td>0.465</td>
<td>0.844</td>
<td>0.796</td>
<td>0.750</td>
<td>0.702</td>
</tr>
<tr>
<td>textual,uri,frame</td>
<td>0.971</td>
<td>0.674</td>
<td>0.465</td>
<td>0.842</td>
<td>0.795</td>
<td>0.749</td>
<td>0.702</td>
</tr>
<tr>
<td>textual,uri,time</td>
<td>0.971</td>
<td>0.674</td>
<td>0.465</td>
<td>0.840</td>
<td>0.791</td>
<td>0.747</td>
<td>0.700</td>
</tr>
<tr>
<td>textual,uri</td>
<td>0.971</td>
<td>0.674</td>
<td>0.465</td>
<td>0.837</td>
<td>0.791</td>
<td>0.747</td>
<td>0.700</td>
</tr>
<tr>
<td>textual,type,frame,time</td>
<td>0.943</td>
<td>0.674</td>
<td>0.471</td>
<td>0.848</td>
<td>0.799</td>
<td>0.745</td>
<td>0.700</td>
</tr>
<tr>
<td>textual,type,time</td>
<td>0.943</td>
<td>0.674</td>
<td>0.471</td>
<td>0.843</td>
<td>0.794</td>
<td>0.743</td>
<td>0.697</td>
</tr>
<tr>
<td>textual,type,frame</td>
<td>0.943</td>
<td>0.674</td>
<td>0.468</td>
<td>0.847</td>
<td>0.797</td>
<td>0.743</td>
<td>0.695</td>
</tr>
<tr>
<td>textual,frame,time</td>
<td>0.943</td>
<td>0.674</td>
<td>0.462</td>
<td>0.842</td>
<td>0.793</td>
<td>0.741</td>
<td>0.693</td>
</tr>
<tr>
<td>textual,type</td>
<td>0.943</td>
<td>0.674</td>
<td>0.468</td>
<td>0.842</td>
<td>0.792</td>
<td>0.740</td>
<td>0.693</td>
</tr>
<tr>
<td>textual,time</td>
<td>0.943</td>
<td>0.669</td>
<td>0.462</td>
<td>0.836</td>
<td>0.786</td>
<td>0.737</td>
<td>0.689</td>
</tr>
<tr>
<td>textual,frame</td>
<td>0.943</td>
<td>0.674</td>
<td>0.453</td>
<td>0.839</td>
<td>0.789</td>
<td>0.737</td>
<td>0.686</td>
</tr>
</tbody>
</table>

NDCG, and NDCG@10 (significance threshold 0.05), based on the p-values computed with the paired t-test (claimed as one of the best tests for IR by Sanderson and Zobel, 2005) and the approximate randomization test (Noreen, 1989). With respect to the Google baseline, the marked difference of performances derives from Google returning far less results than KE4IR for the evaluation queries. Indeed, large-scale (web-scale) IR systems such as Google are heavily tuned for precision, as any query usually matches a large number of documents and the problem is to discard the irrelevant ones. In our context, small-scale IR, systems as our tool deal with fewer documents and hence must be tuned also for recall.

**Impact of various layer combinations**  A detailed analysis of the results obtained using different layer combinations in KE4IR is shown in Table 8.3. Combining all the semantic layers produces the best performances for all the considered metrics. In particular, the URI layer is the most effective, as it is always included in the top settings for MAP. These results show that the integration of different semantic information leads to a general improvement of IR effectiveness, in line with the purpose of the proposed approach.
Table 8.4. NDCG@10 and MAP of textual plus semantic layer against textual, query-by-query. Queries with no performance change are omitted (“q01”, “q04”, “q06”, “q12”, “q26”, “q32”, “q34”, and “q46”). Note that semantic information may be available even if no difference is observed.

<table>
<thead>
<tr>
<th>Query</th>
<th>Δ NDCG@10</th>
<th>Δ MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>TYPE</td>
<td>FRAME</td>
</tr>
<tr>
<td>q02</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q03</td>
<td>0.002</td>
<td>–</td>
</tr>
<tr>
<td>q07</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q08</td>
<td>–</td>
<td>0.049</td>
</tr>
<tr>
<td>q09</td>
<td>0.029</td>
<td>–</td>
</tr>
<tr>
<td>q10</td>
<td>–</td>
<td>0.093</td>
</tr>
<tr>
<td>q13</td>
<td>0.066</td>
<td>0.066</td>
</tr>
<tr>
<td>q14</td>
<td>–</td>
<td>0.015</td>
</tr>
<tr>
<td>q16</td>
<td>0.018</td>
<td>–</td>
</tr>
<tr>
<td>q17</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q18</td>
<td>–</td>
<td>0.002</td>
</tr>
<tr>
<td>q19</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q22</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q23</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q24</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q25</td>
<td>0.007</td>
<td>0.002</td>
</tr>
<tr>
<td>q27</td>
<td>-0.007</td>
<td>0.011</td>
</tr>
<tr>
<td>q28</td>
<td>-0.117</td>
<td>-0.016</td>
</tr>
<tr>
<td>q29</td>
<td>0.002</td>
<td>–</td>
</tr>
<tr>
<td>q36</td>
<td>–</td>
<td>-0.016</td>
</tr>
<tr>
<td>q37</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>q38</td>
<td>–</td>
<td>0.028</td>
</tr>
<tr>
<td>q40</td>
<td>0.054</td>
<td>0.007</td>
</tr>
<tr>
<td>q41</td>
<td>0.104</td>
<td>-0.004</td>
</tr>
<tr>
<td>q42</td>
<td>–</td>
<td>0.011</td>
</tr>
<tr>
<td>q44</td>
<td>0.149</td>
<td>0.091</td>
</tr>
<tr>
<td>q45</td>
<td>-0.002</td>
<td>-0.002</td>
</tr>
</tbody>
</table>

8.3.3 Query-by-query Analysis

To complete the analysis of the overall results, we investigate the performances of the system query-by-query, discussing four representative queries more in-depth.

Impact of each single layer  Table 8.4 shows, query by query, the impact of each semantic layer on system effectiveness. The first column contains the query identifier; from the second to the fifth columns, we show the comparison between the NDCG@10 computed using textual and single-layer semantic information, and the NDCG@10 computed by using only the textual information. From the sixth to the ninth columns, the same values are shown for the MAP metric. We selected only the MAP and the NDCG@10 metrics because those are the most indicative metrics for evaluating the performances of IR systems in general (MAP), and for deployment in a real-world environment (NDCG@10).
<table>
<thead>
<tr>
<th>Query</th>
<th>Query Text</th>
<th>(\Delta) NDCG@10</th>
<th>(\Delta) MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>q27</td>
<td>Nazis confiscate or destroy art and literature</td>
<td>0.154</td>
<td>0.099</td>
</tr>
<tr>
<td>q28</td>
<td>Modern Age in English Literature</td>
<td>-0.117</td>
<td>-0.095</td>
</tr>
<tr>
<td>q44</td>
<td>Napoleon’s Russian Campaign</td>
<td>0.151</td>
<td>0.147</td>
</tr>
<tr>
<td>q46</td>
<td>First woman who won a Nobel Prize</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The **type** layer affects the highest number of queries, but for some of them (e.g., “q28”) its contribution is negative. This issue is likely a consequence of the large quantity of information inserted when the query is expanded with **type** terms, especially the ones corresponding to super-types of entities and concepts mentioned in the query, which may lead documents scarcely related to the query being matched. Indeed, by injecting too much information in queries, it is possible to obtain a detrimental effect as shown by Abdelali et al. (2007), who discuss query expansion trade-offs and impact on IR effectiveness.

The **uri** layer also impacts on many queries with both positive and negative effects. Differences on NDCG@10 and MAP scores are larger than the ones resulting from the **type** layer (see, e.g., queries “q16” and “q28”), reflecting the fact that **uri** terms impact more on document scores (with respect to the textual layer) since they are generally more selective (high IDF) and often correspond to entities mentioned multiple times in a document (high TF).

The **frame** and **time** layers, where available, have almost always a positive impact on performances (esp. for “q27” and “q44”). The **frame** layer affects the smallest number of queries. As described in Section 8.1, this layer describes relations between entities detected in the text, and thus requires a query structure that is more complex with respect to a simple keyword-based one.

**Analysis of selected queries** We select four queries giving hints about pros and cons of using semantic information in IR. Table 8.5 shows these queries and their performances when using all the semantic layers, compared to the use of textual information only.

Query “q46” is an example where semantic information has no effects. This is because entities at different granularities are injected in the **uri** layers of query and documents. Specifically, the query is annotated with `dbpedia:Nobel_Prize`, while relevant documents have annotations like `dbpedia:Nobel_Prize_in_X`, where X is one of the disciplines for which Nobel Prizes are assigned. Unfortunately, these entities are not related in DBpedia (also in terms of types), thus it is not possible to expand the query in order to find matches with relevant documents.

Query “q28” is an example where worse performances are achieved by using semantic information, due to EL errors. From the query, two **uri** terms (and related **type** terms) are correctly extracted: `dbpedia:Modern_history`, with no matches in the document collection, and `dbpedia:English_literature`, with 12 matches. Of these matches, 11 are incorrect and
refer to irrelevant documents where the instance `dbpedia:English_literature` is wrongly linked to mentions of other “English” things in the text (e.g., “English scholar”, “English society”, “English medical herbs”).

Queries “q27” and “q44” are examples where semantic information significantly boost performances. In “q44”, the correct link to `dbpedia:Napoleon` and the type and time information associated to that entity in DBpedia allow extracting URI, TYPE, and TIME terms that greatly help ranking relevant documents higher. In “q27”, the major improvement derives from the extraction and matching of FRAME term ⟨`framebase:frame-Destroying`, `dbpedia:Nazism`⟩; while TIME information is also available (as `dbpedia:Nazism` is linked to category `dbc:20th_century` in DBpedia), our KE4IR implementation is not sophisticated enough to exploit it.

8.3.4 Balancing Semantic and Textual Content

In our work, we combine both textual and semantic content to improve the performances of IR. While in previous analyses we assigned equal weights to semantic and textual information, here we experiment with different balances. Figure 8.2 shows how the NDCG@10 and MAP metrics change when the importance given to the semantic information changes as well. On the y-axis, we have the NDCG@10 (Figure 8.2a) and MAP (Figure 8.2b) values, while on the x-axis we have the weight $w(\text{semantics})$ assigned to all the semantic information and divided equally among semantic layers, with $w(\text{textual}) = 1 - w(\text{semantics})$; a $w(\text{semantics})$ value of 0.0 means that only textual information is used (and no semantic content), while a value of 1.0 means that only semantic information is used (and no textual content). Results show that semantic information impacts positively on system performances up to $w(\text{semantics}) \leq 0.89$ for NDCG@10 and $w(\text{semantics}) \leq 0.92$ for MAP, reaching the highest scores around 0.61 and 0.65, respectively. Similar behaviors can be observed for NDCG and MAP@10. The highest scores obtained (NDCG@10 = 0.809, MAP = 0.763) are better than the scores reported by intuitively using equal textual and semantic weights, suggesting the importance of a methodology for optimal weight tuning.

8.4 Related Work

The use of semantic information in IR is not new. An early tentative in injecting domain knowledge information for improving the effectiveness of IR systems was presented by Croft (1986). In this work, the author manually built a thesaurus supporting the expansion of terms contained in documents and queries. Such a thesaurus modeled a set of relations between concepts including synonymy, hyponymy, instantiation, meronymy, and similarity. An approach based on the same philosophy was presented by Gonzalo et al. (1998), who proposed an indexing technique where WordNet synsets, extracted from each document word, are used in place of textual terms in the indexing task.
In the last decade, semantic IR systems started to embed ontologies for addressing the task of retrieving domain-specific documents. Two models for the exploitation of ontology-based knowledge bases were presented by Vallet et al. (2005) and Castells et al. (2007). Both models aim at improving search over large document repositories, and feature ontology-based schemes for document annotation and retrieval models adapting VSM. A review of IR techniques based on ontologies was presented by Dridi (2008), while Jimeno-Yepes et al. (2010) analyzed the usefulness of ontologies in IR, and Tomassen (2006) studied the application of ontologies to a large-scale IR system for Web usage.

More recently, approaches combining many different semantic resources for retrieving documents have been proposed. Fernández et al. (2011) described an ontology-enhanced IR platform where a repository of domain-specific ontologies is exploited for addressing the challenges of IR in the massive and heterogeneous Web environment. Waitelonis et al. (2015) described two IR techniques that exploit EL to annotate documents and queries with DBpedia instance URIs, and import and leverage typing and relational background knowledge associated to those instances in DBpedia.

While these works typically build on terminological knowledge (i.e., types and their taxonomy) and possibly EL (i.e., entities and associated background knowledge), to the best of our knowledge KE4IR is the first approach exploiting a large variety of automatically
extracted semantic content (i.e., entities, types, frames, temporal information, with associated background knowledge) for IR.

A further problem in IR is the ranking of retrieved results. Users typically make short queries and tend to consider only the first ten to twenty results (Spink et al., 2006), which justifies our choice of considering (also) Prec@N, MAP, and NDCG metrics for the first ten retrieved results in Section 8.3. An approach that specifically addresses the evaluation of document relevance and the ranking of documents in ontology-based IR is the one by Stojanovic (2005), which adopts a retrieval model different from VSM.

When IR approaches are applied in a real-world environment, the computational time needed to evaluate the match between documents and the submitted query has to be considered too. Systems using the well-known VSM, such as KE4IR, typically have higher efficiency in retrieving documents with respect to systems adopting more complex models to account for semantic information. For instance, the work presented by Baziz et al. (2007) implements a non-vectorial data structure that exhibits high computational times for both indexing and retrieving documents.

8.5 Summary

In this chapter we investigated the benefits of using semantic content automatically extracted from text by PIKES for IR. Building on VSM, we designed and implemented an approach, KE4IR, where both queries and documents are processed to extract semantic content such as entities, types, semantic frames, and temporal information. By evaluating our approach on a state-of-the-art document collection, we showed that complementing the textual information of queries and documents with the content resulting by processing them with frame-based ontology population tools, enables to outperform document retrieval performances when only textual information is exploited. Specifically, experiments using different layer combinations show that the aggregation of different semantic layers leads to effective rankings of relevant documents even in a multi-value relevance setting. The analysis of the NDCG and MAP values, representing the most meaningful metrics for evaluating an IR system, both in general and with respect to common user behaviors, validated the possibility of deploying KE4IR in a real-world environment.

The evaluation here reported may be considered a first step for observing the behavior of KE4IR under different configurations and for enabling an analysis of the impact of each semantic layer. The results obtained provide some interesting insights about the components that should be improved for augmenting the effectiveness of the retrieval system. As shown in Table 8.5, concerning queries “q28” and “q46”, single issues in the linking phase may lead to poor results. Thus, instead of trying to enrich as much text as possible with linked information coming from different knowledge bases, the use of approaches favoring precision of suggested links instead of recall may be the best strategy for obtaining a better average improvement of system effectiveness. Similar considerations can be done about all the other semantic layers considered in KE4IR. On the other hand,
relational information of the FRAME layer, while consistently beneficial, affects only few queries, thus suggesting a revision of the layer aimed at increasing its impact on IR performances. Finally, the weighting of textual and semantic layers is an aspect deserving further attention, as confirmed by Section 8.3.4 where the best MAP and NDCG scores were obtained using non-equal weights for semantic and textual layers.
Chapter 9

Using the KnowledgeStore and RDFpro in NewsReader

The goal of the NewsReader\(^1\) EU Project (Jan 2013 – Dec 2015) was to build a news processing infrastructure (Rospocher et al., 2016) for extracting events (i.e., what happened to whom, when and where, such as “The Black Tuesday, on October 24th, 1929, when United States stock market lost 11% of its value”), and organizing these events in coherent narrative stories, combining new events with past events and background knowledge. These stories are then offered to users (e.g., professional decision-makers), that by means of visual interfaces and interaction mechanisms are able to explore them, exploiting their explanatory power and their systematic structural implications, to make well-informed decisions. Achieving this challenging goal required NewsReader to address several objectives:

- to process document resources, detecting mentions of events, event participants (e.g., persons, organizations), locations, time expressions, and so on;
- to link extracted mentions with instances, either previously extracted or available in background knowledge resources such as DBpedia (Lehmann et al., 2015), and corefer mentions of the same instance;
- to complete instance descriptions by complementing extracted mention information with available structured background knowledge;
- to interrelate instances to support the construction of narrative stories;
- to store all this huge quantity of information (on resources, mentions, instances) in a scalable way, enabling efficient retrieval and intelligent queries;
- to effectively offer narrative stories to decision makers.

In this chapter, we report on how the KnowledgeStore (Chapter 6) and RDFpro (Chapter 7) were used in NewsReader to help achieving these objectives.\(^2\) Multiple KnowledgeStore instances were configured and deployed to store the texts, Natural Language Processing (NLP) annotations, and knowledge processed in NewsReader in different scenarios (instance sizes: \(\sim 18K – 2.3M\) news, \(\sim 105M – 1.25B\) triples), with some instances exploited to build enhanced applications for decision making and data journalism (Section 9.1). RDFpro was used to support the post-processing of the RDF knowledge extracted from news before loading it in the KnowledgeStore (Section 9.3.2), and to assemble the DBpedia background knowledge datasets used in the project (Section 9.3).

\(^1\)http://www.newsreader-project.eu/

\(^2\)PIKES (Chapter 5) and PreMOn (Chapter 3) were not used in NewsReader, which instead developed its own multi-language knowledge extraction pipelines for processing news articles.
9.1 Using the KnowledgeStore as a Data Hub

The KnowledgeStore played a central role in addressing the objectives of the NewsReader project, acting as a sort of data hub populated with news articles and RDF knowledge extracted by the NewsReader knowledge extraction pipelines, and accessed by applications presenting the users with comprehensive views on the heterogeneous content stored in it (see Figure 9.1). More in details, the KnowledgeStore was successfully deployed, populated, and exploited to build enhanced applications in a number of concrete NewsReader scenarios, in different domains, with real content varying in size. Before detailing these scenarios, we describe the KnowledgeStore data model configuration adopted in NewsReader, as well as the processing tools used to populate the various KnowledgeStore instances.

Data model  Figure 9.2 shows how the KnowledgeStore data model was manually configured for the NewsReader scenarios. Note that while all the elements of KEM (Chapter 4) can be recognized in the data model, many properties have different names and kem:SemanticAnnotations are not used, replaced by a hierarchy of mentions: this is
because KEM has evolved further after the KnowledgeStore data model was configured for NewsReader. Starting from the Resource layer, the original news articles are stored together with their corresponding annotated versions obtained by processing them with NLP knowledge extraction processors,\(^3\) and each resource is described with metadata from the Dublin Core\(^4\) and Nepomuk vocabularies. In the Mention layer several types of mentions are stored, denoting either an instance (e.g., person, organization, event), a relation among instances (e.g., participation links between event and participant mentions, as well as causal, temporal and subordinate links among event mentions and/or time expressions, based on TimeML), or a numerical quantity; several specific mention attributes were defined to

\(^3\)Hence, for each news article, two resources are stored: the original news article, and its fully annotated version. Note that mentions are only linked to the original news article.

\(^4\)http://purl.org/dc/terms/
store information extracted from NLP processing. As usual, the Instance layer contains instances and triples both extracted from text or imported from background knowledge, using however property `gaf:denotedBy` to link them to mentions, since NewsReader made use of GAF (Ground Annotation Framework, Fokkens et al., 2013). See NewsReader Deliverable D6.2.3 (Corcoglioniti et al., 2015b) for more details.

**Population** The KnowledgeStore population in NewsReader was organized in two phases, in accordance with the two information processors developed in the project (Figure 9.1).^5^

- **NewsReader NLP Pipeline.**^6^ This single-resource processor performs the NLP analysis of one news at a time, producing the corresponding NAF (NLP Annotation Format, Fokkens et al., 2014) annotations about: Tokenization, Lemmatization, POS-Tagging, Parsing, WSD (Word Sense Disambiguation), NERC (Named Entity Recognition and Classification), EL (Entity Linking) against DBpedia, SRL (Semantic Role Labeling), Coreference Resolution for entities and events, TERN (Temporal Expression Recognition and Normalization), and Opinion Mining. At the end of the NewsReader NLP Pipeline, the KnowledgeStore NAF Populator is invoked to upload in the KnowledgeStore Resource layer the complete NAF annotated version of the source news article, and to inject in the KnowledgeStore the mentions extracted by processing the news article. The NAF Populator is also used to upload into the Resource layer all the source news articles, setting the values of the several metadata attributes attached to each news article (e.g., publication date, author, title).

- **VUA Event Coreference Module.**^7^ This cross-resource processor works on the results of the NewsReader NLP Pipeline by processing extracted mentions. Clusters of mentions referring to the same instance (e.g., event, person, organization) are identified using machine learning techniques and several features, including mention extents, links to DBpedia, and SRL event-actor links (Cybulska and Vossen, 2014). An instance is created for each cluster of mentions, and triples describing and linking these instances are asserted based on attributes and relations in the Mention layer. These instances and triples are loaded into the Instance layer of the KnowledgeStore.

**Applications** The contents loaded in the KnowledgeStore instances were accessed by users via the web UI, and by two applications via the SPARQL endpoint and ReST API:

- **NewsReader Simple API (Hopkinson et al., 2014).** To support people not familiar with Semantic Web (SW) technologies such as RDF and SPARQL, the NewsReader Simple API was developed by ScraperWiki to act as a mediator between the KnowledgeStore and the end user or application. The NewsReader Simple API exposes an

---

^5^As the processing and population of the several KnowledgeStore instances occurred at various stages of the project, different development versions of the processors were used for preparing them.

^6^http://ixa2.si.ehu.es/nrdemo/demo.php

^7^http://ic.vupr.nl/~ruben/vua-eventcoreference.ttl/
HTTP ReST API developed in python that uses JSON and is easily accessible from JavaScript, where each method is implemented by evaluating a SPARQL query on the KnowledgeStore starting from a template that is instantiated at runtime with the actual parameters passed to the method (e.g., the method “Actors of a specified type” implements a query that returns all instances having as RDF type the value of the parameter passed to the method).

- **SynerScope.** SynerScope is a visual analytics application delivering real time interaction with network-centric data. SynerScope interacts with the KnowledgeStore through the KnowledgeStore Exporter (Bogaard et al., 2015), a tool that converts selected data stored in the KnowledgeStore to the format digested by SynerScope. SynerScope offers different views (e.g., table view, hierarchical view, map view) on the KnowledgeStore content, enabling users to navigate it through various interaction methods (e.g., selection/highlight, drill down/up, expansion). This way, it is possible to visually browse all events that involve a given person or company, or to build networks of persons/companies based on event co-participation.

**Deployment** Several populated KnowledgeStore instances were deployed within NewsReader in a span of two years. Their characteristics are summarized in Table 9.1, in particular in terms of size. To interpret these numbers, note that different versions of the Knowledge Extraction Processors, Populators, and KnowledgeStore software were used in each scenario (and its versions), leading to different population rates, space requirements, and amounts of information extracted from a news. All the KnowledgeStore instances were deployed on a cluster of five Linux server machines: a machine with the KnowledgeStore servers and the Virtuoso databases, and four additional machines hosting Hadoop HDFS and HBase. In the following sections, we present the different deployment scenarios. Based on this experience, in Section 9.1.5 we discuss open issues and relevant lessons learned applicable to the KnowledgeStore but also to any system with the same goals.

### 9.1.1 Scenario 1: Wikinews

Wikinews is a source of general domain news in different languages (English news were only considered). Two versions were processed at different times during NewsReader: Wikinews (Ver. 1) and Wikinews (Ver. 2) (cf. corresponding columns in Table 9.1). Differently from the other scenarios considered in NewsReader, Wikinews data are publicly available (Creative Commons Attribution 2.5 License). This allows exposing the corresponding

---

9 A server with two Intel Xeon E5-2440 CPUs (12 cores), 32 GB RAM, and 1 TB HD, later replaced with a server with two Intel Xeon E5-2430 CPUs (24 cores), 192 GB RAM, and 480 GB SSD.
Table 9.1. Overview of KnowledgeStore instances deployed in NewsReader; n/a means that the value was not recorded (data no more available); starred values for Wikinews (Ver. 2) refer to an alternative backend (ElasticSearch) instead of HBase.

<table>
<thead>
<tr>
<th></th>
<th>Wikinews Ver. 1</th>
<th>Wikinews Ver. 2</th>
<th>FIFA World Cup Ver. 1</th>
<th>FIFA World Cup Ver. 2</th>
<th>Cars Ver. 1</th>
<th>Cars Ver. 2</th>
<th>Cars Ver. 3</th>
<th>Dutch Parliament</th>
</tr>
</thead>
<tbody>
<tr>
<td>News</td>
<td>18,510</td>
<td>19,755</td>
<td>212,258</td>
<td>63,635</td>
<td>1,259,748</td>
<td>2,316,158</td>
<td>597,530</td>
<td></td>
</tr>
<tr>
<td>words/news</td>
<td>314</td>
<td>268</td>
<td>597</td>
<td>531</td>
<td>387</td>
<td>394</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Mentions per news</td>
<td>2,629,176</td>
<td>5,206,202</td>
<td>76,165,114</td>
<td>9,110,683</td>
<td>205,114,711</td>
<td>842,639,827</td>
<td>9,231,113</td>
<td></td>
</tr>
<tr>
<td>Instances</td>
<td>670,031</td>
<td>673,018</td>
<td>10,246,338</td>
<td>2,212,691</td>
<td>27,123,724</td>
<td>44,559,443</td>
<td>5,495,077</td>
<td></td>
</tr>
<tr>
<td>events persons</td>
<td>624,439</td>
<td>632,704</td>
<td>9,387,356</td>
<td>1,783,991</td>
<td>25,156,574</td>
<td>42,296,287</td>
<td>5,383,498</td>
<td></td>
</tr>
<tr>
<td>in DBpedia</td>
<td>19,677</td>
<td>17,617</td>
<td>403,021</td>
<td>199,999</td>
<td>729,797</td>
<td>895,541</td>
<td>43,546</td>
<td></td>
</tr>
<tr>
<td>organizations</td>
<td>9,744</td>
<td>10,784</td>
<td>40,511</td>
<td>16,787</td>
<td>128,183</td>
<td>126,140</td>
<td>13,942</td>
<td></td>
</tr>
<tr>
<td>in DBpedia</td>
<td>15,559</td>
<td>14,358</td>
<td>431,232</td>
<td>187,842</td>
<td>947,262</td>
<td>1,139,170</td>
<td>44,139</td>
<td></td>
</tr>
<tr>
<td>locations</td>
<td>6,317</td>
<td>4,940</td>
<td>15,984</td>
<td>8,695</td>
<td>60,547</td>
<td>44,458</td>
<td>12,907</td>
<td></td>
</tr>
<tr>
<td>in DBpedia</td>
<td>10,356</td>
<td>8,339</td>
<td>24,729</td>
<td>40,859</td>
<td>290,091</td>
<td>228,445</td>
<td>23,894</td>
<td></td>
</tr>
<tr>
<td>in DBpedia</td>
<td>7,773</td>
<td>7,369</td>
<td>16,372</td>
<td>11,364</td>
<td>88,695</td>
<td>76,341</td>
<td>11,167</td>
<td></td>
</tr>
<tr>
<td>Triples from Mentions</td>
<td>105,675,519</td>
<td>110,861,823</td>
<td>240,731,408</td>
<td>316,034,616</td>
<td>535,035,576</td>
<td>1,240,774,944</td>
<td>188,296,316</td>
<td></td>
</tr>
<tr>
<td>from DBpedia</td>
<td>9,700,585</td>
<td>16,688,833</td>
<td>136,135,841</td>
<td>46,359,300</td>
<td>439,060,642</td>
<td>1,146,601,954</td>
<td>65,631,222</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>15,974,934</td>
<td>94,172,990</td>
<td>104,595,567</td>
<td>269,675,316</td>
<td>95,974,934</td>
<td>94,172,990</td>
<td>122,665,094</td>
<td></td>
</tr>
<tr>
<td>time (hrs)</td>
<td>2</td>
<td>*</td>
<td>n/a</td>
<td>56</td>
<td>30</td>
<td>160</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>rate (news/h)</td>
<td>9,300</td>
<td>*</td>
<td>n/a</td>
<td>4,000</td>
<td>2,250</td>
<td>7,800</td>
<td>~6,400</td>
<td></td>
</tr>
<tr>
<td>Disk space (GB)</td>
<td>17.64</td>
<td>16.33</td>
<td>82.48</td>
<td>30.67</td>
<td>260.20</td>
<td>967.99</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Resource layer</td>
<td>1.25</td>
<td>*</td>
<td>1.40</td>
<td>16.55</td>
<td>3.10</td>
<td>108.27</td>
<td>48.87</td>
<td></td>
</tr>
<tr>
<td>Mention layer</td>
<td>1.49</td>
<td>*</td>
<td>1.64</td>
<td>41.72</td>
<td>4.77</td>
<td>112.00</td>
<td>558.74</td>
<td></td>
</tr>
<tr>
<td>Instance layer</td>
<td>14.90</td>
<td>13.29</td>
<td>24.21</td>
<td>22.80</td>
<td>39.93</td>
<td>66.88</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

KnowledgeStore instances\textsuperscript{14} populated with a complete dataset consisting of structured content (mentions, instances, triples) linked to the source news from which it was extracted, thus favoring the dissemination of the project results and enabling other researchers and developers to exploit this content for various purposes, such as benchmarking their knowledge extraction pipelines, or building and testing new Linked Open Data (LOD) applications. Given its controlled size, substantially smaller than the other scenarios here reported, Wikinews data—and in particular the MEANTIME (Minard et al., 2016) subset manually annotated by a team of linguists as part of the project—was used in NewsReader to support benchmarking of the knowledge extraction processors.

\subsection*{Scenario 2: FIFA 2014 World Cup}

This scenario is about revealing hidden facts and people networks behind FIFA World Cup 2014, by building web-based applications on top of the KnowledgeStore. A total of 212,258 football-related news articles, from various providers (LexisNexis,\textsuperscript{15} BBC,\textsuperscript{16} The Guardian\textsuperscript{17}) and distributed over a time period of ten years (2005–2014), were processed and uploaded into the KnowledgeStore (cf. column “FIFA World Cup” in Table 9.1).

While data collection and preparation required significant time and effort, the development of applications on top of stored content was realized as part of a hackathon event (held in London, 10th June 2014)\textsuperscript{18} where 40 people, a mixture of LOD enthusiasts and data journalists, gathered for one day to collaboratively develop web-based applications on top of the KnowledgeStore, using its interfaces as well as the NewsReader Simple API. Ten web-based applications, implemented in different programming languages, were developed in roughly 6 working hours. Each application was developed with a focused purpose: among them, to determine which teams a football player had played in during his career (by looking at transfer events); to discover which football teams were most commonly associated with violence; to determine people and companies related to gambling; and, to establish the popularity of people, companies, and football teams in different locations.

During the hackathon, the KnowledgeStore received 30,126 queries (on average, 1 query/s, with peaks of 20 queries/s), issued either directly through the SPARQL endpoint or via the NewsReader Simple API, and successfully served them on average in 654 ms (only 40 queries out of 30,126 took more than 60 seconds to complete).

\subsection*{Scenario 3: Global Automotive Industry Crisis Review}

This scenario is about analyzing the news related to the last decade’s financial crisis, with a special focus on the global automotive industry sector, in order to mine its key events, and to understand the role of major players (e.g., CEOs, companies) in it. The news

\textsuperscript{14}Wikinews (Ver. 2) instance: \url{http://knowledgestore2.fbk.eu/nwr/wikinews/}
\textsuperscript{15}\url{http://www.lexisnexis.nl/}
\textsuperscript{16}\url{http://www.bbc.com/}
\textsuperscript{17}\url{http://www.theguardian.com/}
\textsuperscript{18}\url{http://www.newsreader-project.eu/newsreader-world-cup-hack-day/}
articles were made available for project purposes by LexisNexis, and three KnowledgeStore instances were prepared: Cars (Ver. 1), Cars (Ver. 2), and Cars (Ver. 3), with the number of news ranging from 63,635 to 2,316,158 (cf. corresponding columns in Table 9.1).

The main application in this scenario is SynerScope. In addition, the capability to query the KnowledgeStore content was exploited to deliver automatically generated reports (and plots) supporting decision makers. For instance, by retrieving the different events involving the ten major car companies, it was possible to generate a report showing the trend of the quantity of events per year in which these companies were involved in the considered period, and therefore to assess their popularity (according to the considered dataset) during the economic crisis. Similarly, by retrieving the different events with their locations and times, it was possible to produce maps (one per year) providing insights into how the localities of the global automotive industry changed during the crisis.

The Cars (Ver. 2) KnowledgeStore instance was exploited in two hackathon events held in Amsterdam, 21st January 2015,19 and London, 30th January 2015,20 where enhanced applications were built to conduct exploratory investigations on top of the KnowledgeStore: among them, an in-depth analysis of the age of CEOs when they get hired or fired from companies, analysis of the most dangerous cars around, car companies with high cars recall rate, and so on. During the hackathons, the KnowledgeStore received 118,094 requests (3 requests/s avg., with peaks of 40 requests/s), issued either directly through its endpoints or via the NewsReader Simple API, and successfully served them on average in 31 ms. The Cars (Ver. 3) instance was also used for an end-user evaluation in November 2015.

9.1.4 Scenario 4: The Dutch House of Representatives

In this scenario, a KnowledgeStore instance (Dutch Parliament in Table 9.1) was populated with content extracted from texts about an inquiry of the House of Representatives on the financial system, with the aim of making this information more insightful. The corpus consists of news and magazine articles, debate transcripts, and parliamentary papers provided by the Information Provision Department of the Dutch House of Representatives, and ~50K news articles about ABN-AMRO (one of the main financial players) by LexisNexis.

Differently from the previous scenarios, all the texts considered in this scenario are in Dutch and were processed with a version of the NewsReader knowledge extraction pipeline specifically tailored for this language. To account for this aspect, the KnowledgeStore instance was loaded with a multilingual version of the DBpedia background knowledge that contains textual attributes also in the Dutch language (see Section 9.3).

In June 2015, the navigation of the KnowledgeStore instance via SynerScope was presented to about 10 members of the Information Provision Department of the Dutch House of Representatives (including the head of the department), and to 3 members of De Nederlandsche Bank (The Dutch Bank) who had expressed their interest in this use case.

19http://www.newsreader-project.eu/amsterdam-hackathon-recap/
20http://www.newsreader-project.eu/london-hackathon/
9.1.5 Discussion

Two years of concrete usage of the KnowledgeStore in NewsReader (including hackathons with real users) provided valuable insight on the practical issues and the user expectations encountered when deploying a system like the KnowledgeStore, permitting us to validate our design and identify its weaknesses. In this section we discuss our findings, that we believe are of general interest for any system addressing the goals of the KnowledgeStore.

Unified query language  Concrete usage of the system in NewsReader showed that users appreciate the expressivity of SPARQL for accessing Instance layer data, and they ask for a unified, SPARQL-like language targeting all the contents of the KnowledgeStore, in place of (or augmenting) the CRUD methods exposed to access the other layers of the system. Providing such a language is however a challenging task due to the volume of data and the different storage backends involved.

Analytical queries  The KnowledgeStore logs in NewsReader showed the submission of many analytical SPARQL queries that match and/or extract a sensible amount of stored data. Users submitted SPARQL queries to compute statistics, to analyze loaded corpora and to assess the results and performance of knowledge extraction processors. As discussed at the end of Section 6.3.2, also the NewsReader Simple API contains queries that can be classified as analytical. While SPARQL can be used for these investigations, some analytical queries take long times to execute (or even fail or time out), in some cases due to improper query planning but most often due to their inherent complexity. While we improved some of these queries on an ad-hoc basis, e.g., via careful rewriting or by materializing some properties that help speeding up queries, a more general and principled approach to handle analytical requests is clearly needed in the KnowledgeStore.

Read/write separation  Practical experience in NewsReader showed a sharp separation between read and write accesses to the KnowledgeStore, with populators and knowledge extraction processors performing large, infrequent write operations, whereas access from applications was essentially read-only. Many KnowledgeStore instances were populated exactly once, with their contents never modified after. Also in cases were incremental updates were implemented, to handle streams of news, these updates were designed to be performed periodically (e.g., every day) in batch mode. This evidence opens the possibility to redesign and optimize the system for a write once, read many load, rather than the mixed write many, read many load the KnowledgeStore was mainly built for.

Flexible access control  Access control becomes a requirement in presence of copyrighted content whose provision and consumption involve different parties. This aspect

\[\text{For instance, we materialized rdfs:isDefinedBy} \]

\[\text{annotations that link each vocabulary term to the URI of the ontology defining it, so to ease querying for concepts of a specific vocabulary.}\]
turned out to be particularly relevant in research scenarios such as NewsReader, where dissemination needs conflict with the need of content providers to protect intellectual property. In general, different access control policies apply to resources from different sources and, within a resource, to its text and various metadata attributes (e.g., title and date can be publicly accessible whereas author and text cannot). Access control policies also apply to mention and instance data derived from copyrighted resources, with the situation being more complex for instance data as they combine information extracted from multiple resources, possibly with different distribution policies. As we had to adapt several times access control in NewsReader, we stress the importance for systems like the KnowledgeStore of flexible access control mechanisms able to accommodate known requirements and cope (as far as possible) with their unanticipated change in time.

9.2 Post-processing Extracted RDF with RDFpro

RDFpro was used for post-processing the RDF data extracted by the NewsReader knowledge extraction pipeline before loading it the KnowledgeStore. Specifically, RDFpro was involved in the three post-processing tasks described next.

Data merging and filtering The RDF output of the VUA Event Coreference Module, consisting of several RDF files, was processed with RDFpro to merge all these files in a single RDF dataset without duplicate quads, using a few Groovy scripts (via the @groovy processor) to: (i) generate the rdfs:labels of owltime:DateTimeDescription s; (ii) convert xsd:gYear literals to xsd:int, as the former were problematic to load into Virtuoso; and (iii) to generate rdfs:isDefinedBy triples to link each concept to the URI of the vocabulary defining it, enabling the efficient SPARQL querying of concepts of selected vocabularies.

Materialization of pre-/post-/during- situations Given the declarations in the ESO (Event Situation Ontology, Segers et al., 2015) ontology and an event typed according to ESO, in many cases it is possible to materialize a number of triples describing the situations holding immediately before, during, or after that event. For example, for a “giving” event where a person gives an object to another person, in the pre-situation the first person owns the object, while in the post-situation it is the second person who owns the object. The materialization of these triples is performed by the ESO Reasoner (Corcoglioniti et al., 2015b, Section 8), a software component implemented as a processor of RDFpro and executed within an RDFpro pipeline. Taking advantage from the fact that this kind of reasoning is defined on a per-event basis, the ESO Reasoner specializes the @mapreduce processor in order to partition the input triples by event instance and process each partition separately to materialize the triples of its pre-, post- and during-situations.

22The ESO Reasoner itself is not a contribution of this thesis, but rather an example of the tasks that can be achieved by leveraging RDFpro.
Merging of old and new event data  The NewsReader infrastructure can handle a stream of news by processing them in small batches, incrementally updating the data in the KnowledgeStore based on the results of each batch. While data in the Resource and Mention layers are only added with this approach, for data in the Instance layer it may happen that new events are added but also that events previously asserted in the KnowledgeStore are merged (smushed) together and enriched with new data, as a result of cross-document coreference. This situation is handled by invoking RDFpro (via a “custom API” specifically added to the KnowledgeStore) for each batch of RDF data produced by the VUA Event Coreference Module, fetching (and removing) the RDF data in the KnowledgeStore affected by the operation and using the @smush, ESO Reasoner, and @groovy processors to perform the necessary post-processing and add back the resulting triples to the Instance layer of the KnowledgeStore.

9.3 Preparing Background Knowledge with RDFpro

Within NewsReader, RDF background knowledge from DBpedia is collected and stored in the Instance layer of the KnowledgeStore, where it is integrated with knowledge extracted from texts and made available for consumption to use case applications (e.g., the hackathon ones) and NLP modules. This background knowledge consists of terminological and assertional data about instances, events, and their relations, extracted and assembled from selected DBpedia files using RDFpro. In the following, we describe how data selection and processing were done (Sections 9.3.1 and 9.3.2), and provide some statistics on the background knowledge datasets obtained, whose utility is not restricted to NewsReader (Section 9.3.3).

9.3.1 Data Selection

DBpedia data is organized primarily by the language of the Wikipedia chapter it has been extracted from. As NewsReader considers both single-language (EN) and multiple-language scenarios (EN, ES, IT, NL), both an English (en) version and a multilingual (ml) version of the background knowledge were produced. The en version maps to the selection of the EN DBpedia chapter and includes only EN literals, while the ml version maps to the selection (and integration) of the EN, ES, IT, and NL DBpedia chapters, with literals in these languages. It must be noted, however, that also other DBpedia chapters can contribute with relevant non-localized data, such as numeric and relational data, when they describe instances that occur also in one of the considered EN, ES, IT, and NL DBpedia chapters. In these cases, the descriptions of the instance in the different chapters are often different and complementary (in general, the richest description is the one in the chapter of the language associated to the instance country), thus making possible to build a richer merged description. As this merging may also introduce inconsistencies (e.g., incompatible values assigned to the same functional property), two variants were produced for each localization of the background knowledge (en, ml): one enriched with data from all the 18
available DBpedia chapters (ext variant) and the other one not enriched. Summing up, the following four background knowledge datasets were generated:

- **en** dataset, with EN-only literals and data of DBpedia EN;
- **en_ext** dataset, with EN-only literals and data of all DBpedia chapters;
- **ml** dataset, with EN, ES, IT, NL literals and data of corresponding DBpedia chapters;
- **ml_ext** dataset, with EN, ES, IT, NL literals and data of all DBpedia chapters.

For each language, DBpedia data is divided in multiple files based on topic. The following files were selected for inclusion in the background knowledge:

- instance types and properties based on FOAF (Friend of a Friend\(^23\)) and DBpedia vocabularies (files `instance_types`, `instance_types_heuristic`, `mappingbased_properties_cleaned`, `persondata`);
- instance names based on Wikipedia titles (file `labels`);
- instance links to WordNet (Fellbaum, 1998) synsets (file `wordnet_links`);
- geographic coordinates of location instances (file `geo_coordinates`), keeping only property `georss:point` and dropping redundant data;
- links to instance images in Wikipedia (file `images`), including property `foaf:depiction` and excluding thumbnails data and copyright metadata (images are all open licensed);
- links to instance home pages (file `homepages`);
- brief language-dependent textual description of instances (file `short_abstracts`);
- `owl:sameAs` links among DBpedia chapters and among URIs and IRIs assigned to the same instance\(^24\) (files `interlanguage_links`, `iri_same_as_uri`).

In addition to the assertional data listed above, the terminological data from the following vocabularies was also selected: DBpedia OWL ontology, Dublin Core, FOAF, SKOS (Simple Knowledge Organization System, Miles and Bechhofer, 2009), WGS84\(^25\) and GeoRSS\(^26\) for geographic data.

### 9.3.2 Data Processing

Selected files cannot be simply “concatenated” to produce the background knowledge datasets. Data must be filtered on a per-file basis, in order to remove unwanted data. `owl:sameAs` smushing\(^27\) and deduplication must then be applied to filtered data from different datasets, so to ensure that duplicate triples are removed (preserving provenance metadata) and that each instance identified by multiple URIs (connected by `owl:sameAs`)

---

\(^{23}\)http://www.foaf-project.org/

\(^{24}\)The newer IRIs supports a broader set of characters and are more readable for non-English languages. As tools may still use URIs rather than IRIs (e.g., for linking a mention to an instance), both kinds of identifiers were included in the background knowledge, together with the `owl:sameAs` triples linking them.

\(^{25}\)http://www.w3.org/2003/01/geo/wgs84_pos

\(^{26}\)http://www.w3.org/2005/Incubator/geo/XGR-geo/W3C_XGR_Geo_files/geo_2007.owl

\(^{27}\)http://patterns.dataincubator.org/book/smushing.html
Figure 9.3. Example of SPARQL query (a) with and (b) without smushing and RDFS inference.

RDFS inference and owl:sameAs smushing greatly ease the use and querying of RDF data once stored in the KnowledgeStore, as user queries can now assume that full knowledge about an instance (including implicit knowledge) is materialized in the KnowledgeStore and associated to a single, reference instance URI notwithstanding the many URI aliases the instance can have. This results in more simple and efficient queries, as shown in the example of Figure 9.3 where a simple query extracting names and surnames of persons is written both (a) assuming and (b) not assuming owl:sameAs smushing and RDFS inference.

In order to perform the required data filtering, owl:sameAs smushing, and inference materialization, a processing workflow was assembled based on RDFpro. The workflow automates these tasks based on a simple script and some configuration data specifying the URLs of the files to process and how to process them, customized for each of the four background knowledge datasets selected in Section 9.3.1. Both workflow script and configuration data are published on the KnowledgeStore website, so that the whole process is repeatable and reconfigurable by anyone. The workflow is organized in the four download, filtering, merging, and analysis stages described next.

**Download stage** Dataset and vocabulary files listed in the workflow configuration are downloaded from their source locations (if locally missing or newer), and trigger further processing in the next stages of the workflow.

**Filtering stage** Each downloaded file is parsed, filtered and saved by RDFpro using a common format (Turtle Quads, i.e., N-Quads with Turtle encoding\(^{28}\)) and compression method (gzip, due to good compression/speed trade-off). Filtering is performed in a single

\(^{28}\)http://wiki.dbpedia.org/Internationalization/Guide
pass on a per-triple basis, using the @transform RDFpro processor. It allows dropping triples with specific predicates and types on a per-file basis and, for every file, to remove literals in a language not supported and to rewrite blank nodes making them globally unique (this avoids clashes when merging data from multiple files). Triples in each filtered file are placed in a named graph associated to the DBpedia chapter it comes from, so to keep track of provenance in the next processing.

Merging stage This stage merges the filtered files previously generated, performing owl:sameAs smushing and inference materialization and producing the final background knowledge dataset. Merging is implemented with two invocations of RDFpro:

- first, terminological definitions are extracted from filtered files using the @tbox processor, deprived of axioms leading to unwanted inferences using the @transform processor (e.g., rdfs:subClassOf axioms leading to inferring that everything is an owl:Thing or rdfs:Resource), and saved in a TBox output file.

- then, a more complex RDFpro pipeline is executed that calls in sequence the @smush, @rdfs, @transform, and @unique processors to perform owl:sameAs smushing, RDFS inference, filtering out of unwanted triples, and deduplication of filtered files, using the TBox file previously created. The @unique processor is configured so to place each produced triple in a graph linked to all the DBpedia chapters asserting (or leading via RDFS inference) to that triple. The @rdfs processor is configured so to disable rules RDFS2 and RDFS3 on rdfs:domain and rdfs:range axioms, as many of them are imprecise and cause the materialization of a large number of ‘incorrect’ triples (e.g., that almost every dbo:place is also a dbo:person).

Analysis stage This stage computes some statistics for the generated background knowledge dataset, calling RDFpro twice:

- first, complete VOID (Vocabulary of Interlinked Datasets, Alexander et al., 2009) statistics are extracted from the generated dataset using the @stats processor. These annotations and the TBox file previously extracted can be imported in tools such as Protége enabling an easy navigation of extracted statistics.

- then, additional statistics about the numbers of instances in the dataset for some specific types considered in NewsReader (e.g., PER, ORG, LOC) are extracted using an RDFpro pipeline that (i) keeps only rdf:type triples in the dataset using the @transform processor, (ii) perform RDFS reasoning using custom rdfs:subClassOf axioms that map DBpedia types to the specific types considered, using the @rdfs processor with only rules RDFS4A and RDFS9 enabled, and (iii) invokes the @stats processor to compute the number of instances per type (plus other statistics).

http://protege.stanford.edu/
Table 9.2. DBpedia background knowledge datasets: numbers of triples.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>rdf:type</th>
<th>owl:sameAs</th>
<th>ABox (other)</th>
<th>TBox</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_39</td>
<td>11,583,289</td>
<td>783,200</td>
<td>49,312,382</td>
<td>15,039</td>
<td>61,693,910</td>
</tr>
<tr>
<td>en_ext_39</td>
<td>13,239,452</td>
<td>783,200</td>
<td>61,153,352</td>
<td>15,041</td>
<td>75,191,045</td>
</tr>
<tr>
<td>ml_39</td>
<td>16,142,863</td>
<td>4,239,891</td>
<td>76,073,537</td>
<td>15,043</td>
<td>96,471,334</td>
</tr>
<tr>
<td>ml_ext_39</td>
<td>16,917,881</td>
<td>4,239,891</td>
<td>83,220,239</td>
<td>15,043</td>
<td>104,393,054</td>
</tr>
<tr>
<td>en_2014</td>
<td>15,184,293</td>
<td>859,158</td>
<td>59,774,033</td>
<td>19,798</td>
<td>75,837,282</td>
</tr>
<tr>
<td>en_ext_2014</td>
<td>17,279,671</td>
<td>859,158</td>
<td>77,751,702</td>
<td>19,816</td>
<td>95,910,347</td>
</tr>
<tr>
<td>ml_2014</td>
<td>23,881,931</td>
<td>4,691,477</td>
<td>94,079,675</td>
<td>19,810</td>
<td>122,672,893</td>
</tr>
<tr>
<td>ml_ext_2014</td>
<td>24,872,604</td>
<td>4,691,477</td>
<td>104,800,908</td>
<td>19,816</td>
<td>134,384,805</td>
</tr>
<tr>
<td>en_2015</td>
<td>13,138,091</td>
<td>910,725</td>
<td>59,268,670</td>
<td>19,718</td>
<td>73,337,204</td>
</tr>
<tr>
<td>en_ext_2015</td>
<td>15,539,931</td>
<td>1,097,251</td>
<td>78,436,030</td>
<td>19,735</td>
<td>95,092,947</td>
</tr>
<tr>
<td>ml_2015</td>
<td>22,378,634</td>
<td>4,981,271</td>
<td>88,542,689</td>
<td>19,746</td>
<td>115,922,340</td>
</tr>
<tr>
<td>ml_ext_2015</td>
<td>22,669,522</td>
<td>5,151,730</td>
<td>101,268,784</td>
<td>19,735</td>
<td>129,109,771</td>
</tr>
</tbody>
</table>

9.3.3 Results

The workflow was configured and executed to generate all the background knowledge datasets listed in Section 9.3.1: en, en_ext, ml, ml_ext. DBpedia version 3.9 was used initially, producing the datasets used during the second year of the NewsReader project. With the release of DBpedia 2014 and DBpedia 2015, the workflow was reconfigured and executed again to generate the new versions of the datasets used in the third year of the project. The remainder of this section reports some statistics about the processing done and the resulting datasets, covering the versions based on DBpedia 3.9 (tagged with _39), DBpedia 2014 (tagged with _2014), and DBpedia 2015 (tagged with _2015) so that a comparison between the three can be made.

Table 9.2 reports the number of triples in each dataset, distinguishing between rdf:type triples, owl:sameAs triples, other ABox triples (essentially expressing instance properties) and TBox triples. Moving from DBpedia 2014 to DBpedia 2015 causes a small decrease of dataset size. Enrichment with data from other DBpedia chapters causes an increase in size of 22 – 27% for the en datasets, and 8 – 10% for the ml datasets. Multilingual (ml) datasets contain more owl:sameAs triples, mainly consisting in owl:sameAs links from canonical instance URIs to their aliases in the integrated DBpedia chapters.

Table 9.3 reports the number of instances 30 in each dataset, divided based on some NERC types considered in NewsReader, i.e.: persons (PER), organizations (ORG), geopolitical entities and locations (GPELOC), facilities (FAC), products (PROD), works of art (WOA), and events (EVENT), with MISC representing DBpedia instances that could not be classified under previous types. 31 As the classification of instances is not exclusive...

---

30 Instances have been counted by selecting distinct URIs appearing as the subject of some rdf:type statement and having a named OWL class as its object. This broad definition covers both ABox and TBox concepts, differently from the statistics provided by DBpedia that accounts only for ABox instances.

31 Classification according to these specific types is performed based on the DBpedia classes (dbo: namespace) associated to instances using the following mapping: Person \(\rightarrow\) PER; Organisation \(\rightarrow\) ORG;
### Table 9.3. DBpedia background knowledge datasets: numbers of instances.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PER</th>
<th>ORG</th>
<th>GPELOC</th>
<th>FAC</th>
<th>PROD</th>
<th>WOA</th>
<th>EVENT</th>
<th>MISC</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_39</td>
<td>1,124,450</td>
<td>329,693</td>
<td>621,437</td>
<td>144,143</td>
<td>119,734</td>
<td>372,953</td>
<td>70,464</td>
<td>1,270,659</td>
<td>4,049,227</td>
</tr>
<tr>
<td>en_ext_39</td>
<td>1,246,076</td>
<td>344,802</td>
<td>666,932</td>
<td>165,266</td>
<td>123,814</td>
<td>399,611</td>
<td>94,049</td>
<td>1,292,297</td>
<td>4,299,465</td>
</tr>
<tr>
<td>ml_39</td>
<td>1,368,929</td>
<td>350,210</td>
<td>748,181</td>
<td>273,780</td>
<td>132,853</td>
<td>462,536</td>
<td>112,982</td>
<td>1,910,307</td>
<td>5,342,574</td>
</tr>
<tr>
<td>ml_ext_39</td>
<td>1,427,713</td>
<td>362,106</td>
<td>764,098</td>
<td>285,686</td>
<td>135,415</td>
<td>482,721</td>
<td>123,790</td>
<td>1,945,551</td>
<td>5,493,613</td>
</tr>
<tr>
<td>en_2014</td>
<td>1,649,672</td>
<td>302,727</td>
<td>849,711</td>
<td>164,646</td>
<td>128,196</td>
<td>389,118</td>
<td>89,216</td>
<td>1,226,617</td>
<td>4,634,402</td>
</tr>
<tr>
<td>en_ext_2014</td>
<td>1,673,873</td>
<td>324,190</td>
<td>915,939</td>
<td>187,701</td>
<td>133,324</td>
<td>424,830</td>
<td>115,921</td>
<td>1,306,946</td>
<td>4,863,236</td>
</tr>
<tr>
<td>ml_2014</td>
<td>2,069,158</td>
<td>352,411</td>
<td>1,275,738</td>
<td>423,843</td>
<td>144,297</td>
<td>499,760</td>
<td>220,940</td>
<td>2,056,611</td>
<td>6,606,109</td>
</tr>
<tr>
<td>ml_ext_2014</td>
<td>2,076,553</td>
<td>366,079</td>
<td>1,310,817</td>
<td>434,989</td>
<td>147,350</td>
<td>525,553</td>
<td>232,569</td>
<td>2,111,575</td>
<td>6,738,541</td>
</tr>
<tr>
<td>en_2015</td>
<td>2,135,045</td>
<td>220,422</td>
<td>596,509</td>
<td>143,052</td>
<td>119,393</td>
<td>347,496</td>
<td>80,663</td>
<td>584,115</td>
<td>4,225,234</td>
</tr>
<tr>
<td>en_ext_2015</td>
<td>2,164,052</td>
<td>252,859</td>
<td>668,125</td>
<td>178,130</td>
<td>135,244</td>
<td>426,225</td>
<td>116,485</td>
<td>670,048</td>
<td>4,564,208</td>
</tr>
<tr>
<td>ml_2015</td>
<td>2,669,196</td>
<td>275,377</td>
<td>787,692</td>
<td>413,737</td>
<td>142,034</td>
<td>471,046</td>
<td>310,106</td>
<td>1,393,960</td>
<td>6,446,328</td>
</tr>
<tr>
<td>ml_ext_2015</td>
<td>2,682,263</td>
<td>296,880</td>
<td>811,283</td>
<td>436,526</td>
<td>150,094</td>
<td>532,231</td>
<td>329,807</td>
<td>1,469,789</td>
<td>6,661,598</td>
</tr>
</tbody>
</table>

in DBpedia, the total values in the table do not represent the sum over the number of instances for each specific type.

Table 9.4 reports some statistics on the processing performed with the pipeline for each dataset: number of input files; number of triples resulting from each stage of the pipeline; and execution times of each stage and of the pipeline as a whole. Times were measured on a RedHat 6.4 (Linux 2.6) workstation with an Intel Core i7 CPU, 16 GB RAM and 500 GB disk. Execution times of the download stage are omitted as they depend on bandwidth.

### Table 9.4. DBpedia background knowledge: processing statistics.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Files</th>
<th>Output triples</th>
<th>Execution time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Download</td>
<td>Filter</td>
</tr>
<tr>
<td>en_39</td>
<td>17</td>
<td>77,770,116</td>
<td>70,268,829</td>
</tr>
<tr>
<td>en_ext_39</td>
<td>85</td>
<td>139,812,619</td>
<td>110,394,636</td>
</tr>
<tr>
<td>ml_39</td>
<td>43</td>
<td>178,654,555</td>
<td>120,011,745</td>
</tr>
<tr>
<td>ml_ext_39</td>
<td>98</td>
<td>219,324,567</td>
<td>145,968,914</td>
</tr>
<tr>
<td>en_2014</td>
<td>17</td>
<td>98,897,879</td>
<td>82,371,053</td>
</tr>
<tr>
<td>en_ext_2014</td>
<td>85</td>
<td>176,899,005</td>
<td>133,811,216</td>
</tr>
<tr>
<td>ml_2014</td>
<td>43</td>
<td>236,496,207</td>
<td>144,612,632</td>
</tr>
<tr>
<td>ml_ext_2014</td>
<td>98</td>
<td>287,518,712</td>
<td>177,634,142</td>
</tr>
<tr>
<td>en_2015</td>
<td>17</td>
<td>75,875,870</td>
<td>65,707,375</td>
</tr>
<tr>
<td>en_ext_2015</td>
<td>85</td>
<td>142,990,109</td>
<td>102,832,650</td>
</tr>
<tr>
<td>ml_2015</td>
<td>43</td>
<td>202,306,201</td>
<td>115,089,945</td>
</tr>
<tr>
<td>ml_ext_2015</td>
<td>98</td>
<td>242,809,561</td>
<td>135,211,128</td>
</tr>
</tbody>
</table>
availability. Filtering is slow for DBpedia 3.9 and 2014 and faster for DBpedia 2015, as the former use Turtle while the latter uses the TQL line-oriented RDF format, which is faster to parse. On average, throughputs are 78K triples/s (DBpedia 3.9 and 2014) and 338K triples/s (DBpedia 2015) for the filtering stage, 136K triples/s for the merging stage and 234K triples/s for the analysis stage. We also tested the workflow on the more powerful server where the KnowledgeStore servers are deployed, equipped with two Intel Xeon E5-2430 CPUs (12 physical cores, 24 logical), 192 GB RAM, and 480 GB SSD. On this machine, the throughputs on the DBpedia 2015 datasets are 470K, 249K, and 276K triples/s for the filtering, merging and analysis stages, respectively, with the improvements largely determined by the higher number of available CPU cores.

All the produced datasets and their statistics are available on the KnowledgeStore website. In particular, the statistics of a dataset are distributed as an annotated statistics ontology in two versions: a full version covering all the terminological concepts and a more compact (and manageable) version having only the concepts with more than 100 instances. These ontologies can be imported in tools for ontology editing and browsing such as Protégé, as shown in Figure 9.4, and can help understanding and using the datasets, e.g., by supporting the construction of SPARQL queries.

9.4 Summary

In this chapter we described the use of the KnowledgeStore and RDFpro in NewsReader, the project where both tools originated from. We described the role, data model, and four application scenarios characterizing the use of the KnowledgeStore in NewsReader, and we discussed some lessons learned that, overall, recommend optimizing the KnowledgeStore architecture for a write-once, read-many load, providing unified SPARQL querying over all the KnowledgeStore layers with better support for analytical queries. We then described how RDFpro was used, directly or indirectly (i.e., as a component of other tools such as the ESO Reasoner), for the post-processing of RDF data extracted from news, and how it supported the preparation of the DBpedia background knowledge datasets used in the project, which is the task RDFpro was originally created for at the end of 2013.

Based on these experiences, and in the light also of the specific evaluations of the KnowledgeStore and RDFpro previously reported (Chapters 6 and 7), in Chapter 10 we will analyze possible improvement directions and future works for both tools.
Figure 9.4. DBpedia background knowledge datasets: browsing statistics in Protégé.
Chapter 10

Conclusions

In this Chapter we summarize the contributions of the thesis (Section 10.1) and analyze the results obtained from two perspectives. First, we consider limitations and extension opportunities related to the presented work, describing some interesting directions for future research (Section 10.2). Then, we consider the practical use of the resources and systems developed in this thesis outside of a research scenario, and report some considerations for their further engineering (Section 10.3).

Acknowledgments

The discussion of limitations, future research directions, and engineering aspects reported in this chapter was influenced by previous discussions with co-authors and collaborators, and by the practical experience achieved by using the results of this thesis in the NewsReader\textsuperscript{1} project funded by the European Union (FP7 ICT-316404). The author would like to thank all the persons involved.

10.1 Summary and Contributions

In this thesis we considered the task of Frame-based Ontology Population from English text from the points of view of extraction approach, representation, and storage of all the involved data (to support application consumption). We addressed this task with the investigation and development of models, systems, and applications:

- **Models.** We set the basis by developing two ontological models: PreMOn (Chapter 3), targeting the ontological representation of the predicate models used in SRL (Semantic Role Labeling) and their mappings to frame-based ontologies such as FrameBase (Rouces et al., 2015); and KEM (Chapter 4), targeting the representation of all the information involved in Knowledge Extraction (KE).

- **Systems.** Based on the PreMOn and KEM models, we developed PIKES (Chapter 5), a 2-phase approach leveraging KEM, SRL, and PreMOn mappings for performing Frame-based Ontology Population from text. We also exploited KEM for building the KnowledgeStore (Chapter 6), a scalable system supporting the joint storage of all the information represented in KEM. During this work it was necessary to implement several RDF processing tasks coping with large amounts of RDF data, and this led us to develop RDFpro (Chapter 7), a tool for non-distributed RDF processing based on streaming, sorting, and arbitrary composition of processing tasks in pipelines.

\textsuperscript{1}http://www.newsreader-project.eu/
Applications. We reported two applications of the previous works: (i) KE4IR (Chapter 8), an Information Retrieval (IR) approach where both documents and queries are augmented with semantic terms extracted by PIKES to improve search performances; and (ii) the use of the KnowledgeStore and RDFpro in the NewsReader EU project (Chapter 9), for storing news article and the RDF knowledge extracted from them, together with specifically-prepared RDF background knowledge.

In the following, we go through the contributions reported in Section 1.3 and graphically depicted in Figure 10.1, and we discuss, for each contribution, the results obtained.

Contribution C1 – PreMOn An ontological model for representing predicate models and their mappings, based on lemon and covering PropBank, NomBank, VerbNet, FrameNet, SemLink, and FrameBase (Coccoglioniti et al., 2016b, Chapter 3).

PreMOn\(^2\) (Predicate Model for Ontologies) consists of: (i) an ontology extending lemon (McCrae et al., 2012; Cimiano et al., 2015) for representing the PropBank (Palmer et al., 2005), NomBank (Meyers et al., 2004), FrameNet (Baker et al., 1998), and VerbNet (Kipper Schuler, 2005) predicate models with their common aspects, specificities, and mappings; and (ii) a Linked Open Data (LOD) dataset of \(~\!20M triples with interlinked data for the considered resources, as well as the SemLink (Palmer, 2009) mappings between predicate models and from predicate models to FrameBase concepts. PreMOn ontologies and data are publicly available for download, and accessible via SPARQL querying and URI dereferencing. We believe that the utility of PreMOn goes beyond its use as a support resource in this thesis. On the one hand, the PreMOn ontology endorses the goal of lemon to provide proper linguistic grounding for ontologies, enabling the grounding of event and frame ontologies, such as FrameBase and ESO (Event Situation Ontology, Segers et al., 2015), on a proper representation of the linguistic information for predicates. On the other hand, PreMOn adoption of a common RDF format, stable URIs, and LOD best practices makes easier to reuse and jointly query data of heterogeneous predicate resources, enabling the use of Semantic Web (SW) technologies such as automated reasoning and SPARQL querying. In turn, these capabilities open up new opportunities for analyzing, validating, and possibly cleaning up or extending predicate model data.

Contribution C2 – KEM An ontological model for representing all the contents involved in KE along three layers: Resource, Mention, and Instance (Chapter 4).

With KEM\(^3\) (Knowledge Extraction Model) we provided a solid model for representing KE data, aligned to the semiotic notions of reference and meaning, and to relevant ontologies such as DOLCE (Gangemi et al., 2002), LMM (Linguistic Meta-Model, Picca et al., 2008), and NIF (NLP Interchange Format, Hellmann et al., 2013). The uses of KEM in PIKES

\(^2\)http://premon.fbk.eu/
\(^3\)http://knowledgestore.fbk.eu/ontologies/kem/
and the KnowledgeStore provide concrete examples of the practical usability of the model, and demonstrate the advantages of using a single RDF model to encode all the data involved in KE, together with KEM named-graph mechanism for representing metadata and fine-grained provenance information.

**Contribution C3 – PIKES** A 2-phase, SRL-based Frame-Based Ontology Population approach for English, extracting FrameBase semantic frames and instances aligned to DBpedia, YAGO, and SUMO (Corcoglioniti et al., 2016c, 2015e, Chapter 5).

PIKES⁴ (PIKES Is a Knowledge Extraction Suite) is publicly available as an open source tool with an online demo, and neatly separates processing in two phases. First, several Natural Language Processing (NLP) tasks are performed on text, exploiting two SRL systems—Mate-tools (Björkelund et al., 2009) and Semafor (Das et al., 2014)—to produce an RDF graph of mentions that integrates and exposes the NLP annotations produced by different tools in an homogeneous way according to KEM. Then, the mention graph is transformed into a knowledge graph aligned to DBpedia (Lehmann et al., 2015), YAGO (Hoffart et al., 2013), and SUMO (Niles and Pease, 2001), using rules and the FrameBase mappings of PreMOn. This decoupling allows an easy replacement and combination of different NLP tools, even for the same task, as well as an easy combination of the outputs of different NLP tasks to support non-trivial linguistic phenomena (e.g., argument nominalization); this flexibility is further enhanced by the use of semantic frames to represent extracted knowledge, as they can accommodate the output of additional NLP tools (not considered in PIKES, so far), e.g., for Causal and Temporal Relation Extraction (Mirza and Tonelli, 2014), or for various flavors of Sentiment or Opinion Detection (Dragoni et al., 2014, 2015; Palmero Aprosio et al., 2015). PIKES was successfully applied to a corpus of 100K documents in ~500K core hours (∼32 hours on a single server) and its precision and recall were evaluated on a small gold standard and compared to the

---

⁴http://pikes.fbk.eu/
ones of FRED (Presutti et al., 2012), a state-of-the-art tool for KE in the SW, overall showing that PIKES is capable of extracting quality knowledge from large text corpora.

**Contribution C4 – KnowledgeStore** A SW-grounded scalable system for storing all the Resource, Mention, and Instance data described and interlinked according to KEM (Corcoglioniti et al., 2015c, 2013; Rospocher et al., 2014a,b, Chapter 6).

The KnowledgeStore\(^5\) integrates a triplestore for managing Instance layer data, and a scalable, distributed HBase\(^6\) and Hadoop\(^7\) HDFS infrastructure for storing Resource and Mention layer data, providing end users and applications with a ReST API and a user interface for accessing all these contents. Experiments measuring the data population and data retrieval performances of the system, and its concrete usage in NewsReader, demonstrated the appropriateness and adequateness of the KnowledgeStore in coping with the goals it was designed for. Thanks to the joint storage of structured knowledge (both background and extracted knowledge), the resources it derives from, and mention information, all effectively accessible through a single API, the KnowledgeStore enables the development of enhanced applications and provides an ideal scenario for developing, debugging, training, and evaluating tools for a number of KE and NLP tasks.

**Contribution C5 – RDFpro** A tool for non-distributed processing of large amounts of RDF data, based on streaming, sorting, and the arbitrary composition of processing tasks in complex pipelines (Corcoglioniti et al., 2015d,a, 2014, Chapter 7).

RDFpro\(^8\) (*RDF Processor*) provides out-of-the-box implementations—called *processors*—of common tasks such as data filtering, rule-based inference, `owl:sameAs` smushing\(^9\), and statistics extraction, as well as easy ways to add new processors and arbitrarily compose processors in complex pipelines. Through an evaluation of RDFpro in four application scenarios, we showed that many RDF processing tasks common in practice can be performed on billions of triples on a single commodity machine using streaming and sorting, with execution times compatible with batch RDF processing and better than approaches using a local data index (e.g., a triplestore). All these characteristics make RDFpro a sort of “Swiss army knife” for exploring and manipulating RDF datasets, and a viable compromise between local RDF processing based on memory-intensive, non-scalable data indexing and distributed processing based on heavyweight infrastructures (e.g., Hadoop).

**Contribution C6 – KE4IR** An IR approach using PIKES to augment documents and queries with URI, TYPE, FRAME, and TIME semantic terms, which are indexed and matched using an adaptation of VSM (Corcoglioniti et al., 2016a, Chapter 8).

\(^5\)http://knowledgestore.fbk.eu/
\(^6\)http://hbase.apache.org/
\(^7\)http://hadoop.apache.org/
\(^8\)http://rdfpro.fbk.eu/
KE4IR\textsuperscript{10} \textit{(Knowledge Extraction for Information Retrieval)} augments VSM (Vector Space Model, Salton et al., 1975) with semantic terms, and has been implemented within an evaluation infrastructure that allows assessing its performances on arbitrary documents and queries. We evaluated it on the dataset by (Waitelonis et al., 2015) against a purely textual baseline, obtaining statistically significant improvements of MAP (Mean Average Precision), NDCG (Normalized Discounted Cumulated Gain, Järvelin and Kekäläinen, 2002), and their MAP@10 and NDCG@10 restrictions to the top 10 results, when all the semantic layers of KE4IR are used. This experience indicates another area where KE, and specifically the Frame-based Ontology Population approach of PIKES, can be beneficial, further showing the potential of the research directions pursued in this thesis.

10.2 Limitations and Future Research Directions

In this section, we list some limitations of the work presented in this thesis that can be addressed by pursuing certain research directions, indicated with numbers 1–5 in Figure 10.1 and in the following (in parenthesis the affected components).

1 \textbf{Improved predicate models mappings (PreMOn, PIKES)} Although PreMOn mappings to ontological concepts play a central role in PIKES, they currently provide full coverage only for frames and frame elements of FrameNet 1.5, while only a small part of PropBank and NomBank is mapped to FrameBase concepts; furthermore, other ontologies such as ESO are not considered, and FrameNet and VerbNet selectional constraints are not mapped at all to ontological concepts (e.g., YAGO types), although this would enable assigning ontological types to the instances participating to a semantic frame.

Considered that the benefits of having comprehensive ontological mappings go beyond the use of mappings in PIKES, an obvious research direction is thus to improve mappings to frame-base ontologies and make them publicly available in a new release of PreMOn. This can be achieved by leveraging existing mappings between predicate models in order to propagate ontological concepts mapped in one model to another (also among different versions of the same model, as e.g., from FrameNet 1.5 to FrameNet 1.6), and by applying WSD (Word Sense Disambiguation) to definitions and examples of semantic classes in order to assign them to a WordNet (Fellbaum, 1998) synset and thus reduce the set of possible FrameBase classes (tied to synsets) they can be mapped to. Some mapping resources that can be exploited are: (i) the mappings between predicate models of the Predicate Matrix (Lacalle et al., 2014); (ii) the existing mappings from ESO to FrameNet; and (iii) the FrameNet selectional constraints by Bryl et al. (2012).

2 \textbf{Inconsistency detection and repair (PIKES, RDFpro)} Currently, there are no ontological constraints (e.g., \texttt{owl:disjointWith} axioms, OWL cardinality constraints) in

\textsuperscript{10}http://pikes.fbk.eu/ke4ir.html

155
the FrameBase and YAGO ontologies populated by PIKES that would allow tagging an extracted ABox triple as inconsistent. This results in intuitively incompatible instances (e.g., a person and a location) being possibly coreferred and smushed together, or semantic frames being generated with incompatible participant types (e.g., a location in an agentic role). The problem further worsen when merging knowledge extracted from multiple texts.

An interesting research direction consists thus in: (i) introducing a rich axiomatization of populated ontological properties and classes, including inconsistency-related constraints such as disjointness axioms; and (ii) taking these axioms into consideration during knowledge distillation (or the subsequent post-processing step) to avoid generating an inconsistent ABox. The first step can reuse axioms from DOLCE, via the mappings from DBpedia (and thus YAGO) to DOLCE, and the rdfs:domain and rdfs:range axioms that would result from mapping FrameNet selectional constraints to FrameBase (see above); additional constraints (e.g., temporal) can be added manually. The second step, inspired by other state-of-the-art approaches (e.g., Sofie by Suchanek et al., 2009), may consist in removing the triples making the extracted ABox inconsistent, possibly selected by minimizing their number (or sum of confidences) using techniques such as Weighted Max SAT or Integer Linear Programming; the RDFpro tool can be extended to evaluate constraints (which can be realized as rules) and detect violations. Particular importance should be given to removing wrong owl:sameAs triples coming from coreference resolution and entity linking, which represent frequent sources of noise in extracted data.

A possible extension of this technique would be considering all the linking and typing alternatives produced by NLP modules for a given mention, scored by respective confidence values, and then add constraints imposing that only one of them can be chosen in the ABox.

3 Unified SPARQL querying with analytical loads (KnowledgeStore) The practical use of the KnowledgeStore has highlighted the benefits of having a unified query language—possibly SPARQL—providing access to all the representation layers of the system and supporting analytical queries (Section 9.1.5). However, SPARQL querying in the KnowledgeStore is currently restricted to instance data, as resource and mention data are stored in HBase; moreover, no special support for analytical queries is offered.

Given the large amount of data stored in the Resource and Mention layers (a few thousand of triples per resource, on average), if compared to data in the Instance layer, devising an architecture supporting unified SPARQL access to all layers is a research

---

11See also the work by Paulheim and Gangemi (2015) regarding the use of these mappings for detecting inconsistencies in an ABox, in this case the one of DBpedia.

12These techniques can also be used to recover implicit information (e.g., facts implied by the text) in order to satisfy logical constraints (e.g., defined on classes of semantic frames), as shown in the business process domain by Bertoli et al. (2013). The notion of (in)consistency can be extended to cover temporal constraints, e.g., on the legal successions or compositions of semantic frames representing events. A possible way to consider these constraints is to treat event instances as activities of some formal process, and then use existing techniques from the state of the art to define and validate temporal and logical constraints on top of the process (see, e.g., Ghidini et al., 2012).

13From manual error analysis of several extraction errors.
challenge on its own. The problem can be simplified by restricting the kinds of interaction supported by the KnowledgeStore, providing a write-once, read many solution where writes are targeted at a master storage, which is then (asynchronously) replicated to one or more slave storages serving a read-only load. This solution is justified by the findings of Section 9.1.5, and would allow: (i) the adoption of specialized indexes in the read-only storage that do not have to account for online writes; and (ii) a more easy distribution and replication of data in the read-only storage, to achieve horizontal scalability. An interesting implementation, that we partially investigated, consists in organizing the read-only storage in a core triplestore, holding the subset of data accessed more frequently (mainly Instance data), and a constellation of peripheral triplestores holding resource and mention data partitioned by resource. Using federated query execution techniques (Prud’hommeaux and Aranda, 2013), SPARQL queries would hit the core triplestore first, with portions of them being forwarded to specific peripheral triplestores based on the resources they access.

Supporting analytical queries is another challenge, whose solution depends on how unified SPARQL querying is realized. Possible approaches include: (i) adopting query execution techniques that rely on external storage (rather than RAM) and possibly on distributed paradigm such as MapReduce for robustly dealing with large amounts of data (e.g., Kotonlas et al., 2012); and (ii) providing mechanisms where analytical queries can be registered in the system and computed on a periodic basis or when data change, so that their results are always readily available.

Learning to rank and relational information for IR (KE4IR) The investigation of KE4IR is at its very beginning, and in discussing its performances (Section 8.5), we noted that relational information is currently under-exploited in the FRAME layer (few terms extracted, lack of terms from background knowledge), and that the weighting of layers used in our experiment is sub-optimal and asks for a proper methodology to optimally weigh and combine the contributions of the various layers.

Concerning the use relational information, a first option would be to relax the current restriction of using only disambiguated DBpedia instances in the ⟨frame, participant⟩ pairs of the FRAME layer, which favors precision but reduces the impact of this layer; the problem here is finding a new definition of FRAME terms that do not penalize precision excessively (as we verified would happen if we simply remove this restriction). Another direction for further exploiting relational information is importing FRAME terms about mentioned instances from background knowledge, by mapping triples of DBpedia (or other LOD sources) to FrameBase frames, using the mapping rules (and/or methodology) provided by FrameBase authors; in addition, it is also conceivable to extract background knowledge FRAME terms directly from Wikipedia pages, by processing them with PIKES. Concerning the weighting of layers, a possible direction is the use of supervised learning to rank (Liu, 2011) methods to automatically determine the optimal weights of layers (for a certain corpus), given a training set of few queries with their associated gold rankings.

\footnote{http://www.framebase.org/data}
Experiment on larger and/or domain-specific datasets (PIKES, KE4IR)

Both PIKES and KE4IR were evaluated on small, domain-general corpora (8 sentences in PIKES for assessing precision/recall, ∼300 documents in KE4IR). Although a challenging task, considering larger corpora would enable more articulated analyses and fine tuning opportunities for both systems, while considering domain-specific corpora would allow investigating their performances and applicability in specific domains (e.g., biomedical).

While large IR datasets, such as the TREC AdHoc Test Collection\(^{15}\) and the TREC WT10g,\(^{16}\) do exist for testing KE4IR, the problem being scaling the evaluation infrastructure to their large dimensions (from few hundreds of documents to millions of them), a large corpus for assessing PIKES precision and recall against FrameBase does not exist. Possibly, such corpus can be created starting from the manually-crafted full text annotations of FrameNet, by semi-automatically mapping SRL annotations to expected semantic frames (this requires manually identifying and coreferring instances within gold SRL annotations).

Validation of PIKES and KE4IR in specific domains can be done, e.g., against the dataset and queries of the BIOASQ challenge\(^{17}\) (biomedical domain). The deployment in domain-specific contexts like this one would require the use of domain-specific resources able to provide more effective NLP annotations, in particular for what concerns EL (Entity Linking) and SRL. For the first task, domain-specific KBs can be used by integrating KB-agnostic EL tools such as AGDISTIS (Usbeck et al., 2014) in PIKES, and the resulting domain-specific annotations can be picked up by KE4IR to generate domain-specific URI and TYPE terms. For the second, domain-specific frames might have to be defined (by extending FrameBase and the predicate models it is mapped to) and annotated manually in a training corpus to retrain the SRL tools used in PIKES, after which they become available to KE4IR for generating FRAME terms.

10.3 Engineering Perspectives

The outcomes of this thesis include also a number of systems—PIKES, the KnowledgeStore, RDFpro, KE4IR—which, given further engineering efforts and to different degrees, can potentially be used outside the research context they originated from. In this concluding section we thus consider, from a purely engineering perspective, another three future works that can greatly impact the usability of these tools by a wider community of users, indicated with numbers 6–8 in the following and in Figure 10.1.

System simplification and scaling down (KnowledgeStore, PIKES) Both the KnowledgeStore and PIKES feature complex architectures that make difficult their deployment on a commodity machine, e.g., for processing small datasets, or for evaluation or demonstration purposes. Complexity derives, for the KnowledgeStore, from the adoption

\(^{15}\)http://trec.nist.gov/data/test_coll.html

\(^{16}\)http://ir.dcs.gla.ac.uk/test_collections/wt10g.html

\(^{17}\)http://www.bioasq.org/participate/challenges
of external (distributed) components such as HBase, Hadoop HDFS, and Virtuoso\(^{18}\) (a minimal setup requires 7 processes), while for PIKES it derives from the execution as separate processes of NLP modules like UKB (Agirre et al., 2014) and DBpedia Spotlight (Mendes et al., 2011), each one requiring separate installation and configuration.

While the use of these components is justified by scalability and performance reasons, it is also important to additionally provide simpler, scaled-down configurations of the KnowledgeStore and PIKES that either do not require these components (e.g., by replacing them with embeddable storage libraries in the KnowledgeStore, such as ElasticSearch\(^{19}\) investigated in NewsReader), or that streamline and automatize their installation and configuration, with the ultimate goal of making accessible both the KnowledgeStore and PIKES to a larger audience of possible users.

**Tight system integration (KnowledgeStore, PIKES, KE4IR)** While a user can recur to PIKES for generating the data populating the KnowledgeStore, the two systems are currently loosely integrated. At the same time, KE4IR is implemented only in an evaluation infrastructure specifically targeting the experiments of Chapter 8, although it could provide a useful keyword-based search mechanism for accessing data in the KnowledgeStore, exploiting Instance and Mention layer data already stored in the system.

Therefore, integrating PIKES and KE4IR in the KnowledgeStore appears a natural direction to pursue from an engineering perspective. Specifically, the integration of KE4IR would require the adoption and customization of a full-text index for managing the indexing and search of textual and semantic terms, such as Lucene\(^{20}\) and derivatives (e.g., ElasticSearch, already mentioned above for storing mention and resource data), with the challenge being scaling the implementation to the large amount of data possibly stored in the KnowledgeStore. Instead, the integration of PIKES can be realized by defining an extension point in the KnowledgeStore where standardized, possibly pre-packaged and pre-configured knowledge extraction pipelines can be plugged in to be automatically invoked by the system when a resource is uploaded. This would provide casual users with a ready-to-use system for extracting and storing knowledge from text, while advanced users would have at the same time the possibility to assemble and plug in their own pipelines.

**Support of RDF validation (RDFpro, KnowledgeStore, PreMON)** A problem we frequently faced when post-processing extracted data in NewsReader and building the PreMON dataset is the validation of RDF data. By validation we mean checking that data satisfy specific constraints going beyond logical consistency, such as the mandatory presence (or lack) of certain triples (with specific cardinalities), the use of certain URIs, namespaces, and datatypes, and other constraints generally expressible via rules

---

\(^{18}\)http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/

\(^{19}\)http://www.elastic.co/products/elasticsearch

\(^{20}\)http://lucene.apache.org/
(e.g., a rule prohibiting the existence of a ontolex:evokes triple without a corresponding pmo:Conceptualization instance).

Recognizing the importance of this task (and based on prior research work), the W3C has started the standardization of SHACL (Shapes Constraint Language\textsuperscript{21}), a language that allows specifying and automatically validating the patterns that must be satisfied by descriptions of RDF instances in a dataset. Validation of SHACL patterns on large amounts of data can be implemented quite straightforwardly in RDFpro by grouping triples by instance (via the @mapreduce processor) and validating the patterns (defined on a per-instance basis) on the resulting partitions, possibly performing multiple iterations in case of patterns involving multiple instances. This feature would further extend the practical usability of RDFpro, providing a standard-compliant validation mechanism that can be exploited in the preparation of the next PreMOn dataset releases and in the engineering of the KnowledgeStore (e.g., for automatically checking that loaded data satisfy the constraints of the configured data model).

\textsuperscript{21}http://www.w3.org/TR/shacl
**Bibliography**


Presutti, V., Draicchio, F., and Gangemi, A. (2012). Knowledge extraction based on discourse representation theory and linguistic frames. In *Proc. of Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*, pages 114–129. Springer. See also http://stlab.istc.cnr.it/stlab/FRED.


167


Appendix A

KnowledgeStore Evaluation Details

We report here the complete specification of the parametric requests used in the data retrieval evaluation of the KnowledgeStore (Chapter 6) reported in Section 6.3.2. We start with defining the parameters used, describing their relations and computation. We then specify the 14 parametric requests that were previously informally described in Table 6.4.

A.1 Parameter Specification

Table A.1 describes the parameters used in the parametric requests. The parameters are not independent but their values must satisfy some dependencies, indicated in the descriptions in the table, in order for the instantiated requests to properly simulate the behavior of a user exploring the dataset, either directly via the UI or indirectly using some application built on top of the ReST API. Each parameter-value mapping allows instantiating the parametric requests and obtaining a request mix. Given a dataset, a user-configurable number of mappings can be automatically constructed by evaluating 7 auxiliary queries (available on the KnowledgeStore website) each one extracting all the admissible values for overlapping subsets of parameters: (event, event-year, resource), (event, event-type), (event, event-term), (event, actor, actor-related), (actor, actor-type), (actor, actor-term), (actor, actor-property). Using the query test generator tool, the resulting tuple sets are then joined and the result is randomly sampled to build the desired number of mappings and thus of request mixes.

A.2 Request Specification

We report below the specification (SPARQL expression, ReST URL) of the parametric requests selected for the evaluation, using notation $\{\text{par}\}$ for parameters and indicating for each request the query(ies) of the NewsReader Simple API (Hopkinson et al., 2014) it has been derived from.\(^1\) The selection process can be summarized as follows:

- we discarded 3 queries of the Simple API out of 18: query ‘11. Get situation graph’ because subsumed by query ‘5. Get event precis’; query ‘18. Types of actors’ and query ‘9. Get the properties of a type’ because precomputable offline and thus of little interest for our evaluation;
- of the chosen 15 queries, we aggregated query ‘14. Get events with a specific eso value’ and query ‘15. Get events with a specific framenet value’ because very similar;

\(^1\)See http://newsreader.scraperwiki.com/ for the specification of the original Simple API queries.
Table A.1. KnowledgeStore retrieval evaluation: request parameters (with examples).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>The URI of an event in the dataset</td>
<td>⟨<a href="http://bit.ly/1zkELHj%E2%9F%A9">http://bit.ly/1zkELHj⟩</a></td>
</tr>
<tr>
<td>event_term</td>
<td>A term in one of the textual labels of the event</td>
<td>‘hire’</td>
</tr>
<tr>
<td>event_year</td>
<td>The year when the event happened</td>
<td>2011</td>
</tr>
<tr>
<td>event_type</td>
<td>The URI of an ontological type associated to the event</td>
<td>eso:JoiningAnOrganization</td>
</tr>
<tr>
<td>actor</td>
<td>The URI of an actor participating in the event</td>
<td>dbpedia:Burson-Marsteller</td>
</tr>
<tr>
<td>actor_term</td>
<td>The URI of an ontological type associated to the actor</td>
<td>dbo:Company</td>
</tr>
<tr>
<td>actor_property</td>
<td>A term in one of the labels of the actor (e.g., its name)</td>
<td>‘burson’</td>
</tr>
<tr>
<td>actor_related</td>
<td>The URI of a related actor sharing an event with actor</td>
<td>dbpedia:Facebook</td>
</tr>
<tr>
<td>resource</td>
<td>The URI of a news article in the KS mentioning the event</td>
<td>⟨<a href="http://bit.ly/1BkEZE7%E2%9F%A9">http://bit.ly/1BkEZE7⟩</a></td>
</tr>
</tbody>
</table>

- we added a file download request (crud 2) that is missing in the Simple API as news article files contain copyrighted material;
- we set OFFSET to 0 and LIMIT to 20 where used (frequent values during hackathons);
- we finally did some semantics-preserving cleanup of the SPARQL query expressions.

sparql 1  1. Actors of a specified type

```
SELECT ?actor ?comment (COUNT(DISTINCT ?event) AS ?count)
WHERE { ?event sem:hasActor ?actor.
  ?g1 dct:source <http://dbpedia.org/>.
  GRAPH ?g1 { ?actor a ${actor_type} } ?g2 dct:source <http://dbpedia.org/>.
  GRAPH ?g2 { ?actor rdfs:label ?l. ?l bif:contains ${actor_term} }
  OPTIONAL { ?actor rdfs:comment ?comment } }
GROUP BY ?actor ?comment ORDER BY DESC(?count) LIMIT 20
```

sparql 2  10. Property of mentioned actors of type

```
SELECT DISTINCT ?actor ?value
WHERE { ?event sem:hasActor ?actor.
  ?actor a ${actor_type}; ${actor_property} ?value; rdfs:label ?l.
  ?l bif:contains ${actor_term}. }
ORDER BY DESC(?value) LIMIT 20
```

sparql 3  4. Frequency of event labels in events

```
SELECT ?event_label (COUNT(DISTINCT ?event) AS ?count)
  ?event_label bif:contains ${event_term}. }
GROUP BY ?event_label ORDER BY DESC(?count) LIMIT 20
```

sparql 4  12. Get events mentioning a named actor

```
SELECT ?event (COUNT(*) AS ?event_size) ?datetime ?event_label
WHERE { { SELECT DISTINCT ?event ?datetime ?event_label
```

170
WHERE { ?event sem:hasActor|sem:hasPlace ${actor};
  rdfs:label ?event_label; sem:hasTime ?t.
  ?t owltime:inDateTime ?d.
  ?d owltime:year ${event_year}; rdfs:label ?datetime. }
ORDER BY ?datetime LIMIT 20
?event ?p ?o. }
GROUP BY ?event ?datetime ?event_label ORDER BY ?datetime

sparql 5 17. Get events mentioning 2 named actors
SELECT ?event (COUNT(*) AS ?event_size) ?datetime ?event_label
WHERE { { SELECT DISTINCT ?event ?datetime ?event_label
  WHERE { ?event sem:hasActor ${actor}, ${actor_related};
    sem:hasTime ?t; rdfs:label ?event_label.
    ?t owltime:inDateTime ?d.
    ?d owltime:year ${event_year}; rdfs:label ?datetime. }
  ORDER BY ?datetime LIMIT 20 }
?event ?p ?o. }
GROUP BY ?event ?datetime ?event_label ORDER BY ?datetime

sparql 6 13. Get events mentioning a type of actor
SELECT ?event (COUNT(*) AS ?event_size) ?datetime ?actor
WHERE { { SELECT DISTINCT ?event ?datetime ?actor
  WHERE { ?event sem:hasTime ?t; sem:hasActor|sem:hasPlace ?actor.
    ?actor a ${actor_type}.
    ?t owltime:inDateTime ?d.
    ?d owltime:year ${event_year}; rdfs:label ?datetime. }
  ORDER BY ?datetime LIMIT 20 }
?event ?p ?o. }
GROUP BY ?event ?datetime ?actor ORDER BY ?datetime

sparql 7 15. Search for events with a specified label
SELECT ?event (COUNT(*) AS ?event_size) ?datetime ?event_label
WHERE { { SELECT DISTINCT ?event ?datetime ?event_label
  WHERE { ?event sem:hasTime ?t; rdfs:label ?event_label.
    ?event_label bif:contains ${event_term}.
    ?t owltime:inDateTime ?d.
    ?d owltime:year ${event_year}; rdfs:label ?datetime. }
  ORDER BY ?datetime LIMIT 20 }
?event ?p ?o. }
GROUP BY ?event ?datetime ?event_label ORDER BY ?datetime

sparql 8 14. Get events with specific eso + 16. Get events with specific framenet type
SELECT ?event (COUNT(*) AS ?event_size) ?datetime ?event_label
WHERE { { SELECT DISTINCT ?event ?datetime ?event_label
  WHERE { ?event a sem:Event, ${event_type};
    rdfs:label ?event_label; sem:hasTime ?t.
    ?event_label bif:contains ${event_term}.
    ?t owltime:inDateTime ?d.}
?d owltime:year ${event_year}; rdfs:label ?datetime. }

ORDER BY ?datetime LIMIT 20 }

GROUP BY ?event ?datetime ?event_label ORDER BY ?datetime

sparql 9 5. Get event triples

SELECT DISTINCT ?subject ?predicate ?object ?graph
WHERE {{ ${event} eso:hasPreSituation|eso:hasPostSituation|
eso:hasDuringSituation ?graph.
   GRAPH ?graph { ?subject ?predicate ?object. } } UNION
   { BIND (${event} AS ?subject)
     GRAPH ?graph { ${event} ?predicate ?object }
     FILTER (?predicate = sem:hasActor || ?predicate = sem:hasPlace ||
     ?predicate = rdf:type && EXISTS {?object rdfs:isDefinedBy eso:}
     || EXISTS {?predicate rdfs:isDefinedBy eso:}) } UNION
   { GRAPH ?graph { ${event} sem:hasTime ?t }
     ?t owltime:inDateTime ?object
     BIND (nwr:cleanedTime AS ?predicate) } UNION
   { SELECT (<ex:numDocuments> AS ?predicate)
     (COUNT(DISTINCT strbefore(str(?mention), "#")) AS ?object)
     WHERE { ${event} gaf:denotedBy ?mention } } } }

sparql 10 2. Details of URI from DESCRIBE query

DESCRIBE ${actor}

sparql 11 8. People sharing event with named person

SELECT (${actor} AS ?actor) ?actor2 ?comment (COUNT(DISTINCT ?event) AS ?numEvent)
   FILTER(?actor2 != ${actor}) OPTIONAL { ?actor2 rdfs:comment ?comment } }
GROUP BY ?actor2 ?comment ORDER BY DESC(?numEvent) LIMIT 20

sparql 12 3. Details of events with specified actor

SELECT ?event ?predicate ?object (SAMPLE(?t) AS ?object_type)
WHERE { { SELECT DISTINCT ?event
   WHERE { ?event sem:hasActor ${actor}. } }
   ORDER BY DESC(?event) LIMIT 20 }
OPTIONAL { ?object a ?t. FILTER ( ?t = sem:Actor || ?t = sem:Place ||
   ?t = sem:Time || ?t = sem:Event ) }


http://host/resources?id=<${resource}>
http://host/mentions?ks:mentionOf=<${resource}>

crud 2 (not in Simple API)

http://host/files?id=<${resource}>