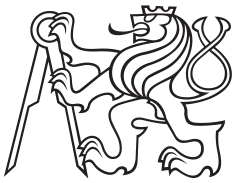
Dissertation Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Leveraging Semantic Web Technologies in Domain-specific Information Systems

Ing. Martin Ledvinka

Supervisor: Ing. Petr Křemen, Ph.D.
Field of study: Electrical Engineering and Information Technology
Subfield: Artificial Intelligence and Biocybernetics
August 2020

Acknowledgements

First and foremost, I want to thank my family for their everlasting care and support. I would like to that my supervisor, Dr. Petr Křemen, for teaching me that research texts should also be good stories. I would also like to thank my colleagues at the Knowledge-based and Software Systems group for creating a pleasant and challenging working environment. I would like to thank Michal Med – our friendly competition was a big motivation for working on this thesis.

Declaration

I declare that I have written my dissertation thesis independently and consistently quoted the sources in the submitted work.

In Prague, 2020-08-11.

Abstract

Although envisioned as a successor of the ubiquitous Web, the Semantic Web, and its related technologies had been for a long time primarily a matter of academic research. Only in recent years has it started to make its way into the mainstream, mainly as a technology stack behind Linked (Open) Data. Nevertheless, Semantic Web technologies can be a welcome asset of domain-specific information systems that are used to create, manage, and process reusable data.

However, Semantic Web technologies still represent rather uncharted territory for developers of such systems. This can be attributed to the amount of non-trivial knowledge a Semantic Web developer has to posses, the lack of mature tools, and still incomparable performance of Semantic storage systems in contrast to relational databases, as well as the semantic impedance between the two.

This thesis tries to facilitate the adoption of Semantic Web technologies by developers of domain-specific information systems by building a stack of principles and tools that they can use to seamlessly integrate such technologies into their regular development stack. In particular, it provides a formal framework for the often problematic access to Semantic data and introduces a persistence library built upon this formalization. In addition, a tool for Semantic Web-based information system integration is presented. The thesis also discusses whether the utilization of Semantic Web technologies influences the architecture of an information system as a whole. Finally, several real-world examples showcase the presented results.

Keywords: Semantic Web, Information System, Ontology, Linked Data, Object-oriented Programming

Supervisor: Ing. Petr Křemen, Ph.D.
Department of Computer Science,
Faculty of Electrical Engineering,
Czech Technical University in Prague,
Karlovo náměstí 13,
121 35 Praha 2,
Czech Republic

Abstrakt

Ačkoliv se měl stát nástupcem dnes všudypřítomného webu, sémantický web a jemu příbuzné technologie byly po dlouhou dobu předmětem především akademického výzkumu. Teprve v posledních letech začal tento koncept pronikat i do běžného použití, hlavně jako technický prostředek využívaný iniciativou propojených (otevřených) dat. Nicméně, technologie sémantického webu mohou být vítanou posilou i doménových informačních systémů používaných k tvorbě, správě a zpracování znovupoužitelných dat.

Bohužel, sémantické technologie zatím pro vývojáře takových systémů zůstavají spíše neprozkoumaným územím. Tento problém lze připsat množství nových znalostí, které musí vývojář načerpat, nedostatku pokročilých vývojářských nástrojů a stále ještě výrazně horšímu výkonu úložišť sémantických dat ve srovnání s relačními databázemi. Dalším faktorem jsou pak principiální rozdíly mezi oběma typy databází.

Cílem této práce je podpořit využití technologií sémantického webu vývojáři doménových informačních systémů vybudováním ekosystému principů a nástrojů, díky kterým lze sémantické technologie snadno začlenit do běžně používaných postupů a knihoven. Konkrétně, text představuje formální rámec pro popis aplikačního přístupu k sémantickým datům a softwarovou knihovnu, která je na tomto formalismu založená. Dále je prezentován nástroj pro integraci aplikací pomocí sémantických technologií. Práce též diskutuje, zda a do jaké míry ovlivňuje využití sémantického webu architekturu informačního systému jako takového. Nakonec je uvedeno několik příkladů reálných informačních systémů vybudovaných na základě prezentovaných výsledků.

Klíčová slova: Sémantický web, Informační systém, Ontologie, Linked Data, Propojená data, Objektově orientované programování

Překlad názvu: Využití technologií sémantického webu v doménových informačních systémech

Contents

Figures

Tables

Part I

Motivation and Problem Statement

Chapter 1

Introduction

The concept of the *Semantic Web* was popularized in the widely cited article by Tim Berners Lee, Jim Hendler, and Ora Lassila [1]. It was envisioned as an evolution of the *Web*, which consisted of websites with mostly unstructured data (text, images) and which was intended for human consumers. The Semantic Web would add languages and principles to allow machines to aid humans by being able to "understand" resources accessible on the Internet. For instance, as Berners Lee et al. write, a user's digital assistant could validate a procedure with their insurance plan, check for available physical therapist's appointment slots and compare them with the user's schedule by being able to communicate with different services (physical therapist's scheduler, the insurance company, user's calendar) without having to be previously configured to support their data schema or the protocols they use.

The Semantic Web is built around the idea of *resources* with globally valid identifiers in the form of IRIs (Internationalized Resource Identifiers). The properties of these resources are themselves identified by IRIs. Semantic Web data are described using structured languages like RDF [2] which are designed to be processed primarily by machines. The types of resources, their properties, and relationships among them are described in schemas called *ontologies*. Such ontologies can be shared, allowing various Semantic Web agents, for example, to process data produced by external entities. Consider the aforementioned example – based on an ontology, the user's agent can compare appointment slots with their calendar because they both represent temporal data concerning scheduling. Moreover, based on an ontology, software agents can *infer* additional knowledge not explicitly stated in the data, for example, an ontological rule may allow inferring that a specific insurance plan covers a certain procedure, because that procedure is defined in an inferior plan.

For many years, the Semantic Web had been primarily confined to the area of academic research. During this period, important standards and notions like RDF, SPARQL [3] (a query language for Semantic data[1]), or *Linked Data* [4] (a set of principles for publishing and interconnecting machine-readable data) were born. Over time, Semantic Web technologies[2] started to gain traction. One of the first and most successful practical applications of Semantic Web ontologies were medical terminologies like GALEN [5] or SNOMED CT [6], where the ability to uniquely identify and disambiguate concepts was exploited. Linked Data caught the attention of public institutions and governments which use them to provide open and standards-based access to their assets.[3] As global technology leaders like Facebook,[4] Google,[5] IBM [7], or Microsoft[6] have adopted Semantic Web principles in some of their mainstream products, the Semantic Web seems to be finally on track to becoming as ubiquitous as the original Web, or even more, given the predicted rise of the Internet of Things (appliances, sensors and regular hardware being connected to the Internet).

1.1 Motivation

Upon closer examination, one finds out that many contemporary use cases of the Semantic Web, including the examples presented so far, are centered around the ability to retrieve or query knowledge. Indeed, even the principles of the (Semantic) Web concern primarily the consumption of data and information.

In most cases, however, a concrete *information system* is responsible for data creation, processing, and visualization. The raw data may be created by users or gathered automatically (e.g., using sensors), but information systems provide the necessary plumbing and business logic operating on them. Businesses and other kinds of organizations or individuals rely daily on information systems to correctly process and manage their data. In fact, while data are considered the most valuable thing, it is the information systems operating on them that allow one to extract the value. Consider a fictive civil aviation authority (CAA) and its safety data collecting and processing system (SDCPS). Most users will use it to analyze trends in aviation safety using its business intelligence (BI) module, view safety occurrences and follow their investigation, or view results of safety audits in organizations. However, it is vital that it allows the CAA employees to enter data gathered during the occurrence

[1] Data expressed using Semantic Web standards – usually RDF(S) and OWL, serialized as RDF.

[2] I shall use the term *Semantic Web technologies* to group standards, notions, and technologies related to the Semantic Web. Relevant ones will be discussed in Chapter 2.

[3] For instance, the Czech Linked Open Data cloud `https://linked.opendata.cz`, the Austrian one `https://www.data.gv.at/linked-data`, both accessed 2020-08-11.

[4] `http://ogp.me`, 2020-08-11.

[5] `https://developers.google.com/knowledge-graph/`, accessed 2020-08-11.

[6] `https://concept.research.microsoft.com/Home/Introduction`, accessed 2020-08-11.

investigations or audits, and other involved parties to report occurrences. In other words, the system manipulates data in the whole create, retrieve, update, and delete (CRUD) spectrum.

Such information systems can be classified using various categories depending on various points of view. The one interesting for this thesis is the way they treat the underlying domain. In this regard, two main categories can be identified [8]:

1. Domain-independent information systems
2. Domain-specific information systems

Domain-independent systems make no assumptions about the underlying domain. Their approach is intentionally generic in order to accommodate various topics. The most prominent examples of such systems are data editors, including triple store or relational database management tools like GraphDB Workbench,[7] TopBraid Composer[8] and PgAdmin.[9] They are useful for direct data authoring and editing by experts on the selected data paradigm, however, their value for *domain experts*, i.e., users not necessarily possessing advanced IT knowledge, is limited.

Domain-specific systems, on the other hand, are built specifically for use in a particular domain. They contain a model of the domain and algorithms to operate on it (the *business logic*). Examples include the aforementioned safety data collecting and processing system, banking systems, systems used in offices, and many more. Such systems are more efficient for the domain experts, as they allow to encode the rules of the domain and the users can remain oblivious to the technologies used for data storage, exchange, etc. In a sense, a terminology editor like Protége [9] is both domain-independent and domain-specific. It is independent of the domain for which the terms are created, yet, the terms, their properties, and relationships are represented by a particular model. My interest in this work concentrates on domain-specific information systems, as they represent the vast majority of information systems out there and are more efficient for domain experts – their users.

The prevailing paradigm used to develop domain-specific information systems (and one may even dare to say information systems in general) is the *object-oriented programming* (OOP). OOP has been a dominant software development technique in the last two decades, mainly due to its ability to represent the underlying domains in a natural and understandable way [10]. The main distinguishing feature of OOP in

[7] `http://graphdb.ontotext.com/documentation/standard/workbench.html`, accessed 2020-08-11.

[8] `https://www.topquadrant.com/products/topbraid-composer/`, accessed 2020-08-11.

[9] `https://www.pgadmin.org/`, accessed 2020-08-11.

comparison to the previously popular *procedural programming* is the encapsulation of data and behavior into objects. Such objects are instances of classes, which represent a conceptualization of domain types. The classes can be structured into inheritance hierarchies, where each subclass inherits behavior and data from its superclass. *Object-oriented modeling* – modeling of an application developed using OOP – is in many aspects similar to *ontological modeling*, where a domain ontology is created. This should not come as a surprise, as both approaches aim at describing the structure of the domain using types, their instances, and relationships. Object-oriented modeling adds the behavior modeling aspect, which is then realized using OOP in the information system. For these reasons, when discussing domain-specific information systems in this thesis, I will always mean systems developed using object-oriented programming.

Domain-specific information systems may not appear directly relevant for the Semantic Web, but appearances are often deceiving. Consider the aforementioned fictive national civil aviation authority. The CAA is a public governmental body obliged to publish its data as open data. Its officials decide to develop an SDCPS for the oversight of aviation organizations under its jurisdiction, involving safety occurrence reporting, the definition of safety issues, and an aviation organization performance dashboard. First, the system is relevant for the Web, as it is likely to be a *Web application*, i.e., an application with a Web-based user interface. Given the ubiquity and performance of the Internet, most information systems are nowadays accessed via a Web browser. Next, building this system based on Semantic Web technologies has major advantages. The CAA has to deal with heterogeneous data involving international and local audits, safety occurrences reported by various parties (airlines, airports, civilians) in various formats. An ontological model can be used for the unification of such data, allowing the system to provide a more comprehensive picture of the situation in the domain. Global identifiers of types (like aircraft, organization), instances (concrete organizations), and attributes (label, a factor of an occurrence) allow the system to be interoperable with other compatible systems like an aircraft registry, a registry of aviation organizations, etc. Taxonomies of types of events that were a part of an occurrence (for instance, an aircraft first lost its speed and then collided with the ground) can be internationalized because different labels can represent the same event type. The ontological domain model can be shared, aiding to interoperability with other systems. Based on the model, the system can also infer knowledge not apparent from the data. For instance, if each report contains a tree of events involved in the occurrence, it is possible to infer what events frequently cause/contribute to other events. Last but not least, publishing the data as 5-star Linked (Open) Data [4] not only fulfills the CAA's obligation but enhances their reusability. A simplified diagram depicting the structure of such an SDCPS is shown in Figure 1.1.

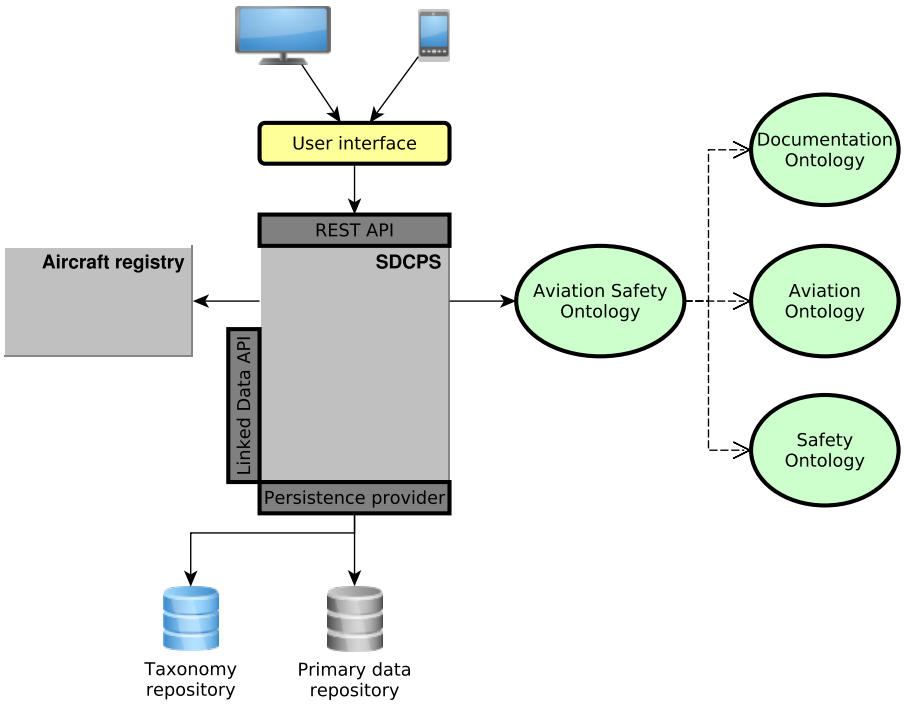


Figure 1.1: Simplified structure of a Semantic Web-based safety data collecting and processing system. Full edges represent dependencies, dashed edges represent ontology import.

1.2 Problem Statement

Despite the aforementioned benefits of using Semantic Web technologies when developing domain-specific information systems, their slow adoption in the area suggests that there are obstacles in the way. Arguably the most prominent are:

Learning curve Using Semantic Web incurs a burden of learning new non-trivial technologies. Developers who have already had to learn sophisticated programming languages and technologies must also learn a new paradigm with complex standards and data formats.

Software libraries Software development is nowadays connected with using middleware tools and auxiliary libraries that speed up the process. Yet, there is a clear lack of mature tools supporting Semantic Web technologies. A classical chicken and egg problem may be spotted here – application developers cannot rely on prototypical Semantic Web tools, and developers of such tools are not motivated to improve them due to their small user base.

Performance Contemporary Semantic storage systems have also a significant performance disadvantage compared to the more widespread relational databases [11].

Consider a software developer starting a new project – a domain-specific information system. If they wanted to build it upon Semantic Web technologies, they would have to learn at least basics of RDF, SPARQL, perhaps principles of Linked Data and JSON-LD (JavaScript Object Notation for Linked Data). In addition, they would often run into the problem of having to write a library for tasks for which there are many alternatives in the regular, relational database-based world (e.g., database access, serialization and deserialization of data in Web services), or, if they are lucky, accepting the risk of using a prototype developed by researchers funded on a per-project basis. Finally, they run into performance issues sooner than users of relational databases, who enjoy the spoils of decades of optimization and indexes over a static schema. Given the arguably unsure benefits of using Semantic Web technologies – shared schemas, global identifiers and machine-readable data formats vitally depend on other systems also supporting them – one cannot blame such a developer for choosing the safe path of a relational database-based system with regular REST (Representational State Transfer) Web services.

1.3 Thesis Goals

The ultimate goal of this thesis is to facilitate the adoption of Semantic Web technologies into domain-specific information systems development by building a stack of principles and tools for efficient software development. Such a stack should (at least partially) solve the issues of the learning curve and the lack of tools discussed above by allowing software developers to use techniques and approaches they know and keeping the Semantic nature of the stack transparent in the majority of cases. While not being able to address the performance issues directly, the tools developed as part of this thesis provide comparable or better performance than existing alternatives (when they exist).

A literature review [12, 11, 13] and my own experience with developing domain-specific Semantic Web-based information systems (many authors also use the term *Domain-specific Linked Data Applications* [12, 11]) suggest two areas where utilizing Semantic Web technologies in development may be problematic:

- Application access to Semantic data
- Integration of application interfaces using Semantic Web technologies

Thus, these two areas will be of particular interest to me. It is suggestive that they concern the boundaries of an information system – be it data access, integration

with other systems or user interface – while the core – the business logic – remains virtually oblivious to the choice of Semantic Web technologies (consider the SDCPS in Figure 1.1, where Semantic Web-related components are marked with bold borders). While it would be in accordance with software architecture best practices, as discussed, for instance, in [14], it remains one of the goals of this thesis to confirm or refute this idea.

The particular goals of this thesis can be thus enumerated as follows:

1. Design a framework for formally sound application access to Semantic data and provide its implementation.
2. Identify and classify the issues of building and integration of Web services based on Semantic Web technologies. Provide a showcase implementation of such integration.
3. Analyze how the decision to build an information system based on Semantic Web technologies influences its architecture and design. Provide guidelines for developing such systems.

Fulfilling these goals should help expand the rather small niche of domain-specific Semantic Web-based information systems by simplifying the use of Semantic Web technologies by common software development folk.

The rest of the thesis is structured as follows: Chapter 2 provides the necessary theoretical background, Chapter 3 discusses the current state of the art and works related to my approach. The main contributions of the thesis are presented in the following two chapters – Chapter 4 provides an overview of Semantic data access solutions and builds a formal framework for this access, whereas Chapter 5 introduces its implementation, together with another library created as part of the intended development stack – the Web service integration tool. Chapter 6 provides an evaluation of the work – it shows the faithfulness of the developed data access library to its formal underpinnings, presents several showcases of real-world domain-specific information systems based on Semantic Web technologies, and discusses the architecture of such systems. The thesis is concluded in Chapter 7.

Chapter 2

Background

This chapter provides the necessary background on standards, languages, and technologies used in this thesis. It starts with the fundamental standard of the Semantic Web – the data description language RDF. The principles of Linked Data are discussed next, as they are closely related to RDF and important for Semantic Web-based information system integration. SPARQL and SPARQL Update are presented as languages for querying and manipulation of Semantic (Linked) data respectively. Moving on, the notion of ontologies is discussed, followed by sections presenting standard languages for their description – RDFS and OWL. Special care is devoted to the formal logics upon which OWL is based – description logics – more specifically, the logic $\mathcal{SROIQ}(D)$. Next up is another idea important for information systems – integrity constraints and their utilization in description logics. Finally, F-logic is introduced as a language this thesis uses to bridge the gap between object-oriented programming languages and description logics.

2.1 RDF

The fundamental standard of the Semantic Web is the Resource Description Framework (RDF) [2]. It is a data modeling language built upon the notion of *statements* about *resources*. These statements consist of three parts, the *subject* of the description, the *predicate* describing the subject, and the *object*, i.e., the predicate value. Such statements – *triples* – represent elements of a labeled directed graph, an *RDF graph*, where the nodes are resources (IRIs or blank nodes[1] in the roles of subjects or

[1] Blank nodes are resources without global identity.

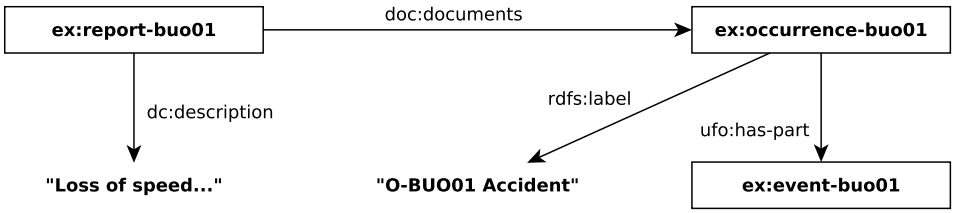


Figure 2.1: Simple visualization of an RDF graph using a labeled directed graph. XML-based namespace prefixes are used to shorted IRIs. Borderless nodes represent literal values while nodes with border are resources.

objects) or literal values (in the role of objects) and the edges are properties (in the role of predicates) connecting them. This can be seen in a simple visualization in Figure 2.1. Note that properties are also resources, so they can appear as nodes in an RDF graph as well. RDF can be serialized in many formats, including RDF/XML, Turtle, or N-triples. Listing 2.1[2] complements Figure 2.1 with a Turtle serialization of the same data. Taking an RDF graph and a set of *named graphs* (RDF graphs identified by IRIs), we get an *RDF dataset*.

Listing 2.1: An example of a small RDF dataset based on Figure 2.1 written in Turtle.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc: <http://purl.org/dc/terms/> .
@prefix doc: <http://onto.fel.cvut.cz/ontologies/documentation/> .
@prefix ufo: <http://onto.fel.cvut.cz/ontologies/ufo/> .
@prefix ex: <http://example.com/> .

ex:report-buo01 doc:documents ex:occurrence-buo01 ;
                dc:description "Loss of speed..." .
ex:occurrence-buo01 rdfs:label "O-BUO01 Accident" ;
                    ufo:has-part ex:event-buo01 .
```

2.2 Linked Data

The term Linked Data denotes a set of practices for publishing and connecting structured data on the Web using W3C standards [4]. This way, the Web could be turned into a massive, distributed database – the Web of Data. Such a database allows one to gather data from various sources, traverse related topics by following links between datasets, or reuse data schemas, facilitating system interoperability.

There are multiple levels of adherence to the Linked Data principles, denoted, in an analogy to ranking systems, by the number of stars a data publisher achieves.

[2] Prefixes declared in listings are assumed to exist in subsequent listings as well.

- ★ Data are available on the Web, in whatever format.
- ★★ Data are available as machine-readable structured data.
- ★★★ Data are available in a non-proprietary format.
- ★★★★ Data are published using W3C open data formats.
- ★★★★★ All of the above plus data are linked to other people's data.

The connection to RDF can be seen in 4-star Linked Data, as RDF is the primary W3C format for publishing machine-readable data.

If, in addition, the data are published under an open license, they become *Linked Open Data*.

2.3 SPARQL and SPARQL Update

The SPARQL Query Language (SPARQL) is a query language for RDF [3]. It is based upon the matching of triple patterns to the data. A *triple pattern* is essentially a triple, where variables can appear in the position of subject, predicate, and object. A query engine then attempts to replace the variables with existing values and check whether the underlying database contains such a triple. A set of triple patterns comprises a *basic graph pattern* – the base of most SPARQL queries. In addition, SPARQL supports various constructs like OPTIONAL for optional matching, FILTER for value filtering, ORDER BY for result ordering etc.

A short example of a SPARQL query selecting all distinct properties whose subject or object is `ex:report`, ordered lexicographically by their identifiers, can be found in Listing 2.2.

Listing 2.2: A SPARQL query example.

```
SELECT DISTINCT ?outgoing ?incoming WHERE {
   { ex:report ?outgoing ?object . }
   UNION
   { ?subject ?incoming ex:report . }
} ORDER BY ?outgoing ?incoming
```

SPARQL Update is, on the other hand, a data manipulation language for RDF [15]. Its syntax is derived from SPARQL and allows creating, updating, and deleting RDF data. The query in Listing 2.3 showcases how data (a type declaration in this case) can be inserted into the repository using SPARQL Update.

Listing 2.3: A SPARQL Update example.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

INSERT DATA {
   ex:report rdf:type doc:occurrence_report .
}
```

2.4 Ontologies

The expressive power of RDF is rather small – it allows one to describe data, but it is not possible to describe the structure of the data – their schema. That is where ontologies come into play. The term *Ontology* originates from philosophy where it is a discipline studying the nature and structure of reality. An ontology in the computer science sense is a computational artifact. The most widely cited definition of an ontology in computer science comes from Tom Gruber, who defines an ontology as "an explicit specification of conceptualization [16]." This somewhat cryptic definition can be informally translated as a written (or otherwise expressed) specification of a conceptual model of a particular domain. Guarino et al. provide a detailed definition of an ontology which can be summed up as follows: "An ontology is a formal, explicit specification of a shared conceptualization [17]." Where formal and explicit corresponds to a specification expressed in a formal language, usually machine-readable, with well-defined intensional semantics, i.e., describing the concepts and relationships using axiomatic rules rather than enumerating their extension (instances) in all possible worlds. Shared conceptualization then indicates that a mental model of an area of interest, again, defined in an intensional way, is *agreed on* by multiple agents.

The gist for the purpose of this thesis is that an ontology is a computational artifact describing the conceptualization (mental model) of a domain, defining its concepts and their properties. Ontologies in computer science are thus defined using structured languages, so that they may be processed by machines. These languages will be discussed next.

2.5 RDFS

RDF Schema (RDFS) is a data modeling language for RDF [18]. It allows one to declare classes (domain concepts), build hierarchies of both classes and properties, specify ranges and domains of properties, etc. Besides, the RDF Schema *reification*

vocabulary provides a way to specify provenance metadata of the data themselves. E.g., one can declare who and when created a particular triple/resource.

The expressiveness of RDFS is still low and it does not support, for example, declaration of equivalence of two RDFS classes, resources. Yet, RDFS is expressive enough for a *reasoner* to infer certain implicit facts from explicit data, which is one of the big benefits of Semantic Web ontologies. Consider the following triples:[3]

```
doc:occurrence_report rdfs:subClassOf doc:report .

ex:a rdf:type doc:occurrence_report .
```

The first triple is a part of the schema and says that `doc:occurrence_report`[4] is a subclass of `doc:report`. The second triple describes actual data. Given RDFS interpretation rules, an RDFS-aware reasoner can infer that

```
ex:a rdf:type doc:report .
```

2.6 OWL

The Web Ontology Language (OWL) is an ontology language for the Semantic Web [19]. It provides, besides basic modeling constructs like classes and properties, more expressive features, such as inverse properties, equivalent classes or properties, intersection or union of classes, etc. OWL 2 is the latest version of this W3C-standardized language.[5] The high expressiveness of OWL comes with a cost of increased computational complexity. Indeed, depending on the chosen semantics (see Section 2.6.1 below), reasoning in OWL is undecidable or N2EXPTIME-complete.[6] Thus, several sub-languages, called *profiles*, were introduced to provide sets of constructs with more favorable computational properties, suitable for particular often appearing problems identified by the community. Examples of OWL profiles are OWL 2 RL (allows rule-based reasoning), OWL 2 QL (supports sound, and complete query answering with LOGSPACE complexity and was specifically designed for ontology-based data access [20]), OWL 2 EL (a restricted profile primarily for classification hierarchies and simple constraints), etc.

OWL ontologies are sets of structural statements which are, due to its logic roots,

[3] Turtle RDF serialization is used throughout the thesis.

[4] I will use the `typewriter font` to denote ontological expressions in text throughout the thesis.

[5] For simplicity, OWL will be used to denote both the original OWL standard as well as its successor OWL 2. I will point out the difference if necessary.

[6] See `https://www.w3.org/TR/owl2-profiles/#Computational_Properties`, accessed 2020-08-11.

traditionally called *axioms*, and statements about the data called *assertions*. Axioms are used, on the one hand, to build hierarchies of classes and properties, often by placing restrictions on other classes and properties, and, on the other hand, to specify various characteristics of the properties, e.g., their reflexivity, or transitivity. OWL ontologies are typically serialized as RDF, however, since OWL is a more expressive, higher-level language, writing an OWL axiom may require several RDF triples. Consider the following axiom declaring class `Parent` to be equivalent to a union of classes `Father` and `Mother` (written in the functional-style syntax used by the OWL specification):

```
EquivalentClasses(ex:Parent ObjectUnionOf(ex:Father ex:Mother))
```

Such an axiom is serialized in RDF (Turtle) as follows:

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .

ex:Parent rdf:type owl:Class ;
       owl:equivalentClass [ rdf:type owl:Class ;
                             owl:unionOf (
                                ex:Father
                                ex:Mother
                             )
                           ] .
```

OWL splits the domain into the sets of classes, properties, individuals, and literals. In contrast to RDFS, properties in OWL can be further divided into:

- *object properties*, which represent relationships between two individuals,
- *data properties*, which have literal values as objects,
- *annotation properties*, which can have either an individual or a literal as a value, but they do not partake in reasoning and are mainly used to provide metadata.

Such a division represents yet another data modeling feature built into the language itself.

2.6.1 OWL Semantics

The semantics of OWL is defined in two variants:

RDF-based Semantics RDF-based semantics of OWL is fully compatible with the semantics of RDF [21]. That is, an OWL ontology is viewed as an RDF

graph. RDF-based Semantics is a superset of the Direct Semantics (see below) and reasoning in it is, in fact, *undecidable*. One of the key reasons for such a conclusion is that under RDF-based semantics, individuals comprise the whole domain, i.e., classes are also individuals (resources in the RDF vernacular). An OWL ontology utilizing the RDF-based semantics is considered to use the OWL 2 Full profile.

Direct Semantics Direct semantics of OWL is based on the formalism of *description logics* (DLs) [22]. Description logics are a set of decidable sub-languages of the first-order logic. Thus, inference under the direct semantics is also decidable, because restrictions are placed on the language usage (e.g., classes and individuals are disjoint under direct semantics[7]). The most expressive OWL profile based on direct semantics is the OWL 2 DL (a successor of OWL DL) which is based on the description logic $\mathcal{SROIQ}(D)$.

Given the undecidability of RDF-based semantics, direct semantics is of more relevance for most applications. It is also closer to the logic roots of OWL, which stem from the DAML+OIL language [23]. Since the formalism for Semantic data access developed in this thesis is based on logics, direct semantics will be meant whenever OWL semantics is discussed in the text, unless explicitly specified otherwise. Description logics are paramount to OWL, it is thus time to introduce the description logic $\mathcal{SROIQ}(D)$ – the formalism underpinning OWL 2 DL.

2.7 $\mathcal{SROIQ}(D)$

$\mathcal{SROIQ}$ [24] is an expressive description logic, i.e., a decidable sub-language of the first-order logic (FOL), used to describe ontologies. $\mathcal{SROIQ}(D)$ is then the description logic $\mathcal{SROIQ}$ extended with *datatypes* – a definition of literal value types.

2.7.1 $\mathcal{SROIQ}(D)$ Syntax

Each $\mathcal{SROIQ}(D)$ ontology $\mathcal{O}$ is comprised of a *terminology* (TBox and RBox), which describes the schema of the ontology, and a set of individual assertions representing actual data (ABox).[8] A TBox consists of a *concept hierarchy* where concepts can be

[7] Although, *punning* in OWL 2 DL allows to syntactically treat them as the same to some extent.

[8] $\mathcal{SROIQ}$ allows expressing individual assertions using TBox axioms with nominals. However, ABox assertions provide a natural, easy to read syntax which I will use throughout this thesis.

either *atomic* or *concept descriptions* of the following forms:

$$C \leftarrow \neg C,\ C \sqcap D,\ C \sqcup D,\ \geqslant nR.C,\ \leqslant nR.C,\ \exists R.Self,\ \{a\},\ \forall R.C,\ \exists R.C,$$
$$\geqslant nT.d,\ \leqslant nT.d,\ \forall T.d,\ \exists T.d,$$

where C, D are concepts, R, T are roles (T is called *concrete*), n is a non-negative integer, d is a datatype and a is an individual. Each datatype d belongs to a set of datatypes $\mathbf{D}$ and is associated with a set $d^D \subseteq \Delta_D$ of concrete values from the concrete domain Δ_D [25, 26]. The concept hierarchy is built using *general concept inclusion* (GCI) axioms of the form $C \sqsubseteq D$. An RBox consists of a hierarchy of roles (built using *role inclusion* axioms (RIA) of the form $R \sqsubseteq S$) and axioms stating their properties: $Sym(R)$, $Asy(R)$, $Tra(R)$, $Ref(R)$, $Irr(R)$, and $Dis(R, S)$. The schema also contains built-in concepts $\top$, $\bot$ and a built-in universal role R_U.

Individual assertions are of the form $C(a)$, $R(a, b)$, $T(a, v)$, $a = b$ and $a \neq b$, where a and b are individuals, v is a concrete literal, C is a concept, R is a role, and T is a concrete role. The set N_C represents concept names, N_R role names, N_I denotes the set of individual names, and N_D denotes the union of the sets of constants assigned to each datatype in $\mathbf{D}$ (datatype literal values) [26].

2.7.2 $\mathcal{SROIQ}(D)$ Semantics

The semantics of a $\mathcal{SROIQ}(D)$ ontology $\mathcal{O}$ is given by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain* of the interpretation and $\cdot^{\mathcal{I}}$ is the *interpretation function*. $\Delta^{\mathcal{I}}$ is disjoint from Δ_D. The interpretation function assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to every individual an element of $\Delta^{\mathcal{I}}$. $\top^{\mathcal{I}}$ is $\Delta^{\mathcal{I}}$, $\bot^{\mathcal{I}}$ is the empty set $\emptyset$ and $R_U^{\mathcal{I}}$ is

$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Concept descriptions are interpreted as follows:

$$
\begin{aligned}
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\geqslant nR.C)^{\mathcal{I}} &= \{x \mid \#\{y \,.\, \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\} \\
(\leqslant nR.C)^{\mathcal{I}} &= \{x \mid \#\{y \,.\, \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\} \\
(\exists R.Self)^{\mathcal{I}} &= \{x \mid \langle x, x\rangle \in R^{\mathcal{I}}\} \\
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y \,.\, \langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y \,.\, \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \\
(\geqslant nT.d)^{\mathcal{I}} &= \{x \mid \#\{v \,.\, \langle x, v\rangle \in T^{\mathcal{I}} \text{ and } v \in d^{\mathcal{D}}\} \geqslant n\} \\
(\leqslant nT.d)^{\mathcal{I}} &= \{x \mid \#\{v \,.\, \langle x, v\rangle \in T^{\mathcal{I}} \text{ and } v \in d^{\mathcal{D}}\} \leqslant n\} \\
(\forall T.d)^{\mathcal{I}} &= \{x \mid \forall v \,.\, \langle x, v\rangle \in T^{\mathcal{I}} \text{ implies } v \in d^{\mathcal{D}}\} \\
(\exists T.d)^{\mathcal{I}} &= \{x \mid \exists v \,.\, \langle x, v\rangle \in T^{\mathcal{I}} \text{ and } v \in d^{\mathcal{D}}\}
\end{aligned}
$$

Where C and D are concepts, R is a role, T is a concrete role, a is an individual, d is a datatype, n is a non-negative integer and $\#M$ denotes the cardinality of a set M. Terminological axioms are interpreted as follows:

$$
\begin{aligned}
&\mathcal{I} \models C \sqsubseteq D \text{ if } C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
&\mathcal{I} \models R \sqsubseteq S \text{ if } R^{\mathcal{I}} \subseteq S^{\mathcal{I}} \\
&\mathcal{I} \models Sym(R) \text{ if } \langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } \langle y, x\rangle \in R^{\mathcal{I}} \\
&\mathcal{I} \models Asy(R) \text{ if } \langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } \langle y, x\rangle \notin R^{\mathcal{I}} \\
&\mathcal{I} \models Tra(R) \text{ if } \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } \langle y, z\rangle \in R^{\mathcal{I}} \text{ imply } \langle x, z\rangle \in R^{\mathcal{I}} \\
&\mathcal{I} \models Ref(R) \text{ if } \forall x \in \Delta^{\mathcal{I}} \,.\, \langle x, x\rangle \in R^{\mathcal{I}} \\
&\mathcal{I} \models Irr(R) \text{ if } \forall x \in \Delta^{\mathcal{I}} \,.\, \langle x, x\rangle \notin R^{\mathcal{I}} \\
&\mathcal{I} \models Dis(R, S) \text{ if } R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset
\end{aligned}
$$

Where C and D are concepts, R and S are roles, and $\emptyset$ denotes the empty set. Finally, ABox assertions are interpreted as follows:

$$
\begin{aligned}
&\mathcal{I} \models C(a) \text{ if } a^{\mathcal{I}} \in C^{\mathcal{I}} \\
&\mathcal{I} \models R(a, b) \text{ if } \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}} \\
&\mathcal{I} \models T(a, v) \text{ if } \langle a^{\mathcal{I}}, v^{\mathcal{D}}\rangle \in T^{\mathcal{I}} \\
&\mathcal{I} \models a = b \text{ if } a^{\mathcal{I}} = b^{\mathcal{I}} \\
&\mathcal{I} \models a \neq b \text{ if } a^{\mathcal{I}} \neq b^{\mathcal{I}}
\end{aligned}
$$

Where C is a concept, R is a role, T is a concrete role, a and b are individuals, and v is a concrete literal.

$\mathcal{I}$ is a model of an ontology $\mathcal{O}$ consisting of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$, and an ABox $\mathcal{A}$ ($\mathcal{I} \models \mathcal{O} = \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}$) if it satisfies all the axioms in $\mathcal{O}$. A set of axioms Θ *logically entails* an axiom θ ($\Theta \models \theta$) if and only if all models of Θ are also models of θ.

Example. Ontology $\mathcal{O}$ below represents a simple $\mathcal{SROIQ}$ knowledge base. In it, each `Report` has exactly one author and documents exactly one object. The class `OccurrenceReport` is then a subclass of `Report`. Finally, some instance data are declared.

$$\begin{aligned}\mathcal{O} = \{&OccurrenceReport \sqsubseteq Report, Report \sqsubseteq= 1 hasAuthor.\top,\\ &Report \sqsubseteq= 1 documents.\top,\\ &OccurrenceReport(report{-}buo01), Occurrence(occurrence{-}buo01),\\ &User(ThomasLasky), hasAuthor(report{-}buo01, ThomasLasky),\\ &documents(report{-}buo01, occurrence{-}buo01)\}\end{aligned}$$

2.8 Integrity Constraints

Description logics adopt the so-called *open world assumption* (OWA) – if a fact cannot be proven from the data, it is not necessarily false, it is just unknown and may be true in some models. This is in conformance with the distributed nature of the Semantic Web, where a fact may be stated in a dataset somewhere on the Web and not reachable from the data currently available to the reasoner. Nevertheless, most domain-specific information systems operate under the *closed world assumption*, that is, if a fact cannot be proven, it is treated as false. Consider the example CAA safety management system mentioned in Chapter 1: the system may have a condition that for an occurrence of type Runway incursion, the perpetrating aircraft has to be specified. Such a constraint would look as follows in $\mathcal{SROIQ}$: $RunwayIncursion \sqsubseteq \exists hasParticipant.Aircraft$. Now, if there is an instance of Runway incursion without the associated aircraft: $RunwayIncursion(a)$, a reasoner will, because of OWA, find a model of such an ontology by introducing a fresh individual o and connecting it with a: $hasParticipant(a, o)$. However, this behavior is hardly what the designers of the system intended – they wanted the user who classifies an occurrence as a Runway incursion to fill in the perpetrator so that the quality of the data does no suffer. OWA allows DLs to remain *monotonic* – new facts added to the knowledge base do not decrease the number of possible inferences. But, for most domain-specific information systems, CWA is more suitable.

A way to resolve the issue is to introduce *nonmonotonic* behavior to the knowledge base, so that new axioms may, in fact, decrease the number of inferences, e.g., by restricting possible models to only those which contain a known value for some property. Such restrictions are known as *integrity constraints* (ICs).

2.8.1 Integrity Constraints in Description Logics

As mentioned above, the ability of description logics to express integrity constraints is, due to their adoption of OWA, limited. But since integrity constraints are such an important data modeling concept, several approaches to augmenting DL knowledge bases with ICs have been developed over the years. This section introduces in detail the approach used in this thesis. Alternative solutions will be discussed in Chapter 3.

Tao et al. [27] propose to build *extended* knowledge bases, where, in addition to axioms with standard semantics, a set of axioms is interpreted with constraint semantics. This solution uses the same syntax for both regular and integrity constraint axioms and supports the description logic $\mathcal{SROIQ}$ (its extension to $\mathcal{SROIQ}(D)$ would be trivial).

Integrity Constraints Semantics

IC semantics is built around an *IC-interpretation* which is defined as a pair $\mathcal{I}, \mathcal{U}$, where $\mathcal{I}$ is a $\mathcal{SROIQ}$ interpretation defined over $\Delta^{\mathcal{I}}$ and $\mathcal{U}$ is a set of $\mathcal{SROIQ}$ interpretations. The IC-interpretation function $\cdot^{\mathcal{I},\mathcal{U}}$ then maps concepts to subsets of $\Delta^{\mathcal{I}}$ and roles to subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ as follows:

$$C^{\mathcal{I},\mathcal{U}} = \{x^{\mathcal{I}} \mid x \in N_I \text{ s.t. } \forall \mathcal{J} \in \mathcal{U}, x^{\mathcal{J}} \in C^{\mathcal{J}}\}$$
$$R^{\mathcal{I},\mathcal{U}} = \{\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \mid x, y \in N_I \text{ s.t. } \forall \mathcal{J} \in \mathcal{U}, \langle x^{\mathcal{J}}, y^{\mathcal{J}} \rangle \in R^{\mathcal{J}}\}$$

Where C is an atomic concept and R is a role. Such interpretation extends to inverse roles and complex concept descriptions as one would expect, for example:

$$(R^{-})^{\mathcal{I},\mathcal{U}} = \{\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \mid \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in R^{\mathcal{I},\mathcal{U}}\}$$
$$(C \sqcap D)^{\mathcal{I},\mathcal{U}} = C^{\mathcal{I},\mathcal{U}} \cap D^{\mathcal{I},\mathcal{U}}$$
$$(\neg C)^{\mathcal{I},\mathcal{U}} = N_I \setminus C^{\mathcal{I},\mathcal{U}}$$

Basically, $C^{\mathcal{I},\mathcal{U}}$ is the interpretation of all named individuals which are instances of C in all interpretations from $\mathcal{U}$. Such semantics clearly adopts the closed world view. For example, if a named individual cannot be proved to be an instance of C in all interpretations, it is considered to be an instance of $\neg C$.

Axiom satisfaction under the IC semantics is analogous to regular $\mathcal{SROIQ}$ axiom satisfaction, but using the IC-interpretation. For example:

$$\mathcal{I},\mathcal{U} \models C \sqsubseteq D \text{ iff } C^{\mathcal{I},\mathcal{U}} \subseteq D^{\mathcal{I},\mathcal{U}}$$
$$\mathcal{I},\mathcal{U} \models R \sqsubseteq S \text{ iff } R^{\mathcal{I},\mathcal{U}} \subseteq S^{\mathcal{I},\mathcal{U}}$$
$$\mathcal{I},\mathcal{U} \models Ref(R) \text{ iff } \forall x \in N_I, \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$$

Where C, D are concepts and R, S are roles.

IC semantics neatly solves the problem discussed at be beginning of this section – considering $RunwayIncursion \sqsubseteq \exists hasParticipant.Aircraft$ as an IC axiom, $RunwayIncursion(a)$ alone will now be inconsistent because there is no known aircraft that would be a participant of the incursion.

Regular $\mathcal{SROIQ}$ semantics lacks one more notion relevant for the declaration of integrity constraints – *unique name assumption* (UNA). Without UNA, any two individuals may be considered the same in a model, unless explicitly declared otherwise. Again, such an approach is logical in the world of the Semantic Web, where multiple resources may easily represent the same thing. However, in a domain-specific information system, the lack of UNA may go against the logic of the domain model, which assumes (locally) complete knowledge. IC semantics by Tao et al. adopts the notion of *weak UNA*, where two individuals are considered different unless their equality is required to satisfy the KB axioms. Weak UNA is based on *Minimal Equality* models. Informally, a model is a minimal equality model if there is no other model with fewer equality relations between named individuals. A set of such models is denoted by $Mod_{ME}(\mathcal{K})$, where $\mathcal{K}$ is a knowledge base. I do not introduce ME models here formally, as an analogous notion will be formally defined in Chapter 4. Nevertheless, a curious reader may refer to [27].

Now, with the IC semantics under our belt, given a $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$ and a $\mathcal{SROIQ}$ integrity constraint α, the IC-satisfaction of α by $\mathcal{K}$ is defined as follows:

$$\mathcal{K} \models_{IC} \alpha \text{ iff } \forall \mathcal{I} \in \mathcal{U},\ \mathcal{I},\mathcal{U} \models \alpha, \text{where } \mathcal{U} = Mod_{ME}(\mathcal{K})$$

An extended knowledge base is then a pair $\langle \mathcal{K}, \mathcal{C} \rangle$, where $\mathcal{K}$ is a $\mathcal{SROIQ}$ KB interpreted with standard semantics and $\mathcal{C}$ is a set of $\mathcal{SROIQ}$ axioms interpreted with IC semantics. $\langle \mathcal{K}, \mathcal{C} \rangle$ is valid if $\forall \alpha \in \mathcal{C}, \mathcal{K} \models_{IC} \alpha$. If $\langle \mathcal{K}, \mathcal{C} \rangle$ is not valid, there is an integrity constraint violation.

Example. The following set of integrity constraints can be used to constrain ontology $\mathcal{O}$ from Section 2.7 in that occurrence reports document only occurrences

and their authors must be instances of type `User`:

$$\begin{aligned}\mathcal{IC} = \{&OccurrenceReport \sqsubseteq \forall documents.Occurrence,\\ &OccurrenceReport \sqsubseteq \forall hasAuthor.User\}\end{aligned}$$

Integrity Constraints Validation

Tao et al. propose to validate the ICs by their translation to *distinguished conjunctive queries* (DCQ) with the *Negation as Failure* (NAF) operator **not** (DCQ$^{\mathbf{not}}$). I will first introduce DCQ$^{\mathbf{not}}$ and then present the IC translation rules.

Let N_V be a non-empty set of variable names disjoint from N_C, N_R and N_I. A *query atom* is an ABox axiom where variables can be used in the place of individuals, i.e.,

$$q \leftarrow C(x) \mid R(x,y) \mid \neg R(x,y) \mid x = y \mid x \neq y$$

Where $x, y \in N_I \cup N_V$, C is a concept, and R is a role. A *conjunctive query* is a conjunction of query atoms, i.e.,

$$Q \leftarrow q \mid Q_1 \wedge Q_2$$

A distinguished conjunctive query can contain only distinguished variables – variables mapped to elements of N_I. The semantics of DCQ is given by regular $\mathcal{SROIQ}$ interpretations. An *assignment* σ is defined as a mapping from the set of variables used in the query into the KB's named individuals, i.e., $\sigma : N_V \rightarrow N_I$. $\sigma(Q)$ denotes the application of σ to the query Q. A knowledge base $\mathcal{K}$ entails a DCQ Q with an assignment σ ($\mathcal{K} \models^{\sigma} Q$) if:

$$\begin{aligned}&\mathcal{K} \models^{\sigma} q \text{ iff } \mathcal{K} \models \sigma(q)\\ &\mathcal{K} \models^{\sigma} Q_1 \wedge Q_2 \text{ iff } \mathcal{K} \models \sigma(Q_1) \text{ and } \mathcal{K} \models \sigma(Q_2)\end{aligned}$$

The set of *answers* to a query is a set of assignments for which the knowledge base entails the query. A query is true w.r.t. a KB $\mathcal{K}$ if the set of its answers is not empty.

Now, a DCQ$^{\mathbf{not}}$ query is defined as follows:

$$Q \leftarrow q \mid Q_1 \wedge Q_2 \mid \mathbf{not} Q$$

With the semantics of **not** being:

$$\mathcal{K} \models^{\sigma} \mathbf{not} Q \text{ iff } \nexists \sigma' \text{ s.t. } \mathcal{K} \models^{\sigma'} \sigma'(Q)$$

Translation Rules from IC Axioms to DCQ$^{\mathbf{not}}$. The translation from IC axioms to DCQ$^{\mathbf{not}}$ consists of two operators – $\mathcal{T}_C$ for concepts and $\mathcal{T}$ for axioms. $\mathcal{T}_C$ takes a concept expression and a variable as an input and returns a DCQ$^{\mathbf{not}}$ as a result according to the following rules:

$$\begin{aligned}
\mathcal{T}_C(C_a, x) &= C_a(x) \\
\mathcal{T}_C(\neg C, x) &= \mathbf{not}\mathcal{T}_C(C, x) \\
\mathcal{T}_C(C \sqcap D, x) &= \mathcal{T}_C(C, x) \wedge \mathcal{T}_C(D, x) \\
\mathcal{T}_C(\geqslant nR.C, x) &= \bigwedge_{1 \leq i \leq n} (R(x, y_i) \wedge \mathcal{T}_C(C, y_i)) \bigwedge_{1 \leq i < j \leq n} \mathbf{not}(y_i = y_j) \\
\mathcal{T}_C(\exists R.Self, x) &= R(x, x) \\
\mathcal{T}_C(\{a\}, x) &= (x = a) \\
\mathcal{T}_C(\forall R.C, x) &= \mathbf{not}(R(x, y) \wedge \mathbf{not}\mathcal{T}_C(C, y)) \\
\mathcal{T}_C(\exists R.C, x) &= R(x, y) \wedge \mathcal{T}_C(C, y)
\end{aligned}$$

Where C_a is an atomic concept, C and D are concepts, R is a role, a is an individual, x is a variable, and y_i is a fresh variable. $\mathcal{T}$ is a function that maps a $\mathcal{SROIQ}$ IC axiom to a DCQ$^{\mathbf{not}}$, for example:

$$\begin{aligned}
\mathcal{T}(C \sqsubseteq D) &= \mathcal{T}_C(C, x) \wedge \mathbf{not}\mathcal{T}_C(D, x) \\
\mathcal{T}(R \sqsubseteq S) &= R(x, y) \wedge \mathbf{not}S(x, y) \\
\mathcal{T}(Ref(R)) &= \mathbf{not}R(x, x)
\end{aligned}$$

Where C and D are concepts, R and S are roles, and x and y are variables. Intuitively, $\mathcal{T}$ switches the polarity of the axiom so that it is not satisfied if the query is true.

The complete list of translation rules for $\mathcal{T}_C$ and $\mathcal{T}$, together with a proof of the translation faithfulness and restrictions on the extended knowledge base expressiveness, can be found in [27] or in Jiao Tao's PhD thesis [28].

Example. The set of integrity constraints $\mathcal{IC}$ introduced above can be translated using operators $\mathcal{T}_C$ and $\mathcal{T}$ into the following queries:

$$\begin{aligned}
&OccurrenceReport(x) \wedge documents(x, y) \wedge \mathbf{not}(Occurrence(y)) \\
&OccurrenceReport(x) \wedge hasAuthor(x, y) \wedge \mathbf{not}(User(y))
\end{aligned}$$

If either query yields results over a knowledge base, it indicates that the corresponding integrity constraint is violated.

2.9 F-logic

F-logic [29, 30, 31] is a formalism rooted in FOL that can be used to describe structural aspects of object-oriented or frame-based languages. It has model-theoretic semantics and a sound and complete proof theory. In the discussion of F-logic syntax, the revised version of [31] is used and w.l.o.g. the distinction between inheritable and non-inheritable methods is omitted. F-logic allows one to specify relatively complex programs. However, for the purpose of this thesis, where it is used to represent object-oriented languages in the formalization of mapping between DL ontologies and object models, this complexity is unnecessary. Therefore a restricted variant of F-logic is introduced – it does not contain, for instance, methods with arbitrary arity (only *attributes* – parameterless methods are used). Also, to provide a close match to DLs (and $\mathcal{SROIQ}$ in particular), *sorted* F-logic is used, so that (atomic) classes are disjoint from individuals and methods.

F-logic Syntax

The alphabet of an F-logic language $\mathcal{L}$ consists of:

- A set of *object constructors* $\mathcal{F} = \mathcal{C} \cup \mathcal{R} \cup \mathcal{E} \cup \mathcal{A}$, where $\mathcal{C}$ is a set of class names (0-ary function symbols), $\mathcal{R}$ is a set of methods (0-ary function symbols), $\mathcal{E}$ is a set of instances (0-ary function symbols), and $\mathcal{A}$ is a set of function symbols (it essentially allows to *parameterize* concept constructors, as will be seen in Section 4.2). $\mathcal{C}$, $\mathcal{R}$, $\mathcal{E}$, and $\mathcal{A}$ are mutually disjoint,
- a set of predicate symbols $\mathcal{P}$,
- an infinite set of variables $\mathcal{V}$,
- auxiliary symbols like (,), [,], $\rightarrow$, etc.,
- logical connectives and quantifiers – $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$.

An *id-term* is a first-order term composed of an object constructor and variables. A variable-free object constructor is called a *ground id-term* and the set of all ground id-terms is denoted $U(\mathcal{F})$. Formulas in F-logic can be either *molecular formulas* (F-molecules or just molecules), or complex formulas consisting of other formulas connected using logical connectives and quantifiers. Molecular formulas can be:

1. *Is-a* assertions of the form $A :: B$ or $o : A$, where o, A, B are id-terms,

2. *Object molecules* of the form O[a ';' separated list of method expressions]. Where method expressions can be:
   - *data expressions* of the form $m \rightarrow v$, where m and v are id-terms (v is the attribute value),
   - *signature expressions* of the form $m \Rightarrow (T_1, ..., T_n)$, where $n \geq 1$ and m and T_i are id-terms (T_i are the return types).

In short, data expressions represent attribute values, whereas signature expressions represent their return types (data expressions correspond to DL (concrete) role assertions and signature expressions to local domain/range restrictions). Listing 2.4 shows a short example of F-logic syntax. : represents class membership, whereas :: represents the subclass relationship. The declaration of class *OccurrenceReport*[9] shows several signature expressions. Then, *report* is declared as an instance of *OccurrenceReport*, together with some data expressions. The current version of F-logic also supports built-in types based on the XML Schema datatypes,[10] their names start with an underscore, e.g., *_string*. *occurrence-buo01* and *ThomasLasky* are declared as instances of their respective types to complete the KB.

Listing 2.4: Example of a small knowledge base written in F-logic.

```
OccurrenceReport::Report .
OccurrenceReport[
   description => _string ;
   occurrence => Occurrence ;
   author => User ;
   created => _dateTime
] .
report-buo01:OccurrenceReport [
   description -> "Loss of speed in-flight resulted in..."^^_string ;
   occurrence -> occurrence-buo01 ;
   author -> ThomasLasky ;
   created -> "2019-08-28T01:17:00"^^_dateTime
] .
occurrence-buo01:Occurrence .
ThomasLasky:User .
```

F-logic Semantics

Semantics of F-logic is specified using *F-structures*. Before defining an F-structure, several additional notions are needed.

For a pair of sets U, V, $Total(U, V)$ denotes the set of all *total* functions from U to V. Similarly, $Partial(U, V)$ denotes the set of all *partial* functions from U to V. $\mathcal{P}(U)$

[9] I will use the *slanted typewriter font* to denote ontological terms written in F-logic in text throughout this thesis.

[10] See `https://www.w3.org/TR/xmlschema-2/#built-in-datatypes`, accessed 2020-08-11.

is used to express the power set of U. $\mathcal{P}_{\uparrow}(U)$ is the set of all *upward-closed* subsets of U. A set $V \subseteq U$ is upward closed if for $v \in V$, $u \in U$, $v \prec_U u$ implies $u \in V$, where $\prec_U$ is an irreflexive partial order on U (see below). $PartialAM_{\prec_U}(U, \mathcal{P}_{\uparrow}(U))$ denotes the set of all partial *anti-monotonic* functions from U to $\mathcal{P}_{\uparrow}(U)$. A function f is *partial anti-monotonic* if for vectors $\vec{u}, \vec{v} \in U^k$, $\vec{v} \prec_U \vec{u}$, if $f(\vec{u})$ is defined, then $f(\vec{v})$ is also defined and $f(\vec{u}) \subseteq f(\vec{v})$.

An F-structure is then a tuple $\mathbf{I} = \langle U, \prec_U, \in_U, I_{\mathcal{F}}, I_{\mathcal{P}}, I_{\rightarrow}, I_{\Rightarrow} \rangle$, where:

- U is the domain of $\mathbf{I}$ consisting of disjoint subdomains $U_{\mathcal{E}}$, $U_{\mathcal{C}}$, $U_{\mathcal{R}}$, $U_{\mathcal{A}}$,
- $\prec_U$ is an irreflexive partial order on $U_{\mathcal{C}\cup\mathcal{A}}$ representing the subclass relationship,[11]
- $\in_U$ is a binary relationship on $U_{\mathcal{E}} \times U_{\mathcal{C}\cup\mathcal{A}}$ specifying instance membership in classes,
- $I_{\mathcal{F}} : \mathcal{F} \rightarrow \bigcup_{k=0}^{\infty} Total(U^k, U)$ is a mapping which represents function symbols from $\mathcal{F}$ by functions from U^k to U. For $k = 0$, $I_{\mathcal{F}}(f)$ can be identified with an element of U. $I_{\mathcal{F}}$ maps names to their respective subdomains, e.g., class names from $\mathcal{C}$ to $U_{\mathcal{C}}$,
- $I_{\mathcal{P}}(p) \subseteq U^n$ for any n-ary predicate symbol $p \in \mathcal{P}$,
- $I_{\rightarrow} : U_{\mathcal{R}} \rightarrow Partial(U_{\mathcal{E}}, \mathcal{P}(U_{\mathcal{E}}))$,
- $I_{\Rightarrow} : U_{\mathcal{R}} \rightarrow PartialAM_{\prec_U}(U_{\mathcal{C}\cup\mathcal{A}\cup\mathcal{E}}, \mathcal{P}_{\uparrow}(U_{\mathcal{C}\cup\mathcal{A}}))$.

Remarks. The use of upward-closed sets is important for class hierarchies – it means that along with each class, the set also contains all its superclasses. The relationship between $I_{\rightarrow}$ and $I_{\Rightarrow}$ is such that $I_{\Rightarrow}$ defines the target type (range) of an attribute, whereas $I_{\rightarrow}$ defines particular values. The definition of an F-structure can be summarized by several properties which provide a more intuitive picture of F-logic semantics (see [29], Sec. 7, for full discussion of F-structure properties):

- Equality (=) is, as one would expect, a congruence relation on $U(\mathcal{F})$, i.e., it is reflexive, symmetric, transitive, and supports substitution.
- The subclass relationship (::) is reflexive, transitive, and acyclic.
- An important relationship between the :: and : operators is called subclass inclusion, that is: if $\mathbf{I} \models p{:}q$ and $\mathbf{I} \models q{::}r$, then $\mathbf{I} \models p{:}r$.

[11] $U_{\mathcal{C}\cup\mathcal{A}}$ is an abbreviation for $U_{\mathcal{C}} \cup U_{\mathcal{A}}$

- Type inheritance ensures that subclasses inherit signature expressions, i.e., if $\mathbf{I} \models p[m \Rightarrow s]$ and $\mathbf{I} \models q :: p$, then $\mathbf{I} \models q[m \Rightarrow s]$.
- Output relaxation means that superclasses can be returned instead of subclasses, i.e., if $\mathbf{I} \models p[m \Rightarrow s]$ and $\mathbf{I} \models s :: r$, then $\mathbf{I} \models p[m \Rightarrow r]$.

A variable assignment ν is a mapping from the set of variables, $\mathcal{V}$, to the domain U, which extends to id-terms as follows: $\nu(d) = I_{\mathcal{F}}(d)$ if $d \in \mathcal{F}$ has arity 0 and $\nu(f(..., t, ...)) = I_{\mathcal{F}}(f)(..., \nu(t), ...)$. Intuitively, given an F-structure $\mathbf{I}$ and a variable assignment ν, a molecule $t[...]$ is *true* under $\mathbf{I}$ w.r.t. to ν, written $\mathbf{I} \vDash_\nu t[...]$, if and only if the object $\nu(t)$ has the properties defined by the F-molecule. For example, $\mathbf{I} \vDash_\nu (O :: P)$ iff $\nu(O) \preceq_U \nu(P)$. For attributes, this means that there exist functions interpreting them and they have the right return values (types), e.g., $\mathbf{I} \vDash_\nu q[m \rightarrow v]$ iff $I_{\rightarrow}(\nu(m))(\nu(q))$ is defined and contains $\nu(v)$. An object molecule is considered a conjunction of method expressions. Precise definitions of logical implication in F-logic can be found in [29], Sec. 5.2. Satisfaction of complex formulas (using logical connectives and quantifiers) is defined in the usual first-order sense. An F-logic theory $\mathcal{S}$ logically implies an axiom α ($\mathcal{S} \vDash_\nu \alpha$) iff all models of $\mathcal{S}$ are also models of α. Since only closed formulas will be dealt with in this work, the variable assignment identifier will be omitted. Instead, F-logic semantic implication will be denoted by $\models^F$ to distinguish it from DL entailment.

Queries

An F-logic *query* Q is a molecule. The set of answers to Q w.r.t. a set of formulas P is the smallest set of molecules that:

- contains all instances of Q (variable assignments for all variables in Q) that are found in the model of P,
- is closed under $\models^F$ (see [29], Sec. 12.1.2).

Example. F-logic queries traditionally use capital letters to denote variables and the Prolog-style query prompt `?-` to indicate query start. The first query bellow searches for instances of class `OccurrenceReport` with author `ThomasLasky`, the second then returns the value of attribute `occurrence` of instance `report-buo01` and its type:

$$?- X : OccurrenceReport \wedge X[author \rightarrow ThomasLasky]$$
$$?- report\text{-}buo01[occurrence \rightarrow X] \wedge X : Y$$

Chapter 3

State of the Art

This chapter discusses the current state of the art of developing (domain-specific) Semantic Web-based information systems. To improve the coherence of the thesis' story, the chapter is split into sections corresponding to the particular sub-goals defined in Section 1.3. The first section is devoted to an overview of approaches to application access to ontologies and Semantic data. From a solely practical point of view, existing solutions and their principles are discussed first. A detailed comparison of selected tools for accessing Semantic data will be provided later (Section 4.1), as it represents one of the contributions of my research. From the point of view of theory, attention is paid to the different ways the closed world view is introduced to open world assumption-based DL knowledge bases. Then, based on the choice of F-logic as a formalism representing object-oriented programming languages, existing approaches of mapping between description logics and F-logic are compared.

What follows is a review of the ways applications can be integrated through Semantic Web technologies, ranging from using a shared ontological schema, over integration via SPARQL endpoints to semantically-enriched Web services.

The chapter is concluded by taking a look at surveys, classification, and examples of existing Semantic Web-based applications, as they help shed light on the areas where such software already exists and what its structure is.

Nomenclature. A short side note on the nomenclature appearing in this chapter and later in the thesis: various authors use different names for applications or information systems based on or using Semantic Web technologies and principles. The titles "Semantic Web applications" and "Linked Data applications" appear in

the literature most often, e.g., [32] or [11] respectively. I prefer "Semantic Web-based information systems" (SWIS), as such a name indicates that the system is not just an application, but may involve the underlying ontological model, possibly a methodology, or a business process model. For the purpose of this chapter, I will use the names interchangeably, and according to the referenced works. In later chapters, I will use the term Semantic Web-based information systems, as it is, in my opinion, more general and better captures the nature of systems at which my research primarily aims.

3.1 Accessing Semantic Data

Access to Semantic data may be viewed from two perspectives. One is the actual engineering problem of accessing the data efficiently. In this area, a wide variety of solutions already exists. The most basic approaches can be based on Semantic Web standards and require no particular tools. Applications can thus access data using a SPARQL [33] or Linked Data Platform [34] endpoint. However, such an approach would likely be inefficient and would require a lot of boilerplate code to transform the data from server responses to a form usable by the application. Therefore, specialized tools have been developed for Semantic data persistence. These will be dealt with first. The other perspective is on which formal bases (if any) this access is built.

3.1.1 Existing Tools for Semantic Data Access

Semantic data are typically stored in dedicated semantic databases – *triple stores* – such as RDF4J (former Sesame) [35], GraphDB (former OWLIM) [36] or Virtuoso [37]. To be able to make use of them, an information system needs means of accessing the triple store. Unfortunately, there is no common standard for accessing semantic databases (like ODBC [38] for relational databases). This lack of standardization gave rise to multiple approaches that can be split into two main categories [39]:

Domain-independent APIs work with the data on statement (triple or axiom, depending on the language) level. Such an approach provides great flexibility, as it allows to access all aspects of the data. Examples of domain-independent APIs are Jena [40], OWL API [41], and Eclipse RDF4J (formerly known as Sesame API) [35].

Domain-specific APIs employ a form of mapping between the data stored as triples and a domain-specific object model of the application. This allows

the application code to exploit the benefits of object-oriented programming (classes representing domain concepts, encapsulating related data and behavior), without having to deal with the statement-based nature of the data. They often internally make use of domain-independent APIs.

Domain-independent APIs are suitable for use in generic tools like ontology editors or vocabulary explorers, where no assumptions are made about the underlying domain. However, domain-specific information systems are built on top of object models representing the underlying domains. An information system using a domain-independent API would either have to be built without an object model of the domain, which would make it extremely difficult to develop and maintain or would have to contain a module for mapping the statement-level data to the object model and vice versa. But, that is exactly what domain-specific APIs, besides other functionalities, do. Listings 3.1 and 3.2 show the difference in readability and conciseness of data retrieval using the domain-specific and domain-independent approach respectively. Note that class Occurrence[1] from Listing 3.1 can be reused throughout the application.

Listing 3.1: Example of a domain class and retrieval of its instances using a domain-specific data access API.

```
@OWLClass(iri = Vocabulary.s_c_Occurrence)
public class Occurrence {

  @Id
  private URI id;

  @OWLAnnotationProperty(iri = RDFS.LABEL)
  private String label;

  @OWLDataProperty(iri = DC.Terms.DESCRIPTION)
  private String description;

  @OWLDataProperty(iri = Vocabulary.s_p_has_start_time)
  private LocalDateTime start;

  @OWLDataProperty(iri = Vocabulary.s_p_has_end_time)
  private LocalDateTime end;

  @Types
  private Set<String> types;

  // Getters and setters follow
}

public class OccurrenceDao {
  // Connection setup omitted

  public Occurrence find(URI id) {
    return em.find(Occurrence.class, id);
  }
}
```

[1] Sans-serif font will be used throughout this thesis to denote programming language code excerpts in text.

Listing 3.2: Example of retrieval of data corresponding to the domain class from Listing 3.1, but using a domain-independent API.

```
public class OccurrenceDao {
  public Map<String, Object> find(String id) {
    final Map<String, Object> data = new HashMap<>();
    try (final RepositoryConnection connection = repository.getConnection()) {
      final ValueFactory vf = connection.getValueFactory();
      final IRI iri = vf.createIRI(id);
      final Set<String> types = new HashSet<>();
      RepositoryResult<Statement> res = connection.getStatements(iri, RDF.TYPE, null, false);
      boolean found = false;
      while (res.hasNext()) {
        final Statement s = res.next();
        if (s.getObject().stringValue().equals(Vocabulary.s_c_Occurrence)) {
          found = true;
        }
        types.add(s.getObject().stringValue());
      }
      if (!found) {
        return Collections.emptyMap(); // Type does not match
      }
      data.put("types", types);
      res = connection.getStatements(iri, RDFS.LABEL, null, false);
      getValue(res, Literal.class).ifPresent(lit -> data.put("label", lit.stringValue()));
      res = connection.getStatements(iri, DCTERMS.DESCRIPTION, null, false);
      getValue(res, Literal.class).ifPresent(lit -> data.put("description", lit.stringValue()));
      res = connection.getStatements(iri, vf.createIRI(Vocabulary.s_p_has_start_time), null, false);
      getValue(res, Literal.class).ifPresent(lit -> data
          .put("start", lit.calendarValue().toGregorianCalendar().toZonedDateTime().toLocalDateTime()));
      res = connection.getStatements(iri, vf.createIRI(Vocabulary.s_p_has_end_time), null, false);
      getValue(res, Literal.class).ifPresent(lit -> data
          .put("end", lit.calendarValue().toGregorianCalendar().toZonedDateTime().toLocalDateTime()));
      return data;
      }
  }

  private <T> Optional<T> getValue(RepositoryResult<Statement> r, Class<T> cls) {
    final Optional<Object> value = r.hasNext() ? Optional.of(r.next().getObject()) : Optional.empty();
    return (Optional<T>) value.filter(v -> cls.isAssignableFrom(v.getClass()));
  }
}
```

An analogous division into *indirect* (corresponding to domain-independent) and *direct* (corresponding to domain-specific) models is used by Puleston et al. [42], who argue that direct models are more suitable for domain-specific applications and propose a hybrid approach where parts of the application are domain-independent and parts are domain-specific.

The difference between domain-independent and domain-specific APIs is similar to the difference between JDBC [43] and JPA [44] in the relational database access world. JDBC is a row-based access technology requiring a lot of boilerplate code and suffering from occasional compatibility issues. JPA, on the other hand, provides object-relational mapping and makes access to various databases transparent.

Domain-Specific Data Access APIs

As mentioned, domain-specific data access libraries provide a *mapping* between the object-oriented and Semantic Web paradigms. This mapping is in literature called *object-ontological* (OOM) or *object-triple* (OTM) mapping (OTM is typically used for RDFS-based solutions). In conformity with the comparison of domain-independent and domain-specific data access APIs above, Quasthoff and Meinel, in their survey among developers of OTM solutions [45], show that the main reason for developing OTM tools was increasing the productivity of programmers using Semantic data. Indeed, using an OTM library allows developers to work in the widespread and well-understood object-oriented paradigm and greatly simplifies access to the underlying triple store.

OTM can take two major forms depending on the programming language used by the library. OTM in *dynamically-typed* programming languages like Python or Ruby often relies on the character of the language and does not use a predefined object model. Oren et al., authors of one such library – *ActiveRDF* – argue, that this approach more closely corresponds to the dynamic nature of RDF and OWL, where classes are not predefined and can change at any time [46]. Similarly, the set of types to which an instance belongs is not immutable and may change. Statically-typed languages like Java and C++, on the other hand, require that the mapping is specified at compilation time. In Java, this is done usually via annotations (see the entity example in Listing 3.1). Quasthoff and Meinel challenge the claim of Oren et al., stating that even in statically-typed languages, one can load an individual as an instance of different classes [45]. In addition, most existing OTM libraries provide (although often limited) ways of accessing properties and classes not mapped by the object model.

The following paragraphs introduce five examples of contemporary object-triple mapping solutions. A more comprehensive list can be found in a paper I published together with my supervisor [47].

ActiveRDF. ActiveRDF [46] is an object-oriented API for Semantic Web data access written in Ruby. ActiveRDF represents resources by object proxies, attaching to them methods representing properties. Invocations of these methods translate to read/write queries to the storage. In addition, it supports an object-oriented query syntax based on PathLog [48] and can generate convenience filter methods automatically. One caveat of the highly dynamic nature of ActiveRDF is pointed out in [49] – it does not offer type correctness and typographical error checking present in libraries based on statically-typed languages such as Java.

AliBaba. AliBaba [50] is a Java library for developing complex RDF-based applications. It is an evolution of the Elmo [51] library. AliBaba is an extension to the Sesame API, so it supports only storage accessible via a Sesame/RDF4J connector. It uses dynamically generated proxy objects to track updates. AliBaba also allows using SPARQL queries to support more complex strategies of mapping values to Java attributes. In addition to OTM, AliBaba contains an HTTP server that can automatically make resources accessible as Web services, providing querying, inserting, and deleting capabilities.

Empire. Empire [52] is an RDF-based implementation of the JPA standard [44]. Therefore, its API should be familiar to developers used to working with relational databases in Java. However, since parts of JPA deal with the physical model of the relational database under consideration (e.g., the `@Table` and `@Column` annotations), Empire is not fully JPA-compliant. On the other hand, it does support the basic `EntityManager` API, query API, and entity lifecycle. Unfortunately, Empire's documentation is limited and it is often unclear which parts of JPA are implemented. On top of JPA annotations, a set of RDF-specific annotations that are used to express the mapping of Java classes and attributes to RDFS classes and properties is provided.

KOMMA. The Knowledge Management and Modeling Architecture (KOMMA) [53] is a Java library for building applications based on Semantic data. A part of this system is an object-triple mapping module, but the library itself has much richer functionality, including support for building graphical ontology editors based on the Eclipse Modeling Framework.[2] OTM in KOMMA is based on Java interfaces representing RDFS classes, for which it generates dynamic implementations at runtime. Parts of the KOMMA code base reuse code from AliBaba since both frameworks are based on similar principles.

RDFBeans. RDFBeans [54] is another OTM library built on top of RDF4J. It allows two forms of object models: 1) the object model may consist of Java classes representing the RDFS classes; 2) the object model may consist of interface declarations forming the mapping and RDFBeans would generate appropriate implementations at runtime using the *dynamic proxy* mechanism. RDFBeans is a lightweight library with a small memory footprint.

Contemporary OOM solutions suffer from several fundamental problems and one of the goals of this thesis is to overcome them. One such problem is the ad hoc nature of the existing implementations. They lack any formal basis and the mapping principles are based solely on the convention their developers had in mind. Of course,

[2] `http://www.eclipse.org/modeling/emf/`, accessed 2020-08-11.

in most cases, this convention-based approach will work just fine. However, certain aspects of the mapping (for example, treatment of anonymous individuals, inferred knowledge, class hierarchies) are not straightforward and their formal underpinning could prevent misinterpretations and inconsistent behavior.

Another issue related to the nonexistent mapping formalism is the conflict between the CWA and the OWA. As was discussed in Section 2.8, for domain-specific information systems, the open world assumption of DL ontologies may be unsuitable, for instance, due to its tendency to create new anonymous instances for the sake of knowledge base consistency.

3.1.2 Closed World Reasoning in Description Logics

The mismatch between the OWA and the CWA actually concerns all monotonic FOL-based languages (including DLs). Therefore, several approaches introducing (local) non-monotonic behavior to such languages have been developed.

Donini et al. [55] introduce the *Minimal Knowledge and Negation as Failure* logic with autoepistemic operators **K** (knows) and **A** (assumes) as an extension of the description logic $\mathcal{ALC}$. Grimm and Motik show how it can be used for IC definition [56]. Motik et al. [57] use minimal Herbrand models to provide integrity constraint validation. They extend knowledge bases with a set of selected TBox axioms which are treated as integrity constraints. Such constraints are checked when performing ABox reasoning. Unfortunately, both of these approaches can lead to counter-intuitive results, as was pointed out by Tao et al. [27]. Their approach will be used in this thesis, as already discussed in Section 2.8. A more recent effort by Patel-Schneider and Franconi [58] discusses the flaws of all of the aforementioned solutions and proposes the use of DBox-based *completely specified* concepts and roles. However, not even this approach is immune to debatable results. Consider the following example:

$$\begin{aligned}\mathcal{T} &= \{Employee \sqsubseteq Person, Flight \sqsubseteq \exists hasPassenger.Person\}\\ \mathcal{A} &= \{Flight(c), Flight(d)\}\\ \mathcal{DB} &= \{Person(a), Employee(b), hasPassenger(c,a), hasPassenger(d,b)\}\end{aligned}$$

We put $hasPassenger$, $Person$ and $Employee$ into the DBox, so that no unexpected instances are generated. However, this will cause an IC violation, because $Person(b)$ will be inferred for a completely specified concept $Person$. In other words, DBoxes are too coarse-grained to be effectively used for IC specification. Ren et al. [59] replace DBoxes with NBoxes which contain concepts and roles for which *negation as*

failure reasoning will be applied. That is, inference is still possible for predicates in an NBox, but such predicates are treated with the CWA.

Application of integrity constraints to DL ontologies is closely related to *(local) closed world reasoning*. A significant amount of work has been done in this area in connection with rule-based languages. They often split the knowledge base into a DL-based OWA part and a rule-based CWA part with stable [60, 61] or well-founded [61, 62] model semantics.

Another approach similar to [58] is based on *grounded circumscription* where selected concepts and roles are closed and minimized, i.e., they contain only the minimum necessary *known* individuals [63].

3.1.3 Mapping Between Description Logics and F-logic

The relationship between ontologies and F-logic (my solution of choice for representing object-oriented programming languages) can be approached from different directions. Firstly, F-logic could be used as an ontology modeling language directly, as is discussed by its authors themselves in [30, 64]. However, description logics are the standard ontology definition language nowadays. Another approach, which has been investigated by Grosof et al. [65] or Kattenstroth et al. [66], enriches a DL knowledge base with (F-logic) rules to provide additional or more efficient inferences ([65] considers logic programming languages in general).

The last direction tries to develop a mapping between description logics and F-logic. De Bruijn et al. exploit the fact that DLs are a subset of FOL and map them to the FOL flavor of F-logic, i.e., concepts to unary predicates and roles to binary predicates [67]. On the other hand, Balaban attempts to map DL constructs to F-logic frames [68]. However, his article deals only with less expressive DLs ($\mathcal{ALC}$). Close to the approach I will introduce later in this thesis is also the work of de Bruijn and Heymans [69] which maps $\mathcal{SHIQ}$ to F-logic by first translating it to predicate-based FOL and then mapping it to F-logic. Compared to Balaban, the approach presented in Section 4.2 deals with more expressive languages and considers the mapping of integrity constraints as well. De Bruijn and Heymans' work presents, in my opinion, a less readable, although arguably more straightforward, approach to the mapping.

3.2 Integrating Applications Using Semantic Web Technologies

With the growing size of information systems and the amount of data they have to cope with, software development trends have shifted towards more modular styles like the *service-oriented architecture* and *microservices*. Such design allows composing the system out of loosely coupled components, which can be developed and deployed independently. Moreover, information systems nowadays do not interact only with human users, but they communicate more and more with other information systems. Consider the safety data collecting and processing system exemplified throughout this thesis. European CAAs have to report to and, conversely, receive reports from the European Central Repository (ECR),[3] a repository for occurrence reports in aviation. Obviously, a national SDCPS in Europe should be integrated with ECR to spare its users having to interact with both systems separately.

On the other hand, many information systems today benefit from integrating data from various sources, providing their users with a more comprehensive picture of the domain of interest. For instance, Flightradar24[4] gathers data from different providers (aircraft position broadcasters, radar data, map services) to provide a realtime overview of air traffic around the world.

Both of these viewpoints illustrate how data and service integration are used to improve information systems' capabilities. This section reviews existing approaches to using Semantic Web technologies to facilitate such integration.

3.2.1 Data-level Integration

Information system integration at the data level can range from systems using the same data schema to complex issues of mapping heterogeneous schemas to a common model. In this regard, Semantic Web technologies can be used for *ontology-based data integration* [70], where heterogeneous data are mapped by an ontology representing the common domain model. The advantage of using an ontology in such cases is that it provides a semantically rich domain description with a formal basis. If the data sources already use Semantic technologies, the problem becomes that of *ontology mapping* – finding a way to unify the ontological schemas of the data [71]. Examples of information systems integrating data using Semantic Web technologies will be discussed in Section 3.3.2.

[3] `https://bit.ly/2pBYlXK`, accessed 2020-08-11.
[4] `https://www.flightradar24.com`, accessed 2020-08-11.

3.2.2 Service-level Integration

At the level of service integration, an information system exposes an API that can be used by another system. Such API may allow it to query or manipulate data, or invoke business logic.

The most basic and, at the same time, the most expressive approach is when an application exposes a SPARQL endpoint so that other systems may invoke it using the SPARQL Protocol [33]. However, such an approach has several disadvantages. On the one hand, it is difficult to restrict the constructs or data available to the caller. The same holds for data manipulation. Allowing clients to run SPARQL queries can also incur heavy load on the target server. Verborgh et al. mention this as one of the main reasons for developing the Linked Data Fragments (LDF) framework [72]. LDF transfer part of the query processing workload to the client by allowing it to execute parts of the query, fetch results in chunks, and combine them in place instead of having the server do all the work. LDF are thus optimization of the read-only data access. Another SPARQL-based approach is presented by Michel et al., who wrap existing Web APIs[5] so that they can be queried using SPARQL [73]. Their goal is to present a unified interface for SPARQL endpoints and non-semantic Web APIs and they choose SPARQL as the primary communication mechanism. Of course, since the semantics of Web APIs is different (primarily subject-centered), only a restricted subset of query patterns is supported. Nouwt uses a similar approach [74]. Strictly speaking, the aforementioned approaches do not represent service-level application integration, because SPARQL endpoints provide access to data, not application logic. They are suitable in certain cases, like publishing open data, but, by analogy, relational database-based information systems also do not expose the underlying database for the client to directly run SQL queries. Besides security and access control reasons mentioned above, using SPARQL (or SQL for that matter) also requires the caller to understand the underlying technology – a hurdle for many information system developers. Another issue may be schema evolution, which is practically impossible to communicate to the clients in this scenario.

There exist other techniques which essentially provide data access similar to the SPARQL endpoints, but do it in a more service-oriented way. That is, they provide an HTTP-based Web API for the clients to query and manipulate data. Linked Data Platform is a W3C standard for providing an API for Linked Data manipulation [34]. It is based upon the notion of containers of data regarding specific resources. The API URLs are based on resource identifiers and thus can adhere to REST principles, HTTP methods are also used to specify operations to perform with the data. However, the data produced and consumed by the platform are purely RDF and their vocabulary and structure are restricted by having to adhere to the container-based design. Schröder et al., on the other hand, build a REST API on top of arbitrary

[5] Server-side API accessible over Web, typically REST (less frequently SOAP).

SPARQL endpoints, allowing one to navigate the graph by alternating resource and property IRIs (namespace-based names to be precise) in the service URL [75]. For instance, `/api/class/foaf:Person` can be used to retrieve all instances of class `foaf:Person`[6] and `/api/class/foaf:Person/dbr:George_S._Patton` then to retrieve data about the instance representing George Patton. The data are transferred using a custom JSON structure (although, JSON-LD would be arguably more appropriate) and the API supports also insertions, updates, and deletes. BASIL represents a similar approach, where SPARQL endpoints are wrapped by a Web API (BASIL API) [76]. This allows developers not familiar with Semantic Web technologies (SPARQL in particular) to exploit their data.

Lastly, there exist approaches that provide semantic annotation of regular non-semantic Web APIs and the data they produce. Due to the overwhelming popularity of REST in comparison to SOAP, I will concentrate on REST APIs in this work. Lanthaler and Gütl proposed the SEREDASj (Semantic RESTful Data Services) framework [77]. This structured framework can be used to describe a Web API. Based on this description, automatic translation from SPARQL queries to Web API calls can be performed. The authors exemplify this architecture on a use case of data integration, where the data integration layer transparently accesses data from SPARQL endpoints, triple stores, static RDF files, and, through SEREDASj, Web APIs. Insertion and removal are also possible, but only without using variables (they are generally not supported by Web APIs). Lanthaler and Gütl later proposed a new framework for semantically describing Web APIs – the Hydra vocabulary [78]. Compared to SEREDASj, Hydra is more lightweight, uses JSON-LD instead of JSON, and is more closely integrated into the Web API. This way, clients can retrieve Hydra description of a service – e.g., valid state transitions and available endpoints with the structure of the data they provide/expect – and directly act upon it. Listing 3.3 shows a short example of a Hydra description of an API endpoint.

Listing 3.3: Example of a Hydra description of a Web API entry point. `issues`, `register_user` and `users` point to endpoints with supported operations.

```
{
  "@context": "/hydra/api-demo/contexts/EntryPoint.jsonld",
  "@id": "/hydra/api-demo/",
  "@type": "EntryPoint",
  "issues": "/hydra/api-demo/issues/",
  "register_user": "/hydra/api-demo/users/",
  "users": "/hydra/api-demo/users/"
}
```

With respect to the discussed approaches, the information system integration goal of this thesis is arguably less ambitious. The target is to facilitate building RESTful interfaces of domain-specific information systems using Semantic Web technologies.

[6] `foaf` is a prefix for `http://xmlns.com/foaf/0.1/`.

3.3 Semantic Web-based Information Systems: A Survey

Although Semantic Web-based information systems are far from being mainstream, research concentrating on the topic does exist. I will now first review the theoretical approach – works on principles of developing Semantic Web applications, their classification, and exploitation; second, a short survey of typical Semantic Web applications described in the literature will be provided. Note that this section does not concern Semantic Web tools and software libraries. Rather, end user-facing business information systems are of interest.

3.3.1 Literature Concerning Semantic Web-based Information Systems

In the last decade, several authors have discussed the principles upon which typical Semantic Web-based information systems are built. Hausenblas, in his technical report [11], first reviews several existing Linked Data applications (LDAs) and proceeds to discuss the principles of building LDAs. He then reviews the anatomy of typical LDAs and categorizes them into three groups: 1) Generic Linked Data browsers, 2) Linked Data search engines and indexers, and 3) Domain-specific LDAs. Hausenblas' idea of a domain-specific LDA concentrates on the integration of data from various, potentially non-Semantic, data sources, and their presentation to the user. In his view, a triple store acts mainly as a cache for RDF-ised data needed by the application's business logic. While the data integration aspect is certainly one of the big benefits of ontology-based applications, the data authoring aspect of such systems should be taken into account as well. In contrast to accessing legacy non-Semantic data, it is reasonable to employ Semantic technologies when new data are created for an LDA – fewer paradigm boundaries are crossed, Semantic Web standards and languages can be taken advantage of, etc.

Martin and Auer provide a categorization of Semantic Web applications based on a number of criteria [12], which are briefly reproduced in Table 3.1. Their work is referenced by the authors of project EUCLID's [13] description of LDAs, who, in Chapter 5 of the tutorial on the usage of Linked Data,[7] also discuss the tiered architecture of an LDA. Similar to Hausenblas, they stress the data integration function of the data layer. For this layer, they introduce several mechanisms of access to Linked Data:

[7] See `http://euclid-project.eu/modules/chapter5.html`, accessed 2020-08-11.

Criterion	Value	Description
Semantic technology depth	Extrinsic	Semantic Web technologies are used only on the surface of the application, not internally.
	Intrinsic	Semantic Web technologies are used to replace (at least partially) conventional technologies like relational databases.
Information flow direction	Consuming	LD are consumed from a different source, e.g., in a mashup application.
	Producing	Application produces LD.
Semantic richness	Shallow	Simple taxonomies using RDF(S).
	Strong	Expressive languages like OWL.
Semantic integration	Isolated	Limited reuse of identifiers, vocabularies, and ontologies.
	Integrated	Extensive reuse of identifiers, vocabularies, and ontologies.

Table 3.1: Categorization of Semantic Web applications according to [12].

- **Linked Data crawlers** which allow crawling Linked Data datasets by following links between RDF resources. For example, LDSpider [79] or Squirrel.[8]
- **Linked Data client libraries** which allow querying the Semantic Web as a single repository. For example, the Semantic Web Client Library.[9]
- **SPARQL client libraries** which allow executing SPARQL queries on datasets. For example, Jena [40].
- **Federated SPARQL engines** which allow querying multiple target repositories through a single endpoint. For example, RDF4J [35].
- **Search engine APIs** which allow searching and retrieving Semantic data on the Web. For example, the late Sindice engine [80].

Once again, this description concentrates on applications providing primarily read access to Semantic data. The read-only access theme is clear in the Semantic Web literature, as is demonstrated by Barbosa et al., who, in their recent literature review [81], show that 78% of research articles concern systems providing access to Semantic data.

Lytras and García, on the other hand, ponder why the Semantic Web has not been as successful in the industry [32]. They emphasize the business point of view, where the stakeholders need to see the eventual financial or organizational benefits of switching from traditional technologies like relational databases to a new paradigm. One may, in their work, spot an appearance of a principle briefly touched upon in Chapter 1 – the fact that the business logic core of a system stays virtually oblivious to the choice of the technologies and paradigms of the system around it. In the

[8] `https://dice-group.github.io/squirrel.github.io/`, accessed 2020-08-11.

[9] `http://wifo5-03.informatik.uni-mannheim.de/bizer/ng4j/semwebclient/`, accessed 2020-08-11.

article, Lytras and García discuss the four layers which need to be analyzed to evaluate the switch to Semantic Web technologies:

- **Data layer** where data representation standards and technologies are the primary concern. Here, Semantic Web offers the RDF and SPARQL standards.
- **Semantic Web and ontological engineering layer** where ontologies allow creating and reusing data schemas. In addition, relevant development tools needed for efficient information system development are mentioned.
- **Semantic Web-based information systems layer** concerns the integration of Semantic Web-based components into an information system via well-defined interfaces, Web services, etc.
- **Business logic/intelligence layer** represents the core of the system's functionality from the organization's perspective.

In the end, the authors propose a three-tiered decision model for the adoption of Semantic Web technologies in organizations:

1. **Business strategy** – Semantic Web use must be in alignment with the key strategic objectives of the company.
2. **Semantic/knowledge performance** – an analysis of business performance factors that can be enhanced/supported by the Semantic Web and ontological engineering must be performed.
3. **Technical aspect** – finally, technical aspects need to be considered.

Heitmann et al. surveyed Linked Data applications presented at application development challenges of two major Semantic Web conferences – the International Semantic Web Conference (ISWC) and the European Semantic Web Conference (ESWC) [82]. Based on this survey, they decompose a typical Linked Data application into several components, identify four challenging areas of its implementation, and propose ideas on how to simplify Semantic Web-based application development. An important point they make in the survey is that one of the reasons for the low percentage of systems allowing users to create or edit data (29% of the surveyed applications) may have been the lack of data definition and manipulation language at the time the survey took place (the newest examined systems were from 2009). Indeed, the SPARQL Update language standard was proposed in 2012 (submission had been made in 2008). Nevertheless, APIs like Jena [40] and Sesame [35] allowed data manipulation long before the inception of SPARQL Update. The components of a Semantic Web-based information system identified by Heitmann et al. are

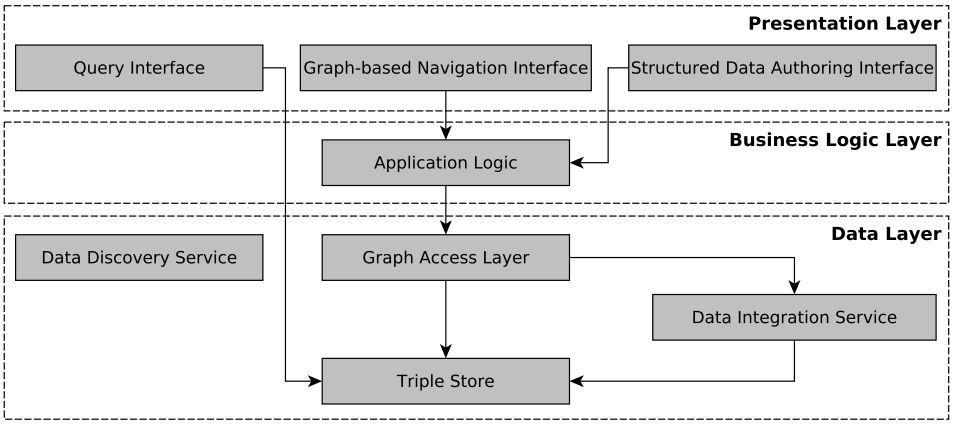


Figure 3.1: Linked Data-based application components by Heitmann et al. [82]. Edges represent component dependency. Layered structure introduced by me.

visualized in Figure 3.1 – their names should be self-explanatory (*Graph-based navigation interface* denotes user interface which exploits the underlying domain schema). The implementation challenges identified by the authors concern:

- **Integration of noisy and heterogeneous data** from different sources, some of which may not be under the control of the application developers.
- **Mismatch of data models between the components** refers to the fact that application data are stored in RDF, but the business logic is written in the object-oriented paradigm. In addition, data being integrated from external sources are often stored in relational databases, so a paradigm threesome can emerge.
- **Distribution of application logic across the components** means that while the bulk of the application logic is in the business logic layer, parts of it can be expressed in the ontological domain model or as rules specified for the storage. More logic is then encoded in various techniques concerning data integration.
- **Missing guidelines and patterns** pertain mostly to guidelines for the implementation of standards-based libraries.

The authors propose the following recommendations to simplify Linked Data application development – 1) creation of guidelines; 2) best practices and design patterns; 3) development of middleware software libraries and factories able to generate complete applications based on an entered configuration. While the third goal is hard (and often infeasible) to achieve even in traditional software development, the first two goals are aligned with the goals of this thesis.

A more recent overview of trends in Semantic Web application development, this time from the point of view of the industry, can be found in the Dagstuhl Workshop summaries [83, 84]. In these, central European (Austria, Germany, Switzerland) Semantic Web practitioners discuss the main challenges, benefits, and trends of using Semantic Web technologies in the corporate environment. The first important thing to note is the difference between the open, academic Semantic Web, and the Corporate Semantic Web, which mostly concerns a particular domain segment. This has certain benefits, including fewer issues with resource identity or meaning disambiguation. It also often means that the open world nature of the Semantic Web is replaced by a centralized closed world view of the corporate data. Nevertheless, using Semantic Web technologies in a corporate environment has also specific challenges, ranging from the need to fit the paradigm into the organization's processes, the requirement of profitability (for companies), and the need for quality data. Regarding trends in the Corporate Semantic Web, the authors discuss ontological modeling, ontology evolution, using machine learning and natural language processing to extract data from plain text or, conversely, to generate unstructured text based on ontological models. An important topic discussed in the reports is also the need for mature and efficient tools and software libraries for the development of ontology-based corporate information systems.

3.3.2 Examples of Existing Semantic Web-based Information Systems

Although the literature is not overflowing with descriptions of successful Semantic Web-based information systems, there exist many. This section introduces just a handful of them to illustrate the typical tasks in which Semantic Web technologies are advantageous.

Systems Providing Data Integration

A recurring topic of Semantic Web technologies exploitation is data integration. This could have been already seen in Section 3.3. O'Riain et al. describe such an application in the financial domain [85]. In their case study, the system integrates data from several heterogeneous sources, including stock exchange public API, news publishers, geospatial databases, etc. Such data are transformed to RDF, integrated using bridge ontologies, and indexed in storage. A user interface then allows browsing and searching the data in an intuitive way.

The *GetThere* system, on the other hand, allows real-time updates to the data [86].

It gathers and integrates data about public transportation, which are then available to users via a mobile application. The mobile application serves two purposes – it provides users with information about public transportation services and in turn, once the user enters a public transportation vehicle, it is able to upload sensory data to the server, where they are used to ascertain the vehicle's current location and compare it to its expected schedule.

Systems with Linked Data API

One of the best-known examples of information systems providing Linked Data API for consumers is the BBC content annotation, linking, and publishing [87].[10] BBC used Semantic technologies to annotate existing content and interlinked it with DBPedia[11] thus providing machine-readable access to a large corpus of structured and quality data.

Knowledge-based Systems

Semantic Web technologies, more specifically expressive ontologies, can play a fundamental role in knowledge-based systems (KBSs). Based on the ontological schema, inference can reveal hidden relationships in the data or generate explanations for particular conclusions. Carvalho et al. describe such a knowledge-based system – the *InvestigationOrganizer* [88, 89]. It was developed at NASA and serves as a support system for mishap investigators. It allows them to enter large amounts of heterogeneous data, build fault trees, or preserve the investigation process history. Its ontological base also allows executing inference rules to extract additional, implicit information from the data. Another example of the utilization of ontologies in KBSs is described by Angele et al. [31]. In their case, F-logic ontologies are used to describe the domain and express the rules and constraints for a test car configuration checking system. Another example of a knowledge-based information system using ontologies is *MONDIS* (MONument Damage Information System) [90]. MONDIS is based on the Monument Damage Ontology [91], which allows describing, besides classification of constructions, materials, etc., also causality chains relevant to damage of cultural heritage structures. The system consists of a mobile application used to collect on-site data, including photographic evidence and geographic location. This application, in addition, contains an adaptable user interface generated from the ontological model. The second part of the information system is an advanced search application, which allows to mine information from the stored Semantic data. The authors themselves

[10] Also `https://bbc.in/1SLBRLO`, accessed 2020-08-11.
[11] Semantic Web-enabled Wikipedia – `https://wiki.dbpedia.org/`, accessed 2020-08-11.

mention the lack of adequate software tools as a major obstacle in the development of the system.

The preceding literature review represents a baseline for the analysis and recommendation of architectural and design guidelines for developing domain-specific Semantic Web-based information systems, which will be provided later in this thesis.

Part II

Contribution

Chapter 4

Theoretical Basis for Application Access to Semantic Data

This chapter presents my contribution to the topic of efficient application access to Semantic data. In particular, it starts with the development of a framework for comparison of object-triple mapping SW libraries and its application to a set of contemporary OTM solutions (Section 4.1). The framework represents a novel, comprehensive tool that helps software developers to choose the most suitable OTM library for their use case.

In the sequel, theoretical foundations describing application access to Semantic data are laid down in two phases. The first phase (Section 4.2) is devoted to a formal definition of object-ontological mapping built on the axis description logics – F-logic – an object-oriented programming language. This formalism comprises the bulk of my (theoretical) contribution. The remainder of the chapter (Section 4.3) presents a formalization of the operations an information system needs to perform on the underlying data. Sections 4.2 and 4.3 provide means for a formal definition of the object-ontological mapping, the lack of which concerns all contemporary OTM/OOM solutions (as was discussed in Section 3.1). Chapter 5 will later show how these principles are carried over to an actual implementation. Figure 4.1 illustrates the focus of this chapter.

A short sidenote on nomenclature is again in order: borrowing the terminology of the Java Persistence API [44], *entity class* is used to denote programming language classes mapped to persistent concepts (OWL or RDFS classes in our case), an *attribute* denotes an instance field together with the methods which get (getter) and set (setter) its value. Finally, an *entity* is an instance (object) of an entity class.

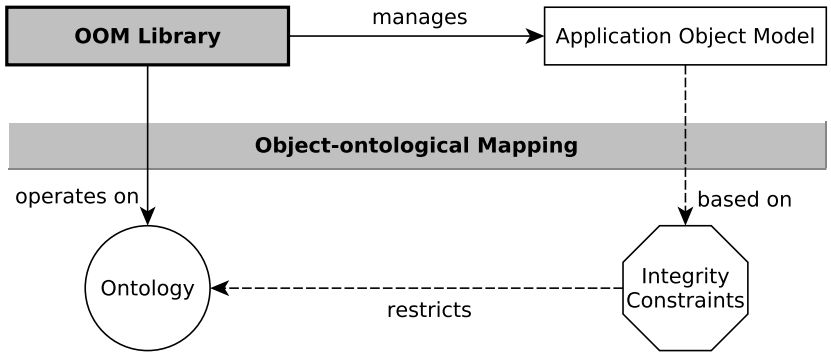


Figure 4.1: Overview of the focus of this chapter. Relevant parts are marked with bold font and grey background. Solid lines depict direct usage.

4.1 Comparison of Object-triple Mapping Libraries

Chapter 3 argued that domain-specific persistence libraries are more suitable for domain-specific Semantic Web-based information systems. Examples of such libraries presented in that chapter comprise but a subset of existing solutions. These solutions predominantly support accessing RDF(S) data, as more expressive languages like OWL often represent a significant performance obstacle in the development of real-world information systems. Thus, the term object-triple mapping (OTM) libraries has been used to denote such tools. When a software developer decides to use an OTM library to access Semantic data in their application, they are faced with an uneasy task of choosing the best applicable one. Right from the start, even discovering existing solutions is difficult. The primary culprit is that most of them have originated in the academic sphere as small tools intended to streamline the work on research projects. Thus, their documentation is scarce, their user base small and it is difficult to find even basic information like features, performance characteristics, or support for integration with application development libraries like Spring.[1] Many of them are also obsolete, as their development stopped years ago, e.g., Elmo [51] or Owl2Java [92].

Therefore, we created a framework for comparison of OTM libraries and used it to provide an overview of existing solutions [47]. The contributions of this endeavor are threefold:

1. A framework for qualitative (feature-wise) and quantitative (performance-wise) comparison of OTM libraries.
2. A survey of existing OTM solutions using this framework.

[1] `https://spring.io/`, accessed 2020-08-11.

3. Minimal working usage examples of libraries evaluated in the performance part of the comparison.

The third point is a by-product of the performance benchmark. However, given the lack of proper documentation for the libraries, it is a welcome result.

There exist very few comparisons of OTM libraries, as most Semantic Web-related comparisons concentrate on the underlying storage – for example, the Berlin SPARQL Benchmark (BSBM) [93] or the Lehigh University Benchmark (LUBM) [94]. Holanda et al. provide a minimalistic performance comparison between their solution – JOINT-DE – and AliBaba [95]. Cristofaro compares the features of selected OTM libraries [96], as does Sieland [97]. Sieland, in addition, reviews the development activity, documentation, and ease of use of the compared tools. Nevertheless, these works are rather narrow in their selection of survey subjects and shallow in the depth of the comparison.

4.1.1 Design of the Comparison Framework

Our comparison framework consists of two parts – a set of criteria used to evaluate the features supported by the compared libraries, and a performance benchmark.

Feature criteria

The qualitative part of the framework consists of a set of twelve criteria that can be used to assess an OTM library's suitability in a particular use case. The criteria may be split into three main categories:

General (GC) General criteria are based on the principles known from application development and relevant object-relational mapping features. The framework contains six general criteria.

Ontology-specific (OC) These criteria take into account specific features of the Semantic Web language stack based on RDF and its design. They are not exclusive to OTM libraries and could be applied to libraries used to access Semantic data in general. The framework contains three such criteria.

Mapping (MC) Mapping criteria concern important techniques used for the object-triple mapping. They are motivated by the differences of ontologies on the one

side and the object-model on the other [98]. The framework contains three mapping-specific criteria.

The criteria selection is based on existing knowledge from other domains (e.g., object-relational mapping), literature, and our own experience with developing Semantic Web-based information systems. The following paragraphs briefly introduce the criteria.

GC1 – Transactions Indicates whether the OTM library supports transactional processing, as it is one of the fundamental paradigms of computer science allowing to bind related operations into a single unit of work [99]. For the purpose of the comparison, only atomicity, consistency, and durability of transactions are considered.

GC2 – Storage access variability Refers to the ability to connect to storages of different providers. This is important in order to avoid vendor locking. Since there is no standard for Semantic storage access (besides a SPARQL endpoint), triple store vendors provide proprietary APIs. *GC2* thus checks whether the library supports at least two different storage APIs.

GC3 – Query result mapping Checks for support for mapping between results of a SPARQL query and the application object model. Such a mapping greatly simplifies the usage of SPARQL in domain-specific applications.

GC4 – Object-level query language Validates the existence of a query language enhanced by object model constructs. That is, a language allowing to use names of classes and attributes and to perform joins by simple traversal of object attributes (e.g., p.friends.firstName) in the query. This criterion goes one step beyond *GC3* – because SPARQL may be cumbersome to work with (mainly due to having to deal with IRIs), queries based on the application model are easier to write and read.

GC5 – Detached objects Refers to the possibility to detach an object from its connection to the underlying repository, as many OTM libraries use the objects as mere proxies to the triple store and require the objects to be tightly coupled with the data source connection, possibly causing performance or transactional issues.

GC6 – Code/ontology generator Indicates whether the OTM library allows generating the object model based on the ontological model, or vice versa. Such a generator may simplify the development (especially) in its initial phases.

OC1 – Explicit inference treatment is important since while inferred statements are indistinguishable from asserted ones on retrieval, the difference is significant on update. Mainly because inferred statements cannot be directly removed. Therefore, *OC1* validates whether an OTM library makes such a distinction.

OC2 – (RDF) Named graphs Named graphs allow to structure data in the repository. *OC2* evaluates whether an OTM tool supports storing and accessing data in named graphs.

OC3 – Automatic provenance management Checks whether the library allows exploiting standard Semantic Web mechanisms for data provenance (RDF reification vocabulary [18], the PROV-O ontology [100]) to automatically generate these metadata. Provenance data are a common requirement in information systems, due to the need for change tracking and auditing. Being able to generate them automatically instead of having to program a routine for such a task would again simplify development.

MC1 – Inheritance mapping Discusses whether the mapping of class hierarchies is supported by the tool. This is one of the most important criteria, as hierarchies of classes are one of the fundamental parts of ontology modeling.

MC2 – Unmapped data access Indicates whether it is possible to access also data not mapped by the object model. For instance, the set of properties concerning a particular individual may not be completely covered by the object model, yet, the application may need to access such data in specific scenarios.

MC3 – RDF collections and containers are introduced as a means of modeling collections of data beyond the regular set semantics of RDF/OWL, which may be insufficient in many cases – for instance, most data are presented to the user as lists based on different ordering criteria. Whether an OTM library supports such concepts is validated by *MC3*.

Performance Benchmark

The slowest player in Semantic data access will in most cases probably always be the repository. This is because of the volume of data it has to search, update, and store. However, efficient strategies employed by an OTM implementation can have a major influence on the overall performance. Therefore, the performance benchmark allows comparing OTM libraries in terms of common operations executed over the repository. The benchmark is based on a relatively simple ontology from the aviation domain. The ontology is a restriction of a larger ontology used in a safety occurrence reporting and management system (will be discussed in greater detail in Chapter 6). An object model based on this ontology is depicted in Figure 4.2. While relatively simple, the model exercises the most common mapping constructs – singular and plural attributes with literal and reference values, and a subclass relationship between Occurrence and Event.

A set of six common create, retrieve, update, and delete (CRUD) operations was selected to simulate real-world usage scenarios.

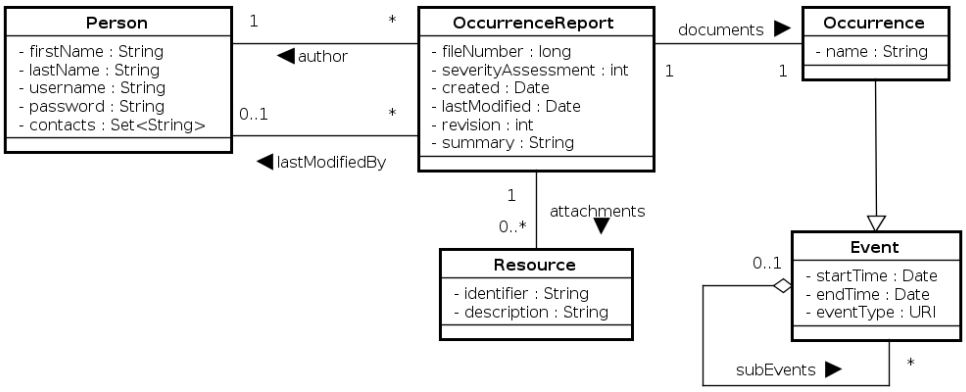


Figure 4.2: UML class diagram of the OTM comparison performance benchmark model.

OP1 – Create represents a typical operation performed by a domain-specific information system. It creates an instance of class OccurrenceReport with a set of related objects and inserts them into the repository.

OP2 – Batch create simulates situations where, for instance, data are imported into the system in bulks. Thus, the mode of operation is almost the same as *OP1*, but a large number of insertions happens in a single transaction.

OP3 – Retrieve stands for an application requesting a specific instance together with its references using its identifier. This can simulate a user viewing the detail of one report in the system's user interface.

OP4 – Retrieve all represents another common functionality of information systems – loading all instances of a certain type, for example, to display them in a list or a table in the UI. In this case, all reports with relevant references are loaded.

OP5 – Update simulates the system updating data of a single report. More specifically, several literal attributes of a report and its related occurrence are updated and a new reference to a Resource is added.

OP6 – Delete removes a report and all its related data except for the users set as the report's author and last modifier, as they are system users and cannot be removed in this operation.

Each operation consists of *setup*, *execution*, and *tear down*, where only the execution phase is measured. The setup and tear down phases are used to generate test data and/or validate operation results.

In order to make the benchmark fair, it is based only on a single platform – Java – and was executed on a single triple store – GraphDB [36]. This narrows down the selection of libraries to evaluate but makes their comparison more representative.

Library	*GC1*	*GC3*	*GC5*	*OC1*	*MC1*
ActiveRDF	×	○	×	×	N/A
AliBaba	✓	✓	×	✓	✓
AutoRDF	×	×	×	×	✓
Empire	○	✓	✓	×	○
JAOB	×	×	✓	×	×
JOPA	✓	✓	✓	✓	○
KOMMA	✓	✓	×	✓	○
RDFBeans	✓	×	✓	×	○
RDFReactor	×	×	×	✓	○
SF	×	×	✓	×	○
Spira	×	×	×	×	○
SuRF	✓	×	×	×	N/A

Table 4.1: Selected OTM libraries compared using a subset of the criteria defined in Section 4.1.1. × means no support, ○ represents partial support, ✓ is full support of the feature and N/A signifies that the feature cannot be evaluated in the particular case. JOPA is highlighted as it is the framework developed by my colleagues and me.

4.1.2 Overview of Comparison Results

The framework introduced above was used to compare a set of OTM libraries selected based on their popularity, maturity, accessibility (some libraries, e.g., JOINT-DE [95] or the Semantic Object Framework [101], are described in the literature, but their source code is not available), and development activity (obsolete libraries were omitted from the selection).

Feature Comparison Results

Twelve libraries were evaluated in the comparison: ActiveRDF [46], AliBaba [50], AutoRDF [102], Empire [52], JAOB,[2] JOPA, KOMMA [53], RDFBeans [54], RDFReactor [103], the Semantic Framework (SF) [104], Spira,[3] and SuRF.[4]

In general, most of them suffer from the fact that there is no formalization of the object-triple mapping. Such a gap leads to potentially inconsistent behavior and straightforward solutions that fail to handle less obvious cases (for example, removal of unmapped data on update in most libraries). A detailed discussion of the individual features is, due to its extent, out of the scope of this work and the reader

[2] `https://github.com/yoshtec/jaob`, accessed 2020-08-11.
[3] `https://github.com/ruby-rdf/spira`, accessed 2020-08-11.
[4] `https://github.com/cosminbasca/surfrdf`, accessed 2020-08-11.

should kindly refer to the published paper [47]. To summarize the conclusions of the comparison, in case of multi-user applications with non-trivial business logic, support for transactions (*GC1*), query result mapping (*GC3*), detached objects (*GC5*), and inheritance mapping (*MC1*) are crucial for any OTM library. Explicit treatment of inferred knowledge (*OC1*) would be important in scenarios where the mapped data are involved in inference. Table 4.1 presents the comparison results for these selected criteria, the following paragraphs then provide further details.

GC1 – Transactions. AutoRDF, JAOB, and the Semantic Framework do not support transactions likely due to their lack of support for triple stores in general. While ActiveRDF, RDFReactor, and Spira do support access to regular triple stores and, for example, RDFReactor internally makes use of the store's transactional mechanism, they do not allow the programmer to control the transactions externally. Empire's API hints towards support for transactional processing, but its implementation is rather strange. Operations that insert/update/remove data are automatically committed without any way of preventing this behavior. So, for example, it is not possible to bind updates to multiple entities into one transaction. This is analogous to the auto-commit feature of many databases. However, the auto-commit is implemented at the storage connection level, where queries in the native language of the storage (e.g., SQL, SPARQL) are used mainly by administrators, and it can be disabled. AliBaba, KOMMA, and RDFBeans support transactions by relying on the underlying storage's transaction management. SuRF, on the other hand, tracks changes to objects by marking their updated attributes as dirty and allows the programmer to commit the changes manually. JOPA handles transactions by creating clones of the manipulated objects and tracking their changes locally, pushing them into the storage on commit.

GC3 – Query Result Mapping. In contrast to JAOB, which does not provide any query API at all, AutoRDF, RDFBeans, the Semantic Framework, and Spira do not provide a query API either but contain at least a find all method, which allows retrieving all instances of the specified type. SuRF allows to execute arbitrary SPARQL queries, but cannot map their results to entities. While ActiveRDF does not support mapping SPARQL query results to entities, its advanced object-based query API obviates this issue by supporting almost complete SPARQL grammar. Finally, AliBaba, Empire, JOPA, and KOMMA support mapping SPARQL query results to entities by allowing specification of the target entity class when retrieving query results.

GC5 – Detached Objects. ActiveRDF, AliBaba, AutoRDF, KOMMA, RDFReactor, Spira, and SuRF do not support detached objects because their entities act as proxies which load attribute values from the repository when they are accessed. Conversely, setting attribute values causes the changes to be written into the storage

immediately (except for SuRF which tracks changes locally). Therefore, they have to hold onto a connection to the storage to provide basic data access functions. Empire, JAOB, JOPA, RDFBeans, and the Semantic Framework, on the other hand, allow working with the entities completely independently of the persistence context from which they were retrieved because they store the data in the actual objects.

MC1 – Inheritance Mapping. Inheritance mapping is probably the most complex feature in the comparison framework, with many subtle issues. Despite this, several evaluated libraries take a straightforward approach which can often lead to unexpected results. Consider Empire which does support multiple inheritance in that it is able to work with classes that inherit mapped getters/setters[5] from multiple interfaces. However, it is unable to persist more than one type for an entity. So if each interface declares a mapped type, only one of them gets persisted. KOMMA and RDFBeans suffer from the same issue, i.e., they correctly interpret inherited attributes, but always persist only a single type. To illustrate the issue, let us have interfaces Student and Employee, which are mapped to RDFS classes `ex:Student` and `ex:Employee` respectively. Then, declare an entity class WorkingStudent, which implements both Student and Employee. Persisting an object `c` of type WorkingStudent into the repository would result in either `ex:c rdf:type ex:Student` or `ex:c rdf:type ex:Employee` being inserted, depending on the order of declarations in the implements clause of WorkingStudent, where `c` is mapped to `ex:c`. Conversely, a straightforward implementation of loading an instance of WorkingStudent would require the corresponding resource to have types `ex:Student` and `ex:Employee`, mapping WorkingStudent to an unnamed intersection of the two, which is inconsistent with the persist strategy. In addition, this behavior is not formalized in any way. The Semantic Framework appears to also support multiple inheritance thanks to its use of Java classes and interfaces. However, the implementation extracts only attributes declared in a particular class, without regard for any fields inherited from its superclass. JAOB does not support inheritance mapping. SuRF allows to specify multiple superclasses when loading a resource, but they do not represent mapped ontological classes, they are regular Python classes adding custom behavior to the instance. On the other hand, similarly to ActiveRDF, since loaded instances contain attributes corresponding to all properties found on the resource (thanks to Python being a dynamic language), the concept of inheritance mapping does not apply in this case. JOPA currently supports only class-based inheritance, so mapping classes with multiple supertypes is not possible. Similarly, RDFReactor and Spira support only single type inheritance. AliBaba and AutoRDF are thus the only libraries fully supporting class hierarchy mapping. Since AliBaba is a Java library, it relies on interfaces to support multiple inheritance. AutoRDF is able to exploit the built-in multiple inheritance support of C++.

[5] Methods for retrieving/setting attribute values.

OC1 – Explicit Inference Treatment. JOPA allows marking entity attributes as potentially containing inferred data. Currently, this makes them effectively read-only. However, in the future, the implementation may permit additive changes. AliBaba, KOMMA, and RDFReactor allow marking attributes as read-only, a mechanism that could be used to prevent attempts at modification of inferred statements.

Overall, the evaluated libraries may be split into several groups based on different points of view. One contains solutions which appear to be proof-of-concept prototypes with limited functionality. This group contains AutoRDF, JAOB, and the Semantic Framework – their usability is restricted to a rather small set of specific cases. Another group consists of tools in which entities are merely proxies of the underlying RDF data. Such libraries could be problematic especially in larger (Web) applications because they generally do not support detached objects. On the other hand, thanks to the fact that they use dynamically generated proxies, they are able to support a wide variety of shared entity behavior. In the case of ActiveRDF and SuRF, this is enhanced by the fact that they do not require a predefined object model. The last group may be denoted as general-purpose persistence libraries and it contains Empire, JOPA, and RDFBeans. They attempt to provide functionality similar to JPA but in the world of ontologies.

Benchmark Results

Since the benchmark is Java-specific and requires the OTM libraries to support GraphDB, of the twelve tools examined in the qualitative part of the comparison, only five were evaluated in the benchmark – AliBaba, Empire, JOPA, KOMMA, and RDFBeans. The benchmark framework itself is scalable, so a size requiring reasonable running time in the test environment was configured. Moreover, to provide statistically sound results [105], each operation was executed in three runs, where each run consisted of twenty warm-up rounds, which were not measured, and 100 measured rounds. Each run was executed in a new Java virtual machine process. The repository was cleared after each round and GraphDB was restarted between the operations. The benchmark was run for several values of heap size (32 MB, 64 MB, 128 MB, 256 MB, 512 MB, 1 GB) to illustrate memory scalability of the OTM tools. The experiments were conducted on the following setup:

- OS: Linux Mint 17.3 64-bit
- CPU: Intel Core i5-750 2.67 GHz (4 cores)
- RAM: 16 GB
- Disk: Samsung SSD 850 EVO 250 GB

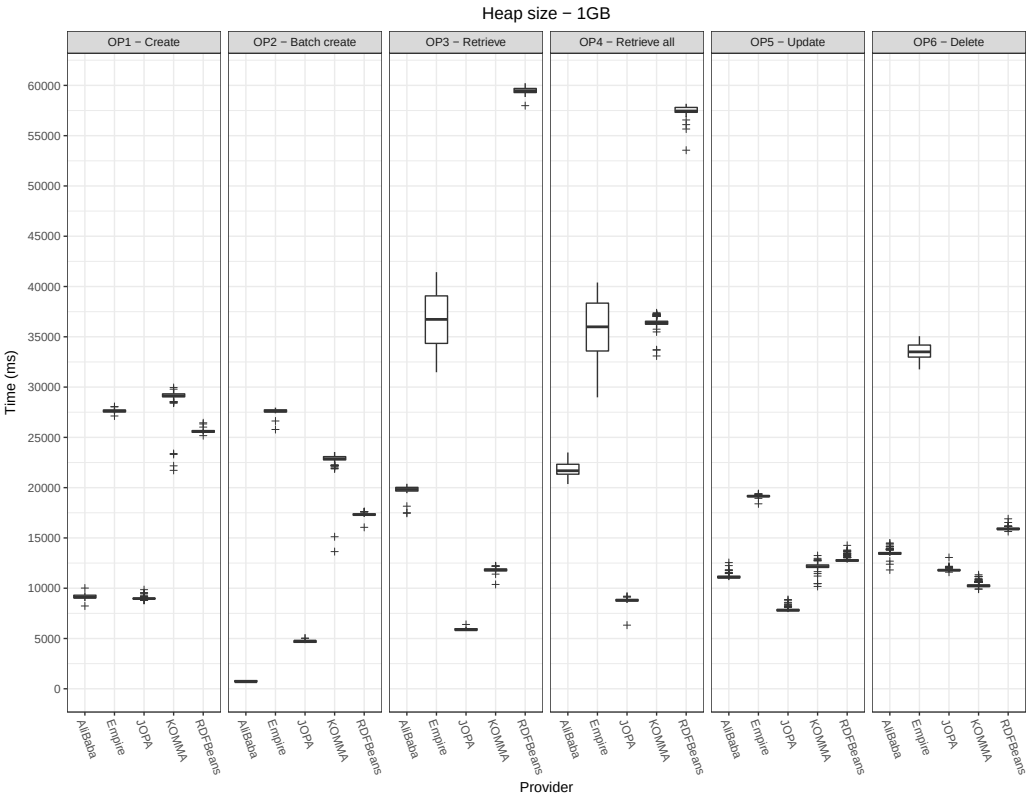


Figure 4.3: Performance of the individual libraries on a 1 GB heap. The plots are grouped by the respective operations. Lower is better. Taken from [47].

- Java: Java HotSpot JDK[6] 8u161, 64-bit
- GraphDB 8.4.1 Free with disabled inference

Figure 4.3 shows performance of each library in each operation with 1 GB heap size. It can be seen that JOPA exhibited the most consistent performance, in most cases outperforming the other libraries.

It turns out that the OTM solutions are not much sensitive to heap size tuning. As long as the heap is large enough for the data to fit in memory, the OTM libraries perform consistently over varying heap sizes. To gain a better insight into the memory efficiency of the OTM libraries, an additional benchmark was performed, in which the libraries performed an equivalent of *OP1*, *OP4*, *OP5*, and *OP6* in a cycle for a given period. More specifically, the test period was set to four hours, with a heap size of 40 MB. After this, Java garbage collector output was analyzed using *GCViewer*.[7] Simplified results of this analysis are shown in Table 4.2.

[6] Java Development Kit

[7] `https://github.com/chewiebug/GCViewer`, accessed 2020-08-11.

Measure	AliBaba	Empire	JOPA	KOMMA	RDFBeans
GCT (s)	496.98	1 326.58	147.78	65.36	40.4
Throughput (%)	96.37	84.56	98.97	99.55	99.72

Table 4.2: Memory utilization summary in a benchmark running for four hours with 40 MB heap. *GCT* is the total time spent in garbage collection and *Throughput* is the corresponding application throughput.

Complete results of the benchmark are published online.[8] The benchmark itself is published online as well,[9] including prototypes showcasing usage of the evaluated OTM libraries.

4.2 Formal Object-ontological Mapping

A formal description of object-ontological mapping allows to precisely describe the contract between the ontology and the object model. Such a description provides a degree of predictability of the Semantic data access, which, as could have been seen in Section 4.1, is often lacking in contemporary Semantic persistence libraries. The choice of mapping OWL ontologies restricted by integrity constraints makes the task both simpler and more difficult at the same time. Object-triple mapping for RDF(S) is, to a certain extent, simple – map RDFS classes to entity classes, RDF properties to attributes, and RDFS class instances to entities. However, this simplicity also brings a lot of uncertainty. For example, properties are first-class citizens in RDF (and in OWL as well), whereas attributes existentially depend on the class in which they are declared in object-oriented programming languages. Therefore, domain and range restrictions are often too general for an object model and their attachment to specific entity classes is purely conventional.

OWL with integrity constraints allows to overcome such problems by localizing domain and range restrictions on properties to particular concepts. For instance,

$$OccurrenceReport \sqsubseteq \forall documents.Occurrence$$

restricts the range of the property `documents` to `Occurrence` only locally for the concept `OccurrenceReport`. On the other hand, OWL is an expressive language with various constructs that have no direct counterpart in programming languages. For example, such a basic axiom as the above local range restriction requires a

[8] `https://kbss.felk.cvut.cz/web/kbss/otm-benchmark`, accessed 2020-08-11.
[9] `https://github.com/kbss-cvut/otm-benchmark`, accessed 2020-08-11.

mapping specifying that it will become an attribute of type Occurrence in the entity class OccurrenceReport. Advanced constructs like anonymous concepts created by concept intersections are even more complicated. The form of the ontology-object model contract is such that the object model is based on *compile-time integrity constraints* [39]. The mapping then ensures that ABox axioms are correctly interpreted as instance data and TBox axioms can be used to infer additional knowledge (mainly concept membership).

Now, on the one hand, OWL ontologies are based on description logics, so one would like to take advantage of the formal apparatus backing these languages. In contrast, mainstream object-oriented programming languages are not based on any logical formalism. Fortunately, F-logic has been specifically designed to allow representing structural aspects of object-oriented languages [29]. Therefore, the vital part of this OOM formalism deals with a mapping between description logics and F-logic, whereas the transition between F-logic and a selected programming language should be relatively smooth.

4.2.1 Mapping between Description Logics and F-logic

The description of the F-logic-based object-ontological mapping can be split into three parts:

1. Ontology mapping – mapping of the TBox and ABox
2. Mapping of integrity constraints
3. Finding means of integrity constraint validation in F-logic

A simple running example will be used to illustrate the mapping, with $\mathcal{O}_T$ representing the DL ontology schema, $\mathcal{O}_A$ the actual data and $\mathcal{IC}$ being integrity constraints placed on the ontology. In the example, an asset is declared and specified to have an author. It may also have a last editor. This generic ontology is restricted by integrity constraints for a safety occurrence reporting system. The constraints specify that reports are kinds of assets, they declare the same cardinalities of both author and last editor as $\mathcal{O}_T$, but require their values to be users of the system.

$$\begin{aligned}
\mathcal{O}_T =& \{Asset \sqsubseteq =1\, author.\top, Asset \sqsubseteq \leqslant 1\, lastEditor.\top, \\
& Report \sqsubseteq Asset, author \sqsubseteq editor, lastEditor \sqsubseteq editor\} \\
\mathcal{O}_A =& \{User(Tom), User(Sarah), Report(report{-}buo01)\} \\
\mathcal{IC} =& \{Report \sqsubseteq \forall author.User, Report \sqsubseteq =1\, author.User, \\
& Report \sqsubseteq \forall lastEditor.User, Report \sqsubseteq \leqslant 1\, lastEditor.User\}
\end{aligned}$$

$\mathcal{SROIQ}$	F-logic	F-logic Semantics
A	$A_\mathcal{C}$	
$\neg C$	$Not(C_\mathcal{C})$	$\mathbf{I} \models^F x{:}Not(C_\mathcal{C})$ iff $I_\mathcal{F}(x) \notin_U I_F(C_\mathcal{C})$
$C \sqcap D$	$C_\mathcal{C}$ and $D_\mathcal{C}$	
$C \sqcup D$	$C_\mathcal{C}$ or $D_\mathcal{C}$	
$\geqslant n\, R.C$	$AtLeast(n, R_\mathcal{R}, C_\mathcal{C})$	$\mathbf{I} \models^F x{:}AtLeast(n, R_\mathcal{R}, C_\mathcal{C})$ iff $\exists y_1 ... y_n$ s.t. $y_i \in I_\rightarrow(I_F(R_\mathcal{R}))(I_F(x)) \wedge y_i \in_U I_F(C_\mathcal{C})$, for $\neg(y_i = y_j)$
$\exists R.Self$	$HasSelf(R_\mathcal{R})$	$\mathbf{I} \models^F x{:}HasSelf(R_\mathcal{R})$ iff $I_F(x) \in I_\rightarrow(I_F(R_\mathcal{R}))(I_F(x))$
$\{a\}$	$Nom(a_\mathcal{E})$	$\mathbf{I} \models^F x{:}Nom(a_\mathcal{E})$ iff $I_\mathcal{F}(x) = I_\mathcal{F}(a_\mathcal{E})$
$\forall R.C$	$All(R_\mathcal{R}, C_\mathcal{C})$	$\mathbf{I} \models^F x{:}All(R_\mathcal{R}, C_\mathcal{C})$ iff $\forall y,\ y \in I_\rightarrow(I_F(R_\mathcal{R}))(I_F(x)) \Rightarrow y \in_U I_F(C_\mathcal{C})$
$\exists R.C$	$Some(R_\mathcal{R}, C_\mathcal{C})$	$\mathbf{I} \models^F x{:}Some(R_\mathcal{R}, C_\mathcal{C})$ iff $\exists y$ s.t. $y \in I_\rightarrow(I_F(R_\mathcal{R}))(I_F(x)) \wedge y \in_U I_F(C_\mathcal{C})$

Table 4.3: Mapping of concept descriptions. x is universally quantified over $\mathcal{E}$, y_i is quantified over $U_\mathcal{E}$. $X_\mathcal{C}$ ($X_\mathcal{R}$) represents a concept (method) name, i.e., a function symbol from $\mathcal{C}$ ($\mathcal{R}$). `AtMost` is defined analogously to `AtLeast` and corresponds to $\leqslant n\, R.C$.

Ontology Mapping

The mapping is inspired by [68], but supports a more expressive DL. The use of sorted F-logic and the proof of entailment equivalence are based on [69]. Table 4.3 shows mapping of concept descriptions. Similar to [68], several new function symbols are introduced – `Not`, `AtLeast`, `AtMost`, `HasSelf`, `Nom`, `All`, `Some` $\in \mathcal{A}$ – which allow representing $\mathcal{SROIQ}$ concept constructs which cannot be directly mapped to F-logic. For instance, $\geqslant nR.C$ does not correspond to $[R_\mathcal{R} \Rightarrow_{\{n:*\}} C_\mathcal{C}]$ because the $\mathcal{SROIQ}$ version admits also R-fillers of other types than C, whereas the F-logic signature expression would require all $R_\mathcal{R}$-fillers to belong to $C_\mathcal{C}$. Also, signature expressions cannot be used to infer the type of values of the corresponding data expressions. For each of the new function symbols, a condition on the underlying F-structures ensuring correct semantics w.r.t. their $\mathcal{SROIQ}$ counterparts is specified. W.l.o.g., concrete role-based concept descriptions like $\forall T.d$ are omitted here for the sake of brevity.

$\mathcal{SROIQ}$ top (bottom) concept $\top$ ($\bot$) is mapped to an F-logic concept $\top_\mathcal{C}$ ($\bot_\mathcal{C}$) for which it must hold $\forall x \in U_\mathcal{E},\ x \in_U I_F(\top_\mathcal{C})$ ($\forall x \in U_\mathcal{E},\ x \notin_U I_F(\bot_\mathcal{C})$). Similarly, $\mathcal{SROIQ}$ universal role R_U is mapped to an F-logic method $M_\mathcal{R}$ such that $\forall x, y \in U_\mathcal{E},\ y \in I_\rightarrow(I_F(M_\mathcal{R}))(x)$.

TBox and RBox axiom mapping is shown in Table 4.4. It makes use of F-logic predicates and defines conditions under which they are true.

$\mathcal{SROIQ}$	F-logic	Condition on $\mathbf{I}$
$C \sqsubseteq D$	$C_\mathcal{C} :: D_\mathcal{D}$	$I_F(C_\mathcal{C}) \preceq_U I_F(D_\mathcal{C})$
$R \sqsubseteq S$	$subPropertyOf_\mathcal{P}(R_\mathcal{R}, S_\mathcal{R})$	$y \in I_\rightarrow(I_F(R_\mathcal{R}))(x) \Rightarrow y \in I_\rightarrow(I_F(S_\mathcal{R}))(x)$
$Sym(R)$	$Sym_\mathcal{P}(R_\mathcal{R})$	$y \in I_\rightarrow(I_F(R_\mathcal{R}))(x) \Rightarrow x \in I_\rightarrow(I_F(R_\mathcal{R}))(y)$
$Asy(R)$	$Asy_\mathcal{P}(R_\mathcal{R})$	$y \in I_\rightarrow(I_F(R_\mathcal{R}))(x) \Rightarrow x \notin I_\rightarrow(I_F(R_\mathcal{R}))(y)$
$Tra(R)$	$Tra_\mathcal{P}(R_\mathcal{R})$	$y \in I_\rightarrow(I_F(R_\mathcal{R}))(x) \wedge z \in I_\rightarrow(I_F(R_\mathcal{R}))(y)$ $\Rightarrow z \in I_\rightarrow(I_F(R_\mathcal{R}))(x)$
$Ref(R)$	$Ref_\mathcal{P}(R_\mathcal{R})$	$x \in I_\rightarrow(I_F(R_\mathcal{R}))(x)$
$Irr(R)$	$Irr_\mathcal{P}(R_\mathcal{R})$	$x \notin I_\rightarrow(I_F(R_\mathcal{R}))(x)$
$Dis(R,S)$	$Dis_\mathcal{P}(R_\mathcal{R}, S_\mathcal{R})$	$y \notin I_\rightarrow(I_F(R_\mathcal{R}))(x) \vee y \notin I_\rightarrow(I_F(S_\mathcal{R}))(x)$

Table 4.4: Mapping of $TBox$ and $RBox$ axioms. $RBox$ axioms are mapped to predicates, for which satisfaction conditions on the F-structure $\mathbf{I}$ are provided. $\Rightarrow$ outside of an F-molecule represents regular logical implication. Variables are universally quantified over $U_\mathcal{E}$.

ABox individual assertions are mapped straightforwardly, $C(a)$ as an is-a assertion $a_\mathcal{E} : C_\mathcal{C}$, $R(a,b)$ as a data expression $a_\mathcal{E}[R_\mathcal{R} \rightarrow b_\mathcal{E}]$ and (in)equality $a = b$ ($a \neq b$) as $a_\mathcal{E} = b_\mathcal{E}$ ($a_\mathcal{E} \neq b_\mathcal{E}$).

Running Example. The running example should help illustrate the mapping. The corresponding F-logic ontology $\mathcal{O}^F$ looks as follows:

$$\begin{aligned}\mathcal{O}^F_T = \{&Asset_\mathcal{C} :: Some(author_\mathcal{R}, \top_\mathcal{C}), Asset_\mathcal{C} :: AtMost(1, author_\mathcal{R}, \top_\mathcal{C}),\\ &Asset_\mathcal{C} :: AtMost(1, lastEditor_\mathcal{R}, \top_\mathcal{C}), Report :: Asset,\\ &subPropertyOf_\mathcal{P}(author_\mathcal{R}, editor_\mathcal{R}),\\ &subPropertyOf_\mathcal{P}(lastEditor_\mathcal{R}, editor_\mathcal{R})\}\\ \mathcal{O}^F_A = \{&Tom_\mathcal{E} : User_\mathcal{C}, Sarah_\mathcal{E} : User_\mathcal{C}, report\text{-}buo01_\mathcal{E} : Report_\mathcal{C}\}\end{aligned}$$

Now, it has to be shown that the mapping preservers entailment. First, I show that a formula θ is satisfiable in a $\mathcal{SROIQ}$ language $\mathcal{L}^{DL}$ if and only if a corresponding formula θ^F is satisfiable in a corresponding F-logic language $\mathcal{L}^F$.

Lemma 4.1. *Let θ be a formula in $\mathcal{L}^{DL}$ and θ^F a corresponding F-logic formula in an F-logic language $\mathcal{L}^F$. Then θ is satisfiable in some interpretation $\mathcal{I}$ of $\mathcal{L}^{DL}$ if and only if θ^F is satisfiable in some F-structure $\mathbf{I}$ of $\mathcal{L}^F$.*

Proof. (Sketch) The lemma is proven by showing how an F-structure $\mathbf{I}$ can be constructed for a $\mathcal{SROIQ}$ interpretation $\mathcal{I}$ and vice versa. The interpretation correspondence is shown for RBox and TBox axioms, with ABox axioms being *internalized* using TBox. The full proof can be found in Appendix B.1. □

The lemma allows to show that entailment is preserved by the mapping.

Theorem 4.2. *Let Θ and Θ^F be corresponding theories in $\mathcal{L}^{DL}$ and $\mathcal{L}^F$. For any formula θ in $\mathcal{L}^{DL}$ holds:*

$$\Theta \models \theta \text{ iff } \Theta^F \models^F \theta^F,$$

where $\models^F$ represents F-logic entailment.

Proof. The proof relies on the fact that entailment checking can be reduced to satisfiability checking, that is, $\Theta \models \theta$ ($\Theta^F \models^F \theta^F$) can be reduced to verification that $\Theta \wedge \neg\theta$ ($\Theta^F \wedge \neg\theta^F$) is not satisfiable. Since Lemma 4.1 showed satisfiability of a formula is equivalent if both $\mathcal{SROIQ}$ and F-logic, it is clear that entailment checking is also equivalent. □

Mapping Integrity Constraints

IC semantics allows imposing a closed world view on a portion of a knowledge base $\mathcal{K}$ affected by the integrity constraints. Analogously to the approach of Tao et al. [27], an augmented F-structure $\mathbf{I}^{IC}$ with IC semantics is defined below. This approach has the advantage of not introducing additional syntactic constructs and giving the IC axioms a natural, easy-to-understand meaning. The semantics uses the notion of *minimal equality models* (Mod_{ME}) to support a *weak* form of unique name assumption. The original definition of Mod_{ME} from [27] can be carried over to F-logic as follows:

Consider a knowledge base $\mathcal{K}^F$ and let $E_{\mathbf{I}}$ be the set of equality relations satisfied by an F-structure $\mathbf{I}$, i.e., $E_{\mathbf{I}} = \{\langle a, b\rangle \mid a, b \in \mathcal{E} \text{ s.t. } \mathbf{I} \models^F I_{\mathcal{F}}(a) = I_{\mathcal{F}}(b)\}$. A relation $\mathbf{I} \prec^F_{=} \mathbf{J}$, where $\mathbf{I}$ and $\mathbf{J}$ are F-structures, holds iff:

- $\forall C \in \mathcal{C} \cup \mathcal{A}$, if $\mathbf{I} \models^F a{:}C$, then $\mathbf{J} \models^F a{:}C$,
- $\forall R \in \mathcal{R}$, if $\mathbf{I} \models^F a[R \rightarrow b]$, then $\mathbf{J} \models^F a[R \rightarrow b]$,
- $E_{\mathbf{I}} \subset E_{\mathbf{J}}$.

$Mod^F_{ME}(\mathcal{K}^F)$ is then defined as $Mod^F_{ME}(\mathcal{K}^F) = \{\mathbf{I} \mid \mathbf{I} \text{ is a model of } \mathcal{K}^F \text{ s.t. } \nexists \mathbf{J}\ \mathbf{J} \prec^F_{=} \mathbf{I}\}$.

The augmented F-structure with IC semantics is a tuple $\mathbf{I}^{IC} = \langle U, \prec_U^{IC}, \in_U^{IC}, I_{\mathcal{F}}, I_{\mathcal{P}}^{IC}, I_{\rightarrow}^{IC}, I_{\Rightarrow} \rangle$, where:

- $\prec_U^{IC} = \{\langle I_{\mathcal{F}}(x), I_{\mathcal{F}}(y)\rangle \mid x, y \in \mathcal{C} \text{ s.t. } \forall \mathcal{J} \in Mod_{ME}^{F}(\mathcal{K}^F), \mathcal{J} \models^F I_{\mathcal{F}}(x) \prec_U I_{\mathcal{F}}(y)\}$,
- $\in_U^{IC} = \{\langle I_{\mathcal{F}}(x), I_{\mathcal{F}}(y)\rangle \mid x \in \mathcal{E}, y \in \mathcal{C} \cup \mathcal{A} \text{ s.t. } \forall \mathcal{J} \in Mod_{ME}^{F}(\mathcal{K}^F), \mathcal{J} \models^F I_{\mathcal{F}}(x) \in_U I_{\mathcal{F}}(y)\}$,
- $I_{\mathcal{F}}(y) \in I_{\rightarrow}^{IC}(I_{\mathcal{F}}(z))(I_{\mathcal{F}}(x))$ iff $x, y \in \mathcal{E}, z \in \mathcal{R} \wedge \forall \mathcal{J} \in Mod_{ME}^{F}(\mathcal{K}^F), \mathcal{J} \models^F I_{\mathcal{F}}(y) \in I_{\rightarrow}^{IC}(I_{\mathcal{F}}(z))(I_{\mathcal{F}}(x))$,
- $I_{\mathcal{P}}^{IC}(p) = \{\langle I_{\mathcal{F}}(y_1), ..., I_{\mathcal{F}}(y_n)\rangle \mid y_i \in \mathcal{F} \text{ s.t. } \forall \mathcal{J} \in Mod_{ME}^{F}(\mathcal{K}^F), \mathcal{J} \models^F \langle I_{\mathcal{F}}(y_1), ..., I_{\mathcal{F}}(y_n)\rangle \in I_{\mathcal{P}}^{IC}(p)\}$, where n is the arity of p,
- And the other parts of $\mathbf{I}^{IC}$ are the same as in a regular F-structure.

Based on $\mathbf{I}^{IC}$, IC semantics of concept descriptions can now be defined in Table 4.5. One modification is the switch from $All(R_{\mathcal{R}}, C_{\mathcal{C}})$ to the built-in signature expression $[R_{\mathcal{R}} \Rightarrow C_{\mathcal{C}}]$. This can be done thanks to the F-logic notion of *typing*, which requires data expressions to correspond to a signature expression declaring their types, e.g., for a signature $C_{\mathcal{C}}[R_{\mathcal{R}} \Rightarrow D_{\mathcal{C}}]$, typing rules require d from the data expression $c[R_{\mathcal{R}} \rightarrow d]$ to be of type $D_{\mathcal{C}}$, i.e., $d{:}D_{\mathcal{C}}$. Typing is an optional, non-monotonic, part of F-logic. It is utilized for IC declaration because it provides a nice, succinct, frame-based syntax. Moreover, `AtLeast` and `AtMost` restrictions combined with universal role restriction can be expressed even more concisely using F-logic cardinality restriction on signature expressions. The running example below demonstrates such a common combination.

Finally, IC semantics of axioms should follow trivially from the definitions in Table 4.5.

Running Example. Reviewing the running example, the biggest change is the use of method signatures with cardinality constraints. This significantly reduces the verbosity of the ICs and makes them arguable easier to understand.

$$\mathcal{IC}^F = \{Report_{\mathcal{C}}[author_{\mathcal{R}} \Rightarrow_{\{1:1\}} User_{\mathcal{C}}; lastEditor_{\mathcal{C}} \Rightarrow_{\{0:1\}} User_{\mathcal{C}}]\}$$

Figure 4.4 illustrates how an integrity constraints-based model from the running example may look in terms of a UML class diagram.

Concept	$\mathbf{I}^{IC} \models^{F} x{:}Concept$ iff
$Not(C_{\mathcal{C}})$	$x \in \mathcal{E} \wedge I_{\mathcal{F}}(x) \notin_{U}^{IC} I_{\mathcal{F}}(C_{\mathcal{C}})$
$C_{\mathcal{C}}$ and $D_{\mathcal{C}}$	$x \in \mathcal{E} \wedge I_{\mathcal{F}}(x) \in_{U}^{IC} I_{\mathcal{F}}(C_{\mathcal{C}}) \wedge I_{\mathcal{F}}(x) \in_{U}^{IC} I_{\mathcal{F}}(D_{\mathcal{C}})$
$C_{\mathcal{C}}$ or $D_{\mathcal{C}}$	$x \in \mathcal{E} \wedge I_{\mathcal{F}}(x) \in_{U}^{IC} I_{\mathcal{F}}(C_{\mathcal{C}}) \vee I_{\mathcal{F}}(x) \in_{U}^{IC} I_{\mathcal{F}}(D_{\mathcal{C}})$
$AtLeast(n, R_{\mathcal{R}}, C_{\mathcal{C}})$	$x \in \mathcal{E} \wedge \exists y_1, ...y_n \in \mathcal{E}$ s.t. $I_{\mathcal{F}}(y_i) \in I_{\rightarrow}^{IC}(I_{\mathcal{F}}(R_{\mathcal{R}}))(I_{\mathcal{F}}(x))$ $\wedge I_{\mathcal{F}}(y_i) \in_{U}^{IC} I_{\mathcal{F}}(C_{\mathcal{C}}) \wedge \neg(I_{\mathcal{F}}(y_i) = I_{\mathcal{F}}(y_j))$
$HasSelf(R_{\mathcal{R}})$	$x \in \mathcal{E} \wedge I_{\mathcal{F}}(x) \in I_{\rightarrow}^{IC}(I_{\mathcal{F}}(R_{\mathcal{R}}))(I_{\mathcal{F}}(x))$
$Nom(a_{\mathcal{E}})$	$x \in \mathcal{E} \wedge I_{\mathcal{F}}(x) = I_{\mathcal{F}}(a_{\mathcal{E}})$
$[R_{\mathcal{R}} \Rightarrow D_{\mathcal{C}}]$	$\mathbf{I}^{IC}$ is a *typed* F-structure [29] (Sec. 13)
$Some(R_{\mathcal{R}}, C_{\mathcal{C}})$	$x \in \mathcal{E} \wedge \exists y \in \mathcal{E}$ s.t. $I_{\mathcal{F}}(y) \in I_{\rightarrow}^{IC}(I_{\mathcal{F}}(R_{\mathcal{R}}))(I_{\mathcal{F}}(x))$ $\wedge I_{\mathcal{F}}(y) \in_{U}^{IC} I_{\mathcal{F}}(C_{\mathcal{C}})$

Table 4.5: Integrity constraint semantics of F-logic concept descriptions. The right hand column specifies a condition under which an individual x is an instance of the concept specified in the left hand column under the IC semantics.

Figure 4.4: UML class diagram of a model based on integrity constraints from the running example.

Integrity Constraint Validation

Being able to define integrity constraints for an ontology is hardly enough. It is necessary to be able to validate them as well. While IC semantics is a convenient theoretical construct, it is impractical, because existing F-logic implementations do not support it. Instead, the ability to query an F-logic knowledge base can be exploited for integrity constraint validation.

Since ICs represent a close-world view of the ontology, *negation as failure* (NAF) is necessary to be able to represent it in the queries. Like in most logic programming languages [106], the NAF operator **not** with the following semantics can be introduced into F-logic: $\mathcal{K}^{F} \models \mathbf{not}(\alpha)$ iff $\mathcal{K}^{F} \not\models \alpha$, where α is an F-formula.

I now show how IC axioms can be translated to F-logic queries with **not**. The rationale is that if the knowledge base entails the query, there is an IC violation. The approach again follows the line of reasoning of Tao et al. [27], who introduce two translation operators: $\mathcal{T}_{C}$ for concepts and $\mathcal{T}$ for axioms. The definition of analogous F-logic translation operators $\mathcal{T}_{C}^{F}$ and $\mathcal{T}^{F}$ can be found in Tables 4.6 and 4.7 respectively.

Concept	$\mathcal{T}_C^F$
$\mathcal{T}_C^F(x{:}C_A)$	$x{:}C_A$
$\mathcal{T}_C^F(x{:}Not(C))$	$\mathbf{not}(x{:}\mathcal{T}_C^F(C))$
$\mathcal{T}_C^F(x{:}(C_1 \text{ and } C_2))$	$x{:}\mathcal{T}_C^F(C_1) \wedge x{:}\mathcal{T}_C^F(C_2)$
$\mathcal{T}_C^F(x{:}(C_1 \text{ or } C_2))$	$x{:}\mathcal{T}_C^F(C_1) \vee x{:}\mathcal{T}_C^F(C_2)$
$\mathcal{T}_C^F(x{:}AtLeast(n,R,C))$	$\bigwedge_{1\leq i\leq n} x[R \to y_i] \wedge y_i{:}\mathcal{T}_C^F(C) \bigwedge_{1\leq i\leq j\leq n} \mathbf{not}(y_i = y_j)$
$\mathcal{T}_C^F(x{:}HasSelf(R))$	$x[R \to x]$
$\mathcal{T}_C^F(x{:}Nom(a))$	$x = a$
$\mathcal{T}_C^F(x[R \Rightarrow C])$	$x[R \to y] \Rightarrow y{:}\mathcal{T}_C^F(C)$
$\mathcal{T}_C^F(x{:}Some(R,C))$	$x[R \to y] \wedge y{:}\mathcal{T}_C^F(C)$

Table 4.6: Integrity constraints validation transformation rules for concepts. C_A is an atomic class name.

Axiom	$\mathcal{T}^F$
$\mathcal{T}^F(C_1 :: C_2)$	$\mathcal{T}_C^F(x{:}C_1) \wedge \mathbf{not}(\mathcal{T}_C^F(x{:}C_2))$
$\mathcal{T}^F(subPropertyOf_{\mathcal{P}}(R_1,R_2))$	$x[R_1 \to y] \wedge \mathbf{not}(x[R_2 \to y])$
$\mathcal{T}^F(Sym_{\mathcal{P}}(R))$	$x[R \to y] \wedge \mathbf{not}(y[R \to x])$
$\mathcal{T}^F(Asy_{\mathcal{P}}(R))$	$x[R \to y] \wedge y[R \to x]$
$\mathcal{T}^F(Tra_{\mathcal{P}}(R))$	$x[R \to y] \wedge y[R \to z] \wedge \mathbf{not}(x[R \to z])$
$\mathcal{T}^F(Ref_{\mathcal{P}}(R))$	$\mathbf{not}(x[R \to x])$
$\mathcal{T}^F(Irr_{\mathcal{P}}(R))$	$x[R \to x]$
$\mathcal{T}^F(Dis_{\mathcal{P}}(R_1,R_2))$	$x[R_1 \to y] \wedge x[R_2 \to y]$

Table 4.7: Integrity constraints validation transformation rules for axioms. C_i is a concept, R_i is a role and x, y_i are variables.

The universal role restriction concept comes with a little twist. Instead of representing a standalone concept, a corresponding signature expression is attached to the target concept, i.e., instead of mapping a GCI $C \sqsubseteq \forall R.D$, we have directly $C_{\mathcal{C}}[R_{\mathcal{R}} \Rightarrow D_{\mathcal{C}}]$. A corresponding IC validation query in F-logic is created by verifying that data expressions of all instances of $C_{\mathcal{C}}$ comply with the signature expression, i.e., $\mathcal{T}_C^F(x[R_{\mathcal{R}} \Rightarrow D_{\mathcal{C}}])$. This can be also seen in the running example below.

Running Example. Since a signature expression with cardinality constraints is essentially a combination of multiple concept descriptions, it results in multiple validation queries. The queries presented below can be executed in F-logic implementations like FLORA-2,[10] which already supports the NAF operator. Presence of results for any of the queries indicates an IC violation. In this case, the constraints are violated by the lack of an explicit author of `report-buo01`, which is manifested by the second query below. Asserting an author, e.g., $report{-}buo01_{\mathcal{E}}[author_{\mathcal{R}} \to Tom_{\mathcal{E}}]$, would fix the IC violation.

[10] `http://flora.sourceforge.net/`, accessed 2020-08-11.

$$
\begin{aligned}
\mathcal{T}^F = \{ & x{:}Report \wedge x[author_{\mathcal{R}} \rightarrow y] \wedge \mathbf{not}(y{:}User_{\mathcal{C}}), \\
& x{:}Report_{\mathcal{C}} \wedge \mathbf{not}(x[author_{\mathcal{R}} \rightarrow y] \wedge y{:}User_{\mathcal{C}}), \\
& x{:}Report_{\mathcal{C}} \wedge x[author_{\mathcal{R}} \rightarrow \{y_1, y_2\}] \bigwedge_{1 \leq i \leq 2} y_i{:}User_{\mathcal{C}} \wedge \mathbf{not}(y_1 = y_2) \\
& x{:}Report \wedge x[lastEditor_{\mathcal{R}} \rightarrow y] \wedge \mathbf{not}(y{:}User_{\mathcal{C}}), \\
& x{:}Report_{\mathcal{C}} \wedge x[lastEditor_{\mathcal{R}} \rightarrow \{y_1, y_2\}] \bigwedge_{1 \leq i \leq 2} y_i{:}User_{\mathcal{C}} \wedge \mathbf{not}(y_1 = y_2)\}
\end{aligned}
$$

Finally, it remains to show that the validation queries faithfully represent the IC semantics, i.e., that validation queries generated from IC axioms return results whenever any of the IC axioms are violated by the knowledge base.

Theorem 4.3. *Consider a knowledge base $\mathcal{K}^F$, a set of integrity constraint axioms $\mathcal{IC}$ and a set of IC validation queries $\mathcal{Q}$, constructed by applying the translation operator $\mathcal{T}^F$ on each IC axiom α in $\mathcal{IC}$. If $\mathcal{K}^F$ violates any of the IC axioms in $\mathcal{IC}$, then $\exists\ q \in \mathcal{Q}$ such that $\mathcal{K}^F \models^F q$.*

Proof. (Sketch) The proof shows for RBox and TBox integrity constraint axioms that if there is a model which violates an IC axiom, it satisfies the corresponding IC validation query. Thus, integrity constraint checking can be reduced to query answering in F-logic. The full proof can be found in Appendix B.2. □

4.2.2 Mapping between F-logic and Programming Languages

The F-logic model represents a convenient intermediate step between a DL ontology and an object model in an object-oriented programming language (OOPL). However, from the description of the mapping so far, one may become reasonably skeptical of the simplicity of the mapping between F-logic and an OOPL. In this section, I will describe this mapping and illustrate its form on an example model in Java.

Mapping Principles

The mapping between an ontology and an application consists of two parts:

- An object model structure based on integrity constraints,

- Instance data (objects with attributes and relationships) based on the ontology ABox.

The role of the ontological schema (TBox) is mainly in that it may allow inferring additional instance knowledge.

Mapping Object Model Structure. Křemen and Kouba have shown [39] that not all IC axioms can be mapped to an object model directly. The main culprits are the inability of programming languages to declare anonymous classes based on restrictions of other existing classes and the fact that class attributes can be either singular or a collection of an unspecified number of values (so an arbitrary number cannot be by default provided). For example, it is not possible to map an IC axiom $A_{\mathcal{C}}[R_{\mathcal{R}} \Rightarrow (B_{\mathcal{R}} \text{ and } C_{\mathcal{R}})]$ to an OOPL directly, because it does not allow to declare an attribute with an anonymous type being an intersection of classes B and C. On the other hand, it can be argued that anonymous classes are less relevant for object models of most information systems and that it is possible to work around such a restriction. Precise cardinalities, as I will show presently, can be treated using different means.

Adhering to the terminology of Křemen and Kouba, the IC axioms relevant for an object model can be split into three categories:

Compile-time which can be built directly into the object model,

Run-time which can be checked efficiently by predefined procedures at execution time,

Reasoning-time which require validation by a reasoner.

My work extends their categorization of IC axioms translatable to OOPLs by showing how additional IC axioms can be mapped. Table 4.8 presents this extended categorisation. For compile-time constraints, the table also provides corresponding code examples in Java. Several comments can be made here. First, the $A_{\mathcal{C}}[R_{\mathcal{R}} \Rightarrow C_{\mathcal{C}}]$ and $A :: AtMost(1, R_{\mathcal{R}}, C_{\mathcal{C}})$ constraints are typically combined together to denote a singular attribute with a value of type $C_{\mathcal{C}}$ ($A_{\mathcal{C}}[R_{\mathcal{R}} \Rightarrow_{\{0,1\}} C_{\mathcal{C}}]$). Second, due to the lack of class-level multiple inheritance in languages like Java, superclass conjunction ($A_{\mathcal{C}} :: (B_{\mathcal{C}} \text{ and } C_{\mathcal{C}})$) requires the use of interfaces on the code level. Languages like C++, which support multiple inheritance for classes, do not need such a workaround.

Mapping Instance Data. Mapping instance data, as has been already shown, is relatively straightforward – individuals become instances of the respective classes,

IC Category	Constraint	Code Snippet
Compile-time	$A_{\mathcal{C}}[R_{\mathcal{R}} \Rightarrow C_{\mathcal{C}}]$	Set<C> r
	$A_{\mathcal{C}} :: AtMost(1, R_{\mathcal{R}}, C_{\mathcal{C}})$	C r
	$A_{\mathcal{C}} :: (B_{\mathcal{C}}$ and $C_{\mathcal{C}})$	interface A extends B, C
	$A_{\mathcal{C}} :: Nom(a_{\mathcal{E}})$	enum A {a}
Run-time	$A_{\mathcal{C}} :: AtLeast(n, R_{\mathcal{R}}, C_{\mathcal{C}})$	
	$A_{\mathcal{C}} :: AtMost(n, R_{\mathcal{R}}, C_{\mathcal{C}})$	
	$A_{\mathcal{C}} :: Some(R_{\mathcal{R}}, C_{\mathcal{C}})$	
	$A_{\mathcal{C}} :: HasSelf(R_{\mathcal{R}})$	
Reasoning-time	$A_{\mathcal{C}} :: (B_{\mathcal{C}}$ or $C_{\mathcal{C}})$	
	$A_{\mathcal{C}} :: Not(B_{\mathcal{C}})$	

Table 4.8: Mapping integrity constraints to OOPL structures. The *AtMost* run-time constraint mapping assumes $n > 1$. Code snippets are in Java.

the values of their ontological properties become values of their attributes (literal values or references to other instances). From a more technical standpoint, while the same individual can be an instance of multiple unrelated classes in an ontology, in an OOPL, each object has to have only one primary class.[11] However, there exist ways to treat multiple separate objects as the same using equality.

Running Example. The running example can now be finished by showing the resulting class Report with its attributes in Listing 4.1. Configuration of the mapping, e.g., to what ontological class entity class Report corresponds, is not expressed in this example, because various solutions use various methods and vocabularies, e.g., annotations in the case of Java. Note that the minimal cardinality on attribute author has to be checked separately as well, because it is a run-time constraint. A UML object diagram presenting data from an ABox $\mathcal{O}_A^{F\prime}$ is displayed in Figure 4.5. $\mathcal{O}_A^{F\prime}$ is an extension of the original ABox $\mathcal{O}_A^F$, demonstrating references between the objects:

$$\mathcal{O}_A^{F\prime} = \mathcal{O}_A^F \cup \{report\text{-}buo01_{\mathcal{E}}[author_R \rightarrow Tom_{\mathcal{E}};\ lastEditor_{\mathcal{R}} \rightarrow Sarah_{\mathcal{E}}]\}$$

Listing 4.1: Java class corresponding to the integrity constraint axioms of the running example. Note that mapping configuration is not specified here.

```
public class Report {
    // Identifier attribute

    private User author;

    private User lastEditor;

    // Additional attributes, getters, setters
}
```

[11] The most specific class of which it is an instance.

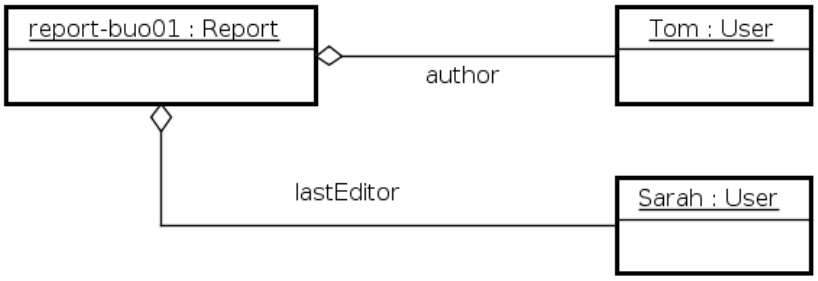


Figure 4.5: UML object diagram of the extended ABox $\mathcal{O}_A^{F\prime}$ from the running example.

4.3 Data Access Operations

Object-ontological mapping formalized in the previous section represents a static artifact consisting of a domain model and a snapshot of the data described by this model. This can be complemented by the definition of the operations performed on the knowledge base. The operations are not as much interesting in the sense of their effects on the data – as has been discussed, individual assertion mapping is relatively straightforward – but it provides an overview of the kinds of requests a repository backing an application using OOM will face. Therefore, arbitrary SPARQL (Update) statements are out of the scope of this section, the focus is on create, retrieve, update, and delete (CRUD) operations exploiting the domain model. In addition, as Chapter 5 will show, such operations represent the core of a data access layer API which allows separating (possibly) vendor-specific triple store access from generic object-ontological mapping code.

4.3.1 Definition of the Operations

The operations are described using DLs because that is the paradigm in which the data are ultimately stored. But an equivalent F-logic description could be provided. Consider thus a $\mathcal{SROIQ}(\mathcal{D})$ ontology $\mathcal{O}$ and a set of integrity constraints $\mathcal{IC}$. A *multi-context ontology* is a tuple $\mathcal{M} = (\mathcal{O}_d, \mathcal{O}_1, ..., \mathcal{O}_n)$, where each $\mathcal{O}_j$ is an ontology identified by a unique IRI (denoted by $IRI(\mathcal{O}_j)$) and is called a *context*. $\mathcal{O}_d$ denotes the *default context*. $\mathcal{M}$ is analogous to an RDF dataset with named graphs [2]. This structure is supported by most contemporary triple stores – including the notion of the default context, which is used in the operations when no specific context is selected. For the purpose of this section, assume that each axiom α in $\mathcal{O}_d \cup \bigcup_{j=1}^{n} \mathcal{O}_j$ has assigned a boolean status indicating whether it is asserted or inferred. While this technique has no formal foundation in DLs, it is used (in various forms) by virtually all contemporary triple stores mainly to facilitate axiom removal (remember that inferred axioms cannot be removed directly).

An *axiom descriptor* δ_a is a tuple $(i, \{(r_1, b_1, o_1), ..., (r_k, b_k, o_k)\})$, where $i \in N_I$, $r_m \in N_R$, $b_m \in \{0, 1\}$, o_m is a context IRI or it is empty (indicating the default context), and $m \in 1...k$. The b_ms indicate whether inferred values for the given role should be included as well. The axiom descriptor is used to specify data retrieval parameters. An *axiom value descriptor* δ_v is a tuple $(i, \{(r_1, v_1, o_1), ..., (r_k, v_k, o_k)\})$, where $i \in N_I$, $r_m \in N_R$, $v_m \in N_I \cup N_D$, o_m is an IRI of one of the ontologies in $\mathcal{M}$ or it is empty (indicating the default context), and $m \in 1...k$. The v_ms represent values for the given individual and role. The axiom value descriptor specifies data which should be inserted into the storage and is essentially an extension of an axiom descriptor where all inference markers b_m are set to 0 and values are provided for the roles.

The following list defines the five basic Semantic data access operations needed by OOM persistence libraries. Besides CRUD operations, there is also one allowing to validate reasoning-time integrity constraints.

- $find(\mathcal{M}, \delta_a)$: $2^{\mathcal{M}} \times N_I \times N_R^k \times \{0, 1\}^k \times \mathcal{M}^k \rightarrow 2^{N_I \times N_R \times (N_I \cup N_D)}$,
  - Given an individual i, find axioms representing assertions of its roles r_m specified by δ_a,
- $persist(\mathcal{M}, \delta_v)$: $\mathcal{M} \times \delta_v \rightarrow \mathcal{M}'$, where $\mathcal{M}' = (\mathcal{O}'_d, \mathcal{O}'_1, ..., \mathcal{O}'_n)$, where each $\mathcal{O}'_t = \mathcal{O}_t \cup \alpha_v$, with $\alpha_v = (i, r_s, v_s)$ such that $o_s = IRI(\mathcal{O}_t)$, where $t \in 1...n$ and $s \in 1...k$,
  - Insert axioms representing the specified values of roles r_m of individual i into the corresponding contexts,
- $remove(\mathcal{M}, \delta_a)$: $\mathcal{M} \times \delta_a \rightarrow \mathcal{M}'$, where $\mathcal{M}' = (\mathcal{O}'_d, \mathcal{O}'_1, ..., \mathcal{O}'_n)$, where each $\mathcal{O}'_t = \mathcal{O}_t \setminus \alpha_v$, with $\alpha_v = (i, r_s, x)$ such that $o_s = IRI(\mathcal{O}_t)$, where $t \in 1...n$, $s \in 1...k$, and $x \in (N_R \cup N_D)$,
  - Remove axioms representing values of roles r_m of individual i from the corresponding contexts,
- $update(\mathcal{M}, \delta_v)$: $persist(remove(\mathcal{M}, \delta_v), \delta_v)$,
  - Update $\mathcal{M}$ by first removing old values of roles r_s (where $s \in 1...k$) of individual i specified by δ_v and then inserting new values specified by δ_v,
- $validateIC(\mathcal{M}, \{\gamma_1, ..., \gamma_k\})$: $2^{\mathcal{M}} \times 2^{N_I \times N_R \times (N_I \cup N_D)} \times \mathcal{IC} \rightarrow \{0, 1\}$, where $\gamma_m \in \mathcal{IC}$ and $m \in 1...k$,
  - Validate reasoning-time integrity constraints from the set $\mathcal{IC}$.

4.3.2 Complexity Analysis

The formal definition of the CRUD operations from Section 4.3.1 can be used to estimate their computational complexity. This theoretical analysis has an important input parameter – the inference strategy. There are two main ways of approaching inference in a triple store:

- Materialization
- Query-time reasoning

Materialization

Materialization means that inference is performed on statement insertion and its results are in turn inserted into the storage as well. On insertion, a set of inferred statements K_I^0 is derived from the set of new statements K_E inserted into the knowledge base. K_I^0 is then also processed by the reasoner, generating a set K_I^1 of further inferences. For total materialization, the reasoning phase is repeated until a set of statements K_I^n yields no new results. Statement removal is even more interesting. For example, the triple store GraphDB [36] performs a combination of forward and backward chaining: in the forward chaining phase, inferences are found for the asserted statements being removed. These inferred results are then used as the starting point of a backward chaining procedure which checks whether there are any other explicit data from which the inferred knowledge can be derived. If not, the inferred statements need to be removed as well.

Materialization is thus theoretically favorable for read-only scenarios because the query engine can directly retrieve the data, but performance may suffer on insertion and removal. For more expressive languages, materialization can also cause major inflation of the database size. For example, a repository in one of our projects, with a relatively inexpressive ontological model, consists of approximately 300 000 asserted and 700 000 inferred statements.

Query-time Reasoning

Query-time reasoning does not materialize inference results and performs reasoning at the moment a query is executed. While most reasoners cache certain inferences, in

Strategy	T_{find}	$T_{persist}$	T_{remove}
Materialization	$O(log_b n)$	$O(\sum_{i=0}^{m} C_{Ri} \times log_b n)$	$O(\sum_{i=0}^{m} C_{Ri} \times log_b n)$
Query-time	$O(C_R) + O(log_b n)$	$O(log_b n)$	$O(log_b n)$

Table 4.9: Asymptotic time complexity of the selected data access operations for a materializing and query-time reasoning storage. b is the branching factor of the index B+ tree, n is the size of the dataset. C_R is the reasoning cost, which depends on the selected language expressiveness, and m is the number of reasoning cycles performed during materialization of statements inserted into the repository.

principle, reasoning has to be performed whenever answering a query. On the other hand, statement insertion or removal requires no reasoner involvement. In addition, inference expressiveness can be selected per query, whereas total materialization typically has to be configured with the language of choice before inserting any data. Stardog[12] is an example of a triple store supporting query-time reasoning.

Another important factor of the complexity analysis of operations over storage is its data indexing strategy. Most triple stores nowadays use B+ trees [107] (including GraphDB and Stardog), an evolution of B-trees [108]. These trees are built over different combinations of statement constituents – (s)ubject, (p)redicate, (o)bject, and (c)ontext. For the sake of simplicity, let me omit the statement context from the discussion. Since all indexes have to be updated when data change, triple stores attempt to balance the number of indexes needed for efficient querying and updating. For example, GraphDB uses PSO and POS indexes by default, whereas RDF4J uses SPOC and POSC.

From the structure of the axiom descriptor δ_a introduced in Section 4.3.1, it should be clear that the SPO and possibly PSO (predicates are bound in the descriptor) indexes are the most relevant. Assuming the presence of either of them, Table 4.9 presents the asymptotic complexity of $find$, $persist$ and $remove$ (*update* consists of $remove$ and $persist$ and its complexity can thus be derived).

The theoretical findings suggest that a materializing store like GraphDB is more suitable for read-oriented applications, especially when reasoning expressiveness increases. In this case, the inference cost is paid when data are loaded into the repository and the actual queries should then be faster. On the other hand, applications performing lots of data modifications would benefit from the non-materializing approach of query-time reasoning stores. However, as our benchmark paper has shown [109], the actual performance greatly depends on the implementation efficiency, and a materializing store may outperform a non-materializing one even for data modifying operations.

[12] `https://www.stardog.com/`, accessed 2020-08-11.

Chapter 5

Practical Solutions of Thesis Goals

This chapter builds upon the theoretical foundations provided in Chapter 4 and presents the *Java OWL Persistence API (JOPA)* – a Java persistence library for accessing Semantic data. Analogously to the world of relational databases, data access is split into the object-ontological mapping part represented by JOPA and the layer responsible for communication with the underlying repository – the *OntoDriver*. Typically, applications would directly use the higher-level API of JOPA and leave the OntoDriver as an implementation detail of storage access. Table 5.1 illustrates how the OntoDriver and JOPA correspond to their standard counterparts in the world of relational database access in Java. The OntoDriver and JOPA are introduced in Sections 5.1 and 5.2 respectively.

In addition, this chapter deals with another interface of a Semantic Web-based information system – the Web services. The *Java Binding for JSON-LD (JB4JSON-LD)*, presented in Section 5.3, is, as its name suggests, a library for translating Java objects into JSON-LD and vice versa. Such a library allows exploiting some of the Semantic Web capabilities in a regular Web service interface (for example, a REST API).

Figure 5.1 illustrates the structure of a domain-specific Semantic Web-based information system utilizing the software stack proposed in this thesis. The libraries described in this chapter are emphasized with bold borders. Note that the stack is built in Java – one of the most popular programming languages and a prominent language in the Semantic Web community as well.

Level	Relational World	Semantic Web World
Storage access	JDBC	OntoDriver
Mapping	JPA	JOPA

Table 5.1: Correspondence of the OntoDriver and JOPA to their relational database-access counterparts in Java.

5.1 OntoDriver

The OntoDriver is a data access layer that mediates access to the underlying Semantic data storage. In its nature, it is similar to a JDBC [43] driver (for Java) or an ODBC [38] driver (in general) for relational databases. The main motivation for separating storage access from the object-ontological mapping is to enable the development of drivers for various storage implementations, ranging from simple RDF files to SPARQL protocol [33] endpoints and vendor-specific APIs. The last point is actually the most important – different triple store vendors provide different APIs which are optimized for accessing the respective implementations. Therefore, it is logical to attempt to exploit these optimized APIs. This is, again, analogous to various JDBC driver implementations accessing different relational databases. Another important advantage of using vendor-specific APIs is that the most popular ones – Jena [40], OWL API [41], and RDF4J [35] – allow distinguishing asserted and inferred statements. While this may seem superfluous for read access, it is vital for data modifications, because, as has been discussed, inferred knowledge cannot be directly removed and attempts to do so should have no effect.

5.1.1 Structure of OntoDriver

The OntoDriver layer is described by a unified API that abstracts away the specifics of access to different storages. The API operates with the concept of an Axiom which represents a statement – either an RDF triple or an OWL axiom. A client (usually JOPA) instantiates an implementation of the DataSource implementation, which provides it with Connection instances – the main means of communication between the client and the driver. Connection realizes the operations defined in Section 4.3.1. In addition, it allows to create and execute SPARQL and SPARQL Update statements and contains transaction management methods.

In the current OntoDriver implementations, there is an *adapter* underneath the Connection [110]. This adapter ensures correct mapping of the Axiom-based API to the API of the underlying storage and implements the data access methods declared in Connection. The described structure is visualized in Figure 5.2.

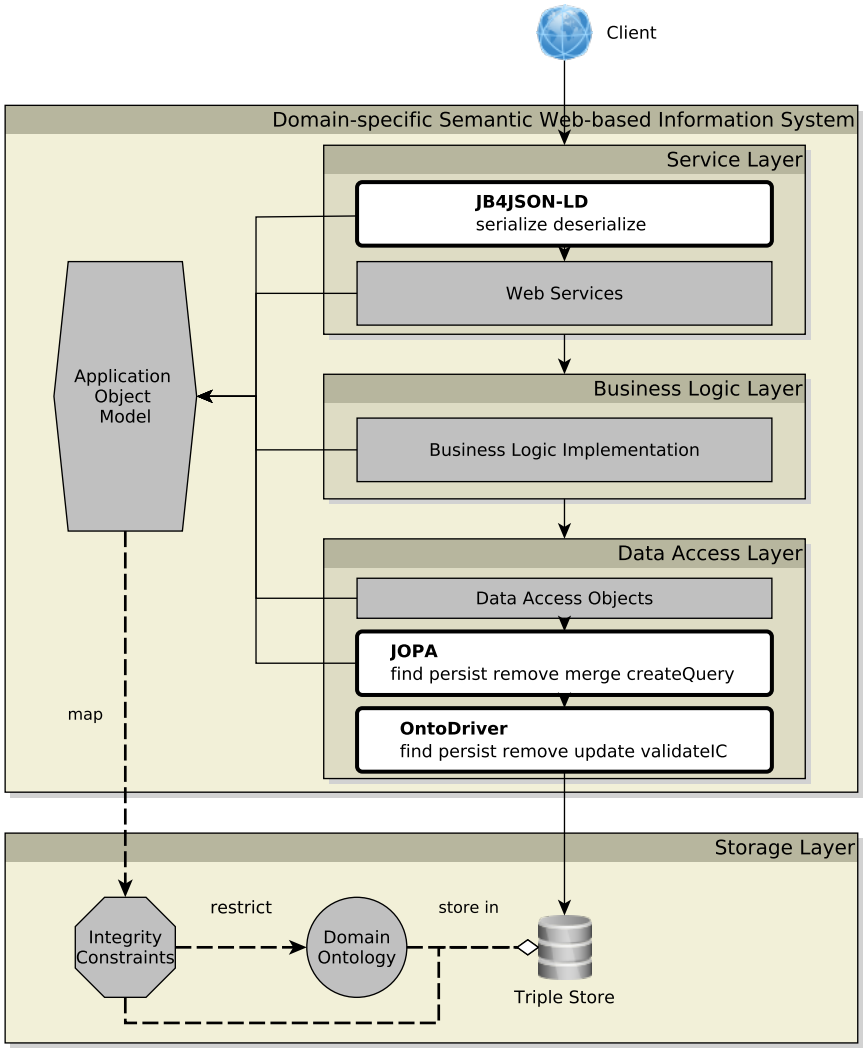


Figure 5.1: Simplified structure of a DSSWIS with emphasis on software libraries described in this chapter. Solid lines represent invocation or direct dependence. Dashed lines represent various relationships indicated by the labels on the lines.

5.1.2 Implementations

There currently exist OntoDriver implementations for all three major Java Semantic data access libraries – Jena, OWL API, and RDF4J. Together, they cover a wide variety of use cases – files containing RDF and OWL data can be accessed using the Jena or OWL API driver; local repositories supported by Jena and RDF4J can be used as well as remote RDF4J-compatible servers (including GraphDB). This range also covers various levels of data expressiveness – from simple RDFS up to OWL 2 DL – depending on the reasoner used by the underlying platform (OWL API-compatible reasoners like Pellet [111] support OWL 2 DL inference).

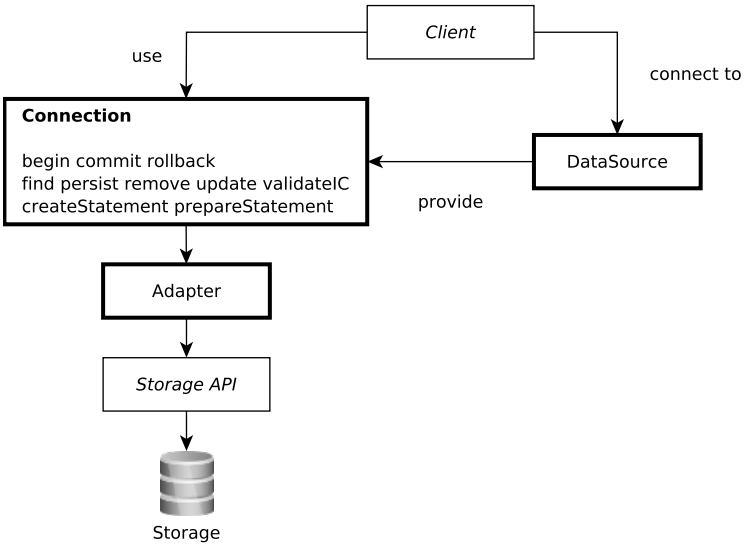


Figure 5.2: Simplified visualization of the OntoDriver structure. Components with a bold border are a part of the driver.

An important feature of the existing OntoDriver implementations is their support for caching transactional changes. When a driver-level transaction is started, a local cache of changes is created. Transactional data modifications are stored in this cache and used to augment results of read operations performed during the transaction. Such a strategy has two main benefits:

1. It allows isolating client transactions,
2. Inference can be applied to the transactional changes as well (currently supported only in the OWL API driver).

On commit, this transactional cache is written into the repository.

Source code of the OntoDriver API and all the OntoDriver implementations are publicly available as part of the JOPA repository on GitHub.[1]

5.2 Java OWL Persistence API

The Java OWL Persistence API (JOPA) is a persistence library for efficient access to Semantic data from object-oriented Java applications. Its main goal is to streamline

[1] `https://github.com/kbss-cvut/jopa`, accessed 2020-08-11.

the development of Semantic Web-based applications by allowing the developers to work with domain objects and taking care of the object-ontological mapping. This allows to reduce the extent of knowledge of Semantic Web technologies and principles developers need to possess and the amount of boilerplate code usually required by lower-level APIs like the OWL API.

The API of JOPA is designed to be as similar to the Java Persistence API (JPA) [44] as possible, so that developers used to working with relational databases can more easily transition to Semantic Web-based development. This means, for example, that ontological concepts are mapped to entity classes, the mapping is based on annotations, and objects processed during one transaction are gathered in a *persistence context* which is accessed via an instance of the EntityManager. However, in contrast to Empire [52], JOPA is not an implementation of JPA, because there are differences in both underlying paradigms which have to be taken into account, for instance, named graphs, inferred knowledge, or unmapped properties. Listing 5.1 provides an example entity class declaration, which will be referenced throughout this section when explaining some of the features of JOPA. It follows the running example from the previous chapter and adds new attributes to showcase the functionalities of the library and mapping configuration.

Listing 5.1: Example of an entity class declaration in JOPA. Mapping is specified via annotations. Important notions are explained in the text.

```
1  @Namespaces({
2      @Namespace(prefix="ex", namespace="http://example.org/"),
3      @Namespace(prefix="rdfs", namespace="http://www.w3.org/2000/01/rdf-schema#")
4  })
5  @OWLClass(iri="ex:Report")
6  public class Report {
7
8      @Id
9      private URI id;
10
11     @ParticipationConstraints(nonEmpty=true)
12     @OWLDataProperty(iri="rdfs:label")
13     private String headline;
14
15     @ParticipationConstraints(nonEmpty=true)
16     @OWLObjectProperty(iri="ex:author")
17     private User author;
18
19     @OWLObjectProperty(iri="ex:lastEditor")
20     private User lastEditor;
21
22     @Types
23     private Set<String> types;
24
25     @Properties
26     private Map<String, Set<Object>> properties;
27
28     // getters, setters
29 }
```

5.2.1 History

My supervisor, Dr. Petr Křemen, developed the first version of JOPA in the late 2000s as a more formally-grounded alternative to persistence libraries like AliBaba [50] or Empire [52]. The formalism was based on OWL integrity constraints [39], but the mapping tried to bridge the gap between DLs and Java directly, without any intermediate steps. The original implementation supported only OWL API and did not support transactions.

I first encountered JOPA when I was working on my bachelor thesis. Since then, I gradually took over its development and extended the original formalism-based implementation with features improving its applicability. These extensions include:

- Support for transactions,
- Caching and performance improvements in general,
- Support for inheritance and other new mapping features,
- Design and development of the OntoDriver API and its implementations, which allowed to support access to production-ready triple stores,
- Publishing the source code on GitHub,[2]
- Deploying the build artifacts to Maven Central[3] for easier access.

Up until now, JOPA has been used in at least seven real-world information systems (some of them will be discussed in Chapter 6). The statistics from Maven Central show almost 10 000 downloads of the main JOPA artifact between December 2018 and December 2019.

5.2.2 Features

The main features of JOPA can be split into OOM-related and technical. While the OOM-related features had been a part of JOPA from the start, they have been extended in my work as well. Moreover, their implementation is in alignment with the theoretical apparatus developed throughout the previous chapter (as will be demonstrated in Chapter 6).

[2] `https://github.com/kbss-cvut/jopa`, accessed 2020-08-11.
[3] `https://search.maven.org/search?q=cz.cvut.kbss.jopa`, accessed 2020-08-11.

Object-ontological Mapping Features

Explicit treatment of inference allows the developer to mark attributes that may contain inferred values (or even specify that they can contain only inferred values). Since inferred statements cannot be directly removed, this enables JOPA to ensure that such attempts are not even made. Currently, marking an attribute as inferred makes it effectively read-only, prohibiting any modifications to its content. This strategy is unnecessarily strict but is safe w.r.t. the inference results removal problem. In the future, it is planned to allow at least additive changes to inferred attributes, i.e., allow adding values to plural attributes containing inferred values.

Participation constraints fall into the category of *run-time* integrity constraints (see Section 4.2.2) and thus can be verified at execution time using predefined programming procedures. JOPA contains a validator that checks on load and on save (persist or update) that the number of attribute values corresponds to the declared constraint. The constraint specification can be seen on lines 11 and 15 in Listing 5.1.

Types attributes are attributes containing data about the ontological classes to which an individual belongs. In many ontologies, an individual is an instance of multiple classes, because classification is one of the most natural ways of categorizing objects [112]. An attribute annotated with `@Types` (as on lines 22 and 23 in Listing 5.1) contains the set of types to which, besides the type mapped by the object's entity class, the corresponding individual belongs.

Unmapped properties represent properties and values which are not covered by the application object model. While many other OOM solutions expect the mapping to cover the full extent of the ontology schema, it is conceivable that it may not be the case. Consider a terminology editor based on Semantic Web technologies (similar to VocBench [113]). Initially, it is rather difficult to determine the set of properties that users need to fill in. Therefore, it can be beneficial to start with a more generic tool able to add new properties when necessary, and over time adjust the static object model of the application according to the most relevant data. Unmapped properties in JOPA allow this type of access – they are represented by a generic map containing property-set of values pairs allowing to access data which are possibly beyond the current model based on the object-ontological mapping. The map can contain both literal values and identifiers of individuals. Listing 5.1 demonstrates this support on lines 25 and 26.

Technical Features

Transactional processing allows applications to group related operations on entities into indivisible, independent aggregates [99]. JOPA handles transactional changes on objects similarly to EclipseLink[4] (the JPA reference implementation) by providing the client with clones of instances read from the storage. Upon commit, these clones are compared to the unchanged original and only the necessary updates are made. In case of a rollback, the clones are simply thrown away.

Caching is an important aspect allowing to improve the performance of a persistence library. The *first-level cache* is represented by the persistence context, it ensures that asking twice for an object with the same identifier returns the same instance and is required for correct transactional semantics. JOPA, similarly to JPA, also supports the *second-level cache* which spans multiple persistence contexts and stores objects for faster lookup, because this cache is consulted before retrieving data from the storage.

Query result mapping represents the ability to map the results of SPARQL queries to entities. For example, when a developer uses a SPARQL query to select all reports matching some criteria, JOPA is able to automatically return instances of entity class Report, absolving the developer from having to map the results to the Report class manually.

Separate storage access layer has been already discussed in Section 5.1, the main benefit being the ability to easily implement access to new repositories.

Object model generator OWL2Java is a tool capable of generating a Java object model based on OWL integrity constraints. It allows to quickly set up the object-ontological mapping with minimum manual effort. Its main current disadvantage is that it completely overwrites the object model on rerun, removing any potential manual adjustments made by the developer. A less invasive feature of OWL2Java is the ability to generate a vocabulary file – a file containing constants for classes and properties discovered in the ontology. These can be used to manually configure the OOM, reducing the risk of typos.

5.2.3 Structure

Figure 5.3 displays a UML component diagram illustrating the high-level architecture of JOPA together with its relationship to OntoDriver. The Persistence Unit component represents infrastructure for accessing one repository, with the second-level cache available to all persistence contexts. It is also responsible for initializing

[4] `https://www.eclipse.org/eclipselink/`, accessed 2020-08-11.

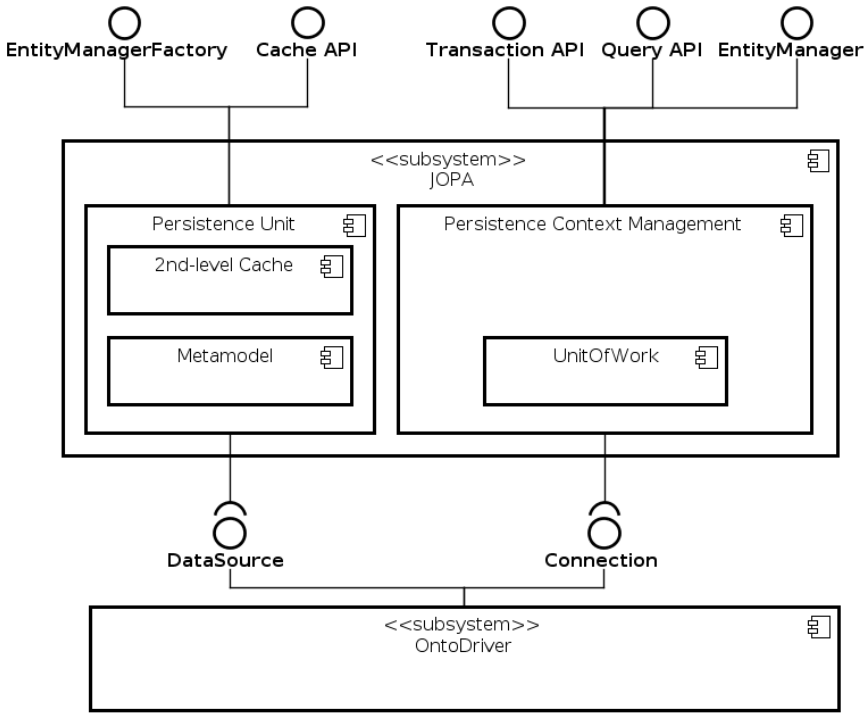


Figure 5.3: UML component diagram of JOPA. UnitOfWork represents an internal component responsible for managing the persistence context during a transaction.

the connection to the underlying repository via the DataSource interface. When a persistence unit starts, it also builds a *metamodel* of the object model. This metamodel contains the necessary configuration of the object-ontological mapping (depicted by the Metamodel component). The application can request an instance of EntityManager for opening a new persistence context via the EntityManagerFactory interface and manage the second-level cache via a dedicated API.

The Persistence Context Management component then, as the name suggests, manages the persistence context. It is intended to be accessed by a single thread (typically) during one transaction. In contrast, a persistence unit exists since its initialization till its shutdown (usually coinciding with application startup and shutdown), spans a number of transactions and can be accessed from multiple threads. A UnitOfWork represents the actions and objects participating in a single transaction [114] and JOPA uses this concept to encapsulate the persistence context during one transaction. When a persistence context is initialized, JOPA opens a Connection to the underlying OntoDriver, so that data can be read and eventually saved to the repository. A persistence context is outwardly represented by the EntityManager interface. In addition, during a transaction, the application can also use the query API and the transaction management API.

5.3 Java Binding for JSON-LD

This thesis hypothesizes is that the choice of using Semantic Web technologies for the development of a domain-specific information system mainly influences the interfaces of the system (of course, this applies to the actual design and programming phase and does not concern, for example, conceptual modeling of the domain). So far, this chapter has dealt with the interface between an application and the underlying data source. However, arguably even more important is the ability to provide Semantic Web-based services for the clients. This can shift the information system towards supporting machine-readable Linked Data. There exist solutions capable of providing Linked Data-compatible access to the data stored in a triple store – one of the most popular being Pubby [115]. But such tools provide direct read-only access to the repository, without any high-level operations or control over the structure of the data.

The Java Binding for JSON-LD (JB4JSON-LD) chooses a different path. It is a small software library allowing *Web services* to produce and consume JSON-LD. Typical Web services – software components providing services over the Web – nowadays rely on JSON or XML as their primary data exchange format. However, neither of these allows to easily convey relationships of the data and their schema to other structures outside of the processed document. JSON-LD,[5] on the other hand, allows to serialize Linked Data in JSON – a popular and easy-to-use format. The goal of JB4JSON-LD is to allow REST Web services to use JSON-LD as another data exchange format so that Linked Data can be not only read directly from the triple store but also produced and consumed by Web services with (domain-specific) business logic.

The approaches of Pubby and JB4JSON-LD are not mutually exclusive. On the contrary, they work best together. One can use a Web service to retrieve domain-specific data and, by dereferencing IRIs from the provided JSON-LD document, get to a Pubby endpoint containing further information, perhaps outside of the scope of the domain model of the Web service which provided the data in the first place.

5.3.1 Principles

JB4JSON-LD uses JOPA metamodel annotations to determine the types and properties to which the corresponding Java classes and their attributes should be mapped. Consider the example entity class in Listing 5.1 and an object of this class roughly

[5] JavaScript Object Notation for Linked Data

corresponding to Figure 4.5. Such an object would be serialized into JSON-LD shown in Listing 5.2. Notice that the author and last editor reference the same instance – since last editor is the second occurrence of this instance, it is only referenced by its IRI. This effortlessly breaks circular dependencies, a common problem when serializing object graphs to JSON.

Listing 5.2: Sample output of JB4JSON-LD serializing an entity into JSON-LD.

```
{
    "@id": "http://example.org/report-buo01",
    "@type": ["http://example.org/Report"],
    "http://www.w3.org/2000/01/rdf-schema#label": "Report BUO01",
    "http://example.org/author": {
        "@id": "http://example.org/Tom",
        "@type": ["http://example.org/User"],
        "http://example.org/firstName": "Tom",
        "http://example.org/lastName": "Lasky"
    },
    "http://example.org/lastEditor": {
        "@id": "http://example.org/Tom"
    }
}
```

JSON-LD is a flexible format with multiple ways of structuring the data. One possibility allowing to improve backward compatibility with JSON is to gather the prefixes into an attribute called `@context`. That way, the rest of the document resembles regular JSON serialization even more and applications can thus support both formats easily. A context-based version of the JSON-LD from Listing 5.2 is shown in Listing 5.3.

Listing 5.3: JSON-LD with a context definition. It is structurally more similar to regular JSON serialization in Java.

```
{
  "@context": {
    "id": "@id",
    "headline": "http://www.w3.org/2000/01/rdf-schema#label",
    "types": "@type",
    "author": "http://example.org/author",
    "lastEditor": "http://example.org/lastEditor",
    "firstName": "http://example.org/firstName",
    "lastName": "http://example.org/lastName"
  },
  "id": "http://example.org/report-buo01",
  "types": "http://example.org/Report",
  "author": {
    "id": "http://example.org/Tom",
    "types": "http://example.org/User",
    "firstName": "Tom",
    "lastName": "Lasky"
  },
  "lastEditor": {
    "id": "http://example.org/Tom"
  },
  "headline": "Report BUO01"
}
```

JB4JSON-LD consists of a core module which processes the object graph and

performs the mapping based on JOPA annotations, and integration modules for existing JSON processing libraries. These integration modules are responsible for the actual handling of the JSON-LD input and output. Currently, there is an integration module available for the Jackson JSON library.[6] The source code of both JB4JSON-LD and JB4JSON-LD-Jackson is publicly available on GitHub[7,8] and their build artifacts are deployed to Maven Central,[9] from which the Jackson integration of JB4JSON-LD has been downloaded over 3500 times between December 2018 and December 2019.

[6] `https://github.com/FasterXML/jackson`, accessed 2020-08-11.
[7] `https://github.com/kbss-cvut/jb4jsonld`, accessed 2020-08-11.
[8] `https://github.com/kbss-cvut/jb4jsonld-jackson`, accessed 2020-08-11.
[9] `https://search.maven.org/search?q=g:cz.cvut.kbss.jsonld`, accessed 2020-08-11.

Part III

Results

Chapter 6

Evaluation

This chapter provides an evaluation of the ideas and solutions presented in the previous part. It shows how the presented approach allows to efficiently build (domain-specific) Semantic Web-based information systems and thus fulfills the thesis goals.

It starts in Section 6.1 by validating that the object-ontological mapping implemented by JOPA corresponds to the formalism developed in Section 4.2. Most importantly, it shows that the object model generator provided by JOPA generates models consistent with the formal transformation rules and that the instance data are equivalent under both models.

Next, several domain-specific Semantic Web-based information systems built using the toolset described in Chapter 5 are introduced in Section 6.2. These systems cover several different domains and show that the approach of this work is viable for various scenarios. Section 6.3 follows this presentation with a discussion of the architecture of the exemplified information systems and its general applicability. Lastly, the results of a short survey among Semantic Web-based information system developers are discussed in Section 6.4.

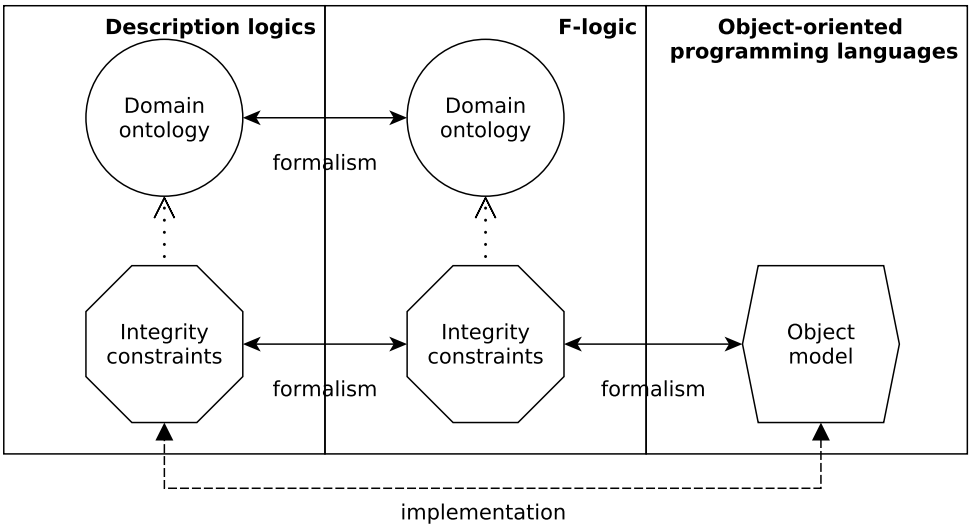


Figure 6.1: Illustration of the formal and technical object-ontological mapping. The formal mapping is depicted using solid edges, whereas the technical solution is represented by the dashed edge.

6.1 Evaluation of the Object-ontological Mapping Formalism

The evaluation of the object-ontological mapping formalism needs to show that the ontological and application models are equivalent so that the operations performed over data based on these models have corresponding results on both levels.

The mapping formalism is built along the axis description logics – F-logic – OO programming languages. As already discussed, F-logic was chosen because of being a logic-based language (similarly to DLs) intended to represent the structural aspects of OO programming languages. Thus, the correctness of the translation between DLs and F-logic presented in this work can be proven using the apparatus of the FOL (already done in Section 4.2). The mapping between F-logic and OOPLs is then based on F-logic's ability to express basic OOP notions such as classes, attributes, or inheritance, with a frame-based syntax and semantics [29]. Of course, in real scenarios, such a two-step translation would be impractical and persistence libraries like JOPA translate data directly between DLs and object models (as is illustrated by Figure 6.1). It is, therefore, necessary to show that this direct mapping is faithful w.r.t. its formal counterpart.

Since the translation between F-logic and OOPLs is not fully formally grounded (i.e., there is no formalism connecting, for instance, F-logic and Java), it is not possible to provide a rigorous proof of equivalence of the DLs – F-logic – OOPLs and

DLs – OOPLs translations. Therefore, this section resorts to a different strategy. It provides example ontologies and sets of integrity constraints covering all the basic DL concept descriptions. These examples illustrate the process of the formalism-based mapping between the DL ontology and its ICs and F-logic, and finally, a Java object model built in JOPA. Next, the same model is generated directly by the OWL2Java tool, which is a part of JOPA. Both of these models are then compared with the expectation that they would be equivalent.

6.1.1 Mapping by Example

Two sample ontologies with corresponding sets of integrity constraints were created to showcase the mapping. $\mathcal{O}_{S1}$ continues in the topic of the running example of Section 4.2, whereas $\mathcal{O}_{S2}$ slightly deviates and presents a hierarchy of safety and security events, threats and their targets. The mapping demonstration is available online.[1] The artifact contains, besides $\mathcal{O}_{S1}$, $\mathcal{O}_{S2}$, and their integrity constraints, also IC validation queries, sample valid and invalid (w.r.t. integrity constraints) datasets, and a small test application. The F-logic data are written in a format compatible with Flora-2,[2] an open-source F-logic implementation [116]. Mapping of $\mathcal{O}_{S1}$ is presented here, whereas mapping of $\mathcal{O}_{S2}$ is provided in Appendix C.2.

The excerpt below shows that, in addition to the previously introduced schema, the occurrence now also has a severity assessment, whose values come from an enumeration, and the user has some basic data attributes. The integrity constraints ensure cardinality and range of the properties on the mapped classes. Namespaces are omitted for the sake of readability.

$$\begin{aligned}\mathcal{O}_{S1} = \{ & Asset \sqsubseteq =1\, author.\top, Asset \sqsubseteq \leqslant 1\, lastEditor.\top, Resource \sqsubseteq Asset,\\ & Report \sqsubseteq Asset, author \sqsubseteq editor, lastEditor \sqsubseteq editor,\\ & Report \sqsubseteq =1\, documents.Occurrence,\\ & Occurrence \sqsubseteq =1\, hasSeverity.Severity,\\ & Severity \equiv \{observation, incident, accident\},\\ & User \sqsubseteq =1\, firstName.string, User \sqsubseteq =1\, lastName.string,\\ & User \sqsubseteq =1\, username.string\}\end{aligned}$$

[1] Available at `https://kbss.felk.cvut.cz/gitblit/summary/ml-oom-validation.git`, accessed 2020-08-11.

[2] `http://flora.sourceforge.net/`, accessed 2020-08-11.

$$\begin{aligned}\mathcal{IC}_{S1} = \{&Report \sqsubseteq =1\, author.User, Report \sqsubseteq \forall\, author.User,\\ &Report \sqsubseteq \leqslant 1\, lastEditor.User, Report \sqsubseteq \forall\, lastEditor.User,\\ &Report \sqsubseteq =1\, documents.Occurrence, Report \sqsubseteq \forall\, documents.Occurrence,\\ &Report \sqsubseteq \forall\, hasAttachment.Resource,\\ &Occurrence \sqsubseteq =1\, hasSeverity.Severity,\\ &Occurrence \sqsubseteq \forall\, hasSeverity.Severity,\\ &User \sqsubseteq =1\, firstName.string, User \sqsubseteq \forall\, firstName.string,\\ &User \sqsubseteq =1\, lastName.string, User \sqsubseteq \forall\, lastName.string,\\ &User \sqsubseteq =1\, username.string, User \sqsubseteq \forall\, username.string\}\end{aligned}$$

Since the ontological schema itself is not important for the mapping between F-logic and the object model, the example here continues only with the set of corresponding F-logic integrity constraints $\mathcal{IC}^F_{S1}$ (the full mapping is shown in Appendix C.1):[3]

$$\begin{aligned}\mathcal{IC}^F_{S1} = \{&Report_{\mathcal{C}}[author_{\mathcal{R}\{1,1\}} => User_{\mathcal{C}};\\ &\quad lastEditor_{\mathcal{R}\{0,1\}} => User_{\mathcal{C}};\\ &\quad documents_{\mathcal{R}\{1,1\}} => Occurrence_{\mathcal{C}};\\ &\quad hasAttachment_{\mathcal{R}} => Resource_{\mathcal{C}}],\\ &Occurrence_{\mathcal{C}}[hasSeverity_{\mathcal{R}\{1,1\}} => Severity_{\mathcal{C}}],\\ &User_{\mathcal{C}}[firstName_{\mathcal{R}\{1,1\}} => \_string;\\ &\quad lastName_{\mathcal{R}\{1,1\}} => \_string;\\ &\quad username_{\mathcal{R}\{1,1\}} => \_string]\}\end{aligned}$$

Mapping to a Java object model based on these integrity constraints is relatively straightforward. Figure 6.2 displays a UML class diagram of the object model, while listings 6.1 and 6.2 show the class Report as an example mapped manually and automatically resp.

[3] Remember that `_string` is a built-in F-logic string type (see Section 2.9).

Listing 6.1: Class Report mapped manually based on the evaluated OOM formalism.

```
@Namespace(prefix = "ev",
    namespace =
      "http://example.org/evaluation-01/")
@OWLClass(iri = "ev:Report")
public class Report implements Serializable {

    @Id
    private String id;

    @ParticipationConstraints(nonEmpty = true)
    @OWLObjectProperty(iri = "ev:documents")
    private Occurrence documents;

    @OWLObjectProperty(iri =
      "ev:hasAttachment")
    private Set<Resource> hasAttachment;

    @ParticipationConstraints(nonEmpty = true)
    @OWLObjectProperty(iri = "ev:author")
    private User author;

    @ParticipationConstraints(nonEmpty = true)
    @OWLObjectProperty(iri = "ev:lastEditor")
    private User lastEditor;

    // Getters and setters follow
}
```

Listing 6.2: Class Report mapped automatically by OWL2Java. Empty lines between attributes were added manually.

```
@OWLClass(iri = Vocabulary.s_c_Report)
public class Report implements Serializable {

    @Id(generated = true)
    protected String id;

    @OWLAnnotationProperty(iri = RDFS.LABEL)
    protected String name;

    @OWLAnnotationProperty(iri =
      DC.Elements.DESCRIPTION)
    protected String description;

    @Types
    protected Set<String> types;

    @Properties
    protected Map<String, Set<String>> properties;

    @OWLObjectProperty(iri =
      Vocabulary.s_p_author)
    @ParticipationConstraints({
        @ParticipationConstraint(owlObjectIRI =
          Vocabulary.s_c_User, min = 1, max = 1)
    })
    protected User author;

    @OWLObjectProperty(iri =
      Vocabulary.s_p_documents)
    @ParticipationConstraints({
        @ParticipationConstraint(owlObjectIRI =
          Vocabulary.s_c_Occurrence, min = 1, max
          = 1)
    })
    protected Occurrence documents;

    @OWLObjectProperty(iri =
      Vocabulary.s_p_hasAttachment)
    protected Set<Resource> hasAttachment;

    @OWLObjectProperty(iri =
      Vocabulary.s_p_lastEditor)
    @ParticipationConstraints({
        @ParticipationConstraint(owlObjectIRI =
          Vocabulary.s_c_User, max = 1)
    })
    protected User lastEditor;

    // Getters and setters follow
}
```

Although it may not appear so, both versions of the class Report are effectively equivalent and most differences are purely technical.

- Whether identifier value is generated or not upon instance persist is up to the

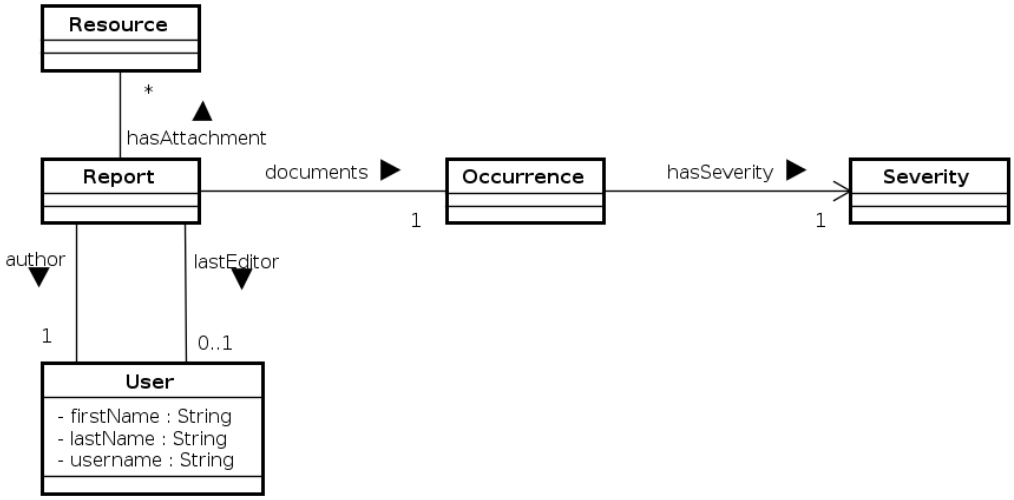


Figure 6.2: UML class diagram of the object model from the first mapping example, based on the ontology $\mathcal{O}_{S1}$. Resources attached to a report could be, for example, files, video and audio recordings or photographic evidence (or, more precisely, references to such assets).

entity class configuration. Even if the identifier is configured for auto-generation, it can be set by the application.

- The name and description attributes are added to all entity classes generated by OWL2Java regardless of the underlying ontology to provide basic information about each instance. Admittedly, it should be possible to disable this feature.
- Similarly, @Types and @Properties attributes are also generated automatically. As the reader may remember from Section 5.2, these attributes allow to capture parts of the ontology not directly described by the object model.
- @ParticipationConstraints(nonEmpty = true) is equivalent to @ParticipationConstraint(min = 1).
- Configuration of participation constraints with a maximum of 1 is superfluous for singular attributes.
- Similarly, @ParticipationConstraint annotations with target type (owlObjectIRI) are not necessary if it can be inferred from the target type of the attribute itself.

As can be seen, OWL2Java in some cases generates unnecessary configuration which may clutter the code. On the other hand, the mapping corresponds to the formalism developed in this thesis and manual fine-tuning is generally more precise and efficient (in terms of code quality and verbosity) than an automatic generation.

The example ontology $\mathcal{O}_{S2}$ showcases additional constructs like a hierarchy with multiple superclasses or disjunction in a property range. Remember, however, that

DL	F-logic	Java
$Report(r)$	$r_{\mathcal{E}} : Report_{\mathcal{C}}$	r **instanceof** Report
$documents(r, o)$	$r_{\mathcal{E}}[documents_{\mathcal{R}} \rightarrow o_{\mathcal{E}}]$	r.getDocuments() == o

Table 6.1: Illustration of the ABox axiom mapping between description logics, F-logic and Java.

only *compile-time* and *run-time* constraints are reflected in the object model. And run-time constraints only in the form of auxiliary Java annotation-based configuration for verification procedures in the OOM library itself. This is the case of the minimum participation constraints in class Report, for example.

Mapping Data

The mapping of the ABox axioms is, as has been already stated, relatively straightforward. DL class assertions map to F-logic is-a assertions between the instance and its class which in turn map to Java instances of the respective classes. Similarly, property assertions translate to data expressions in F-logic and further to instance attribute values in Java. Table 6.1 illustrates this on a couple of examples.

The example ontologies are accompanied by sample databases that demonstrate ABox mapping. In the case of $\mathcal{O}_{S1}$, $\mathcal{A}_{S11}$ represents a database corresponding to the ontology and application integrity constraints. On the other hand, $\mathcal{A}_{S12}$ is invalid w.r.t. the integrity constraints $\mathcal{IC}_{S1}$. Validation queries which fail on $\mathcal{A}_{S12}$ (there exist results for at least one query) are provided in SPARQL (for the DL version, can be verified by uploading the ontology and the dataset into a repository with reasoning support, e.g., Stardog[4] or GraphDB) and in Flora-2 syntax (for the F-logic version, can be verified directly in Flora-2). In addition, a simple Java application shows that the instance data under both manual and generated models are equivalent and that JOPA validates runtime integrity constraints, throwing an IC violation exception in case of $\mathcal{A}_{S12}$. Analogous examples are provided for $\mathcal{O}_{S2}$. However, since it cannot be currently fully mapped in JOPA (see below), no sample application exists for it.

6.1.2 Missing Features

There are several features of the object-ontological mapping formalism missing in its implementation in JOPA. Namely, support for compile-time integrity constraints

[4] `https://www.stardog.com/`, accessed 2020-08-11.

$A_{\mathcal{C}} :: (B_{\mathcal{C}}$ and $C_{\mathcal{C}})$ and $A_{\mathcal{C}} :: Nom(a_{\mathcal{E}})$ and the run-time constraint $A_{\mathcal{C}} :: HasSelf(R_{\mathcal{R}})$. However, Table 4.8 has already shown how the compile-time constraints would be mapped to Java and the validation of the run-time local reflexivity constraint would be straightforward to implement. All of these features are planned in the future development of JOPA.

6.2 Information Systems Built Using the Presented Tools

Over the last few years, the Knowledge-based Software Systems group,[5] of which I am a member, has developed several Semantic Web-based information systems in multiple domains. These systems were mostly built in cooperation with domain experts and relied on the tools and procedures advocated in this thesis. Let me introduce three of these systems in greater detail and at least briefly mention the other ones. This discussion should convince the reader that the approach introduced in the previous chapters is an effective way of creating Semantic Web-based information systems.

6.2.1 INBAS

INBAS[6] (**IN**dicator **BA**sed **S**afety) is an information system for safety occurrence reporting and investigation in high-risk industries. Its goal is to help improve safety culture in organizations by allowing its users to manage, investigate, and analyze safety occurrences and apply corrective measures based on this analysis. The system was built in the aviation domain, but its applicability is not limited to it. On the contrary, what ties INBAS to aviation is a part of the underlying modular ontology – the *Aviation Safety Ontology* (ASO) [117]. However, this module can be easily switched for a module from another high-risk industry, such as power engineering or railroad transportation. The ASO consists of the following main modules:

- Documentation ontology
- Safety ontology
- Aviation ontology

[5] `https://kbss.felk.cvut.cz/`, accessed 2020-08-11.
[6] `https://www.inbas.cz`, accessed 2020-08-11.

Moreover, the aviation ontology can be complemented by submodules suitable for different types of aviation organizations, for example, airports, airline operators, maintenance providers, or air navigation services centers. All of the core ontologies of the ASO are based on the top-level *Unified Foundational Ontology* (UFO) [112], which improves their reusability and anchors them in a more general modeling framework.

System Principles

Besides the underlying ontology, the *Reporting tool*[7] is the most important (or perhaps most visible) part of INBAS. The Reporting tool is a Semantic Web-based Web application through which users manage safety occurrence reports. Upon signing in, the user can create new occurrence reports, view or edit existing reports, and view statistical data.

In each report, the user can fill in basic information about the occurrence like its date and time, location, and, most importantly, classification and severity assessment. Then the user can model the *chain of factors* – a graph of events that were a part of the occurrence and their relationships. This factor chain is the primary investigation tool – an analysis of relevant factor chains in the statistics can reveal patterns of interest that need to be acted upon by the organization. For example, if most vehicle collisions on an airport are caused by problems with situational awareness and disorientation, the organization should take steps to improve the clarity and visibility of the signs and routes at the premises. Figure 6.3 shows the factor chain designer component of the INBAS Reporting tool. In addition, for each event, a generated form specific for the given occurrence category can be used to provide more relevant information.

The statistics view in the Reporting tool provides the user with trends in reported occurrence severity, most frequent occurrence categories and event types, and a graph of event factors, i.e., for a selected event type, event types which caused, contributed to, or mitigated it are summarized.

Relevance to the Thesis

INBAS is an example of a domain-specific Semantic Web-based information system. It is based on an ontological conceptualization of the domain, it uses ontological

[7] Source code is available at `https://github.com/kbss-cvut/reporting-tool`, accessed 2020-08-11.

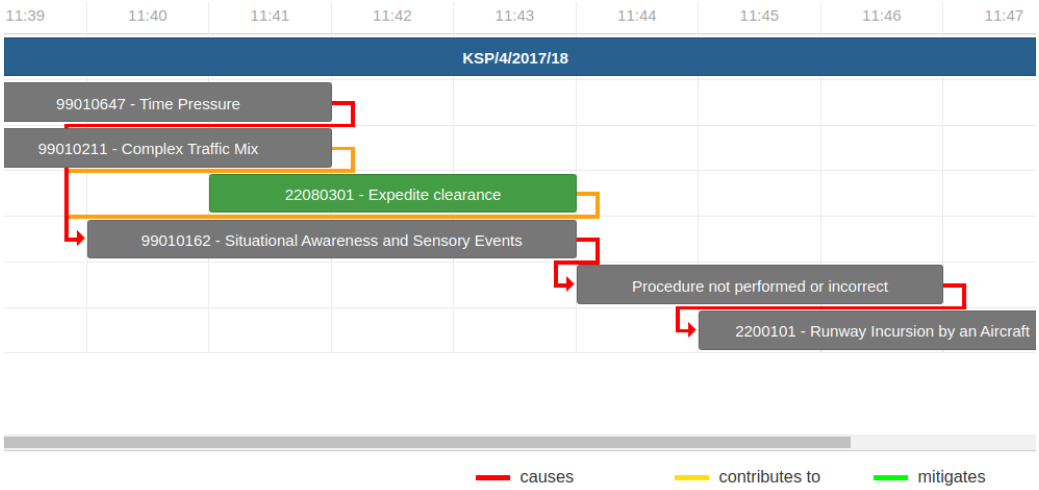


Figure 6.3: Factor chain designer in the INBAS Reporting tool. Top node represents the reported occurrence, its sub-nodes are events which were a part of the occurrence. The green node is an *explanatory factor*, whereas the grey nodes are events.

taxonomies of event types and occurrence categories. These taxonomies are based on existing taxonomies of the ECCAIRS ecosystem,[8] but their ontologization allowed to fix some of the structural and semantic problems they suffer from [118]. The taxonomies are stored in a triple store, so various combinations can be created just by changing the queries which load the classification options. The data backing the system have relatively low expressiveness, and the Reporting tool itself does not rely on any inference on the storage level. However, the SPARQL queries providing statistics for the system to perform some basic inference, e.g., using transitivity of property paths and property and class hierarchies.

The Reporting tool uses a triple store to save all its data and JOPA to access the repository. The application itself is a regular Web application written in Java, with user interface written in JavaScript using the React framework.[9] JOPA-based *data access objects* [114] are integrated with the transactional system of the popular Spring framework[10] used to power the application.

The Web services of the published version of the Reporting tool do not support JSON-LD as a data exchange format, however, an experiment with integrating JB4JSON-LD in the existing application showed that it could be done relatively easily. Such integration would increase the relevance of the system for the Semantic Web. On the other hand, this shows that using Semantic Web technologies at this level is noninvasive in terms of the application structure. This has been further proven in another manner, where we used the Reporting tool as a demonstration application in an advanced programming course at the university by simply switching

[8] `https://eccairsportal.jrc.ec.europa.eu/`, accessed 2020-08-11.
[9] `https://reactjs.org/`, accessed 2020-08-11.
[10] `https://spring.io/`, accessed 2020-08-11.

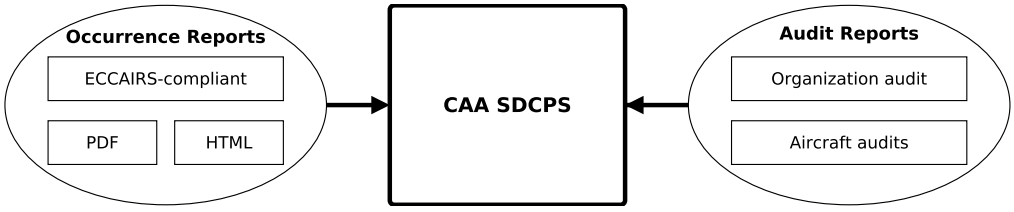


Figure 6.4: Simplified visualization of the various input formats a CAA SDCPS has to deal with.

to a relational database and reimplementing the data access objects to use JPA instead of JOPA. The rest of the application did not have to change.

6.2.2 SISel

SISel is a safety data collecting and processing system (SDCPS) developed for the Civil Aviation Authority (CAA) of the Czech Republic. The philosophy and principles of the system are similar to INBAS, but its use case is specific to the Czech CAA. While INBAS is intended for organizations that, in the case of aviation, are mandated to report safety-related occurrences to an authority, SISel is used by this authority to collect and process the received reports.

The CAA has a bit broader requirements on the system. It must be able to import reports sent to the CAA by the overseen organizations. However, not all organizations use the standardized ECCAIRS-compatible data format. In addition, voluntary reports from other organizations or even individuals are received in various formats. Besides occurrence reports, the CAA has another type of relevant reports. European CAAs perform safety audits in organizations and even aircraft under their jurisdiction. In case of an audit of a foreign aircraft, the audit results are shared via the European Union Aviation Safety Agency with the CAA of the country of origin of the aircraft's operator. This represents yet another source of data coming into SISel, as is visualized in Figure 6.4.

System Principles

The goal of the system is to gather a broader range of data to provide a more comprehensive picture of the safety situation. For this reason, data from audit and occurrence reports are transformed into a common model based on the Aviation safety ontology. An external analytical module (developed by a third-party company) then provides safety inspectors at the CAA with various statistical views. Two types

of outcomes of the situation analysis are expected. First, based on the nature and number of occurrences and audit findings related to an organization, CAA safety inspectors can plan audits concentrating on problematic areas. Second, a board of experts – the *safety action group* – at the Czech CAA uses outputs of SISel to review the safety situation in the country and possibly provide guidelines or regulations to address areas of interest.

To facilitate the investigation of occurrences, SISel supports a third kind of reports – safety issues. Safety issues are based on frequently occurring patterns of factor chains discovered in the data.

SISel is based on INBAS and uses an extended version of the Reporting tool, with the added support for the new types of reports. It employs an external statistical module implemented using Pentaho.[11] Moreover, it provides connectors to external services to be able to import the different kinds of reports. Namely, it has a connector to the ECCAIRS system and a mail server connector, because most of the reports the Czech CAA receives come via email.

Relevance to the Thesis

All the arguments discussed in Section 6.2.1 apply to SISel as well. In addition, given the variety of input formats it has to process, ontology-based data integration using the extended Aviation safety ontology comes into play.

From a technical point of view, SISel is a modular information system whose core (the extended Reporting tool) is based on Semantic Web technologies, but whose other components (namely the analytical module) need not be. This shows that Semantic Web technologies can be noninvasively integrated with other paradigms. We published our experience with designing and developing the system in a journal article [119].

[11] `https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho-platform.html`, accessed 2020-08-11.

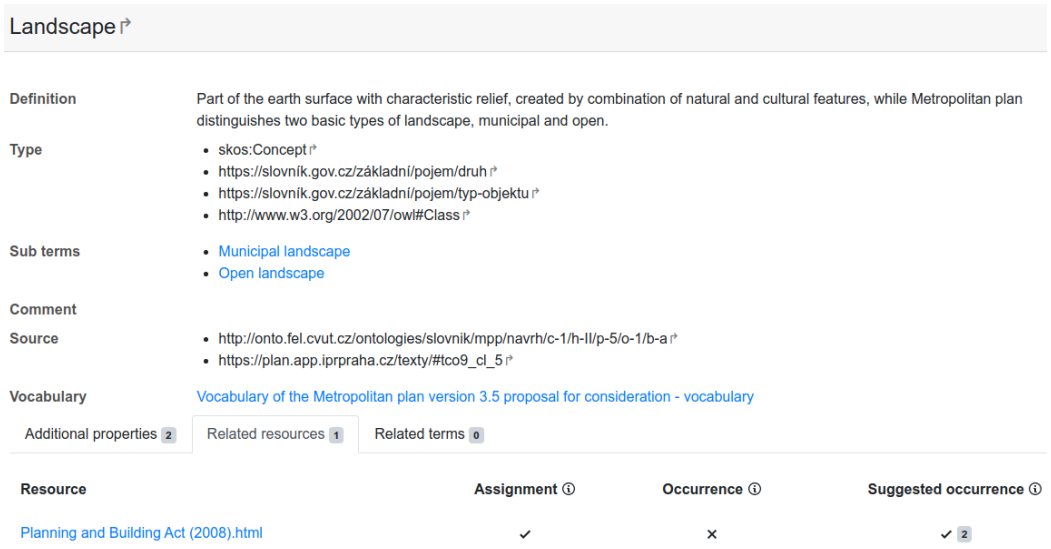


Figure 6.5: Detailed view of a term in TermIt. Notice that it is classified as both a SKOS concept and an OWL class. In addition, it is a UFO kind and object-type (IRI's are in Czech).

6.2.3 TermIt

TermIt[12] [120] is a vocabulary manager and glossary editor for domain experts. Consider the following sentences: "The construction of the Large Hadron Collider took ten years." and "The construction of the Large Hadron Collider is hidden in a 27 km-long underground tunnel." The word *construction* is used in different meanings conveyed by the context. These two meanings represent two *terms* which happen to have an identical label. In complex domains (for example, various regulations), unambiguous terms whose definition has been agreed upon are an important asset.

TermIt enables domain experts, who need not be fluent in the Semantic Web, to create and maintain glossaries organized using the *Simple Knowledge Organization System* (SKOS) [121] – a scheme allowing to build simple hierarchies of related terms. The reason for providing domain experts primarily with such a simple organization tool as SKOS is that, from our experience, their ability to exploit advanced modeling techniques and languages is usually limited [122]. Our proposal is to split the work between a domain expert who maintains a glossary of the vocabulary with terms, their definitions and simple (hierarchical) relationships, and an ontology engineer who can create a model of the vocabulary, expressing the complex relationship of the terms. Figure 6.5 shows the term detail view in TermIt. The term in question comes from the metropolitan plan of Prague, but it is assigned to a single resource – a file containing the text of the Planning and Building Act of Norway.

[12]Source code available at `https://github.com/kbss-cvut/termit` and `https://github.com/kbss-cvut/termit-ui`, accessed 2020-08-11.

System Principles

TermIt is a DSSWIS with a strong formal background. It is based on a hierarchy of ontologies starting with the Unified Foundational Ontology [112], containing an ontology for data description,[13] and finally an application-specific technical ontology. The data description ontology is based on both UFO and SKOS so that the glossaries managed by TermIt are SKOS-compatible.

TermIt provides two main modules. In the vocabulary management module, the user can maintain domain vocabularies. They are able to manage the vocabulary's glossary and, to a certain extent, also view and modify the ontological model of the vocabulary. However, the main focus of TermIt is on glossaries and the model editing features are limited. Vocabularies can be structured in hierarchies, for example, a vocabulary of a law would be more general than the vocabulary of a city's building regulations. Terms from more specific vocabularies can then depend on terms from more general vocabularies.

The second module supports resource management. The users can register various resources in TermIt and use the vocabulary terms to annotate them. These annotations can be used for a more accurate search. For instance, a dataset can be annotated by the terms it contains. Later, the user can search for terms they are interested in. Once such a term is found, all relevant resources are readily available for it. In fact, related resources can be discovered in a similar manner as well – the current version supports resources related by being annotated by a common term, but more complex relationships can be utilized. TermIt is also integrated with Annotace,[14] a semantic text analysis service which is able to discover term occurrences in text files and even suggest new terms based on their significance in the text. Figure 6.6 illustrates the structure of the TermIt system.

From a technical standpoint, TermIt stores all the data in a single repository, but each vocabulary has its own named graph. The repository is configured with a set of custom rules which allow inference combining selected RDFS (class and property hierarchies) and OWL (inverse properties) features. It is also planned that TermIt would allow importing a vocabulary from a remote location so that the network of TermIt instances managing vocabularies can be decentralized. For instance, a top-level deployment would contain vocabularies of the basic laws in the country, whereas specific ministries and their departments would manage lower-level vocabularies based on the generic ones in their instances of TermIt.

[13] Available at `http://onto.fel.cvut.cz/ontologies/slovnik/agendovy/popis-dat/current/index-en.html`, accessed 2020-08-11.

[14] `https://github.com/kbss-cvut/annotace`, accessed 2020-08-11.

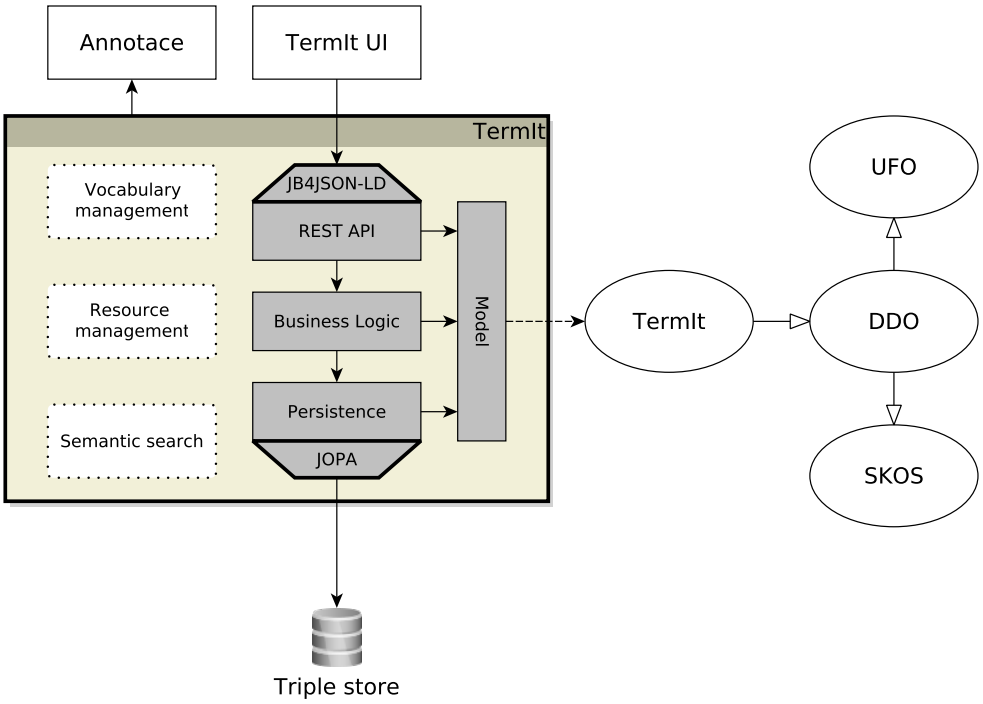


Figure 6.6: Schematic depiction of the structure of TermIt. Oval nodes represent ontologies (DDO is the data description ontology), nodes with a dotted border are functional modules, whereas nodes with a solid border are architectural layers of the application. Components relevant to this thesis are marked with a bold border. The dashed edge means that the model is based on the TermIt ontology.

Relevance to the Thesis

TermIt is a complex DSSWIS which fully exploits the stack described in this thesis. It uses a domain-specific ontology derived from higher-level ontologies and non-trivial inference over a triple store with application data accessed via JOPA. The triple store is at the same time accessible via a Pubby [115] Linked Data endpoint. The development deployment of TermIt works with a repository with approximately 2 million statements, over 1 million of which are inference results.

The structure of the application represents a standard layered Web application with a Java backend and a JavaScript frontend. The Java backend uses Spring with declarative transactions, security, and automatic validation of data. In addition, the backend REST API supports both JSON and JSON-LD (thanks to JB4JSON-LD) as data formats.

6.2.4 The Others

There are several other information systems developed mostly by my colleagues that use the tools presented in this thesis. They are briefly introduced below.

Dataset Dashboard

The *Dataset Dashboard* [123] is a tool for getting an overview of a SPARQL endpoint, or a graph inside a SPARQL endpoint. It can analyze the content of a dataset and generate *descriptors* – easy to interpret and visualize RDF summaries of the dataset. Dataset Dashboard is a Web application based on a formal ontology, it uses a JOPA-accessed repository to cache the calculated descriptors, and visualizes them via various widgets.

Study Manager

The *Study Manager* [124] is an ontology-based manager for clinical trials. Its model consists of a static part concerning user and organization management and a dynamic part which deals with actual studies. Each type of study requires the users to enter different data, so declaratively described forms are used for the individual studies. These forms are generated dynamically at runtime by an RDF data processing pipeline using the *SPipes* framework.[15] JOPA is used to store the data – both the static and dynamic parts of the model (the dynamic part is represented by a generic tree of questions and answers). The stored data are then analyzed by different tools. The Study Manager has been used in several international clinical trials, mostly in the area of cancer treatment research.

SPipes Editor

The *SPipes editor* [125],[16] as the name suggests, is a tool for creating and editing data processing pipelines using the SPipes framework. Figure 6.7 shows a data pipeline visualized by the editor.

[15] `https://kbss.felk.cvut.cz/web/kbss/s-pipes`, accessed 2020-08-11.

[16] Source code available at `https://github.com/kbss-cvut/s-pipes-editor`, accessed 2020-08-11.

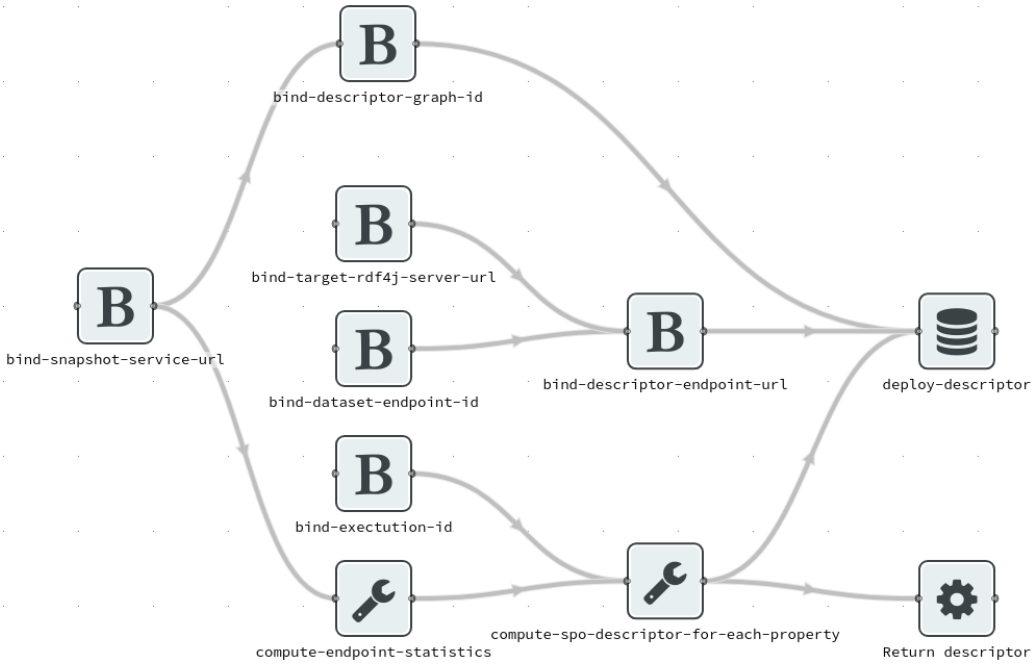


Figure 6.7: Pipeline for dataset descriptor generation (see Section 6.2.4) as visualized by the SPipes editor. Taken from [125].

The SPipes editor implementation differs from all the tools and information systems presented so far in that it does not use a triple store. SPipes modules are described using RDF files containing a declarative description of the module, so the editor uses these files for storage. JOPA is able to provide access to them via its Jena [40] OntoDriver. This architecture allows immediate feedback when editing a pipeline – once a change is made, the pipeline can be run based on the module declaration files being edited and the user can see the effects of the change at once. The frontend of the SPipes editor communicates with the REST services of the backend using JSON-LD, which is a natural representation of the module descriptions. Besides the specific nature of the storage, the SPipes editor is a regular Web application with a layered architecture. It is actually written in Scala, a programming language using the Java virtual machine (JVM) for execution, but providing a different syntax. This shows that the technological stack introduced in this thesis is not restricted to Java, but can be used in connection with other programming languages from the JVM ecosystem, e.g., Kotlin, Groovy, Clojure, or the aforementioned Scala.

6.3 Architecture of Semantic Web-based Information Systems

Sections 3.3.2 and 6.2 have shown that useful real-world information systems can be built using Semantic Web technologies. It is important, however, to be able to develop such systems efficiently. This section first provides some general guidelines to the development of Semantic Web-based information systems, followed by trade-offs and pitfalls one may face when developing them.

6.3.1 General Notes on Developing Semantic Web-based Information Systems

It has been the goal of certain parts of this thesis (Section 3.1) and several other works I co-authored (e.g., [119, 47]) to convince the reader that the development of most Semantic Web-based information systems is more efficient when a domain model is used and data are accessed using a software library providing object-ontological mapping. There are cases where a generic, statement-based, approach is more suitable, but these arguably concern mainly data editors. Even systems dealing with dynamic data, like terminology editors, can benefit from at least a rudimentary domain model (as the example of TermIt has shown in Section 6.2.1).

Once a domain model is established, business logic can be built around it. Such a structure then allows one to leverage the large amount of existing research and software development experience. Indeed, with a domain model-based business logic core, the underlying technology becomes an implementation detail [14]. Similar principles can be found in Alistair Cockburn's *hexagonal* architecture (also known as *ports and adapters*) [126] where adapters are used to transform data to and from a format used by the business core of the system. The adapters are connected to ports – endpoints of the system API. Palermo's *Onion architecture* [127] also pushes infrastructure components, which include data access and Web services, to the outer layer of the architectural onion.

The *layered* architectural style [128], known nowadays mainly from the development of Web applications, supports isolation of business logic as well. In its typical form, the business logic layer is stacked on top of the persistence layer, which consists of *data access objects* [114] separating the technicalities of accessing the underlying data source from the domain logic. Figure 6.8 illustrates this separation of concerns.

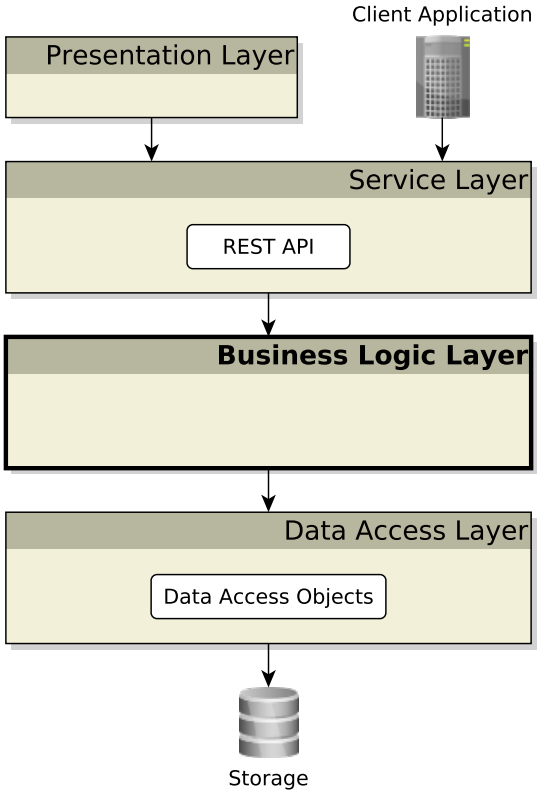


Figure 6.8: Layered architecture overview. Shows how the business logic (emphasized by bold label and border) is separated from external interfaces of the system.

All the aforementioned architectural styles have in common their effort to keep infrastructural details such as database technology or the means of communication with clients separate from the business logic core of the system. That way, the infrastructure can change (of course, to a limited extent) without causing ripple effects in the core implementation.[17] If one adheres to these best practices of software design, introducing Semantic Web technologies into the information system should be feasible. There are, however, questions and trade-offs that may be encountered.

6.3.2 Separating Business Logic from Infrastructure – Pitfalls

Using Semantic Web technologies can bring subtle design issues a developer should be aware of. The following paragraphs describe them and discuss their repercussions.

[17] In fact, one of my students has recently developed a document management system, in which a triple store or a relational database can be used to store data. The choice of the underlying storage is made by changing just one configuration parameter, the business logic of the system remains oblivious of the storage paradigm and the domain model is the same as well.

Is the model stable enough? This problem concerns mainly compiled languages with static typing like Java. Most OOM libraries in such languages declare the mapped model and do not consider any other data. However, certain scenarios require the possibility of reading and writing values of properties which are not a part of the static model. For instance, in the case of a terminology editor, users from different domains may need to specify custom properties (responsible personnel, publication status, etc.) which the original model did not foresee. Or, as discussed in the feature criterion *MC2* in [47], an application may start with only a small mapped model, leaving most of its parts dynamic until the structure is stable enough to become a part of the static model. In these cases, the developer needs to choose an OOM library capable of working with dynamic unmapped parts of the ontology schema.

Is the model expressive? One of the benefits of formal ontologies is the expressive languages that can be used to describe the domain model. Such a model then allows to *infer* implicit knowledge from the data. This, however, is a double-edged blade, because it essentially causes the spreading of business logic across multiple places in the system – the core of the implementation and the underlying repository (which is typically responsible for reasoning). Such a concern has been already identified by Heitmann et al. in their survey of Linked Data applications [82]. On the other hand, by building a model with no expressiveness, one would rob themselves of one of the key benefits of using formal ontologies. Thus the takeaway can be to be aware of this trade-off and clearly document what parts of business logic are covered by the domain model.

Does the system utilize ontology-based data integration? This question is closely related to the previous one because answering 'yes' has similar effects – the business logic of the system seeps into the data-integration module built usually directly in the storage using SPARQL queries and ontology matching. Once again, awareness and caution should be recommended here, as ontology-based data integration is an important selling point of utilizing Semantic Web technologies in an information system.

6.4 Semantic Web-based Information Systems Developer Survey

A survey was conducted among developers of Semantic Web-based information systems. The main goal of the survey was to gain insight into which technologies

such developers use (mainly w.r.t. data access), what are the reasons for choosing specific technologies and the experience developers have.

6.4.1 Survey Audience and Questions

The survey consisted of seven questions, which were:

1. Sent to a selected set of developers of information systems related to the ones discussed in Section 6.2. More specifically, the following were contacted:
   - Developers of *VocBench* [113] – a collaborative terminology and ontology editor,
   - Developers of the *GetThere* system [86],
   - Developers of *LinkedPipes ETL* [129] – a data processing framework similar to the SPipes framework,
   - The developer of the *SPipes Editor* [125]
2. Made into an online form[18] which was then sent to:
   - Users discussion group of the RDF4J framework,[19]
   - Users email list of the Apache Jena framework at `users@jena.apache.org`,
   - Email list of OWL API at `owlapi-developer@lists.sourceforge.net`,
   - The *Linked Data Web*[20] and *Ontologies, OWL-S, SPARQL interest group*[21] LinkedIn groups

The questions were as follows:

1. What technology do you use to access Semantic data?
   - This was a choice question with the following options:
     a. Domain-independent API like Jena or RDF4J
     b. Domain-specific API like Empire or JOPA
     c. SPARQL/SPARQL Update or the Linked Data Platform
     d. Other

[18] `https://forms.gle/yde5oKR6QY6DYM2B6`, accessed 2020-08-11.
[19] `https://groups.google.com/forum/#!forum/rdf4j-users`, accessed 2020-08-11.
[20] `https://www.linkedin.com/groups/60636`, accessed 2020-08-11.
[21] `https://www.linkedin.com/groups/86246`, accessed 2020-08-11.

2. What led you to select the data access technology you use? Did you consider other options?
   - This was an open-ended question.
3. If you are not using higher-level libraries: Do you think you would benefit from using such an API? If not, what are the reasons?
   - This was an open-ended question.
4. Do you need to edit data across multiple triple stores (resp. multiple named graphs in a single triple store)? How do you do it?
   - This was an open-ended question.
5. What is the expressiveness of the data your application works with? What are the inference tasks the application business logic benefits from?
   - This was an open-ended question.
6. What were the biggest development obstacles you had to overcome?
   - This was an open-ended question.
7. Any additional comments, experiences, feedback...

In addition, due to its anonymity, respondents of the online survey form were asked for at least basic information about the systems they develop.

6.4.2 Survey Evaluation

In total, eleven answers were gathered for the survey, which makes it, unfortunately, statistically insignificant. Nevertheless, there is some knowledge to be gathered. In the following, responses to each question are summarized and important points are highlighted. Afterward, an overall commentary is provided. The complete set of answers is available in Appendix D.

1. Eight out of the eleven respondents use primarily statement-based domain-independent APIs, the GetThere system relied directly on SPARQL queries. One respondent of the online survey uses all of the options provided in the question, depending on the use case. This result may seem as though domain-specific APIs do not attract users, but answers to the next question will shed more light on the reasons.

2. Almost none of the respondents of the survey works on a domain-specific information system. Some come close (for instance, one of the developers works on RDF-based power system models), but others work on highly generic tools like a custom triple store, ontology and terminology modeling tools (VocBench), or a SPARQL engine for continuous reasoning (C-SPARQL [130]).
3. Answers to this question are closely related to the previous one – developers of generic Semantic Web-based tools do not see benefits of switching to a domain-specific API. Often cited reasons are limited optimization options of the higher-level APIs or loss of expressiveness. On the other hand, one of the respondents is currently developing a similar high-level API.
4. Most of the developers utilize named graphs in a triple store, some even access multiple triple stores at the same time.
5. Answers to this question varied in scope and technical level. Three respondents specified reasoning levels in terms of OWL profiles (which was the original intention of the question), some mentioned the nature of the data their system deals with, which included power system planning and operation, data streams, or call data records. Three answers indicate that no reasoning is used by their system.
6. Various obstacles were encountered by the survey respondents, ranging from a lack of mature tools (apparently a recurring topic in the Semantic Web world), problematic performance when dealing with highly expressive data, to limited compliance with standards (triple store SPARQL engines in particular).

General Observations

The survey results do not support the approach of this thesis. However, they do not contradict it either. This is mainly because most of the systems whose developers answered the questions are not domain-specific information systems in the sense discussed throughout the thesis.

Nevertheless, several conclusions may be drawn from the survey results.

- The topic of performance and optimizations is recognized and considered important by Semantic Web developers,
- Similarly, the maturity of Semantic Web-based software tools is often problematic,
- A conclusion not directly following from the survey is that there is arguably a lack of information about Semantic Web-based software libraries and tools.

There exist sources and repositories with such tools, but they are often outdated (like the list at W3C's Wiki[22]) or not well-known (like the list of tools at Awesome Semantic Web[23]). This indirectly also represents a lack of a communication platform where new developers could get information or usage examples.

Overall, the survey certainly does not represent a significant contribution to the thesis, mainly due to the small number of responses. Perhaps the target audience should have been chosen differently, or more direct communications mechanisms should have been used. Nevertheless, some of its conclusions are aligned with this thesis' topic (tool maturity, performance, accessibility of information sources).

6.5 Experience with Developing Semantic Web-based Information Systems

My experience with developing DSSWISs can be characterized as a journey from basic to more sophisticated programming techniques employed in SW development. At its beginning stands INBAS. While the Aviation Safety Ontology is relatively expressive ($\mathcal{SHIQ}(\mathcal{D})$), the Reporting tool is based on plain RDF and does not utilize any reasoning over the data. In addition, the system's transaction management is procedural and the REST API is based on JSON.

SISel extends INBAS in many ways. From this thesis' point of view, the biggest addition is the ontology-based integration of data from different sources. SISel also has an external statistical module built in Pentaho. This statistical module is fed by data exported from the application triple store into a relational database. Nevertheless, the object model remains based on RDF, transactions are procedural and the REST API uses JSON.

TermIt is, from the perspective of incorporating Semantic Web technologies into information systems, the most advanced system I have worked on to date. Its domain model is backed by an expressive ontology and the inference results are actually used in it. Transaction management is declarative, thanks to a Spring integration library I created for JOPA,[24] and the REST API primarily uses JSON-LD. In addition, several Semantic Web-based external services that will be used by TermIt are currently under development (an authorization service, a document management system). This should turn TermIt into a modern service-oriented information system with the added benefit of using Semantic Web technologies.

[22] `https://www.w3.org/2001/sw/wiki/Tools`, accessed 2020-08-11.
[23] `https://github.com/semantalytics/awesome-semantic-web`, accessed 2020-08-11.
[24] `https://github.com/ledsoft/jopa-spring-transaction`, accessed 2020-08-11.

As tighter integration of Semantic Web software libraries into common development frameworks and tools like Spring emerges, developing Semantic Web-based applications becomes easier and more efficient. For example, although based on purely subjective opinion, the increase of development productivity from INBAS to TermIt has been, thanks to the improvements in JOPA, JB4JSON-LD, and their integration with Spring, quite substantial. What is showing as increasingly problematic is the performance of such applications when dealing with large amounts of data. Therefore, more effort will need to be devoted to developing optimization techniques for Semantic Web-based information systems.

Chapter 7

Conclusions

This thesis explored the problematique of developing Semantic Web-based information systems and their domain-specific variant in particular. The main motivation was that while the Semantic Web has been getting traction as a paradigm for publishing and consuming data, its utilization in information systems with complex business logic is still limited. Indeed, a survey of relevant literature and existing Semantic Web-based information systems has revealed that the main focus is on Linked Data publishing, consumption, and semantic search. One of the rare use cases of Semantic Web technologies in domain-specific information systems is ontology-based data integration. I argued that a steep learning curve, the lack of mature software libraries, and problematic performance are among the chief reasons for such a slow adoption. Two areas of possible improvement were identified – application access to Semantic data and integration of application interfaces via Semantic Web technologies. Thus, the goals of the thesis were to analyze and describe the design principles of domain-specific Semantic Web-based information systems and improve the client and data source-facing boundaries of such systems.

In case of application access to Semantic data, the contribution is threefold – a feature and performance-based comparison of existing object-triple mapping libraries is provided. Its purpose is to help developers in deciding which tool would be the most suitable for their project. As an added benefit, the performance comparison test applications may act as configuration demonstrations in an area where documentation is scarce. The second part of this thesis' contributions w.r.t. data access is a formal framework describing object-ontological mapping. This framework represents a logic-based description of the contract between an ontology and an application object model. Adhering to this formalism allows to precisely specify the shape and behavior of the object-ontological mapping. Lastly, the Java OWL Persistence API – a persistence library for Semantic data – is presented as an implementation of this

formalism. JOPA has been developed for the past decade, originally maintained by my supervisor, Dr. Petr Křemen. Its popularity has been slowly growing especially in the last few years and it has transitioned from a purely prototypical project to a tool usable in real-world Semantic Web-based applications.

The contribution in the area of Semantic Web-based application integration is more subtle. Since a large body of work has been already done in the area of describing application interfaces using Semantic Web technologies (e.g., Hydra [78]), this thesis merely introduces the JB4JSON-LD library, which allows Java applications to effortlessly consume and produce JSON-LD data. Such an extension could be used to build Semantic REST APIs.

When it comes to the architecture of an information system based on Semantic Web technologies, it turns out that if the developer adheres to the best practices of separating business logic from infrastructural code, which mainly deals with the system's outward interfaces, there is not much difference in developing an information system utilizing the Semantic Web or other well-known paradigms (e.g., relational). There are trade-offs one needs to be aware of, but their significance is rather minor and depends on other factors as well.

Turning our gaze into the future, there are several directions worthy of expansion. First, the missing features in JOPA (see Section 6.1.2) should be addressed. Further work could be also directed into the simplification of generating Semantic Web-compatible REST API documentation. Either by extending the capabilities of Hydra (which has currently a generator only for PHP, descriptions of APIs in other languages have to be created manually) or by adding support for JSON-LD into the OpenAPI standard [131]. Finally, more attention will have to be devoted to the performance of Semantic Web tools and libraries, as it becomes more and more crucial to real-world Semantic Web-based information systems that have to cope with an evergrowing amount of data.

Bibliography

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 28–37, 2001.

[2] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax," W3C, W3C Recommendation, 2014, http://www.w3.org/TR/rdf11-concepts/, accessed 2020-08-11.

[3] S. Harris and A. Seaborne, "SPARQL 1.1 Query Language," W3C, W3C Recommendation, 2013, http://www.w3.org/TR/sparql11-query/, accessed 2020-08-11.

[4] D. Wood, M. Zaidman, L. Ruth, and M. Hausenblas, *Linked Data: Structured Data on the Web*, J. Bleiel, Ed. Shelter Island, NY, USA: Manning Publications Co., 12 2013. ISBN 9781617290398

[5] A. Rector, J. Rogers, and P. Pole, "The GALEN High Level Ontology," in *Fourteenth International Congress of the European Federation for Medical Informatics, MIE-96, Copenhagen, Denmark*, 1996.

[6] D. Lee, N. de Keizer, F. Lau, and R. Cornet, "Literature review of SNOMED CT use," *Journal of the American Medical Informatics Association*, vol. 21, no. e1, pp. e11–e19, 7 2013. DOI 10.1136/amiajnl-2013-001636

[7] A. Kalyanpur, B. K. Boguraev, S. Patwardhan, J. W. Murdock, A. Lally, C. Welty, J. M. Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, Y. Pan, and Z. M. Qiu, "Structured Data and Inference in DeepQA," *IBM Journal of Research and Development*, vol. 56, no. 3, pp. 351–364, 5 2012. DOI 10.1147/JRD.2012.2188737

[8] P. Křemen, "Building Ontology-Based Information Systems," Ph.D. dissertation, Czech Technical University in Prague, Prague, 2012.

[9] M. A. Musen, "The Protégé Project: A Look Back and a Look Forward," *AI Matters*, vol. 1, no. 4, pp. 4–12, 6 2015. DOI 10.1145/2757001.2757003

[10] G. Booch, *Object-oriented Analysis and Design with Applications (2^{nd} Ed.)*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994. ISBN 0-8053-5340-2

[11] M. Hausenblas, "Linked Data Applications—The Genesis and the Challenges of Using Linked Data on the Web," DERI, Tech. Rep., 2009.

[12] M. Martin and S. Auer, "Categorisation of Semantic Web Applications," in *Proceedings of the 4th International Conference on Advances in Semantic Processing (SEMAPRO2010) 25 October – 30 October, Florence, Italy*, 10 2010.

[13] E. Simperl, M. Acosta, M. Dimitrov, J. Domingue, P. Haase, M. Maleshkova, A. Mikroyannidis, B. Norton, and M.-E. Vidal, "EUCLID: EdUcational Curriculum for the usage of LInked Data," in *11th European Semantic Web Conference (ESWC 2014), 25-29 May 2014, Anissaras/Hersonissou, Crete, Greece*, 2014.

[14] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2017. ISBN 0134494164, 9780134494166

[15] P. Gearon, A. Passant, and A. Polleres, "SPARQL 1.1 Update," W3C, W3C Recommendation, 2013, https://www.w3.org/TR/sparql11-update/, accessed 2020-08-11.

[16] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *Int. J. Hum.-Comput. Stud.*, vol. 43, no. 5-6, pp. 907–928, 12 1995. DOI 10.1006/ijhc.1995.1081

[17] N. Guarino, D. Oberle, and S. Staab, *What Is an Ontology?* Springer-Verlag Berlin Heidelberg, 2009, ch. 1, pp. 1–17.

[18] D. Brickley and R. V. Guha, "RDF Schema 1.1," W3C, W3C Recommendation, 2014, https://www.w3.org/TR/rdf-schema/, accessed 2020-08-11.

[19] B. Motik, B. Parsia, and P. F. Patel-Schneider, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax," W3C, W3C Recommendation, 2012, https://www.w3.org/TR/owl2-syntax/, accessed 2020-08-11.

[20] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyaschev, "Ontology-Based Data Access: A Survey," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018. DOI 10.24963/ijcai.2018/777 pp. 5511–5519.

[21] M. Schneider, "OWL 2 Web Ontology Language RDF-Based Semantics," W3C, W3C Recommendation, 2012, https://www.w3.org/TR/owl2-rdf-based-semantics/, accessed 2020-08-11.

[22] B. Motik, P. F. Patel-Schneider, and B. C. Grau, "OWL 2 Web Ontology Language Direct Semantics," W3C, W3C Recommendation, 2012, https://www.w3.org/TR/owl2-direct-semantics/, accessed 2020-08-11.

[23] I. Horrocks, "DAML+OIL: A Reason-able Web Ontology Language," in *Advances in Database Technology — EDBT 2002*, C. S. Jensen, S. Šaltenis, K. G. Jeffery, J. Pokorny, E. Bertino, K. Böhn, and M. Jarke, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. ISBN 978-3-540-45876-0 pp. 2–13.

[24] I. Horrocks, O. Kutz, and U. Sattler, "The Even More Irresistible $\mathcal{SROIQ}$," in *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*. AAAI Press, 2006. ISBN 978-1-57735-271-6 pp. 57–67.

[25] I. Horrocks and U. Sattler, "Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. ISBN 1-55860-812-5, 978-1-558-60812-2 pp. 199–204.

[26] B. Motik and I. Horrocks, "OWL Datatypes: Design and Implementation," in *The Semantic Web - ISWC 2008*, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. DOI 10.1007/978-3-540-88564-1_20. ISBN 978-3-540-88564-1 pp. 307–322.

[27] J. Tao, E. Sirin, J. Bao, D. L. McGuinness, and D. L. McGuinness, "Integrity Constraints in OWL," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, Atlanta, Georgia, USA, 2010. ISBN 9781577354666 pp. 1443–1448. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/viewFile/1931/2229

[28] J. Tao, "Integrity Constraints for the Semantic Web: An OWL 2 DL Extension," Ph.D. dissertation, Rensselaer Polytechnic Institute. ISBN 978-1-267-67967-3 2012.

[29] M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-oriented and Frame-based Languages," *Journal of the Association for Computing Machinery*, vol. 42, no. 4, pp. 741–843, 7 1995. DOI 10.1145/210332.210335

[30] M. Kifer, "Rules and Ontologies in F-Logic," in *Reasoning Web: First International Summer School 2005, Msida, Malta, July 25-29, 2005, Revised Lectures*, N. Eisinger and J. Małuszyński, Eds. Springer Berlin Heidelberg, 2005, pp. 22–34. ISBN 978-3-540-31675-6

[31] J. Angele, M. Kifer, and G. Lausen, "Ontologies in F-Logic," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds. Springer Berlin Heidelberg, 2009, pp. 45–70. ISBN 978-3-540-92673-3

[32] M. Lytras and R. García, "Semantic Web Applications: A framework for industry and business exploitation - What is needed for the adoption of the Semantic Web from the market and industry," *International Journal of Knowledge and Learning*, vol. 4, pp. 93–108, 1 2008. DOI 10.1504/IJKL.2008.019739

[33] L. Feigenbaum, G. T. Williams, K. G. Clark, and E. Torres, "SPARQL 1.1 Protocol," W3C, W3C Recommendation, 2013, https://www.w3.org/TR/sparql11-protocol/, accessed 2020-08-11.

[34] S. Speicher, J. Arwe, and A. Malhotra, "Linked Data Platform 1.0," online, W3C, W3C Recommendation, 2013, http://www.w3.org/TR/ldp/, accessed 2020-08-11.

[35] J. Broekstra, A. Kampman, and F. van Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," in *Proceedings of the First International Semantic Web Conference on The Semantic Web*, 2002, pp. 54–68.

[36] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "OWLIM: A family of scalable semantic repositories," *Semantic Web – Interoperability, Usability, Applicability*, 2010.

[37] O. Erling, "Virtuoso, a Hybrid RDBMS/Graph Column Store," *IEEE Data Engineering Bulletin*, vol. 35, no. 1, pp. 3–8, 2012.

[38] Microsoft, "Microsoft Open Database Connectivity (ODBC)," online, Microsoft Corporation, Tech. Rep., 2016, https://cdn.simba.com/wp-content/uploads/2016/03/ODBC_specification.pdf, accessed 2020-08-11.

[39] P. Křemen and Z. Kouba, "Ontology-Driven Information System Design," *IEEE Transactions on Systems, Man, and Cybernetics: Part C*, vol. 42, no. 3, pp. 334–344, 5 2012.

[40] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: Implementing the Semantic Web Recommendations," in *Proceedings of the 13th international World Wide Web conference (Alternate Track Papers & Posters)*, 2004, pp. 74–83.

[41] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Semantic Web – Interoperability, Usability, Applicability*, 2011.

[42] C. Puleston, B. Parsia, J. Cunningham, and A. Rector, "Integrating Object-Oriented and Ontological Representations: A Case Study in Java and OWL," in *The Semantic Web - ISWC 2008*, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. DOI 10.1007/978-3-540-88564-1_9. ISBN 978-3-540-88564-1 pp. 130–145.

[43] JCP, "JDBC^{TM} 4.2 Specification," Java Community Process, Tech. Rep., 2014.

[44] JCP, "JSR 317: Java^{TM} Persistence API, Version 2.0," Java Community Process, Tech. Rep., 2009.

[45] M. Quasthoff and C. Meinel, "Supporting Object-Oriented Programming of Semantic-Web Software," *Transactions on Systems, Man, and Cybernetics, Part C: Applications*, vol. 42, no. 1, pp. 15–24, 1 2012. DOI 10.1109/TSMCC.2011.2151282

[46] E. Oren, B. Heitmann, and S. Decker, "ActiveRDF: Embedding Semantic Web data into object-oriented languages," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 3, 2008.

[47] M. Ledvinka and P. Křemen, "A comparison of object-triple mapping libraries," *Semantic Web*, vol. 11, no. 3, pp. 483–524, 2020. DOI 10.3233/SW-190345

[48] J. Frohn, G. Lausen, and H. Uphoff, "Access to objects by path expressions and rules," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 273–284.

[49] G. Stevenson and S. Dobson, "Sapphire: Generating Java Runtime Artefacts from OWL Ontologies," in *Advanced Information Systems Engineering Workshops*, C. Salinesi and O. Pastor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. DOI 10.1007/978-3-642-22056-2_46. ISBN 978-3-642-22056-2 pp. 425–436.

[50] J. Leigh, "AliBaba," Online, 2007, https://bitbucket.org/openrdf/alibaba/, accessed 2020-08-11.

[51] P. Mika, "Social Networks and the Semantic Web," Ph.D. dissertation, Vrije Universiteit Amsterdam, 2006, https://research.vu.nl/ws/portalfiles/portal/77648468/complete+dissertation.pdf, accessed 2020-08-11.

[52] M. Grove, "Empire: RDF & SPARQL Meet JPA," *Dataversity*, April 2010. [Online]. Available: http://www.dataversity.net/empire-rdf-sparql-meet-jpa/ ,accessed2020-08-11

[53] K. Wenzel, "KOMMA: An Application Framework for Ontology-based Software Systems," *Semantic Web – Interoperability, Usability, Applicability*, 2010, http://www.semantic-web-journal.net/sites/default/files/swj89_0.pdf, accessed 2020-08-11.

[54] A. Alishevskikh, "RDFBeans," online, 2017, https://rdfbeans.github.io, accessed 2020-08-11.

[55] F. M. Donini, D. Nardi, and R. Rosati, "Description Logics of Minimal Knowledge and Negation As Failure," *ACM Trans. Comput. Logic*, vol. 3, no. 2, pp. 177–225, Apr. 2002. DOI 10.1145/505372.505373

[56] S. Grimm and B. Motik, "Closed World Reasoning in the Semantic Web through Epistemic Operators," in *Proceedings of the OWLED*05 Workshop on OWL: Experiences and Directions*, ser. CEUR Workshop Proceedings, B. C. Grau, I. Horrocks, B. Parsia, and P. Patel-Schneider, Eds., vol. 250. CEUR-WS.org, 1 2005, pp. 1–10.

[57] B. Motik, I. Horrocks, and U. Sattler, "Bridging the gap between OWL and relational databases," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 2, 2009.

[58] P. F. Patel-Schneider and E. Franconi, "Ontology Constraints in Incomplete and Complete Data," in *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, ser. ISWC'12. Berlin, Heidelberg: Springer-Verlag, 2012. DOI 10.1007/978-3-642-35176-1_28. ISBN 978-3-642-35175-4 pp. 444–459.

[59] Y. Ren, J. Z. Pan, and Y. Zhao, "Closed World Reasoning for OWL2 with NBox," *Tsinghua Science and Technology*, vol. 15, no. 6, pp. 692 – 701, 12 2010. DOI 10.1016/S1007-0214(10)70117-6

[60] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits, "Combining answer set programming with description logics for the Semantic Web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 11, pp. 1577–1592, 2010. DOI 10.1109/TKDE.2010.111

[61] C. V. Damásio, A. Analyti, G. Antoniou, and G. Wagner, "Supporting Open and Closed World Reasoning on the Web," in *Principles and Practice of Semantic Web Reasoning*, J. J. Alferes, J. Bailey, W. May, and U. Schwertel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. DOI 10.1007/11853107_11 pp. 149–163.

[62] M. Knorr, J. J. Alferes, and P. Hitzler, "Local closed world reasoning with description logics under the well-founded semantics," *Artificial Intelligence*, vol. 175, no. 9-10, pp. 1528–1554, 2011. DOI 10.1016/j.artint.2011.01.007

[63] K. Sengupta, A. A. Krisnadhi, and P. Hitzler, "Local closed world semantics: Grounded circumscription for OWL," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7031 LNCS, no. PART 1, pp. 617–632, 2011. DOI 10.1007/978-3-642-25073-6_39

[64] G. Yang and M. Kifer, *Reasoning about Anonymous Resources and Meta Statements on the Semantic Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 69–97. ISBN 978-3-540-39733-5

[65] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: Combining logic programs with description logic," in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW '03. New York, NY, USA: ACM, 2003. DOI 10.1145/775152.775160. ISBN 1-58113-680-3 pp. 48–57.

[66] H. Kattenstroth, W. May, and F. Schenk, "Combining OWL with F-Logic Rules and Defaults," in *Proceedings of the ICLP'07 Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services, ALPSWS 2007*, 2007.

[67] J. de Bruijn, R. Lara, A. Polleres, and D. Fensel, "OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web," in *Proceedings of the 14th International Conference on World Wide Web*, ser. WWW '05. ACM, 2005. DOI 10.1145/1060745.1060836. ISBN 1-59593-046-9

[68] M. Balaban, "The F-Logic Approach for Description Languages," *Annals of Mathematics and Artificial Intelligence*, vol. 15, no. 1, pp. 19–60, 1995. DOI 10.1007/BF01535840

[69] J. de Bruijn and S. Heymans, "Translating ontologies from predicate-based to frame-based languages," in *Proceedings of the 2nd International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML2006)*, 2006.

[70] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, *Ontology-Based Data Access and Integration.* New York, NY: Springer New York, 2018, pp. 2590–2596. ISBN 978-1-4614-8265-9

[71] Y. Kalfoglou and M. Schorlemmer, "Ontology mapping: The state of the art," *The Knowledge Engineering Review*, vol. 18, no. 1, pp. 1–31, 1 2003. DOI 10.1017/S0269888903000651

[72] R. Verborgh, M. V. Sande, O. Hartig, J. V. Herwegen, L. D. Vocht, B. D. Meester, G. Haesendonck, and P. Colpaert, "Triple Pattern Fragments: A Low-cost Knowledge Graph Interface for the Web," *Journal of Web Semantics*, vol. 37-38, pp. 184 – 206, 2016. DOI https://doi.org/10.1016/j.websem.2016.03.003

[73] F. Michel, C. F. Zucker, and F. Gandon, "SPARQL Micro-Services: Lightweight Integration of Web APIs and Linked Data," in *LDOW Workshop of the 2018 World Wide Web Conference (WWW'18)*, 4 2018.

[74] B. Nouwt, "Tight integration of Web APIs with Semantic Web," in *SEMANTiCS-WS 2017 Workshops of SEMANTiCS 2017*, A. Fensel and L. Daniele, Eds. CEUR-WS.org, 2017.

[75] M. Schröder, J. Hees, A. Bernardi, D. Ewert, P. Klotz, and S. Stadtmüller, "Simplified SPARQL REST API," in *The Semantic Web: ESWC 2018 Satellite Events*, A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, and M. Alam, Eds. Cham: Springer International Publishing, 2018. ISBN 978-3-319-98192-5 pp. 40–45.

[76] E. Daga, L. Panziera, and C. Pedrinaci, "A BASILar Approach for Building Web APIs on Top of SPARQL Endpoints," in *SALAD 2015 Services and Applications over Linked APIs and Data*, M. Maleshkova, R. Verborgh, and S. Stadtmüller, Eds. CEUR-WS.org, 2015, pp. 22–32.

[77] M. Lanthaler and C. Gütl, "Aligning Web Services with the Semantic Web to Create a Global Read-Write Graph of Data," in *2011 IEEE Ninth European Conference on Web Services*, G. Zavattaro, U. Schreier, and C. Pautasso, Eds. Piscataway, NJ, USA: IEEE, 2011. DOI 10.1109/ECOWS.2011.17 pp. 15–22.

[78] M. Lanthaler and C. Gütl, "Hydra: A Vocabulary for Hypermedia-Driven Web APIs," in *LDOW2013 Linked Data on the Web*, C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas, and S. Auer, Eds. CEUR-WS.org, 2013.

[79] R. Isele, J. Umbrich, C. Bizer, and A. Harth, "LDSpider: An open-source crawling framework for the web of linked data," in *Proceedings of 9th International Semantic Web Conference (ISWC 2010) Posters and Demos*. Aachen: CEUR-WS.org, 2010. [Online]. Available: http://iswc2010.semanticweb.org/pdf/495.pdf

[80] G. Tummarello, R. Delbru, and E. Oren, "Sindice.com: Weaving the Open Linked Data," in *The Semantic Web*, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. DOI 10.1007/978-3-540-76298-0_40. ISBN 978-3-540-76298-0 pp. 552–565.

[81] A. Barbosa, I. I. Bittencourt, S. W. M. Siqueira, R. d. A. Silva, and I. Calado, "The Use of Software Tools in Linked Data Publication and Consumption: A Systematic Literature Review," *Int. J. Semant. Web Inf. Syst.*, vol. 13, no. 4, pp. 68–88, 10 2017. DOI 10.4018/IJSWIS.2017100104

[82] B. Heitmann, R. Cyganiak, C. Hayes, and S. Decker, *Linked Data Management*. Chapman and Hall/CRC, 2014, ch. Architecture of Linked Data Applications, pp. 69–91.

[83] T. Hoppe, B. Humm, U. Schade, T. Heuss, M. Hemmje, T. Vogel, and B. Gernhardt, "Corporate Semantic Web – Applications, Technology, Methodology," *Informatik-Spektrum*, vol. 39, no. 1, pp. 57–63, 2 2016. DOI 10.1007/s00287-015-0939-0

[84] H. Bense, B. Gernhardt, T. Hoppe, M. Hemmje, B. Humm, U. Schade, R. Schäfermeier, A. Paschke, M. Schmidt, P. Haase, M. Siegel, T. Vogel, and R. Wenning, "Emerging Trends in Corporate Semantic Web," *Informatik Spektrum*, vol. 39, no. 6, pp. 474–480, 12 2016.

[85] S. O'Riain, A. Harth, and E. Curry, "Linked Data Driven Information Systems as an Enabler for Integrating Financial Data," in *Information Systems for Global Financial Markets: Emerging Developments and Effects*. IGI Global, 2012, pp. 239–270.

[86] D. Corsar, P. Edwards, J. Nelson, C. Baillie, K. Papangelis, and N. Velaga, "Linking Open Data and the Crowd for Real-time Passenger Information," *Journal of Web Semantics*, vol. 43, no. C, pp. 18–24, 3 2017. DOI 10.1016/j.websem.2017.02.002

[87] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee, "Media Meets Semantic Web — How the BBC Uses DBpedia and Linked Data to Make Connections," in *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*. Springer-Verlag, 2009. DOI 10.1007/978-3-642-02121-3_53. ISBN 978-3-642-02120-6 pp. 723–737.

[88] R. Carvalho, J. Williams, I. Sturken, R. Keller, and T. Panontin, "Investigation Organizer: the development and testing of a Web-based tool to support mishap investigations," in *2005 IEEE Aerospace Conference*, 3 2005. DOI 10.1109/AERO.2005.1559302. ISSN 1095-323X pp. 89–98.

[89] R. Carvalho, S. Wolfe, D. Berrios, and J. Williams, "Ontology Development and Evolution in the Accident Investigation Domain," in *2005 IEEE Aerospace Conference*, 3 2005. DOI 10.1109/AERO.2005.1559634. ISSN 1095-323X pp. 1–8.

[90] R. Cacciotti, J. Valach, P. Kuneš, M. Čerňanský, M. Blaško, and P. Křemen, "Monument Damage Information System (MONDIS): An Ontological Approach to Cultural Heritage Documentation," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-5/W1, pp. 55–60, 07 2013. DOI 10.5194/isprsannals-II-5-W1-55-2013

[91] M. Blaško, R. Cacciotti, P. Křemen, and Z. Kouba, "Monument Damage Ontology," in *Progress in Cultural Heritage Preservation*, M. Ioannides, D. Fritsch, J. Leissner, R. Davies, F. Remondino, and R. Caffo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34234-9 pp. 221–230.

[92] M. Zimmermann, "Owl2Java – A Java Code Generator for OWL," Online, 2009, http://www.incunabulum.de/projects/it/owl2java/, accessed 2020-08-11.

[93] C. Bizer and A. Schultz, "The Berlin SPARQL benchmark," *International Journal On Semantic Web and Information Systems*, vol. 5, no. 2, pp. 1–24, 2009.

[94] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems," *Journal of Web Semantics*, vol. 3, no. 2-3, pp. 158–182, 2005.

[95] O. Holanda, S. Isotani, I. I. Bittencourt, D. Dermeval, and W. Alcantara, "An Object Triple Mapping System Supporting Detached Objects," *Engineering Applications of Artificial Intelligence*, vol. 62, no. C, pp. 234–251, 6 2017. DOI 10.1016/j.engappai.2017.04.010

[96] P. Cristofaro, "Virtuoso RDF Triple Store Analysis Benchmark & mapping tools RDF / OO," Online, 12 2013, https://tinyurl.com/virt-rdf-map-tools, accessed 2020-08-11.

[97] M. Sieland, "Selecting an RDF mapping library for cross-media enhancements," online, March 2015, https://tinyurl.com/sel-rdf-map-lib, accessed 2020-08-11.

[98] W. V. Siricharoen, "Ontologies and Object models in Object Oriented Software Engineering," *IAENG International Journal of Computer Science*, vol. 33, no. 1, 2 2007.

[99] G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN 9780080519562

[100] T. Lebo, S. Sahoo, and D. McGuinness, "PROV-O: The PROV Ontology," W3C, W3C Recommendation, April 2013, http://www.w3.org/TR/prov-o/, accessed 2019-10-24.

[101] P.-H. Chiu, C.-C. Lo, and K.-M. Chao, "Integrating Semantic Web and Object-Oriented Programming for Cooperative Design," *Journal of Universal Computer Science*, vol. 15, no. 9, pp. 1970–1990, 5 2009.

[102] F. Chevalier, "AutoRDF - Using OWL as an Object Graph Mapping (OGM) Specification Language," in *The Semantic Web*, H. Sack, G. Rizzo, N. Steinmetz, D. Mladenić, S. Auer, and C. Lange, Eds. Cham: Springer International Publishing, 2016. ISBN 978-3-319-47602-5 pp. 151–155.

[103] M. Völkel and Y. Sure, "RDFReactor – From Ontologies to Programmatic Data Access," in *POSTER AND DEMO AT INTERNATIONAL SEMANTIC WEB CONFERENCE (ISWC) 2005, GALWAY, IRELAND*, 2005.

[104] P. Ježek and R. Mouček, "Semantic framework for mapping object-oriented model to semantic web languages," *Frontiers in Neuroinformatics*, vol. 9, no. 3, 2015. DOI 10.3389/fninf.2015.00003

[105] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically Rigorous Java Performance Evaluation," in *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, ser. OOPSLA '07. New York, NY, USA: ACM, 2007. DOI 10.1145/1297027.1297033. ISBN 978-1-59593-786-5 pp. 57–76.

[106] J. W. Lloyd, *Foundations of Logic Programming*. Berlin, Heidelberg: Springer-Verlag, 1984. ISBN 0-387-13299-6

[107] S. Heymans, L. Ma, D. Anicic, Z. Ma, N. Steinmetz, Y. Pan, J. Mei, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, C. Feier, G. Hench, B. Wetzstein, and U. Keller, *Ontology Reasoning with Large Data Repositories*. Boston, MA: Springer US, 2008, pp. 89–128. ISBN 978-0-387-69900-4

[108] D. Comer, "Ubiquitous B-Tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, 6 1979. DOI 10.1145/356770.356776

[109] M. Ledvinka and P. Křemen, "Object-UOBM: An Ontological Benchmark for Object-Oriented Access," in *Knowledge Engineering and Semantic Web*, P. Klinov and D. Mouromtsev, Eds. Cham: Springer International Publishing, 2015. DOI 10.1007/978-3-319-24543-0_10. ISBN 978-3-319-24543-0 pp. 132–146.

[110] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2

[111] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 6 2007.

[112] Giancarlo Guizzardi, "Ontological Foundations for Structural Conceptual Models," Ph.D. dissertation, University of Twente, 2005.

[113] A. Stellato, A. Turbati, M. Fiorelli, T. Lorenzetti, E. Costetchi, C. Laaboudi, W. V. Gemert, and J. Keizer, "Towards VocBench 3: Pushing Collaborative Development of Thesauri and Ontologies Further Beyond," in *Proceedings of the 17th European Networked Knowledge Organization Systems Workshop, co-located with the 21st International Conference on Theory and Practice of Digital Libraries 2017 (TPDL 2017)*. CEUR-WS.org, 2017, pp. 39–52. [Online]. Available: http://ceur-ws.org/Vol-1937/paper4.pdf

[114] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, Massachusetts, USA: Addison-Wesley Professional, 2002. ISBN 0321127420

[115] R. Cyganiak and C. Bizer, "Pubby – A Linked Data Frontend for SPARQL Endpoints," Online, 2007, http://wifo5-03.informatik.uni-mannheim.de/pubby/, accessed 2020-08-11.

[116] G. Yang, M. Kifer, and C. Zhao, "$\mathcal{F}$lora-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, R. Meersman, Z. Tari, and D. C. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. ISBN 978-3-540-39964-3 pp. 671–688.

[117] B. Kostov, J. Ahmad, and P. Křemen, *Towards Ontology-Based Safety Information Management in the Aviation Industry*. Cham: Springer International Publishing, 2017, pp. 242–251. ISBN 978-3-319-55961-2

[118] P. Křemen, B. Kostov, M. Blaško, J. Ahmad, V. Plos, A. Lališ, S. Stojić, and P. Vittek, "Ontological Foundations of European Coordination Centre for Accident and Incident Reporting Systems," *Journal of Aerospace Information Systems*, vol. 14, no. 5, pp. 279–292, 2017, https://doi.org/10.2514/1.I010441.

[119] M. Ledvinka, A. Lališ, and P. Křemen, "Towards Data-Driven Safety: An Ontology-Based Information System," *Journal of Aerospace Information Systems*, vol. 16, no. 1, pp. 22–36, 2019. DOI 10.2514/1.I010622

[120] M. Ledvinka, P. Křemen, L. Saeeda, and M. Blaško., "TermIt: A Practical Semantic Vocabulary Manager," in *Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 1: ICEIS*, INSTICC. SciTePress, 2020. DOI 10.5220/0009563707590766. ISBN 978-989-758-423-7 pp. 759–766.

[121] A. Miles and S. Bechhofer, "SKOS Simple Knowledge Organization System Reference," W3C, W3C Recommendation, 2009, http://www.w3.org/TR/skos-reference, accessed 2020-08-11.

[122] P. Křemen and M. Nečaský, "Improving discoverability of open government data with rich metadata descriptions using semantic government vocabulary," *Journal of Web Semantics*, vol. 55, pp. 1 – 20, 2019. DOI https://doi.org/10.1016/j.websem.2018.12.009

[123] P. Křemen, L. Saeeda, M. Blaško, and M. Med, "Dataset Dashboard - a SPARQL Endpoint Explorer," in *Proceedings of the Fourth International Workshop on Visualization and Interaction for Ontologies and Linked Data co-located with the 17th International Semantic Web Conference, VOILA@ISWC 2018, Monterey, CA, USA, October 8, 2018*, 2018, pp. 70–77, http://ceur-ws.org/Vol-2187/paper7.pdf, accessed 2020-08-11.

[124] T. Klíma, "Sémantický manažer prospektivní klinické studie," B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2018, in Czech.

[125] Y. Doroshenko, "Semantic Pipeline Editor," B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2018.

[126] A. Cockburn, "Hexagonal architecture," Online, 2017, https://alistair.cockburn.us/hexagonal-architecture/, accessed 2020-08-11.

[127] J. Palermo, "The Onion Architecture," Online, 2008, https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/, accessed 2020-08-11.

[128] F. Buschman, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, 1996, vol. 1.

[129] J. Klímek, P. Škoda, and M. Nečaský, "LinkedPipes ETL: Evolved Linked Data Preparation," in *The Semantic Web*, H. Sack, G. Rizzo, N. Steinmetz, D. Mladenić, S. Auer, and C. Lange, Eds. Cham: Springer International Publishing, 2016. DOI 10.1007/978-3-319-47602-5_20. ISBN 978-3-319-47602-5 pp. 95–100.

[130] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, "Querying RDF Streams with C-SPARQL," *SIGMOD Rec.*, vol. 39, no. 1, p. 20–26, Sep. 2010. DOI 10.1145/1860702.1860705

[131] D. Miller, J. Whitlock, M. Gardiner, M. Ralphson, R. Ratovsky, and U. Sarid, "OpenAPI Specification," OpenAPI Initiative, Tech. Rep., 2020, http://spec.openapis.org/oas/v3.0.3, accessed 2020-08-11.

Publications by the Author

Journal Publications

1. **M. Ledvinka (80%)** and P. Křemen, "A comparison of object-triple mapping libraries," *Semantic Web*, vol. 11, no. 3, pp. 483–524, 2020. DOI 10.3233/SW-190345. *IF(2018)* = 3.524

2. **M. Ledvinka (45%)**, A. Lališ, and P. Křemen, "Towards Data-Driven Safety: An Ontology-Based Information System," *Journal of Aerospace Information Systems*, vol. 16, no. 1, pp. 22–36, 2019. DOI 10.2514/1.I010622. *IF(2018)* = 0.787

   - Cited by

     a. W. Gao and Y. Chen, "Approximation analysis of ontology learning algorithm in linear combination setting," in *Journal of Cloud Computing*, vol. 9, 2020. DOI 10.1186/s13677-020-00173-y

     b. Richard M. Keller, "Building a Knowledge Graph for the Air Traffic Management Community", in *Companion of the World Wide Web Conference (WWW 2019)*, 2019, pp. 700-704

     c. M. Chen, B. Liu, D. Zeng, and W. Gao, "A Framework for Ontology-Driven Similarity Measuring Using Vector Learning Tricks", in *Engineering Letters*, vol. 27, no. 3, 2019. ISSN 1816-093X, pp. 549-558

Conference Publications

1. **M. Ledvinka** (40%), P. Křemen, L. Saeeda, and M. Blaško, "TermIt: A Practical Semantic Vocabulary Manager," in *Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 1: ICEIS*, INSTICC. SciTePress, 2020. DOI 10.5220/0009563707590766. ISBN 978-989-758-423-7, pp. 759–766

2. **M. Ledvinka** (80%) and P. Křemen, "Formalizing Object-Ontological Mapping Using F-logic," in *Rules and Reasoning*. Cham. Springer, 2019. Lecture Notes in Computer Science. vol. 11784. ISSN 0302-9743. ISBN 978-3-030-31094-3, pp. 97-112

3. **M. Ledvinka** (60%), M. Blaško and P. Křemen, "Factors of Efficient Semantic Web Application Development," in *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, vol. 2. Basel. Springer, 2018. ISSN 0302-9743. ISBN 978-3-030-02670-7, pp. 565-572

4. J. Ahmad, P. Křemen, and **M. Ledvinka** (10%), "Optimization of Queries Based on Foundational Ontologies," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, 2018. ISSN 0302-9743. ISBN 978-3-030-02670-7, pp. 351-367

5. **M. Ledvinka** (80%), P. Křemen, B. Kostov, and M. Blaško, "SISel: Aviation Safety Powered by Semantic Technologies", in *Data a znalosti 2017*, Plzeň. Západočeská univerzita v Plzni, 2017. ISBN 978-80-261-0720-0, pp. 77-82

6. **M. Ledvinka** (80%), P. Křemen and B. Kostov, "JOPA: Efficient Ontology-based Information System Design", in *The Semantic Web: ESWC 2016 Satellite Events*, Cham. Springer International Publishing AG, 2016. ISSN 0302-9743. ISBN 978-3-319-47601-8, pp. 156-160

   - Cited by
     a. G. Bajaj, R. Agarwal, P. Singh, N. Georgantas, and V. Issarny, "4W1H in IoT Semantics," in *IEEE Access*, vol. 6, 2018. ISSN 2169-3536, pp. 65488 - 65506
     b. J. Kraus, A. Lališ, V. Plos, P. Vittek, and S. Stojić, "Utilizing Ontologies and Structural Conceptual Models for Safety Data Management in Aviation Maintenance, Repair and Overhaul Organizations," in *Transportation Research Procedia*, 35, pp. 35-43

7. **M. Ledvinka** (80%), P. Křemen, "JOPA: Stay Object-Oriented When Persisting Ontologies", in *17th International Conference on Enterprise Information Systems ICEIS 2015 (Softcover)*, Porto. SciTePress, 2015. Softcover. DOI 10.1007/978-3-319-29133-8. ISSN 1865-1348. ISBN 978-3-319-29132-1, pp. 408-428

8. **M. Ledvinka** (80%) and P. Křemen, "Object-UOBM: An Ontological Benchmark for Object-Oriented Access," in *Knowledge Engineering and Semantic Web*, P. Klinov and D. Mouromtsev, Eds. Cham: Springer International Publishing, 2015. DOI 10.1007/978-3-319-24543-0_10. ISBN 978-3-319-24543-0, pp. 132–146

   - Cited by
     a. C. Allocca, M. Alviano, F. Calimeri, C. Civili, R. Costabile, B. Cuteri, A. Fiorentino, D. Fuscà, S. Germano, G. Laboccetta, N. Leone, M. Manna, S. Perri, K. Reale, F. Ricca, P. Veltri, and J. Zangari, "Querying Large Expressive Horn Ontologies,", in *Proceedings of the 27th Italian Symposium on Advanced Database Systems*, CEUR Workshop Proceedings, 2019,
     b. C. Allocca, F. Calimeri, C. Civili, R. Costabile, B. Cuteri, A. Fiorentino, D. Fuscà, S. Germano, G. Laboccetta, M. Manna, S. Perri, K. Reale, F. Ricca, P. Veltri and J. Zangari, "Large-scale Reasoning on Expressive Horn Ontologies," in *Datalog 2.0 2019 3rd International Workshop on the Resurgence of Datalog in Academia and Industry*, CEUR Workshop Proceedings, Volume 2368, 2019, pp. 10-21

9. **M. Ledvinka** (80%), and P. Křemen, "JOPA: Accessing Ontologies in an Object-oriented Way," in *Proceedings of the 17th International Conference on Enterprise Information Systems*, Porto. SciTePress, 2015.
DOI 10.5220/0005400302120221. ISBN 978-989-758-097-0, pp. 212-221

   - Cited by
     a. R. Pinka, J. Kaiser, "Multipurpose digital platform in the construction industry development processes for risk and life-cycle assessment," in *Proceedings of the 29th International Business Information Management Association Conference - Education Excellence and Innovation Management through Vision 2020: From Regional Development Sustainability to Global Economic Growth*, 2017, pp. 2602-2616
     b. P. Vittek, A. Lališ, S. Stojić, and V. Plos, "Challenges of implementation and practical deployment of aviation safety knowledge management software," in *Communications in Computer and Information Science*, 518, 2016, DOI 10.1007/978-3-319-45880-9_24, pp. 132-146

10. **M. Ledvinka** (80%) and P. Křemen, "JOPA: Developing Ontology-Based Information Systems," in *Proceedings of the 13th Annual Conference Znalosti 2014*, Praha. VŠE, 2014. ISBN 978-80-245-2054-4, pp. 108-117

11. P. Křemen, M. Blaško, M. Šmíd, Z. Kouba, **M. Ledvinka** (5%), and B. Kostov, "MONDIS: Using Ontologies for Monument Damage Descriptions," in *Proceedings of the 13th Annual Conference Znalosti 2014*, Praha. VŠE, 2014. ISBN 978-80-245-2054-4, pp. 66-69

Methodologies

1. M. Strouhal, P. Vittek, P. Křemen, V. Plos, M. Lánský, S. Szabo, M. Novák, E. Endrizalová, et al., "Possibilities of using SISel to create SSP and SSp," *Methodology*, 2016
2. P. Vittek, A. Lališ, S. Stojić, V. Plos, J. Kraus, M. Lánský, S. Szabo, V. Němec, et al., "The methodology for the implementation of safety indicators and their use for the purposes of safety management in aviation organizations," *Applied Certified Methodology* (certified by the Ministry of Transport of the Czech Republic), 2016
3. M. Strouhal, P. Vittek, V. Plos, M. Lánský, S. Szabo, M. Novák, E. Endrizalová, J. Kraus, et al., "The methodology for establishing, operating and maintaining a national system for the management of safety and supervision of the safety performance of organizations engaged in civil aviation," *Applied Certified Methodology* (certified by the Ministry of Transport of the Czech Republic), 2015

Software

1. **M. Ledvinka**, B. Kostov, P. Křemen, M. Blaško, A. Lališ, S. Stojić, V. Plos, P. Vittek, et al., "SISel," *Software*, 2016
2. **M. Ledvinka**, B. Kostov, M. Blaško, P. Křemen, P. Vittek, J. Kraus, S. Stojić, V. Plos, and A. Lališ, "Reporting Tool," *Software*, 2016

Appendices

Appendix A

Abbreviations and Acronyms

Symbol	Meaning
API	Application Programming Interface
ASO	Aviation Safety Ontology
CAA	Civial Aviation Authority
CPU	Central Processing Unit
CRUD	Create, Retrieve, Update, Delete
CWA	Closed World Assumption
DAO	Data Access Object
DL	Description logic
ECCAIRS	European Coordination Centre for Accident and Incident Reporting Systems
ETL	Extract, transform, load
FOL	First-order Logic
GeoSPARQL	a spatial extension to the SPARQL query language
HTTP	Hypertext Transfer Protocol
IC	Integrity Constraint
IRI	Internationalized Resource Identifier
JDBC	Java Database Connectivity
JDK	Java Development Kit
JOPA	Java OWL Persistence API
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
JVM	Java Virtual Machine
KBS	Knowledge-Based System
NAF	Negation as Failure

ODBC	Open Database Connectivity
OOM	Object-ontological Mapping
OOPL	Object-oriented Programming Language
OS	Operating System
OTM	Object-triple Mapping
OWA	Open World Assumption
OWL	Web Ontology Language
POJO	Plain Old Java Object
RAM	Random Access Memory
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational State Transfer
SDCPS	Safety Data Collecting and Processing System
SKOS	Simple Knowledge Organization System
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Query Language
SQL	Structured Query Language
UFO	Unified Foundational Ontology
UI	User Interface
UML	Unified Modeling Language
UNA	Unique Name Assumption
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Appendix B

Proofs

B.1 Proof of Lemma 4.1

As a reminder, let me first replicate Lemma 4.1 here:

Lemma B.1. *Let θ be a formula in $\mathcal{L}^{DL}$ and θ^F a corresponding F-logic formula in an F-logic language $\mathcal{L}^F$. Then θ is satisfiable in some interpretation $\mathcal{I}$ of $\mathcal{L}^{DL}$ if and only if θ^F is satisfiable in some F-structure $\boldsymbol{I}$ of $\mathcal{L}^F$.*

Proof. The lemma can be proven by showing how an F-structure $\mathbf{I}$ can be constructed for a $\mathcal{SROIQ}$ interpretation $\mathcal{I}$ and vice versa. We will demonstrate that they will have the same truth value for the corresponding axioms.

Let us begin with the RBox axioms.

- $\mathcal{I}$ is a model of $Sym(R)$ iff $\langle x, y\rangle \in R^{\mathcal{I}}$ implies $\langle y, x\rangle \in R^{\mathcal{I}}$. An F-structure $\mathbf{I}$ can be constructed such that x, y are mapped to elements $I_F(x_{\mathcal{E}})$, $I_F(y_{\mathcal{E}})$ such that $I_F(y_{\mathcal{E}}) \in I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(x_{\mathcal{E}}))$ implies $I_F(x_{\mathcal{E}}) \in I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(y_{\mathcal{E}}))$. Then, $\mathbf{I}$ is a model of $Sym_{\mathcal{P}}(R_{\mathcal{R}})$.

  Conversely, suppose $\mathbf{J}$ is a model of $Sym_{\mathcal{P}}(R_{\mathcal{R}})$, thus, it has to adhere to the condition defined in Table 4.4. For such $I_F(x_{\mathcal{E}})$, $I_F(y_{\mathcal{E}})$, we can create an interpretation $\mathcal{J}$ with elements x, y such that $\langle x, y\rangle \in R^{\mathcal{I}}$ implies $\langle y, x\rangle \in R^{\mathcal{I}}$. Thus, the mapping for symmetric role is equivalent.

- The proof for Asy, Tra, Ref, Irr and Dis is analogous. One can easily verify the equivalence of conditions on an interpretation $\mathcal{I}$ described in [24] and the F-structure $\mathbf{I}$ conditions in Table 4.4.
- Consider a role inclusion axiom $R \sqsubseteq S$. For its model $\mathcal{I}$ must hold $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. This can be expanded as $\langle x, y\rangle \in R^{\mathcal{I}}$ implies $\langle y, x\rangle \in S^{\mathcal{I}}$. Once again, we can construct an F-structure $\mathbf{I}$ containing elements $I_F(x_{\mathcal{E}})$ and $I_F(y_{\mathcal{E}})$, corresponding to x and y. Now, whenever $\langle x, y\rangle \in R^{\mathcal{I}}$, we will have $I_F(y_{\mathcal{E}}) \in I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(x_{\mathcal{E}}))$ in $\mathbf{I}$. The same for S and $S_{\mathcal{R}}$. This corresponds to the $\mathbf{I}$-condition for $subPropertyOf_{\mathcal{P}}(R_{\mathcal{R}}, S_{\mathcal{R}})$.

  It should be clear that the converse holds as well. Thus, role inclusion axioms can be faithfully mapped to the $subPropertyOf_{\mathcal{P}}$ predicate and vice versa.

A general concept inclusion axiom (GCI) $C \sqsubseteq D$ is mapped to an is-a expression $C_{\mathcal{C}} :: D_{\mathcal{C}}$. For a DL model $\mathcal{I}$ must hold $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Written explicitly, this means that for all $x \in \Delta^{\mathcal{I}}$, $x \in C^{\mathcal{I}}$ implies $x \in D^{\mathcal{I}}$. In an F-structure $\mathbf{I}$ which is a model of $C_{\mathcal{C}} :: D_{\mathcal{C}}$ must hold $I_F(C_{\mathcal{C}}) \preceq I_F(D_{\mathcal{C}})$. The semantics defined in [29] specifies that if $x \in_U I_F(C_{\mathcal{C}})$ and $I_F(C_{\mathcal{C}}) \preceq I_F(D_{\mathcal{C}})$, then $x \in_U I_F(D_{\mathcal{C}})$. Simply put, in both DL and F-logic, the relationship is that whenever an element is an instance of a subconcept/subclass, it is also an instance of its superconcept/superclass. Therefore, the mapping between DL GCI axioms and subclass expressions in F-logic is equivalent. To ensure a correct baseline for GCI mapping, we will show the correspondence between the basic DL concept descriptions and their F-logic counterparts. Of particular interest are the newly introduced function symbols Not, $AtLeast$ etc. We have to ensure that the extensions of the corresponding concepts are equivalent. The extension of a DL concept C is the set of elements $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. In F-logic, a class' extension is the set of domain elements which are in $\in_U$ relation with its domain representation $I_F(C_{\mathcal{C}})$. Let us now process the concept mapping from Table 4.3.

- For C an atomic concept name the equivalence is trivial.
- For $C \leftarrow \neg D$, $(\neg D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid x \notin D^{\mathcal{I}}\}$. We can construct an F-structure $\mathbf{I}$ where we map $x^{\mathcal{I}}$ to $I_F(x_{\mathcal{E}})$, the fact that $x^{\mathcal{I}}$ does not belong into $D^{\mathcal{I}}$ would be mapped to $\mathbf{I}$ as the lack of $\in_U$-relationship between elements $I_F(x_{\mathcal{E}})$ and $I_F(D_{\mathcal{C}})$. Such elements represent the extension of $Not(D_{\mathcal{C}})$ which proves the DL to F-logic direction.

  Now, for the other direction, the extension of $Not(D_{\mathcal{C}})$ in its model $\mathbf{J}$ is the set of all $I_F(x_{\mathcal{E}})$ such that $I_F(x_{\mathcal{E}}) \notin_U I_F(D_{\mathcal{E}})$. To build an equivalent $\mathcal{SROIQ}$ model $\mathcal{J}$, $I_F(x_{\mathcal{E}})$ will be mapped to $x^{\mathcal{J}}$, $I_F(D_{\mathcal{E}})$ to $D^{\mathcal{J}} \subset \Delta^{\mathcal{J}}$ such that $x^{\mathcal{J}} \notin D^{\mathcal{J}}$. These $x^{\mathcal{J}}$s represent the extension of $(\neg D)$, so the mapping allows to build equivalent models in both directions.
- For $C \leftarrow B \sqcap D$, its extension in a model $\mathcal{I}$ is the intersection of extensions of B and D, i.e., $(B \sqcap D)^{\mathcal{I}} = B^{\mathcal{I}} \cap D^{\mathcal{I}}$. In F-logic, the semantics of ($B_{\mathcal{C}}$ and $D_{\mathcal{C}}$) is the same, i.e., class intersection.

- For $C \leftarrow B \sqcup D$, the reasoning analogous to $B \sqcap D$.
- Take $C \leftarrow \geqslant nR.D$. For its instances $x \in \Delta^{\mathcal{I}}$ in a model $\mathcal{I}$ must hold $|\{y \in \Delta^{\mathcal{I}} \mid \langle x, y\rangle \in R^{\mathcal{I}} \wedge \ y \in D^{\mathcal{I}}\}| \geq n$. We can again construct an equivalent F-structure $\mathbf{I}$ where for each DL domain element y we create a new domain element y^F which belongs to $I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(x_{\mathcal{E}}))$ and is in a $\in_U$-relationship with $I_F(D_{\mathcal{C}})$. All such $x_{\mathcal{E}}$ constitute the extension of $AtLeast(n, R_{\mathcal{R}}, D_{\mathcal{C}})$.
- For $C \leftarrow \exists R.Self$, its extension in a model $\mathcal{I}$ is the set of all $x \in \Delta^{\mathcal{I}}$ such that $\langle x, x\rangle \in R^{\mathcal{I}}$. An equivalent F-structure $\mathbf{I}$ is built by mapping x to elements $I_F(x_{\mathcal{E}})$ such that $I_F(x_{\mathcal{E}}) \in I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(x_{\mathcal{E}}))$. Such elements comprise the extension of $HasSelf(R_{\mathcal{R}})$
- For $C \leftarrow \{a\}$, $(\{a\})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid x = a^{\mathcal{I}}\}$ in a model $\mathcal{I}$. The interpretation of $Nom(a_{\mathcal{E}})$ also relies on equality of domain elements.
- Take $C \leftarrow \forall R.D$ and a model $\mathcal{I}$, $(\forall R.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} \text{ s.t. } \langle x, y\rangle \in R^{\mathcal{I}} \Rightarrow y \in D^{\mathcal{I}}\}$. x and y are mapped to F-structure $\mathbf{I}$ elements $I_F(x_{\mathcal{E}})$ and $I_F(y_{\mathcal{E}})$ in such a way that $\forall I_F(y_{\mathcal{E}}) \in I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(x_{\mathcal{E}}))$ holds that $I_F(y_{\mathcal{E}}) \in_U I_F(D_{\mathcal{C}})$. Such an F-structure provides an extension for $All(R_{\mathcal{R}}, D_{\mathcal{C}})$. Clearly, this transformation also works in the other direction.
- Finally, let $C \leftarrow \exists R.D$. Its extension in a model $\mathcal{I}$ corresponds to $(\exists R.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists\ y \in \Delta^{\mathcal{I}} \text{ s.t. } \langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\}$. A corresponding F-structure $\mathbf{I}$ contains an element $I_F(y_{\mathcal{E}})$ such that $I_F(y_{\mathcal{E}}) \in I_{\rightarrow}(I_F(R_{\mathcal{R}}))(I_F(x_{\mathcal{E}})) \wedge I_F(y_{\mathcal{E}}) \in_U I_F(D_{\mathcal{C}})$. It can be seen that this transformation again works in both directions. Moreover $\mathbf{I}$ provides an extension of $Some(R_{\mathcal{R}}, D_{\mathcal{C}})$, so we can conclude that the mapping is again faithful.

The last part of this proof needs to deal with ABox axioms. However, ABox axioms can be *internalized* using nominals as follows: $C(a)$ to $\{a\} \sqsubseteq C$, $R(a, b)$ to $\{a\} \sqsubseteq \exists R.\{b\}$, $a = b$ to $\{a\} \equiv \{b\}$[1] and $a \neq b$ to $\{a\} \not\equiv \{b\}$. This way, there is no need to treat them explicitly. □

B.2 Proof of Theorem 4.3

For reference, here is Theorem 4.3 replicated again:

Theorem B.2. *Consider a knowledge base $\mathcal{K}$, a set of integrity constraint axioms $\mathcal{IC}$ and a set of IC validation queries $\mathcal{Q}$, constructed by applying the translation operator $\mathcal{T}$ on each IC axiom α in $\mathcal{IC}$. If $\mathcal{K}$ violates any of the IC axioms in $\mathcal{IC}$, then $\exists\ q \in \mathcal{Q}$ such that $\mathcal{K} \models^F q$.*

[1] $C \equiv D$ corresponds to $C \sqsubseteq D \wedge D \sqsubseteq C$.

Proof. First, let me show the equivalence of RBox IC axioms to their validation queries.

- The IC semantics of axiom $\alpha = subPropertyOf_{\mathcal{P}}(R, Q)$ requires an F-structure $\mathbf{I}^{IC}$ to satisfy $I_F(y) \in I^{IC}_{\rightarrow}(I_F(R))(I_F(x)) \Rightarrow I_F(y) \in I^{IC}_{\rightarrow}(I_F(Q))(I_F(x))$ for all individuals $x, y \in \mathcal{E}$. Recall that $\mathbf{I}^{IC}$ is the smallest model w.r.t. equality. For the axiom to be violated, we need to find a model in which the axiom is not true, i.e., an F-structure $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x, y \in \mathcal{E}$ s.t. $I_F(y) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y) \notin I_{\rightarrow}(I_F(Q))(I_F(x))$.

  Now, the corresponding validation query, as specified in Table 4.7, is $x[R \rightarrow y] \wedge \mathbf{not}(x[Q \rightarrow y])$, where **not** represents the absence of knowledge. The query is looking for an F-structure $\mathbf{H} \models^F I_F(y) \in I_{\rightarrow}(I_F(R))(I_F(x))$ and $\mathbf{H} \not\models^F I_F(y) \in I_{\rightarrow}(I_F(Q))(I_F(x))$, where $x, y \in \mathcal{E}$ are existentially quantified. Thus, the two parts can be combined as follows: $\mathbf{H} \models^F \exists x, y \in \mathcal{E}$ s.t. $I_F(y) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y) \notin I_{\rightarrow}(I_F(Q))(I_F(x))$. It can be seen that the query returns results if and only if integrity constraint axiom α is violated.

- For $\alpha = Sym_{\mathcal{P}}(R)$, IC semantics demands an F-structure $\mathbf{I}^{IC} \models^F I_F(y) \in I^{IC}_{\rightarrow}(I_F(R))(I_F(x)) \Rightarrow I_F(x) \in I^{IC}_{\rightarrow}(I_F(R))(I_F(y))$. A series of transformations analogous to the sub-property case above will lead to IC violation in case of an F-structure $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x, y \in \mathcal{E}$ s.t. $I_F(y) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(x) \notin I_{\rightarrow}(I_F(R))(I_F(y))$ and is a model of the knowledge base containing IC axiom α.

  The validation query $x[R \rightarrow y] \wedge \mathbf{not}(y[R \rightarrow x])$ will be looking for an F-structure $\mathbf{H}$ such that $\mathbf{H} \models^F \exists x, y \in \mathcal{E}$ s.t. $I_F(y) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(x) \notin I_{\rightarrow}(I_F(R))(I_F(y))$. Thus, we have again arrived at equivalent formulas for IC violation.

- IC axiom and validation query equivalence for $\alpha = Asy_{\mathcal{P}}(R)$ ($\alpha = Tra_{\mathcal{P}}(R)$, $\alpha = Ref_{\mathcal{P}}(R)$, $\alpha = Irr_{\mathcal{P}}(R)$, and $\alpha = Dis_{\mathcal{P}}(R, Q)$) are proven analogously.

The second part of the proof deals with general concept inclusion axioms for various concept descriptions. Once again, the goal is to show that, for each concept description, the GCI axiom, when taken as an integrity constraint, is violated if and only if a corresponding validation query, constructed using the transformation rules from Table 4.6, finds results.

- Let $\alpha = C :: Not(D)$ be an integrity constraint axiom. An F-structure $\mathbf{I}^{IC}$ satisfies it if $\mathbf{I}^{IC} \models^F I_F(C) \preceq^{IC}_U I_F(Not(D))$. α is violated if there exists an F-structure $\mathbf{J}$ such that $\mathbf{J} \not\models^F I_F(C) \preceq_U I_F(Not(D))$, i.e., $\mathbf{J} \models^F \neg(I_F(C) \preceq_U I_F(Not(D)))$. Now, from the properties of F-structures, we know that if $a \in_U b \wedge b \preceq_U c$, then $a \in_U c$ [29]. So, in $\mathbf{J}$, we have that $\neg(I_F(C) \preceq_U I_F(Not(D)))$ iff $\exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge x \notin_U I_F(Not(D))$. Given the interpretation of $Not(D)$, this can be rewritten as $\exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge x \in_U I_F(D)$.

The validation query for GCI involving $Not(D)$ is as follows: $x : C \wedge \mathbf{not}(x : Not(D))$. Thus, it searches for an F-structure $\mathbf{H}$ such that $\mathbf{H} \models^F I_F(x) \in_U I_F(C)$ and $\mathbf{H} \not\models^F I_F(x) \in_U I_F(Not(D))$, with $x \in \mathcal{E}$ existentially quantified. This can be rewritten as $\mathbf{H} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge I_F(x) \notin_U I_F(Not(D))$. Again, flipping the $\notin_U$ and $Not(D)$, the result is $\mathbf{H} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge I_F(x) \in_U I_F(D)$. So the query has results iff α is violated.

- For an IC axiom $\alpha = C :: (B \text{ and } D)$ an F-structure $\mathbf{I}^{IC}$ must satisfy $I_F(C) \preceq_U^{IC} I_F(B) \wedge I_F(C) \preceq_U^{IC} I_F(D)$. α is violated if there exists a model of the knowledge base $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge \neg(I_F(x) \in_U I_F(B) \wedge I_F(x) \in_U I_F(D))$, so, $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge (I_F(x) \notin_U I_F(B) \vee I_F(x) \notin_U I_F(D))$.

  The validation query for α – $x : C \wedge \mathbf{not}(x : (B \text{ and } D))$ – searches for an F-structure $\mathbf{H}$ such that $\mathbf{H} \models^F I_F(x) \in_U I_F(C)$ and $\mathbf{H} \not\models^F I_F(x) \in_U I_F(B) \wedge I_F(x) \in_U I_F(D)$, where x is existentially quantified over both formulas. This is again combined into a single query formula $\mathbf{H} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U C \wedge (I_F(x) \notin_U I_F(B) \vee I_F(x) \notin_U I_F(D))$. Once again, the query corresponds to the formula for violation of α.

- For an IC axiom $\alpha = C :: (B \text{ or } D)$, the proof goes along the same lines as for $C :: (B \text{ and } D)$. Thus, α is violated if there is a model $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge I_F(x) \notin_U I_F(B) \wedge I_F(x) \notin_U I_F(D)$.

  Similarly, the validation query $x : C \wedge \mathbf{not}(x : (B \text{ or } D))$ searches for an F-structure $\mathbf{H}$ such that $\mathbf{H} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge I_F(x) \notin_U I_F(B) \wedge I_F(x) \notin_U I_F(D)$, so the validation query corresponds to an IC violation.

- Take an IC axiom $\alpha = C :: AtLeast(n, R, D)$. To be satisfied, it requires an F-structure $\mathbf{I}^{IC}$ such that $\mathbf{I}^{IC} \models^F I_F(x) \in_U^{IC} I_F(C) \rightarrow (\exists y_{1..k} \in \mathcal{E}$ s.t. $I_F(y_i) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y_i) \in_U^{IC} I_F(D) \wedge \neg(I_F(y_i) = I_F(y_j))$ for $k \geq n$. Now, for α to be violated, we have to find a model $\mathbf{J}$ in which $k < n$. Thus, α is violated if $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U^{IC} I_F(C) \wedge \exists y_{1..k} \in \mathcal{E}$ s.t. $I_F(y_i) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y_i) \in_U^{IC} I_F(D) \wedge \neg(I_F(y_i) = I_F(y_j)$ for $k < n$. Here, k is treated as maximum, i.e., there can be no more that k ys.

  α is validated by $x : C \wedge \mathbf{not}(\bigwedge_{1 \leq i \leq n} x[R \rightarrow y_i] \wedge y_i : D \bigwedge_{1 \leq i \leq j \leq n} \mathbf{not}(y_i = y_j))$. Let us now rewrite it at the F-structure level. We will also replace the outer big conjunction with existential quantifier (makes no difference in terms of semantics) to allow us to match it to the IC violation formula. Thus, the query is looking for a model $\mathbf{H}$ such that $\mathbf{H} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge \exists y_{1..l} \in \mathcal{E}$ s.t. $I_F(y_i) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y) \in_U I_F(D) \wedge \bigwedge_{1 \leq i \leq j \leq n} \mathbf{not}(I_F(y_i) = I_F(y_j))$ for $l < n$. Again, l is a maximum, so no more unique ys can exist.

- Let $\alpha = C :: HasSelf(R)$ be an IC axiom. It is violated if we find a model $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in I_F(C) \wedge I_F(x) \notin I_{\rightarrow}(I_F(R))(I_F(x))$.

  The corresponding validation query searches for a model $\mathbf{H}$ for which it holds that $\mathbf{H} \models^F I_F(x) \in_U I_F(C)$ and $\mathbf{H} \not\models^F I_F(x) \in_U I_F(HasSelf(R))$ for some x from $\mathcal{E}$. Putting the formulas together and replacing $HasSelf(R)$ with the

corresponding interpretation, we get $\mathbf{H} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge I_F(x) \notin I_{\rightarrow}(I_F(R))(I_F(x))$.

- An IC axiom $\alpha = C :: Nom(a)$ is violated in a model $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in I_F(C) \wedge I_F(x) \neq I_F(a)$. It should be easy to see that the same formula can be constructed from the corresponding validation query.

- As was stated earlier, universal quantification integrity constraints are written using signature expressions in F-logic. Thus, an IC axiom α now takes the form of $\alpha = C[R \Rightarrow D]$. This requires an IC model $\mathbf{I}^{IC}$ to satisfy $I_F(x) \in_U^{IC} I_F(C) \Rightarrow (I_F(y) \in I_{\rightarrow}^{IC}(I_F(R))(I_F(x)) \Rightarrow I_F(y) \in_U^{IC} I_F(D))$ for any $x, y \in \mathcal{E}$. α is violated in a model $\mathbf{J}$ if $\mathbf{J} \models^F \exists x, y \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge I_F(y) \in I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y) \notin_U I_F(D)$.

  Now, for the validation query, we have $\mathcal{T}(C[R \Rightarrow D]) = x : C \wedge \mathbf{not}(x[R \Rightarrow D])$ (recall that signature expressions propagate from classes to instances). According to well-typing rules [29], a matching F-structure $\mathbf{H}$ satisfies $I_F(x) \in_U I_F(C) \wedge \neg(I_F(y) \in_U I_{\rightarrow}(I_F(R))(I_F(x)) \Rightarrow I_F(y) \in_U I_F(D))$, thus $I_F(x) \in_U I_F(C) \wedge I_F(y) \in_U I_{\rightarrow}(I_F(R))(I_F(x)) \wedge I_F(y) \notin_U I_F(D)$ for some $x, y \in \mathcal{E}$.

- Lastly, consider an integrity constraint axiom $\alpha = C :: Some(R, D)$, which requires F-structures to satisfy $\forall x \in \mathcal{E}\ I_F(x) \in_U^{IC} I_F(C) \Rightarrow \exists y \in \mathcal{E}$ s.t. $I_F(y) \in I_{\rightarrow}^{IC}(I_F(R))(I_F(x)) \wedge I_F(y) \in_U^{IC} I_F(D)$. α is violated if we find a model $\mathbf{J}$ such that $\mathbf{J} \models^F \exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge \forall y \in \mathcal{E}\ I_F(y) \notin I_{\rightarrow}(I_F(R))(I_F(x)) \vee I_F(y) \notin_U I_F(D)$.

  Transformation of validation query $\mathcal{T}(C :: Some(R, D)) = x : C \wedge \mathbf{not}(x : Some(R, D))$ follows the path treated several times above and arrives at the same expression as the IC α violation, i.e. $\exists x \in \mathcal{E}$ s.t. $I_F(x) \in_U I_F(C) \wedge \forall y \in \mathcal{E}\ I_F(y) \notin I_{\rightarrow}(I_F(R))(I_F(x)) \vee I_F(y) \notin_U I_F(D)$.

Thus, it has been shown that for both RBox axioms and the GCI axiom involving various concept descriptions integrity constraint checking can be reduced to query answering in F-logic. □

Appendix C

Mapping Examples

C.1 Mapping by Example – $\mathcal{O}_{S1}$

This section reproduces the mapping example based on $\mathcal{O}_{S1}$. The main purpose is to show the F-logic ontology $\mathcal{O}^F_{S1}$, which was not presented in Section 6.1. Let us start with the DL versions of the domain ontology and its integrity constraints.

$$
\begin{aligned}
\mathcal{O}_{S1} = \{ & Asset \sqsubseteq {=}1\, author.\top, Asset \sqsubseteq {\leqslant}1\, lastEditor.\top, Resource \sqsubseteq Asset, \\
& Report \sqsubseteq Asset, author \sqsubseteq editor, lastEditor \sqsubseteq editor, \\
& Report \sqsubseteq {=}1\, documents.Occurrence, \\
& Occurrence \sqsubseteq {=}1\, hasSeverity.Severity, \\
& Severity \equiv \{observation, incident, accident\}, \\
& User \sqsubseteq {=}1\, firstName.string, User \sqsubseteq {=}1\, lastName.string, \\
& User \sqsubseteq {=}1\, username.string\} \\
\mathcal{IC}_{S1} = \{ & Report \sqsubseteq {=}1\, author.User, Report \sqsubseteq \forall\, author.User, \\
& Report \sqsubseteq {\leqslant}1\, lastEditor.User, Report \sqsubseteq \forall\, lastEditor.User, \\
& Report \sqsubseteq {=}1\, documents.Occurrence, Report \sqsubseteq \forall\, documents.Occurrence, \\
& Report \sqsubseteq \forall\, hasAttachment.Resource, \\
& Occurrence \sqsubseteq {=}1\, hasSeverity.Severity, \\
& Occurrence \sqsubseteq \forall\, hasSeverity.Severity, \\
& User \sqsubseteq {=}1\, firstName.string, User \sqsubseteq \forall\, firstName.string, \\
& User \sqsubseteq {=}1\, lastName.string, User \sqsubseteq \forall\, lastName.string, \\
& User \sqsubseteq {=}1\, username.string, User \sqsubseteq \forall\, username.string\}
\end{aligned}
$$

This is then followed by the corresponding F-logic variants. Notice that the `hasAttachment` attribute of `Report` has no cardinality constraint, only its target type is restricted to `Resource`.

$$
\begin{aligned}
\mathcal{O}^F_{S1} = \{ & Asset_{\mathcal{C}} :: Some(author_{\mathcal{R}}, Thing_{\mathcal{C}}), Asset_{\mathcal{C}} :: AtMost(1, author_{\mathcal{R}}, Thing_{\mathcal{R}}), \\
& subPropertyOf(author_{\mathcal{R}}, editor_{\mathcal{R}}), subPropertyOf(lastEditor_{\mathcal{R}}, editor_{\mathcal{R}}), \\
& Resource_{\mathcal{C}} :: Asset_{\mathcal{C}}, \\
& Report_{\mathcal{C}} :: Asset_{\mathcal{C}}, Report_{\mathcal{C}} :: Some(documents_{\mathcal{R}}, Occurrence_{\mathcal{C}}), \\
& Report :: AtMost(1, documents_{\mathcal{R}}, Occurrence_{\mathcal{C}}), \\
& Occurrence_{\mathcal{C}} :: Some(hasSeverity_{\mathcal{R}}, Severity_{\mathcal{C}}), \\
& Severity = Nom(observation_{\mathcal{E}}, incident_{\mathcal{E}}, accident_{\mathcal{E}}), \\
& User_{\mathcal{C}} :: All(firstName_{\mathcal{R}}, \_string), User_{\mathcal{C}} :: All(lastName_{\mathcal{R}}, \_string), \\
& User_{\mathcal{C}} :: All(username_{\mathcal{R}}, \_string), \\
& observation_{\mathcal{E}} : Severity_{\mathcal{C}}, incident_{\mathcal{E}} : Severity_{\mathcal{C}}, accident_{\mathcal{E}} : Severity_{\mathcal{C}} \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{IC}^F_{S1} = \{ & Report_{\mathcal{C}}[author_{\mathcal{R}\{1,1\}} => User_{\mathcal{C}}; \\
& \quad lastEditor_{\mathcal{R}\{0,1\}} => User_{\mathcal{C}}; \\
& \quad documents_{\mathcal{R}\{1,1\}} => Occurrence_{\mathcal{C}}; \\
& \quad hasAttachment_{\mathcal{R}} => Resource_{\mathcal{C}}], \\
& Occurrence_{\mathcal{C}}[hasSeverity_{\mathcal{R}\{1,1\}} => Severity_{\mathcal{C}}], \\
& User_{\mathcal{C}}[firstName_{\mathcal{R}\{1,1\}} => \_string; \\
& \quad lastName_{\mathcal{R}\{1,1\}} => \_string; \\
& \quad username_{\mathcal{R}\{1,1\}} => \_string] \}
\end{aligned}
$$

The Java object model is not reproduced here, but can be found in the published project.[1] Below are the two sample ABoxes, $\mathcal{A}_{S11}$ is valid w.r.t. integrity constraints,

[1] `https://kbss.felk.cvut.cz/gitblit/summary/ml-oom-validation.git`, accessed 2020-08-11.

while $\mathcal{A}_{S12}$ is not.

$$
\begin{aligned}
\mathcal{A}_{S11} = \{&Occurrence(occurrence01), hasSeverity(occurrence01, observation),\\
&Occurrence(occurrence02), hasSeverity(occurrence02, accident),\\
&User(tom), firstName(tom, "Thomas"), lastName(tom, "Lasky"),\\
&username(tom, "lasky@unsc.org"),\\
&User(sarah), firstName(sarah, "Sarah"), lastName(sarah, "Palmer"),\\
&username(sarah, "palmer@unsc.org"),\\
&Report(report01), Report(report02),\\
&author(report01, tom), documents(report01, occurrence01),\\
&author(report02, tom), documents(report02, occurrence02),\\
&lastEditor(report02, sarah)\}\\
\mathcal{A}_{S12} = \{&Occurrence(occurrence01), Report(report01),\\
&documents(report01, occurrence01)\}
\end{aligned}
$$

Finally, the F-logic versions of the ABoxes look as follows:

$$
\begin{aligned}
\mathcal{A}^F_{S11} = \{&occurrence01_{\mathcal{E}} : Occurrence_{\mathcal{C}}, occurrence02_{\mathcal{E}} : Occurrence_{\mathcal{C}},\\
&occurrence01_{\mathcal{E}}[hasSeverity_{\mathcal{R}} \rightarrow observation_{\mathcal{E}}],\\
&occurrence02_{\mathcal{E}}[hasSeverity_{\mathcal{R}} \rightarrow accident_{\mathcal{E}}],\\
&tom_{\mathcal{E}} : User_{\mathcal{C}}, sarah_{\mathcal{E}} : User_{\mathcal{C}},\\
&tom_{\mathcal{E}}[firstName_{\mathcal{R}} \rightarrow "Tom"\hat{}\hat{}\_string;\\
&\quad lastName_{\mathcal{R}} \rightarrow "Lasky"\hat{}\hat{}\_string;\\
&\quad username_{\mathcal{R}} \rightarrow "lasky@unsc.org"\hat{}\hat{}\_string],\\
&sarah_{\mathcal{E}}[firstName_{\mathcal{R}} \rightarrow "Sarah"\hat{}\hat{}\_string;\\
&\quad lastName_{\mathcal{R}} \rightarrow "Palmer"\hat{}\hat{}\_string;\\
&\quad username_{\mathcal{R}} \rightarrow "palmer@unsc.org"\hat{}\hat{}\_string],\\
&report01_{\mathcal{E}} : Report_{\mathcal{R}}, report02_{\mathcal{E}} : Report_{\mathcal{R}},\\
&report01_{\mathcal{E}}[author_{\mathcal{R}} \rightarrow tom_{\mathcal{E}};\\
&\quad documents_{\mathcal{R}} \rightarrow occurrence01_{\mathcal{E}}],\\
&report02_{\mathcal{E}}[author_{\mathcal{R}} \rightarrow tom_{\mathcal{E}};\\
&\quad documents_{\mathcal{R}} \rightarrow occurrence02_{\mathcal{E}};\\
&\quad lastEditor_{\mathcal{R}} \rightarrow sarah_{\mathcal{E}}]\}\\
\mathcal{A}^F_{S12} = \{&occurrence01_{\mathcal{E}} : Occurrence_{\mathcal{C}}, report01_{\mathcal{E}} : Report_{\mathcal{R}},\\
&report01_{\mathcal{E}}[documents_{\mathcal{R}} \rightarrow occurrence01_{\mathcal{E}}]\}
\end{aligned}
$$

C.2 Mapping by Example – $\mathcal{O}_{S2}$

This section presents the second mapping example mentioned in Section 6.1. It is based on an ontology of safety and security events, and particularly concerns threats to airborne and ground objects. Below is the DL ontology $\mathcal{O}_{S2}$ and its corresponding set of integrity constraints $\mathcal{IC}_{S2}$.

$$\begin{aligned}
\mathcal{O}_{S2} =&\{GroundObject \sqsubseteq \neg AirborneObject, SafetyEvent \sqsubseteq Event,\\
&SecurityEvent \sqsubseteq SafetyEvent,\\
&UnlawfulAction \sqsubseteq (Action \sqcap SafetyEvent),\\
&UnlawfulAction \sqsubseteq \exists\, hasPunishment.Punishment,\\
&BombThreat \sqsubseteq (SecurityEvent \sqcap UnlawfulAction),\\
&BombThreat \sqsubseteq \exists\, against.(AirborneObject \sqcup GroundObject)\}\\
\mathcal{IC}_{S2} =&\{GroundObject \sqsubseteq \neg AirborneObject,\\
&BombThreat \sqsubseteq (SecurityEvent \sqcap UnlawfulAction),\\
&UnlawfulAction \sqsubseteq \exists\, hasPunishment.Punishment,\\
&UnlawfulAction \sqsubseteq \forall\, hasPunishment.Punishment,\\
&BombThreat \sqsubseteq \exists\, against.(AirborneObject \sqcup GroundObject),\\
&BombThreat \sqsubseteq \forall\, against.(AirborneObject \sqcup GroundObject)\}
\end{aligned}$$

The F-logic versions – $\mathcal{O}^F_{S2}$ and $\mathcal{IC}^F_{S2}$ are presented next. Of particular importance is the set of integrity constraints $\mathcal{IC}^F_{S2}$, which serves as the base of the final Java object model.

$$\begin{aligned}
\mathcal{O}^F_{S2} =&\{GroundObject_{\mathcal{C}} :: Not(AirborneObject_{\mathcal{C}}), SafetyEvent_{\mathcal{C}} :: Event_{\mathcal{C}},\\
&SecurityEvent_{\mathcal{C}} :: SafetyEvent_{\mathcal{C}},\\
&UnlawfulAction_{\mathcal{C}} :: (Action_{\mathcal{C}} \text{ and } SafetyEvent_{\mathcal{C}}),\\
&UnlawfulAction_{\mathcal{C}} :: Some(hasPunishment_{\mathcal{R}}, Punishment_{\mathcal{C}})\}\\
&BombThreat_{\mathcal{C}} :: (SecurityEvent \text{ and } UnlawfulAction_{\mathcal{C}}),\\
&BombThreat_{\mathcal{C}} :: Some(against_{\mathcal{R}}, AirborneObject_{\mathcal{C}} \text{ or } GroundObject),\\
\mathcal{IC}^F_{S2} =&\{UnlawfulAction_{\mathcal{C}}[hasPunishment_{\mathcal{R}\{1,*\}} \Rightarrow Punishment_{\mathcal{C}}],\\
&BombThreat_{\mathcal{C}}[against_{\mathcal{R}\{1,*\}} \Rightarrow AirborneObject_{\mathcal{C}} \text{ or } GroundObject_{\mathcal{C}}],\\
&BombThreat_{\mathcal{C}} :: (SecurityEvent \text{ and } UnlawfulAction_{\mathcal{C}}),\\
&GroundObject_{\mathcal{C}} :: Not(AirborneObject_{\mathcal{C}})\}
\end{aligned}$$

Now, Section 6.1.2 discussed some mapping features missing in JOPA. The most

prominent of them is arguably multiple inheritance. Listings C.1 and C.2 provide at least a glimpse of how its mapping should eventually look like in the context of the current example. Since Java does not support multiple inheritance at the class level, interfaces are utilized to realize this relationship.

Listing C.1: Mapping of multiple inheritance in Java based on $\mathcal{IC}^F_{S2}$ to Java interfaces.

```
@Namespace(prefix = "ev",
    namespace =
      "http://example.org/evaluation-02/")
@OWLClass(iri = "ev:UnlawfulAction")
public interface UnlawfulAction {

    @ParticipationConstraints(nonEmpty = true)
    @OWLObjectProperty(iri =
      "ev:hasPunishment")
    Set<Punishment> getPunishments();

    void setPunishments(Set<Punishment>
      punishments);
}

@OWLClass(iri = "ev:SecurityEvent")
public interface SecurityEvent {
}

@OWLClass(iri = "ev:BombThreat")
public interface BombThreat extends
  SecurityEvent, UnlawfulAction {

    @ParticipationConstraints(nonEmpty = true)
    @OWLObjectProperty(iri = "ev:against")
    Set<Thing> getTargets();

    void setTargets(Set<Thing> targets);
}
```

Listing C.2: Concrete class BombThreatImpl implements the corresponding interfaces.

```
class BombThreatImpl implements BombThreat {

    @Id
    private String id;

    private Set<Punishment> punishments;

    private Set<Thing> targets;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    @Override
    public Set<Punishment> getPunishments() {
        return punishments;
    }

    @Override
    public void setPunishments(Set<Punishment>
      punishments) {
        this.punishments = punishments;
    }

    @Override
    public Set<Thing> getTargets() {
        return targets;
    }

    @Override
    public void setTargets(Set<Thing> targets) {
        this.targets = targets;
    }
}
```

The sample data consist of two examples, where the first ($\mathcal{A}_{S21}$) is valid w.r.t. integrity constraints and the second ($\mathcal{A}_{S22}$) is invalid. The datasets can be expressed

in description logics as follows:

$$\begin{aligned}\mathcal{A}_{S21} =\{&Punishment(fine), Punishment(imprisonment),\\ &Punishment(probation),\\ &AirborneObject(oceanic815), GroundObject(pragueAirport),\\ &BombThreat(threat01), against(threat01, oceanic815),\\ &hasPunishment(threat01, imprisonment),\\ &BombThreat(threat02), against(threat02, pragueAirport),\\ &hasPunishment(threat02, probation)\}\\ \mathcal{A}_{S22} =\{&Punishment(fine), Punishment(imprisonment),\\ &Punishment(probation),\\ &AirborneObject(oceanic815), GroundObject(pragueAirport),\\ &BombThreat(threat01), against(threat01, oceanic815),\\ &BombThreat(threat02), hasPunishment(threat02, probation),\\ &BombThreat(threat03), hasPunishment(threat03, probation),\\ &against(threat03, unknownObject)\}\end{aligned}$$

Their F-logic translation then has the following structure:

$$\begin{aligned}\mathcal{A}^F_{S21} =\{&fine_{\mathcal{E}}:Punishment_{\mathcal{C}}, imprisonment_{\mathcal{E}}:Punishment_{\mathcal{C}},\\ &probation_{\mathcal{E}}:Punishment_{\mathcal{C}},\\ &oceanic815_{\mathcal{E}}:AirborneObject_{\mathcal{C}}, pragueAirport_{\mathcal{E}}:GroundObject_{\mathcal{C}},\\ &threat01_{\mathcal{E}}:BombThreat_{\mathcal{C}}, threat02_{\mathcal{E}}:BombThreat_{\mathcal{C}},\\ &threat01_{\mathcal{E}}[against_{\mathcal{R}} \rightarrow oceanic815_{\mathcal{E}};\\ &\qquad hasPunishment_{\mathcal{R}} \rightarrow imprisonment_{\mathcal{E}}],\\ &threat02_{\mathcal{E}}[against_{\mathcal{R}} \rightarrow pragueAirport_{\mathcal{E}};\\ &\qquad hasPunishment_{\mathcal{R}} \rightarrow probation_{\mathcal{E}}]\}\\ \mathcal{A}^F_{S22} =\{&fine_{\mathcal{E}}:Punishment_{\mathcal{C}}, imprisonment_{\mathcal{E}}:Punishment_{\mathcal{C}},\\ &probation_{\mathcal{E}}:Punishment_{\mathcal{C}},\\ &oceanic815_{\mathcal{E}}:AirborneObject_{\mathcal{C}}, pragueAirport_{\mathcal{E}}:GroundObject_{\mathcal{C}},\\ &threat01_{\mathcal{E}}:BombThreat_{\mathcal{C}}, threat02_{\mathcal{E}}:BombThreat_{\mathcal{C}},\\ &threat01_{\mathcal{E}}[against_{\mathcal{R}} \rightarrow oceanic815_{\mathcal{E}}],\\ &threat02_{\mathcal{E}}[hasPunishment_{\mathcal{R}} \rightarrow probation_{\mathcal{E}}],\\ &threat03_{\mathcal{E}}:BombThreat_{\mathcal{C}},\\ &threat03_{\mathcal{E}}[against_{\mathcal{R}} \rightarrow unknownObject_{\mathcal{E}};\\ &\qquad hasPunishment_{\mathcal{R}} \rightarrow imprisonment_{\mathcal{E}}]\}\end{aligned}$$

As already explained in Section 6.1, all the examples are available in the sample project, together with the corresponding IC validation queries. The examples from the project can be evaluated using a triple store with at least RDFS inference capabilities (in case of the DL version) or Flora-2 (in case of the F-logic variant).

Appendix D

Semantic Web Developer Survey – Complete Results

This section presents the complete answers of the survey among Semantic Web developers, described in Section 6.4. In the following, each sub-section represents a single question of the survey. Answers are enumerated under each questions. Answers with the same number in different questions are by the same respondent. Answers were not edited, besides correcting occasional typos. Answers of the Linked Pipes ETL editor developers (no. 10) were translated from Czech to English. Empty item means no answer was provided for the respective question.

Special thanks go to Armando Stellato (Semantic Turkey and VocBench), David Corsar (GetThere), and Petr Škoda (Linked Pipes ETL) for their detailed answers.

0. Please, provide a short description of the nature of Semantic Web-based information systems you develop. Names, links to source code or Web page are welcome.

1. C-SPARQL Engine: an RDF Stream Processing engine with stream reasoning capabilities.
2.
3.

4. Model Shape Validation of RDF based power system models (`https://www.entsoe.eu/digital/cim/cim-for-grid-models-exchange/`).
5. Apache Jena
6. Schema managers, datacube-mapping tools, metadata management software, custom applications for multiple domains. Everything that was done with other technologies before.
7.
8. GetThere system
9. VocBench + Semantic Turkey
10. Linked Pipes ETL editor. RDF is used both by backend and frontend directly, but its specifics are limited to minimum.
11. SPipes Editor

1. What technology do you use to access Semantic data?

1. Statement/axiom-based APIs (Jena, OWLAPI, RDF4J, rdflib, RDF.rb, etc.)
2. Statement/axiom-based APIs (Jena, OWLAPI, RDF4J, rdflib, RDF.rb, etc.)
3. Statement/axiom-based APIs (Jena, OWLAPI, RDF4J, rdflib, RDF.rb, etc.)
4. Statement/axiom-based APIs (Jena, OWLAPI, RDF4J, rdflib, RDF.rb, etc.)
5. Statement/axiom-based APIs (Jena, OWLAPI, RDF4J, rdflib, RDF.rb, etc.)
6. Radio button is not smart here. All of the above,[1] depending on the app.
7. Statement/axiom-based APIs (Jena, OWLAPI, RDF4J, rdflib, RDF.rb, etc.)
8. We hosted data in triplestores (Fuseki or RDF4J (or Sesame as it was known), and accessed, queried, updated the data via SPARQL endpoints using Jena's Fuseki SPARQL API.
9. Definitely low-level. In particular I use RDF4J (but I have been a Jena user more than a decade ago). There's a rationale for my choice on low-level. We develop core applications for RDF management, not general applications that simply store data as RDF. While the second could use AliBaba and the likes as the equivalent of what ORMs are for DBs, an application for core dev of RDF needs core access to it Within low-level APIs, then a second question could be: do we use their triple oriented APIs or even higher level, as in Jena,

[1] That is, domain-independent APIs, domain-specific APIs, SPARQL and Linked Data Platform.

for instance, they provide a sort of OO API with methods attached to a given resource type, and asking the relations of that resource, e.g. for a Class, they may have getSubClasses()) or low-level SPARQL access? In the past, Semantic Turkey made large use of triple oriented APIs of Sesame (the old name for RDF4J). However, the necessity to create complex queries and the possibility we gave to connect to external triple stores made this choice inefficient from the point of view of performances. Currently, we go for RDF4J using 99% SPARQL queries and updates. We actually have further query builders and assemblers developed by us for providing more layers of engineerization to the developer of our Web API for ST.

10. RDF4J in the Java-based backend, JS on frontend has custom handling. It was planned to use LDP, but never got to it.

11. JOPA and SPARQL, depending on the use case.

2. What led you to select the data access technology you use? Did you consider other options?

1. Jena is part of the C-SPARQL engine.

2. Maturity of the tools and license (free software).

3. Support for GeoSPARQL.

4. Simple API and little overhead; Need to work on the graph directly.

5. Developing and maintaining a custom quad store required low level API access.

6. Depends on the use-case, there is not one right or wrong way.

7. OWL 2, expressivity that can be compare to what Natural language can do

8. We wanted to use semantic technologies so that we could keep the different datasets separate to simplify updating each independently, but still maintaining the ability to link easily between them. We did consider using a more traditional solution (e.g. database) but data integration required too much coupling between the database and the program code - using linked data it was much easier.

9. I think the answer above mostly replies to this. To add some more history. In past versions of STs (before development of VocBench 3 started, so we are also talking about VB2) we actually created a further middle-layer with our own API (OWLART API). These were meant as bridges over different possible APIs. And, for a while, there has been the possibility to use both Sesame and Jena by switching the implementation of OWLART. With the move to the heavy reengineering we made in ST for the VB3 endeavour, we decided that it was simply too costly to keep up maintaining our abstraction API over different

APIs which were, in turn, middleware as well. So, since we were not interested in the peculiar OO-like API of Jena and since RDF4J seemed to meet more our requirements (we particularly appreciated the sail mechanism) we opted for fully embracing RDF4J.

10. RDF4J (Sesame at the time, 7 years ago) was not chosen for any particular reason over Jena. Since then, we have stuck with it. Using an OTM library like Empire was not considered cost effective – would introduce additional complexity, current RDF handling is short and simple enough.

11. JOPA was chosen due to convenient object-ontology mapping and SPARQL comes in handy for queries where OOM is less desirable.

3. If you are not using higher level libraries: Do you think you would benefit from using such an API? If not, what are the reasons?

1.

2.

3. Likely yes, even though I do not know of any that would suit my use case.

4. Currently developing a higher level framework for the power system modeling domain.

5. Higher levels of abstraction make low level optimisation far more difficult.

6. Again it depends on what one wants to do. Abstractions might be helpful but not always. The point of RDF is the open world model and some of these higher-level libraries might not be able to benefit from that.

7. Expressivity is generally lost at high level. Performance of low level (SPARQL) is also a point when data are big.

8. This was developed before the Linked Data Platform existed, but using SPARQL 1.1 made everything much easier in terms of code for the web services managing the data. Rather than having to worry about mapping data objects to the data store schema, we simply created some template SPARQL queries that were instantiated with values at run time. Also, given the scale of the data, using just Jena or OWL API wouldn't really have been appropriate if it wasn't combined some form of SPARQL querying. It also meant it was easy to test out different triple stores to see which worked well with different datasets.

9. Again, largely answer to 1 replies to this as well.

10. We did not see any benefits in it. Libraries for transformation of RDF to POJOs usually require annotations. Our code has one extra class for dealing with the mapping and the model itself stays simple.

11.

4. Do you need to edit data across multiple triple stores (resp. multiple named graphs in a single triple store)? How do you do it?

1. I need to edit multiple named graphs in a single triple store. I use Jena API.
2. I do it with a text editor because better tools are not available.
3. I don't have this need at the moment.
4. Situation: single triple store with multiple named graphs; Data editing by adding/deleting triples directly; It is defined which kind of information goes in which graph, so easy mapping.
5. Yes, with SPARQL Update.
6. We do that by taking care of it in the application design. Again hard to generalize.
7. No
8. Yes both. Different triple stores stored different data, and within them each had different named graphs for different versions of the dataset. This was all done by adapting the SPARQL query templates and having a configuration file that sets up Jena SPARQL client with the correct SPARQL endpoint address. Then everything was done via SPARQL. Although, we didn't update across multiple stores / graphs in a single query.
9. In VB3, we adopt a repository per project approach. Each repository can be on the same triple store or on different ones but this doesn't change much. In some rare cases (some services for rendering resources in our alignment tool), we adopt internally federated queries (which is a feature of GraphDB), only if the two repositories are on the same triple store instance and this is GraphDB. ST provides aspects in all of its APIs for changing data in different graphs. The "working graph" can be specified in a context of the API (a context is something that can be passed through each API, and thus is not even part of the signature) and can be any of the graphs being maintained by the application. This is powerful at the level of API as the user (and even developer writing the service) can simply use the same services, switching graphs, without even needing to use the GRAPH keyword. However, to the purpose of VocBench application model, we simply have one "main graph" that is meant to contain the data to be edited, while other graphs can be used for putting imported read-only data, application support data (e.g. our staging-graph: `http://vocbench.uniroma2.it/doc/user/history_and_validation.jsf#validation`) that is only managed by the system, inference data etc... So while the WebAPI allow

to switch graphs, our normal policy within the application keeps the main graph as the only "working graph" of the API.

10. We try to avoid triple stores, all the pipelines are stored in RDF files. We haven't had a use case for a classic triple store, yet. We favour approach of avoiding a database as long as it is not strictly necessary to use it.

11. No

5. What is the expressiveness of the data your application works with? What are the inference tasks the application business logic benefits from?

1. Rich Data Streams from Social Media, Call Data Records and Video Analytics IoT devices.

2.

3. The data would in general fit in the Observations & Measurements standard. I am not yet experimenting with reasoning, the hurdles I currently face are at much lower level.

4. The data is the business model; Whole European power system planning and operation depend on it; Business logic: from optimization, over market prediction to accountant.

5. N/A

6. Reasoning is helpful on many levels: Finding more generic things, describing rules etc.

7. OWL-DL-Full, reflexivity, co-graph, transitivity, dynamic extension, coherency check. Business logic is help with the one shoot learning possibilities that are particular system offer.

8. The ontologies were OWL 2 RL. Because of the scale of the data and limited computational resources available to us at the time, there was hierarchical reasoning for subclasses and subproperties but we kept the rest reasonable straightforward with little additional inferencing to ensure the responsiveness of the app (it was using streaming data from users).

9. VocBench fully supports core editing in RDF, RDFS, OWL, OWL2, SKOS, SKOS-XL, Ontolex Lemon. What does "support" mean? Obviously, a pure RDF editor would allow to edit content for any of those core vocabularies. But we know that all these vocabularies have complex expressions that would be humanly unmanageable through a simple RDF editor (think of owl:Restrictions, for which a DL/Manchester syntax is definitely required! Or of Ontolex-lemon looong indirections or even trivially, of SKOS-XL reified labels. In our domain,

all of them (as a mean to analyze what can be inferred given a certain ontology and the data modeled after it) and, strictly, none of them. Like I said, we work on core data development so I guess this question doesn't apply to us.

10. Nothing besides resolving duplicate imports.
11. No

6. What were the biggest development obstacles you had to overcome?

1. Coupling Jena with a Stream Processing system.
2. Lack of decent editors for RDF data, especially for huge datasets.
3. I am yet to find a triple store that can both support GeoSPARQL and is mature enough for production.
4. Complex data model; numerous edits which need to be traceable and revertible.
5.
6. The RDF JavaScript domain was not well developed when we started so we had to bootstrap a whole ecosystem around RDFJS.
7. Volume, even short volume of data make inference hard for computer (OWL-DL-Full is the solution and it's own problem).
8. We created all the SPARQL templates as Java formatted strings; having some Java object to SPARQL mapping (similar to POJO / Spring) would have maybe made it a bit easier. Most of the effort was in writing boilerplate code to convert URL parameters into SPARQL queries and process the results of the SPARQL query into a simple JSON structure that was more compact that SPARQL JSON to send over a mobile data connection with limited connectivity so needed to be simple to process on, potentially old, mobile phones without draining too much resources of the device.
9. Well, there are many and this would require time to reply properly. However, one important thing that comes to my mind is: low interoperability. Despite Semantic Web standards should be all about interoperability, still it's not possible to switch two triplestores and hope to have a 100% compliancy over SPARQL. And this is not only about the coverage of functional aspects (e.g. a certain aspect of SPARQL being implemented or not). Another big issue is that SPARQL is declarative but, obviously, the processes that implement the resolution of its queries are not. And, unfortunately, this might mean even very strong differences in performance. The problem is not that a triple store is simply better than one another. The problem is that knowing how a triple store approaches the resolution of a query influences the way we have to

write the query. In some stores, writing a query in a way could lead to huge computation time because simply it postpones evaluation of a variable which would enormously restrict the evaluation of other variables, thus resulting in explosion of the latter. Our solution has been to use often nested queries, as they guarantee on the order of resolution between the inner and outer.

10. Not really. There were obstacles in using RDF and Linked Data, but most of them have been avoiding by not using a triple store. Maybe just the problem of RDF4J being demanding in terms of memory consumption and having a dynamic schema and incomplete RDF.

11. Inconsistency of in-place data modification.

7. Any additional comments, experiences, feedback...

1. Jena is a wonderful project with some grey unmaintained areas.
2. We badly need better editors for RDF data.
3. The Semantic Web is a great idea, but in most tasks the tools are not really there yet.
4.
5.
6.
7. Bugs in tools are the main concern today. Tomorrow continuous data stream will have to be manage by ontology.
8.
9.
10.
11.