An Approach for Collaborative Ontology Development in Distributed and Heterogeneous Environments

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von
Lavdim Halilaj
aus
Kosovo

Bonn, 25.10.2018

Dieser Forschungsbericht wurde als Dissertation von der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Bonn angenommen und ist auf dem Hochschulschriftenserver der ULB Bonn http://hss.ulb.uni-bonn.de/diss_online elektronisch publiziert.

1. Gutachter:	Prof. Dr. Sören Auer
2. Gutachter:	Prof. Dr. Jens Lehmann
Tag der Promotion:	10.12.2018
Erscheinungsjahr:	2019

Abstract

The era of digitalization poses high demands on capturing and processing knowledge generated in everyday life in formal models. Ontologies provide common means for formal knowledge capturing and modeling for a universe of discourse. Developing ontologies, however, can be a complex, time-consuming and expensive process which requires a significant amount of resource investments. Different stakeholders, such as ontology engineers, domain experts and ultimately users, are usually involved in the development process; they may be geographically distributed and work independently in isolated environments while typically have to synchronize their contributions. Supporting the entire development life-cycle of ontology modeling places a number of challenges. Stakeholders may have different working behaviors and practices that should be accommodated. Concurrent ontology modifications performed using various authoring editors have to be tracked, integrated and may result in synchronization conflicts. Further, ensuring ontology quality according to the domain requirements is another challenge to be tackled. In the past years, several methodologies and tools have been created to enable ontology development for a number of different purposes and applications. Albeit designed to cover a range of development aspects, existing approaches lack comprehensive support of the ontology life-cycle, in particular independent work in disparate environments.

In this thesis, we tackle the problem of collaborative ontology development in distributed and heterogeneous environments, and present a stakeholder-oriented approach able to holistically assist the development of ontologies in diverse and independent scenarios. First, we define Git4Voc, a lightweight methodology comprising a set of guidelines and practices to be followed by stakeholders while modeling ontologies. We then conceive VoCol, a flexible and integrated development platform to address critical requirements from a technical perspective. Moreover, techniques for reducing the number of conflicts and allowing the efficient evaluation of test cases have been designed and implemented. VoCol can be adopted in numerous scenarios and accommodate additional tools in a well-designed and semi-automatic ontology development workflow. The benefits of this flexibility are two-fold: 1) stakeholders do not need to strictly follow a specific methodology; in contrary, they can organize their work in small and incremental development steps; and 2) consumers may provide their feedback, even though they are not directly part of the active development team. VoCol can be utilized to efficiently ensure quality ontologies with respect to the pre-defined requirements. We conducted several empirical evaluations to assess the effectiveness and efficiency of our holistic approach. More importantly, ontologies for various domains, such as Industry 4.0, life sciences and education, have been successfully developed and managed following our approach. The results from the empirical evaluations and concrete applications provide evidence that the methodology and techniques presented in this thesis comply with stakeholders' needs and effectively support the entire ontology development life-cycle in distributed and heterogeneous environments.

Contents

Part I

Preliminaries

CHAPTER 1

Introduction

In an era where data is considered as *the new oil*, the lack of meaning and well-defined structure of data will lead to interoperability and usability problems. To overcome these problems, ontologies are a representation paradigm to capture and represent the knowledge of a universe of discourse and structure it in a machine-comprehensible format. Ontologies are an integral part of a wide range of techniques, such as data integration, entity annotation, and search optimization [1]. Developing ontologies can be a time-consuming and expensive process and requires a significant investment, which is difficult to make by a single person or organization. An effective approach to tackle this problem is building ontologies in a collaborative way where all interested stakeholders are involved. It includes identifying the main terms and concepts by finding a consensus among stakeholders while defining a shared terminology and formalizing it for the intended domain.

The process of jointly building ontologies in distributed environments, which we refer to as *collaborative ontology development*, can be complex. In fact, the main challenge for stakeholders is to work collaboratively on a shared objective in a constructive and efficient way, while avoiding misunderstandings, uncertainty, and ambiguity. The involved stakeholders, which may be geographically distributed, should be able to easily express and integrate their diverse views and ideas without risking to lose the original intention. Researchers have presented methodologies and platforms to allow ontology construction in various scenarios. However, they lack of support for scenarios where stakeholders work independently in isolated and heterogeneous environments. On the other hand, *version control systems*, such as *Subversion* (SVN) or *Git*, are becoming increasingly popular for ontology development. Several aspects of ontology development—in particular with regard to revision management, access control, and governance—are already well covered by version control. Nevertheless, the support for other crucial development aspects, such as *validation*, *consistency checking*, *documentation*, and *visualization*, is missing.

This thesis intends to facilitate the bridging between the *conceptual* development process of ontologies in distributed and heterogeneous environments and *concrete* guidelines and practices to guide this process.[1] Further, our objective is to efficiently ensure qualitative ontologies, where the term "quality" is used as "the indicator of matching between a developed product and user requirements" [3]. Another contribution of the thesis is providing a semi-automatic approach to map the ontology development workflows with concurrent version control methods. The approach features a modular and extensible architecture to allow the integration of additional components or services for addressing specific aspects of ontology development.

[1] In this work, we are focused on lightweight ontologies or "vocabularies", as they are developed in initiatives like *Schema.org* and defined by the W3C [2].

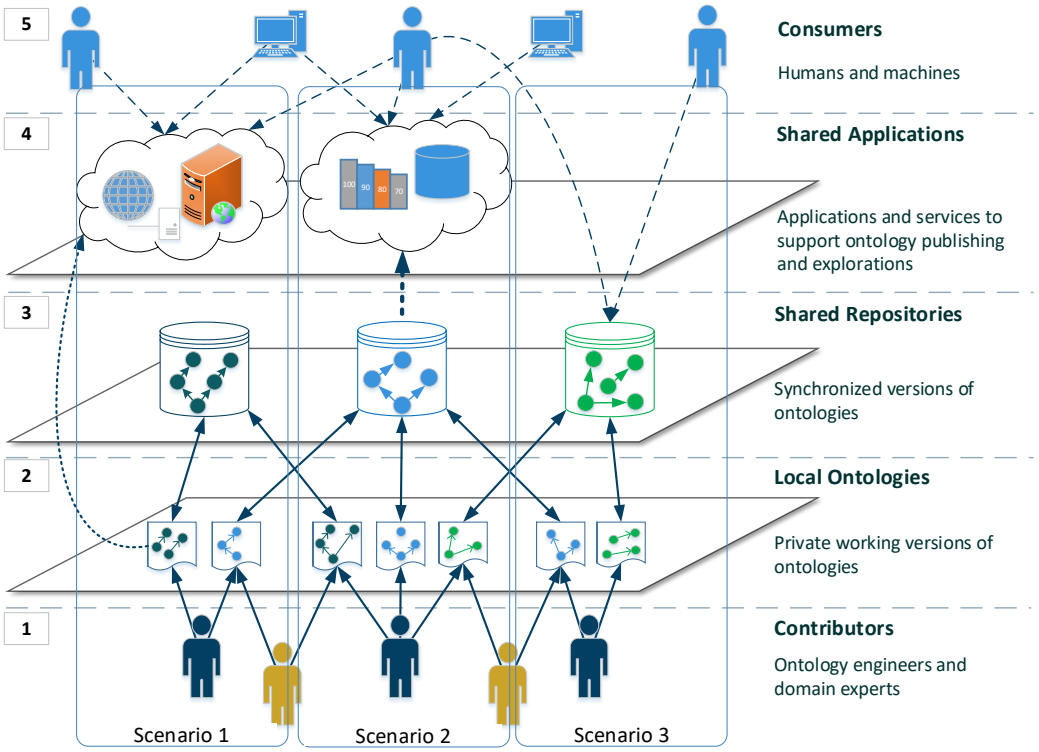


Figure 1.1: **Ontology development in distributed and heterogeneous environments**. A distributed and heterogeneous environment typically consist of several layers. Different stakeholders, such as domain experts and ontology engineers are located in the *Contributors* layer. The *Local Ontologies* layer contains working replicas of ontologies in local machines. The *Shared Repositories* layer allows the distribution and synchronization of changes among replicas. Applications located in the *Shared Applications* layer offer additional possibilities for exploration, visualization and analysis. Third party users or services are located in the *Consumers* layer. The ontology can be developed and deployed according to various scenarios, such as *Scenario 1*: there is no connection between remote *Shared Repositories* and *Shared Applications*, *Scenario 2*: a unidirectional connection exists between *Shared Repositories* and *Shared Applications*, and *Scenario 3*: the ontology is not deployed or can not be used in *Shared Applications*.

1.1 Motivation

Let us suppose a team composed of ontology engineers and domain experts working together to develop an ontology for a specific domain, as depicted in Figure 1.1, *Contributors* layer. After definition of the ontology scope, the stakeholders collect the requirements which later on will be used to define *competency questions*, i.e., questions that the ontology should be able to answer [4]. The stakeholders use a *Version Control System*, such as Git, to track the changes and manage different versions of the ontology. Initially, the stakeholders work in their local machines by performing additions and modifications based on the defined requirements. The *Local Ontologies* layer referes to the working replicas of ontologies in local machines of the contributors. To ensure that changes are conform to the requirements, a suite of test cases can be executed. In addition, the test cases can prevent violation of constraints and bad modeling practices. However, implementing such a quality assurance task is tedious and requires extra efforts. Furthermore,

contributors might use different tools to execute the test suite or automatize this task using the flexibility of version control systems. Without such quality assurance measures, the ontology may suffer from quality issues and not cover the requirements, which requires additional resources to add the missing concepts or correct the errors in later phases.

Additional mechanisms, such as *Remote Hosting Platforms* located in the *Shared Repositories* layer enable the exchanging of local changes. Therefore, changes are distributed and synchronized between local and remote ontology replicas. Ontologies may be built using various ontology authoring editors, such as *Protégé*, *TopBraid Composer* or even any plain text editor. Commonly, ontologies are represented in the RDF format and composed of a set of triples. The ordering of the triples within the file does not play any role, thus the editors can produce arbitrary files being semantically equivalent using different sorting criteria. During the synchronization process, a number of conflicts are generated, which are difficult to resolve, in particular when the ontology is sufficiently large.

The *Shared Applications* layer integrates external tools to offer additional possibilities for exploring, visualization and analysis. Further, this layer enables ontology publishing, a very important factor to further encourage the potential reusability. According to the first scenario, there is no direct connection from the *Shared Repositories* to this layer. Thus, ontology engineers have to set-up a publishing platform, where for each new version, the ontology is manually deployed. Consequently, there is always a possibility that the ontology is not well synchronized and does not reflect the latest version. In the second scenario, a link between layers: *Remote Hosting Platforms* and *Shared Applications* is established. Although the latest version of the ontology is offered, customized applications should be further developed and manually managed. The third scenario illustrates the case where the ontology is not published in any other format, apart from RDF. This forces any potential consumer to directly go to the *Remote Hosting Platform* and try to understand the ontology in the original format in order to reuse it. In all aforementioned scenarios, stakeholders are required to cope with a number of tools for performing their modeling tasks. As a consequence, divergent views of the same ontology may exist, thus making development and discussion activities among team members a challenge.

As depicted in the *Consumers* layers, third-party users or machines should be able to exploit and reuse the ontology via specialized services or rich interfaces. This allows the ontology to be adopted and extended for covering other domains or more specific requirements. Moreover, the need for methodological governance stands orthogonal to the entire development life-cycle. The team should clearly organize its work and follow recommendations for tasks, such as naming conventions, ontology modularization, branching strategies, and utilization of existing ontologies. The lack of methodological support may cause misunderstandings or uncertainty as well as difficulties to maintain and reuse the ontology being developed.

1.2 Problem Definition and Challenges

Ideally, ontology engineers and domain experts should be able to develop ontologies using version control systems and different ontology authoring editors. At conceptual level, this thesis addresses the problem of ontology development in distributed environments with a focus on synchronization of the concurrent ontology changes, while continuously enforcing quality checks on the ontology being developed in a time efficient manner. Therefore, the approach should integrate and facilitate the activities performed during the entire development life-cycle associated with interrelated discussions across various stakeholders.

Figure 1.2 illustrates four main cross-layer challenges identified by this thesis. The first challenge is to ascertain *guidelines and practices* for supporting collaborative ontology development and publication in distributed settings. The community may consist of geographically spread stakeholders with different ways of thinking and working, as well as various interests. Another important factor to be considered in this regard is the ability to contribute in a workflow independent fashion and parallel development in multiple branches. The second challenge is to allow non-expert users and client applications to reuse and consume the developed ontologies in human-friendly representations and machine-understandable formats, respectively. The third challenge is to effectively ensure the synchronization of changes among ontology replicas. Finally, the last challenge is to efficiently execute a set of test cases that check for compliance with competency questions and help avoiding violation of constraints.

As the problem of collaborative ontology development is much larger and poses many issues and obstacles in different scenarios, we consider the following challenges and problems out of the scope of this thesis: logically expressive and very large scale ontologies; heavyweight methodologies; detection and resolution of semantic conflicts; and real-time collaboration support. However, we deem that the results presented in this thesis form the foundation for extending the work also in these aspects.

Challenge 1: Supporting collaborative development of ontologies in distributed environments.

There exists a number of methodologies supporting ontology development in various scenarios. According to [5], these methodologies can be split into two main categories: centralized methodologies and decentralized methodologies. The first category includes methodologies which focus more on activities performed by a number of participants co-located and working together to develop ontologies for a very specific domain or organization. The second category applies to a larger and distributed community working jointly to define ontologies for more general purposes and usage. Other important aspects of the development process are considered by lightweight methodologies which provide generic guidelines and practices.

Apart from ontology engineers, the stakeholder community including domain experts and other third-party users is increasing. Various platforms, such as version control systems, are used to accommodate their needs and enable ontology modeling in these scenarios. These platforms are typically workflow independent where participants can work without obeying to a specific methodology. The challenge that arises here is facilitating the distributed development by providing best practices and guidelines. In particular, crucial guidelines include: modularization of ontologies, communication threads and strategies for branching. Additionally, practices for reusability, naming conventions and labeling of versions are of paramount importance.

Challenge 2: Enabling the availability and reusability of ontologies.

Publishing and consuming ontologies by third-party users and client applications requires features for ready to use and easy to explore through rich visualizations and human-friendly documentation as well as various machine-comprehensible formats. Using individual tools or services for ontology development poses additional overhead for the team, leading to inconsistent results and diverse views. Furthermore, the reusability of the ontology for other purposes or another domains is hampered. The challenge here is providing an integrated and at the same time extensible platform for developing ontologies in distributed and heterogeneous environments.

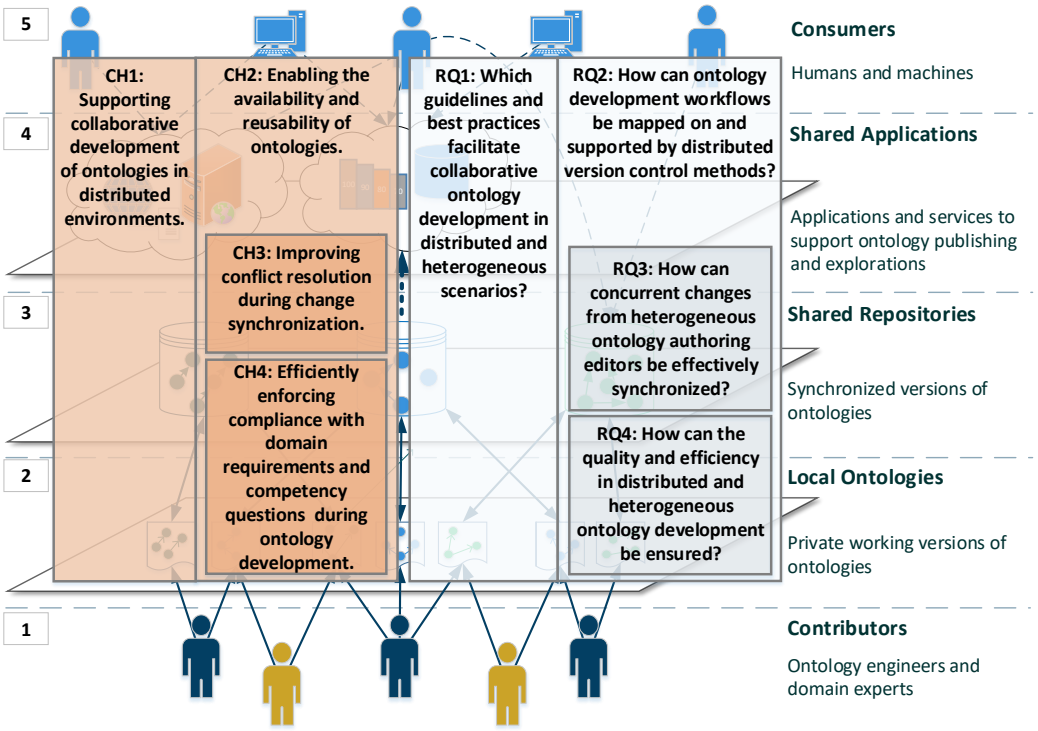


Figure 1.2: **Main Challenges and Research Questions.** This thesis identifies four main challenges to support the ontology development in distributed and heterogeneous environments. Four *research questions* are proposed with the objective of addressing these challenges. Each challenge and research question is located at the respective layer of the problem domain.

This platform should be able to accommodate a number of built-in components and services along with of-the-shelf tools to cover particular aspects of the development process. Moreover, a number of tasks helping team members on modeling activities have to be semi-automatically initiated and the output should be easily accessible.

Challenge 3: Improving conflict resolution during change synchronization.

Resolving conflicts during change synchronization between *local* and *remote replicas* of the ontology is a very time-consuming task. This is exacerbated when heterogeneous ontology authoring editors are used for modeling purposes. Git is able to detect lines within the ontology file that have been changed from the previous version. Ontologies are written in the RDF format, composed of a set of triples where their position in the file does not play any role. Therefore, editors can produce very different serializations for semantically equivalent RDF files, for example, by ordering triples alphabetically or grouping them by categories, such as *classes*, *properties* and *individuals* and later, sorting them within a category. During change synchronization, users will receive many conflicts detected by Git as a result of different sorting criteria. These conflicts should be manually resolved by comparing two versions of the same ontology file. However, in many cases, the synchronization problem is time-consuming and can result on: 1) duplication of RDF triples and several syntactic errors; 2) introducing inconsistencies and violating design decisions; or 3) with the loss of the entire change set, such as classes, properties or individuals. The challenge here is to prevent the version control system from wrongly identifying conflicts.

Challenge 4: Efficiently enforcing compliance with domain requirements and competency questions during ontology development.

The majority of ontologies are dynamic and evolve over time, in order to reflect a changing world. Thus checking for completeness, represented by competency questions and certain quality assessment indicators should be continuously performed, to avoid any possible divergence and mistake. To reduce the cost of later corrections, this task should be realized in advance in the spirit of test-driven development, which means a defined set of test cases are executed before any ontology concept is modeled. Current techniques enable execution of the entire set of test case after each change. Due to high demands of processing time, many stakeholders may avoid this task at all, thus negatively impacting the quality of the ontology with regard to conformance to initially defined requirements. Therefore, the challenge here is providing a technique for the execution of a set of test cases in time-efficient manner. Moreover, in the distributed scenario, where the team is composed of several contributors, another issue is to associate particular test cases to specific roles or contributors.

1.3 Research Questions

Based on the discussion in the motivation section where various scenarios are presented as well as the above identified challenges, we define four main research questions. Figure 1.2 illustrates these questions and their mapping to the corresponding layers.

RQ1: Which guidelines and best practices facilitate collaborative ontology development in distributed and heterogeneous scenarios?

To address this question, we analyze the nature of ontology development processes in distributed environments. The result of this step is a collection of important requirements that should be considered. Driven by the idea and the success of Git used as a version control system, in the context of software development, we study its applicability for collaborative ontology development process. Next, we investigate strategies on how to enable ontology construction even in cases when no specific workflow or methodology is followed. Finally, we examine and evaluate a number of governing aspects and development practices with respect to their application and impact that will have on concrete use cases.

RQ2: How can ontology development workflows be mapped on and supported by distributed version control methods?

With the objective of answering this research question, we devise a semi-automatic approach that is able to perform actions and tasks after occurrence of specific events. As a result, a conceptual architecture for an integrated development environment is designed. To support various scenarios and team compositions, we dive deeper in the following important aspects:

- *extensible* - enable integration of additional tools and services to provide support for specific aspects of ontology development;
- *interoperable* - work in different operating systems and allow the synchronization of results produced by heterogeneous ontology authoring editors;
- *customizable* - based on their requirements, stakeholders are able to select a subset of tools and services to be provided by the platform.

RQ3: How can concurrent changes from heterogeneous ontology authoring editors be effectively synchronized?

To answer this research question, we investigate how parallel changes performed by stakeholders in their local ontology replicas can be effectively synchronized. Particularly, we focus on reducing the number of conflicts resulting from ontology authoring editors that use different ordering criteria to sort triples in the serialization output. Therefore, only the actual conflicts, i.e., overlapping triples, are identified and can easily resolved with human intervention. Next, we study the behavior of the version control systems, such as Git, with respect to the conflict detection whenever the ordering criteria or the size of the ontology is changed.

RQ4: How can the quality and efficiency in distributed and heterogeneous ontology development be ensured?

Here, we study how to efficiently enforce the development of qualitative ontologies by ensuring that pre-defined requirements important for the representation of the domain are fulfilled. In particular, we investigate how the dependency between test cases derived from the requirements can be modeled in a directed acyclic graph. As a result, the ontology can be evaluated in an efficient manner by excluding any test case whose parents have already failed. Considering the distributed scenario where multiple stakeholders are involved and the fact that an ontology may comprise several files, we analyze how to evaluate test cases dedicated to specific users or files.

1.4 Thesis Overview

With the aim of preparing the reader for the rest of the document, we present an overview of: our main contributions and research areas investigated in this thesis; references to scientific publications covering this work; and an overview of the thesis structure.

1.4.1 Contributions

Figure 1.3 illustrates the main research contributions of the approach presented in this thesis, which provides methodological and technical support for ontology development in distributed and heterogeneous environments. In the following, these contributions are described in detail:

1. *A lightweight methodology for collaborative ontology development based on version control.*

   Contribution for RQ1 We propose a methodology for facilitating collaborative ontology development process in distributed environments. It enables experts with different backgrounds, system understanding and domain knowledge to work together while avoiding misunderstandings and lack of communication. The complexity of this process increases with the number of people involved, the variety of systems to be used and the dynamics of their domain. Therefore, we collect a number of crucial requirements by analyzing the nature of collaborative ontology development process and state of the art methods. Drawing from these findings, we define Git4Voc methodology which comprises guidelines and practices on how Git can be adopted to ontology development. In addition, we demonstrate strategies for a clear separation between different branches, and issues associated with as well as practices for naming, reuse, and documentation.

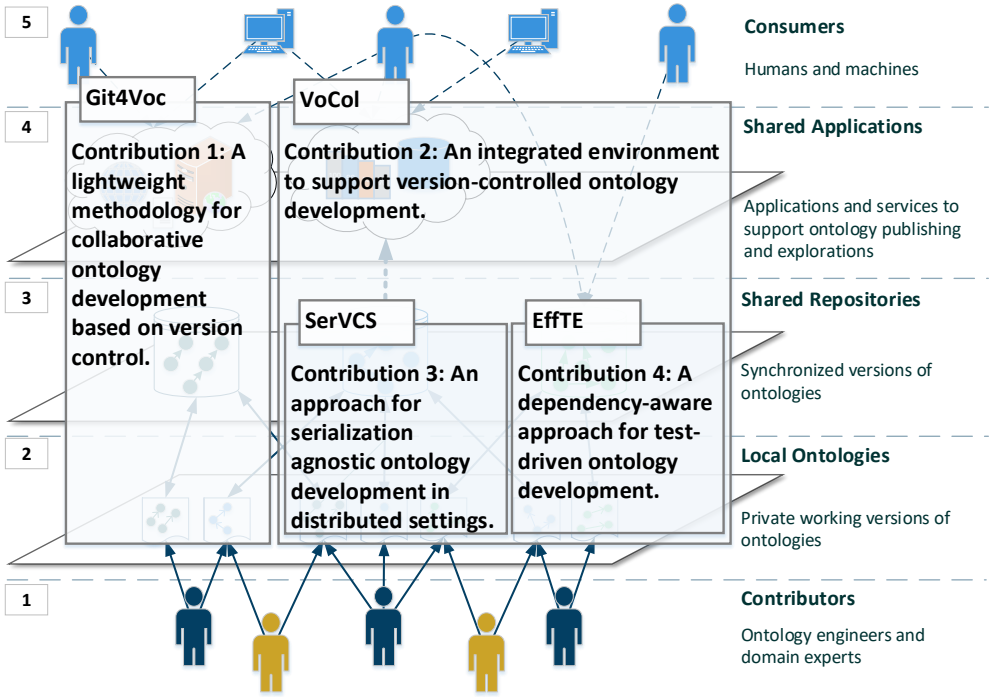


Figure 1.3: **Thesis Contributions**. Four main contributions encapsulated in our holistic approach: a lightweight methodology, an integrated environment for distributed development of ontologies, a mechanism for producing unique serialization and a method for efficiently ensuring the quality ontologies based on the domain requirements. The outcome of this thesis has been used to support modeling of ontologies in a number of different domains, such as manufacturing, health care and education.

2. *An integrated environment to support version-controlled ontology development.*

   Contribution for RQ2 We devise VoCol, a semi-automatic and modular approach for supporting ontology development in distributed and heterogeneous environments. It is based on a fundamental model of ontology development, consisting of the three core activities: modeling, population, and testing. The underlying layer of VoCol, is a pre-defined workflow which orchestrates the execution of tasks after occurrence of specific events. This platform has been implemented following principles for *extensibility*, *interoperability* and *customisability*. The modular architecture allows the platform to be extended by adding or exchanging components or services with the aim of addressing specific development aspects. We demonstrate the applicability of VoCol on a number of concrete use cases where ontologies representing different domains are collaboratively developed.

3. *An approach for serialization agnostic ontology development in distributed settings.*

   Contribution for RQ3 This approach tackles the problem of synchronization of changes performed with heterogeneous authoring editors. VCSs collect metadata describing changes and allow for their propagation to different ontology replicas. For conflict detection, VCSs usually apply techniques where files are compared line by line. We design the *SerVCS* approach, to enhance VCSs to cope with different serializations of the same ontology. Following the principle of *prevention is better than cure*, SerVCS produces a *unified representation* of the concepts before modifications are committed. As a result, the number

of *false-positive* conflicts, i.e., conflicts that do not result from ontology changes but from the fact that two ontology versions are differently serialized is significantly decreased, thus allowing stakeholders to effectively synchronize the ontology replicas.

4. *A dependency-aware approach for test-driven ontology development.*

   Contribution for RQ4 Following the principles of the *test-driven development* technique, we devise a requirement-driven approach for enforcing qualitative and efficient ontology construction. A set of test cases prevents non-intended ontology changes and avoids constraint violations. However, since the number of test cases can be large and their evaluation time may be high, the ontology development process can be negatively impacted. We present *EffTE*, an approach for efficient test-driven ontology development relying on a graph-based model of dependencies between test cases. It enables prioritization and selection of test cases to be evaluated. Traversing the dependency graph is realized via the breadth-first search algorithm along with a mechanism that tracks *tabu* test cases, i.e., test cases to be ignored for further evaluation due to faulty parent test cases. As a result, the number of evaluated test cases is minimized, thus reducing the time required for validating the ontology after each modification. Additionally, the approach performs user- or file-based evaluations, reflecting the diverse team composition and ontology modularity.

1.4.2 List of Publications

Part of the work in this thesis is based on the following publications:

1. **Lavdim Halilaj**, Irlán Grangel-González, Steffen Lohmann, Maria-Esther Vidal, Sören Auer. *EffTE: A Dependency-aware Approach for Test-Driven Ontology Development.* In 33rd ACM/SIGAPP Symposium On Applied Computing (ACM SAC) 2018 Proceedings, ACM. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I devised the formalization of the problem, led the definition and implementation of the proposed approach, reviewed related work, and prepared of the experiments and analysis of the obtained results;

2. **Lavdim Halilaj**, Irlán Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *DemoEffTE: A Demonstrator of Dependency-aware Evaluation of Test Cases over Ontology.* In 13th International Conference on Semantic Systems (Semantics) - Posters and Demo Track, 2017. This demonstration article is joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I conducted the description of the architecture, implementation and demonstration of the prototype;

3. Niklas Petersen, **Lavdim Halilaj**, Irlán Grangel-González, Steffen Lohmann, Christoph Lange, Sören Auer. *Realizing an RDF-based Information Model for a Manufacturing Company – A Case Study.* (One of the two nominees for the Best In-Use Paper Award) In 16th International Semantic Web Conference (ISWC) 2017 Proceedings, 350-366, Springer. This article is a joint work with Niklas Petersen and Irlán Grangel-González, PhD students at the University of Bonn. My contributions focused on the implementation of the proposed approach, the preparation and presentation of the use cases, and analysis of different strategies for data integration for the given scenario;

4. Irlán Grangel-González, Paul Baptista, **Lavdim Halilaj**, Steffen Lohmann, Maria-Esther Vidal, Christian Mader, Sören Auer. *The Industry 4.0 Standards Landscape from a Semantic*

Integration Perspective. In 22nd IEEE International Conference on Emerging Technologies And Factory Automation (ETFA) 2017 Proceedings. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn and Paul Baptista, a student assistant at Fraunhofer IAIS. In this article, I contributed to the implementation of the proposed approach, reviewing of related work and analysis of the obtained results;

5. **Lavdim Halilaj**, Irlan Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *SerVCS: Serialization Agnostic Ontology Development in Distributed Settings.* In Communications in Computer and Information Science (CCIS) 914 - Revised Selected Papers from 8th International Joint Conference, IC3K 2016, Porto, Portugal, 213-232, Springer. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I conducted the formalization of the problem, implementation of the approach, the revision of the state of the art approaches, the presentation of the use cases, as well as the analysis of the results;

6. **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Steffen Lohmann, Sören Auer. *Git4Voc: Collaborative Vocabulary Development Based on Git.* In International Journal of Semantic Computing (IJSC), 1-24, World Scientific. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, my contributions are related with collecting requirements for collaborative ontology development, the description of the approach, the revision of the related work and presentation of the use case evaluation;

7. **Lavdim Halilaj**, Irlan Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *Proactive Prevention of False-Positive Conflicts in Distributed Ontology Development.* (Best Paper Award) In 8th International Conference on Knowledge Engineering and Ontology Development Proceedings (KEOD), 43-51, SciTePress. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I led the formalization of the problem, implementation of the approach, the revision of the related work, the presentation of the use cases, as well as the analysis of the results;

8. **Lavdim Halilaj**, Niklas Petersen, Irlán Grangel-González, Christoph Lange, Sören Auer, Gökhan Coskun, Steffen Lohmann. *VoCol: An Integrated Environment to Support Version-Controlled Vocabulary Development.* (Paper of the Month - Fraunhofer IAIS) In 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW) 2016 Proceedings, 303-319, Springer. This article is a joint work with Niklas Petersen and Irlán Grangel-González, PhD students at the University of Bonn. In this article, I conducted the problem description, definition and implementation of the conceptual architecture, the revision of the state of the art approaches, the presentation of the use cases, and the realization of the user study evaluation;

9. Irlán Grangel-González, **Lavdim Halilaj**, Sören Auer, Steffen Lohmann, Christoph Lange, Diego Collarana. *An RDF-based Approach for Implementing Industry 4.0 Components with Administration Shells.* In IEEE Emerging Technologies and Factory Automation (ETFA) 2016 Proceedings, 1-8, IEEE. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. My contributions to this article are related to the revision of the state of the art approaches, implementation of the proposed approach, the presentation of the use cases, as well as the analysis of the results;

10. **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Sören Auer. *Git4Voc: Git-based Versioning for Collaborative Vocabulary Development.* In IEEE Tenth International Conference on Semantic Computing (ICSC) 2016 Proceedings, 285 - 292, IEEE. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I contributed to collecting requirements for collaborative development of ontologies, definition of the approach, the analysis of the related work and the presentation of the use cases in real world scenarios;

11. Irlan Grángel-González, **Lavdim Halilaj**, Gökhan Coskun, Sören Auer. *Towards Vocabulary Development by Convention.* In 7th Knowledge Engineering and Ontology Development (KEOD) 2015 Proceedings, 334-343, SciTePress. This article is a joint work with Irlan Grángel-González, a PhD student at the University of Bonn. My contributions focused on the review of state of the art approaches, devising, conducting and analyzing the user study and presentation of the outcomes.

The entire list of publications completed during the PhD studies can be found in Appendix A.

1.5 Thesis Structure

This thesis is composed of four main parts each having several chapters. It starts by providing the context, motivation, research questions and challenges in Part I. Further, this part includes the preliminaries and the related work, to help the reader for a better understanding of the thesis. Part II initially explains the collected requirements necessary for supporting collaborative ontology development in distributed and heterogeneous environments. Next, a lightweight methodology comprising a set of best practices and guidelines to be followed by stakeholders involved in the development process, is described. We then continue with the presentation of a conceptual architecture of the integrated environment and its implementation, developed for supporting ontology development centered around version control systems. Part III dives deeper into specific aspects of ontology development, such as synchronization and quality assurance with respect to the defined requirements. An approach for enabling synchronization of changes from heterogeneous ontology authoring editors by providing a unique serialization is presented. Next chapter describes an approach for efficiently ensuring quality of the ontologies by pro-actively preventing from non-indented ontology modifications. In Part IV, using the results from the previous parts, we present and discuss three real-world applications with respect to our holistic approach. Further, we provide usage statistics and other insights on how the VoCol platform is applied in a larger number of projects. Finally, this thesis is concluded by revisiting the research questions and presenting possible future directions in two perspectives: research and technical. An overview at the beginning of each part gives relevant information about the included topics.

CHAPTER 2

Background

In this chapter, we describe relevant terminology that serve as a foundation of the research realized in this thesis. In Section 2.1, we initially discuss important concepts related to ontologies as well as languages and syntaxes used to model them. We continue looking into main topics of the ontology development process and its essential activities in Section 2.2. Finally, in Section 2.3, we discuss techniques of version control systems exploited and adopted in our approach to enable *a collaborative ontology development in distributed and heterogeneous environments.*

2.1 Ontologies

The field of *graph data models* has become very active in recent years. Its main objective is to enable modeling of data and their relationships in a naturally oriented fashion. Angles et al. [6] defines graph data models as "those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors". Over the years, many graph data models are proposed, such as *Logical Data Model (LDM)*, *GROOVY*, *Simatic-XT* and *Gram*. In addition, there exist other models, such as *GraphDB*, *Object Exchange Model (OEM)*, *Database Graph Views* and *Resource Description Framework (RDF)*, that inherits graph-like features. A more detailed survey of the most representative graph data models and similar proposals, is presented in [6]. In this thesis, we are focused on ontologies which offer the means to conceptualize the knowledge of a universe of discourse and the RDF, as a modeling language for ontologies.

The term originally comes from the field of philosophy where an *Ontology* is considered as a systematic account of *being and existence*. It focuses on the study of the things, their categorizations and relations with each other for a particular domain, irrelevant whether they physically exist or not. The earliest notable work is done by Aristotle in his *Metaphysics*, where he dealt with *the study of being qua being*, which involves: 1) a study; 2) a subject matter (being); and 3) a manner in which the subject matter is studied (qua being) [7].

On the other hand, many definitions are given in the field of computer science. However, the most known definition comes from [8], where an ontology is *explicit specification of a conceptualization*. Another definition is given by [9], where an ontology is *a formal and explicit specification of a shared conceptualization*, which is considered to be one of the most complete by [10]. According to [11], a conceptualization is an abstract and simplified view of the world that we wish to represent for some purpose. During this process, a number of relevant concepts, their attributes and relations between each other, are captured. The meaning of these concepts should

be *explicit*, thus avoiding any potential ambiguity among them. In order to enable machines to understand and process the identified concepts, they should be represented in a *formal language*. Furthermore, since the ontology encapsulates the joint efforts of a community, all definitions should derive from a consensual agreement among different stakeholders.

2.1.1 The Resource Description Framework

The Resource Description Framework (RDF) [12] is a W3C Standard for representing information about resources that exists across the Web. Originally designed to allow the annotation and reuse of the resource metadata or *data about data*, it is becoming a data modeling language for encoding information from many different areas. RDF, through its design mechanisms, is intended to support publishing and interlinking of data in a human-readable and machine-understandable format, respectively. It enables the interoperability among intelligent agents by providing a standardized way of encoding and exchanging semantics of the information, regardless of their platform or domain. Thus, RDF bridges between the conceptual and operational levels of information and data representation. The basic structure of RDF, is a statement formed by three elements: **subject**, **predicate**, **object**, called *triple*, formally defined in Definition 2.1.

Definition 2.1: RDF Triple [13]

Let $\mathbf{I}$, $\mathbf{B}$, $\mathbf{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively. A tuple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is denominated an RDF triple, where s is called the subject, p the predicate, and o the object.

The **subject** represents a resource, the **object** denotes either a resource or a literal value, whereas the relationship between them is established through the **predicate**. The RDF resources are typed by simply adding a triple with the `rdf:type` property as the predicate and a suitable object representing the class in which the resource belongs to. Uniform Resource Identifiers (URIs) are used to identify resources unambiguously, while *literals* (consisting of either a string and its language tag or a value and its datatype) describe concrete data values. RDF can be represented as a directed graph composed of vertices (representing subjects and objects) and edges (representing predicates). Properties and classes required to describe and structure data of a certain domain can be defined in RDF using taxonomic relations. Such descriptions of classes and properties are called vocabularies, RDF schemas, or ontologies, and are encoded in an RDF document. An RDF document D, is defined as a set of triples: $D \subset I \times I \times (I \cup L)$, where I represents the set of URIs and L the set of literals. Therefore, RDF can be used to easily represent various types of information and data, including taxonomic/tree data, tabular/relational data, and logical axioms. Since all schema and data entities have URI identifiers associated that are worldwide unique, it is easy to link to other data (instance level) or to reuse vocabulary elements from existing ontologies (schema level).

Let us consider the following natural statement: *Lavdim studies at University of Bonn.* The RDF representation of this statement would start with the definition of: *Lavdim* and *UniversityOfBonn* as resources, where the first is denoted as a subject and the second as an object. The relationship between those two is denoted by *study at*, which is a predicate or property. The statement is encoded as following: *http://sda.uni-bonn.de/onto/staff/Lavdim*, *http://sda.uni-bonn.de/onto/staff/studyAt*, *http://uni-bonn.de/University*, where the URIs are used to provide a unique identification for each of the above elements. Further, this statement is

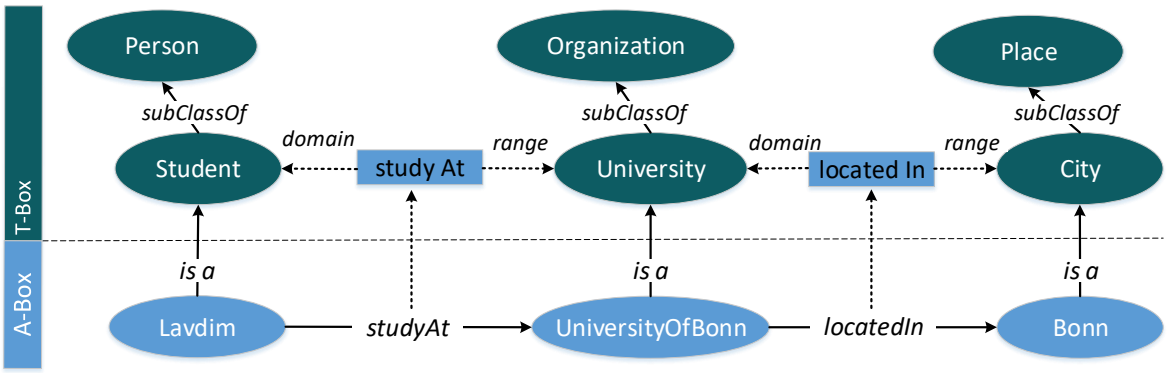


Figure 2.1: **Example of an RDF graph**. **T-Box** represents the conceptual level of entities and their inter-relationships whereas **A-Box** represents concrete instantiations of the defined concepts.

extended with another property, such *located In* to connect with the city of *Bonn*. In addition to concrete instances which belongs to A-Box or *Assertional Knowledge*, RDF allows modeling of the conceptual level in T-Box or *Terminological Knowledge*. More information, represented by *classes* and *properties* are added to better describe the knowledge about this scenario. For instance, *Lavdim* can be seen as a *Student*, *UniversityOfBonn* as a *University* and *Bonn* as a *City*. Since one of the important characteristics of ontologies is *reusability*, concepts such as *Person*, *Organization* and *Place*, can be imported from other external ontologies and interlinked as the *super classes* of *Student*, *University* and *City*, respectively. Moreover, *domain* denotes the type of a *subject* and *range* denotes the type of an *object* that a property can have. The domain values of the property *studyAt* are from *Student* class and range values are from the *University* class whereas the property *locatedIn* has *University* as domain and *City* as range. Figure 2.1 illustrates the graph representation of the above example.

RDF Syntax

In order to enable ontology modeling using RDF, various serialization formats are proposed. In the following, we list some of the most well-known formats:

RDF/XML[1] is a W3C recommendation used to represent the RDF data model. It is based on the XML syntax, imposing hierarchical structure for representation of resources. The root node of an RDF/XML document is and *rdf:RDF*, which is followed by a number of *namespaces*. Multiple nested elements can be encoded within the *rdf:RDF* node. Triples encoded in XML constructs are grouped according to their subject. The XML element, *rdf:Description*, is used to describe subjects and objects of the RDF triples. The unique identifier of a resource is handled by an *rdf:about* attribute, whereas the literal values are encoded in separated tags. Predicates can either be defined via XML attributes or as separate resources. Listing 2.1 shows the RDF/XML shortened representation of the Figure 2.1.

The main advantage of this syntax is the ability to be parsed by many mature tools and libraries compatible with XML. On the other hand, the RDF/XML is not very human readable, making it not attractive for people who model ontologies using plain text editors, in particular when the size is increased. Moreover, due to number of XML encoded constructs, an RDF/XML document tends to grow very quickly, increasing memory and processing requirements.

[1] http://www.w3.org/TR/rdf-syntax-grammar/. Date Accessed: 28 March 2018.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:sda="http://sda.uni-bonn.de/onto/staff#">

  <rdf:Description rdf:about="http://sda.uni-bonn.de/onto/staff#Bonn">
    <rdf:type rdf:resource="http://sda.uni-bonn.de/onto/staff#City"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://sda.uni-bonn.de/onto/staff#
   UniversityOfBonn">
    <rdf:type rdf:resource="http://sda.uni-bonn.de/onto/staff#University"/>
    <sda:locatedIn rdf:resource="http://sda.uni-bonn.de/onto/staff#Bonn"/>
  </rdf:Description>

  <sda:Student rdf:about="http://sda.uni-bonn.de/onto/staff#Lavdim">
    <sda:studyAt rdf:resource="http://sda.uni-bonn.de/onto/staff#
    UniversityOfBonn"/>
  </sda:Student>
</rdf:RDF>
```

Listing 2.1: **RDF/XML Syntax**. RDF/XML serialization of the example in Figure 2.1.

Turtle[2], which stands for *Terse RDF Triple Language*, aims at improving human readability of RDF documents. Using *namespaces*, it is possible to abbreviate the URIs of the internally defined concepts or externally used ones. The namespaces are normally written in the beginning of a Turtle document, in a form of a header. In order to enable their inner usage within the document, a unique prefix is associated for each namespace, respectively. The position and order of triples within the document is not important. Triples are separated with each other using dots, whereas spaces and line-breaks apart from the URIs, are irrelevant and ignored by parsers. Additionally, the so-called *syntactic sugar* allows triples to be grouped according to the same *subject*. In this case, the *subject* is written only once whereas its *predicates* and *objects* are terminated by a semicolon. Furthermore, *objects* sharing the same *subject* and *predicate*, can be grouped together and separated between each other using a comma. As a result, the Turtle format in addition to its human readability, is even more space efficient compared to RDF/XML. Listing 2.2 shows the serialization of the example depicted in Figure 2.1 in the Turtle syntax.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sda: <http://sda.uni-bonn.de/onto/staff#> .

sda:Bonn rdf:type sda:City.

sda:UniversityOfBonn rdf:type sda:University;
        sda:locatedIn sda:Bonn;
        rdfs:label "University of Bonn" .

sda:Lavdim rdf:type sda:Student;
        sda:studyAt sda:UniversityOfBonn;
        rdfs:label "Lavdim" .
```

Listing 2.2: **Turtle Syntax**. Representation of the example in Figure 2.1 using the Turtle serialization.

[2] https://www.w3.org/TR/turtle/. Date Accessed: 30 March 2018.

JSON-LD[3] is designed with the primary objective to enable the serialization of *Linked Data* into JSON format. Thus, RDF information encoded as Linked Data can be processed by any JSON parser and library. Using JSON-LD, applications are able to consume Linked Data and navigate across data sources on the Web, simply by following the embedded links. Moreover, JSON-LD is intended to be easily converted in Turtle, or queried with the SPARQL language, respectively. Important elements that compose a JSON-LD document are: 1) *JSON object* - a compact structure representing an object surrounded by curly brackets with zero or more key-value pairs; 2) *array* - a structure to represent zero or more values surrounded by square brackets; and 3) *string*, *number*, *boolean* and *null* - to allow encoding of values into respective formats. JSON-LD introduces additional features to those inherited from JSON for a unique identification of *JSON objects* via *URI*s, annotation of strings into different languages, ability to express more than one directed graph in a single document, etc. Listing 2.3 represents an excerpt into JSON-LD format of the example graph in Figure 2.1.

```
# prefix declaration

[
   {
      "@id":"http://sda.uni-bonn.de/ontologies/staff#Bonn",
      "@type":[
         "http://sda.uni-bonn.de/ontologies/staff#City"
      ]
   },
   {
      "@id":"http://sda.uni-bonn.de/ontologies/staff#Lavdim",
      "@type":[
         "http://sda.uni-bonn.de/ontologies/staff#Student"
      ],
      "http://sda.uni-bonn.de/ontologies/staff#studyAt":[
         {
            "@id":"http://sda.uni-bonn.de/ontologies/staff#UniversityOfBonn"
         }
      ]
   },
   {
      "@id":"http://sda.uni-bonn.de/ontologies/staff#UniversityOfBonn",
      "@type":[
         "http://sda.uni-bonn.de/ontologies/staff#University"
      ],
      "http://sda.uni-bonn.de/ontologies/staff#locatedIn":[
         {
            "@id":"http://sda.uni-bonn.de/ontologies/staff#Bonn"
         }
      ]
   }
]
```

Listing 2.3: **JSON-LD Syntax**. Representation of the graph in Figure 2.1 into JSON-LD format.

There exist a number of other well-known serialization syntaxes used in different domains or to address specific requirements. RDFa[4] is a W3C recommendation that allows for the addition

[3] http://www.w3.org/TR/json-ld/. Date Accessed: 30 March 2018.
[4] http://www.w3.org/TR/rdfa-syntax/. Date Accessed: 02 April 2018.

of rich metadata with the purpose of providing more information about the resources. As a result, the expressions *subject-predicate-object* can be embedded within the structure of XHTML documents and easily extracted by application agents. Trig[5], is another important syntax, which consists directives for grouping a set of triples into named graphs within an RDF document. For each named graph, an URI is associated, allowing for the identification and referencing from inside or outside of the RDF document. Trig is an extension of the Turtle format, inheriting its features for formatting triples as well as improving the human-readability. Furthermore, it facilitates the encoding of metadata related to the provenance and maintenance.

2.1.2 Expressiveness of Ontologies

The RDF model comprises only minimum number of modeling constructs to allow for the definition of the triples and providing unique identifiers. In order to enable encapsulation of the meaning of the resources as well as a detailed description of their relations, various schemes should be used. As a result, this description is enriched with a formal specification of the resources and their relationships, facilitating the intelligent agents to comprehend this information. In the following, the two most important schemas and their features are described.

RDF Schema

The RDF Schema (RDFS) [14] is created on the top of the RDF model, by extending it with additional modeling constructs, such as `rdfs:Class`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range`. The `rdfs:Class` allows for the specifications of a particular resource as a *Class*. Establishing a hierarchical structure between classes is realized using the `rdfs:subClassOf`. Relationships between classes are defined by properties, i.e., `rdf:Property`. Furthermore, properties use `rdfs:domain` and `rdfs:range` to allow a more detailed specification of these relationships. The domain of a property specifies its *subject* type within a triple whereas by range, the type of an *object* is defined. By doing so, the potential values for a relationship represented by a particular property are restricted to specific types, as defined in the domain and range of that property. Using the `rdfs:subPropertyOf`, properties can be organized in a hierarchical fashion. The modeling constructs, such as `rdfs:comment` and `rdfs:label` are used as annotation properties to enrich the human-readability of the given concepts. The RDF Schema serves as foundation to build classification of the typed hierarchies according to their relationships defined by `rdfs:subClassOf` and `rdfs:subPropertyOf`, as well as the restrictions specified using `rdfs:domain`, and `rdfs:range` [15].

The Web Ontology Language

The Web Ontology Language (OWL) [16] is a standard recommended by W3C, which has a rich set of constructs for modeling of complex ontologies. Thus, OWL is more expressive compared to RDFS, providing a comprehensive coverage with variety of axioms to represent the knowledge of an intended domain. Apart from taxonomic relations, and other simpler relations that can be modeled using RDFS, OWL allows for the specifications of restrictions or cardinalities for objects and literal values. Rigidity is an important feature of OWL, enabling ontology engineers to specify not only how particular axioms can be used but also how cannot be used. OWL offers special axioms to support modeling of the ontology metadata related to provenance, versioning,

[5] http://www.w3.org/TR/trig/. Date Accessed: 15 April 2018.

and maintenance. Apart from allowing the expression of the knowledge in a formal language, through reasoning process, new implicit facts can be inferred from the already explicit modeled ones. Some of the most relevant axioms of OWL are listed as follows:

1. define expected type of properties, i.e., object value, using `owl:ObjectProperty` for resource as value and `owl:DatatypeProperty` for literal as value;
2. distinguish and avoid overlapping between two different classes with `owl:disjointWith`;
3. use logic operators, such as AND, OR and NOT with `owl:intersectionOf`, `owl:unionOf` or `owl:complementOf`;
4. define flat relationships, such as synonyms in the concept level via `owl:equivalentClass` or in the instance level via `owl:sameAs`;
5. restrict type and number of individuals to be used via `owl:allValuesFrom`, `owl:someValuesFrom` or `owl:cardinality`;
6. specify complex relationships using `owl:SymmetricProperty`, `owl:TransitiveProperty` or `owl:InverseFunctionalProperty`.

OWL Profiles To adjust the level of expressivity for various application scenarios, OWL 2 [17] provides the three following *profiles*: OWL EL, OWL SQL, and OWL RL. With the objective of facilitating the reasoning, these profiles restrict the number of modeling axioms that can be used for each of them, respectively.

OWL EL - is suitable for scenarios where ontologies with large number of classes and/or properties need to be modeled. Algorithms dedicated for this profile are highly scalable and can perform reasoning in a polynomial time.

OWL QL - is tailored for scenarios where a large amount of instance data need to be handled and querying them is the most important task. To ensure a polynomial reasoning time, its level of expressivity, is rather limited, although the main modeling features are included.

OWL RL - is designed to balance requirements for scalable reasoning and expressive power. Checking for consistency, satisfiability and subsumption for class expression as well as answering of conjunctive queries is realized in a polynomial time w.r.t. the ontology size.

Therefore, while the ontology is being modeled, a trade-off between the appropriate profile and the reasoning performance should always be considered. A more expressive ontology can lead to the non computational time during reasoning process.

2.1.3 The SPARQL Protocol and RDF Query Language

The SPARQL Protocol and RDF Query Language (SPARQL) is the standard language to query the information represented according to the RDF format [18]. Similar to the Structured Query Language (SQL), used to query and manipulate data in the relational databases, SPARQL supports graph databases with functionalities, such as aggregation, filtering, and nested queries. It enables simultaneous execution of the queries over multiple data sources using a *federated query* mechanism. Moreover, SPARQL allows for querying other types of data sources, like Relational Databases, XML or CSV, whenever they can virtually represented as RDF. With SPARQL, it is possible to explore the RDF graph through traversing the relations between concepts. This is realized by matching *basic graph patterns*, defined in a query, where each

element of the given triples can be a variable. Next, these variables are replaced with concrete values in case the basic graph pattern matches with the resulting triples from the RDF graph.

Definition 2.2: Triple Pattern [19]

Let U, B, L be disjoint infinite sets of URIs, blank nodes, and literals, respectively. Let V be a set of variables such that $V \cap (U \cup B \cup L) = \theta$. A triple pattern tp is member of the set $(U \cup V) \times (U \cup V) \times (U \cup L \cup V)$. Let $tp_1, tp_2, \ldots, tp_n$ be triple patterns. A Basic Graph Pattern (BGP) B is the conjunction of triple patterns, i.e., $B = tp_1 AND tp_2 AND \ldots AND tp_n$

The SPARQL query language provides four forms of queries: 1) `ASK`; 2) `SELECT`; 3) `CONSTRUCT`; and 4) `DESCRIBE`. The `ASK` form returns a boolean value either *true* or *false*, depending on whether the given pattern matches any triple in the RDF graph. `SELECT` retrieves all triples that match the pattern (cf. Definition 2.2) defined in the query, considering clauses for constraints and filters. The result is formatted according to the aggregation, or ordering functions including the limit and pagination. A new RDF graph can be created via the `CONSTRUCT` form, corresponding to the query template. The `DESCRIBE` form shows all information contained in an RDF graph about a particular resource which is specified in the query.

Listing 2.4 depicts a simple `SELECT` query used to retrieve *the information about the name of the university and its actual number of students where the resource named 'Lavdim' is studying.*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sda: <http://sda.uni-bonn.de/onto/staff#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?university ?numberOfStudents WHERE {
    ?university a sda:University;
                  sda:isLocatedIn ?city;
                  dbo:numberOfStudents ?numberOfStudents.

      ?student sda:studyAt ?university;
               rdfs:label ?studentName.
   FILTER(?studentName = 'Lavdim')
}
```

Listing 2.4: **A SPARQL query**. A query to retrieve the name of the university and its number of students where the resource named 'Lavdim' is studying.

2.1.4 The Semantic Web

The idea of extending the Web of Documents towards Semantic Web where things are interconnected and able to exchange information with each other, was introduced by Tim Berners-Lee et al. [20]. The Semantic Web vision can be achieved by structuring the existing information or those being generated using ontologies as one of the important means. As a result, the current unstructured and heterogeneous Web of Documents which is easily understandable and explored by humans, is transfered into structured and integrated global information source. As described by authors, different agents are able to understand the meaning of data and the context on which they are going to be used. *Intelligent* agents collaborate with each other in an autonomous

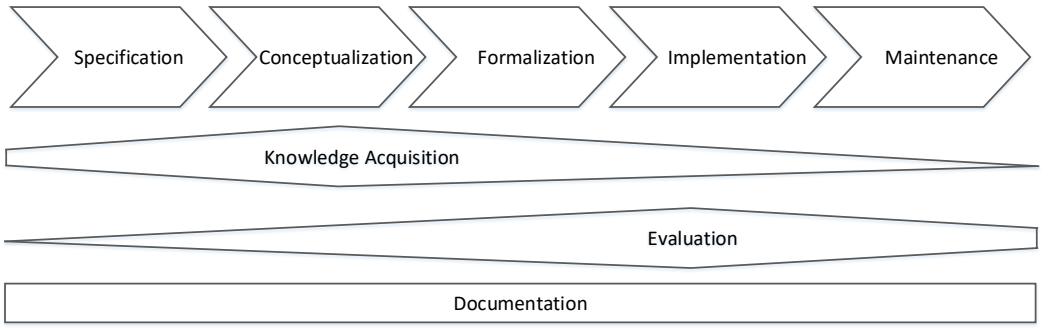


Figure 2.2: **Activities during ontology development**. Core activities performed typically in sequence: specification, conceptualization, formalization, implementation and maintenance. In addition, three activities are performed in parallel: knowledge acquisition, evaluation and documentation [21].

fashion to conduct a particular task at a specific time. As a result, we have a Global Information Source, where heterogeneous data coming from different sources are integrated and ready to be used. However, in order to enable this integration, a flexible and simple, yet powerful enough formal language should be used for modeling purposes. For this reason, the Resource Description Framework (RDF) created by World Wide Consortium (W3C)[6], as a generic framework for representing information on the Web is utilized to develop ontologies.

Ontologies are used for structuring and sharing the knowledge among different organizations, which might be dispersed across the world. They are developed by many people with various interests that contribute on conceptualization of a particular domain and defining the concepts, attributes and relationships. Opinions and interests from stakeholders should be integrated, in order to avoid disagreements and inadequate contribution.

2.2 Ontology Development

Ontological engineering refers to "the set of activities that concern the ontology development process, the ontology life-cycle, principles, methods and methodologies for building ontologies, as well as the tool suites and languages that support them" [10]. During this process, a number of people with different roles and expertises are involved, where two of them are the most important ones: 1) *Ontology Engineers* (OEs); and 2) *Domain Experts* (DEs). Ontology Engineers are skilled people with technical expertise in formal languages and tools, necessary to model a domain of interest. Domain Experts are professional persons who have the expertise on the given domain and its requirements. Figure 2.2 illustrates five core activities and three additional ones performed in parallel during entire ontology development life-cycle [21]:

Specification During this activity, the objective and the scope of an ontology is specified. Usually questions, such as "What is the purpose of the ontology being developed?", "Who are the stakeholders that will be involved in development and maintenance?", "Who will use the ontology?", are clarified. The intention here is to identify the goals and define the granularity of the domain knowledge to be modeled while limiting the complexity. Commonly, a list of *competency questions* derived from the requirements are

[6] https://www.w3.org/TR/rdf-concepts/

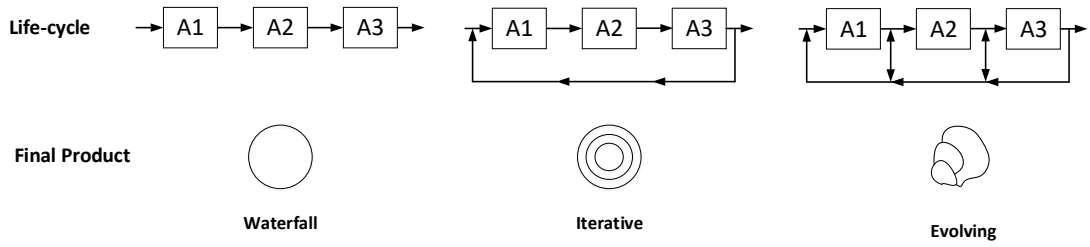


Figure 2.3: **Life-cycles models**. Comparison of the sequence of activities performed in different models, such as: a) Waterfall; b) Iterative; and c) Evolving [22].

used to determine the ontology scope. Later on, these questions are used to evaluate the completeness and avoid any negative impact of modifications to the core concepts.

Conceptualization A number of concepts are derived as an outcome of the previous activity. As a result, a conceptual model is built to represent the concepts and their relationships in a language independent fashion. A number of clusters may be created, since several concepts are strongly connected with each other. These clusters can be aligned as separate modules where the ontology can further be decomposed.

Formalization Next, the conceptual model is converted into a formal model using necessary axioms to reduce the potential interpretations of the meaning for the defined concepts. Usually, a hierarchical structure is created by specifying the relationships between concepts, such as "sub-class-of" or "part-of".

Implementation In this step, the formal model is implemented based on a particular language with a set of strictly defined axioms. As a result, the developed ontology is machine processable and a reasoner can be employed to infer new facts.

Maintenance Commonly, ontology evolves over the time and is subject of changes even after the implementation process is finished. These modifications are performed to reflect the new requirements that may come from the stakeholders or to repair possible bugs.

In addition to the above mentioned activities, there are some activities that are orthogonally performed during the entire ontology life-cycle:

Knowledge acquisition As the development process continues, ontology engineers and domain experts will gather new knowledge about the domain. As a result, the ontology is refined according to the acquired knowledge.

Evaluation Ensures that a certain quality is met by continuously validating the ontology. Moreover, the completeness with respect to competency questions can be assessed and any not intended change to the core concepts and relationships should be avoided.

Documentation Track and monitor the tasks that have been realized and the way of realization as well as design decisions that have been taken. In addition to improve the clarification and reusability of the defined concepts, it facilitates the maintenance of the ontology itself.

As depicted in Figure 2.3, the above mentioned activities can be performed following various life-cycle models, such as *waterfall*, *iterative* and *evolving*. [22] states that the majority of methodologies for ontology development are based on an *evolving prototyping* model. This allows for the switching back and forth to any activity with the objective to improve the prototype, until the defined requirements and the evaluation criteria are fulfilled.

2.2.1 Collaborative Ontology Development

A *collaborative ontology development* process involves many stakeholders working together to model a domain of interest. This process should be supported by specialized methods and tools to accommodate stakeholders with different skills, experience and responsibilities, as well as potentially divergent agendas [5]. Typically, activities starting from *specification*, *conceptualization*, *formalization*, *implementation* and *maintenance* are performed during a collaborative development as well. Stakeholders need to jointly define the domain requirements while proposing and negotiating various use cases. Next, a number of constraints and modeling decisions, such as the level of granularity, naming conventions, and reuse of existing ontologies, are defined. To avoid misunderstandings, stakeholders may define a common glossary including technical terms. They further investigate current tools and frameworks for collaborative development and decide upon which one to use, considering current team composition and their skills. Another important aspect to be carefully analyzed, is the definition of the quality metrics, which should be performed in parallel to each activity during the entire development process. In these scenarios where many people are involved, the roles for each of them have to be clearly defined and the mechanisms to monitor their contributions should be specified.

2.2.2 Test-driven Development

Test-driven Software Development (TDD) is a programming approach where a set of test cases is defined before the code is actually written [23]. Three are the main steps based on the TDD principles: 1) add a test; 2) write the code; and 3) refactor the code. First, test cases which represent the requirements to cover different aspects of the software being developed, should be defined. Next, these test cases have to be executed, which initially will fail. The developer should write only the necessary code to pass the test. The execution process has to be repeated until there is no test case that fails. The refactoring step ensures that any written code meets certain quality criteria, such as: a) purpose and the use of each module, class and variable is clearly defined; b) avoid duplicates; and c) split large modules, classes or methods into smaller ones to support maintainability and readability. On the other hand, the development of domain-specific ontologies requires joint efforts among different groups of stakeholders, such as ontology engineers and domain experts. Following the principles of TDD, the ontology development process is ensured that any ontology modification has only the expected effects, by continuously running a set of test cases. This set of test cases can be defined based on the *Competency Questions* and the constraints derived from the domain requirements. Figure 2.4 illustrates SAMOD, a *simplified agile methodology for ontology development* [24]. It is based on iteration over a *modelet*, which is an isolated model used to conceptualize a specific aspect of the domain with respect to the motivation scenario. SAMOD follows the TDD principles, consisting of three main steps: 1) ontology engineers together with domain experts collect domain requirements, build a *modelet* and create a test case; 2) current model is merged with the modelet of the new test case from last iteration by ontology engineers; and 3) ontology engineers perform refactoring based on the outcome from the previous step [24].

2.3 Version Control Systems

Version Control Systems (VCSs) are software systems used to manage changes made to a repository over the time. Typically, a repository comprises a file or set of files organized into

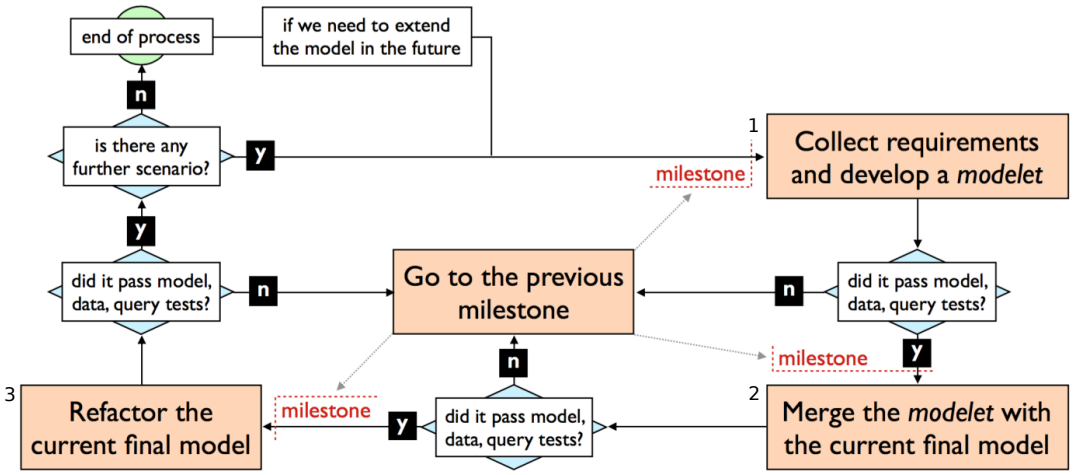


Figure 2.4: **SAMOD - A Simplified Agile Methodology for Ontology Development**. SAMOD follows the TDD principles: 1) collecting the domain requirements; 2) merging models from two different iterations; and 3) refactoring based on the outcome of the previous step [24].

directories. Any performed change is tracked, enabling the VCS to create regular snapshots for the repository. The snapshots represent different versions of the files and allow to reverse back to a previous state of a file, in case of identified mistakes. Developers of a team can work collaboratively while simultaneously performing changes to the same project. If properly used, the version control prevents from a potential loss of the work realized over the time, as well as it avoids the gradual worsening of the work or any unindented action. A hosting platform is chosen to maintain and propagate changes performed to a repository by different developers.

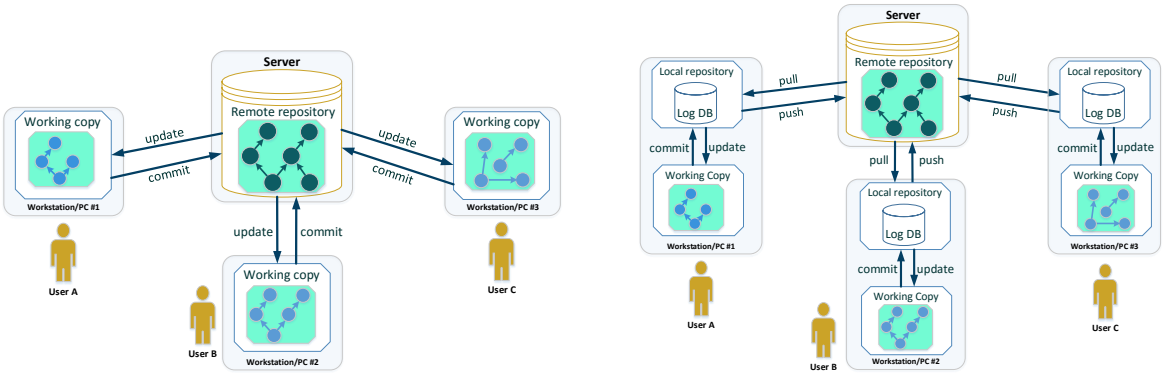
There are two major types of version control: 1) Centralized Version Control Systems; and 2) Distributed Version Control Systems [25].

2.3.1 Centralized Version Control Systems

The main concept in centralized version control is a *central* repository. It primarily serves as a shared space to store the entire history of changes and enables developers to access it following the principles of a client-server architecture.

A new *version* of a repository is created after each change or set of changes. The new *version* contains metadata to describe more details about it, and an identification number, which is automatically incremented or uniquely generated according to a special pattern. Users are able to retrieve the latest or a specific version of the repository using the *checkout* command while providing the version id. Thus, they can work on the personal *working copy*, which does not include the entire history of changes made on it. Further, users share their local modifications with other team members via the *commit* operation. The *update* operation is used to receive the latest changes realized by other users. Changes between different versions of the repository are calculated as *deltas* using various *diff* algorithms. The parallel development is supported by various *branches*, acting as independent lines of the main repository, where users can fix related bugs, implement new features or release the mature versions.

The central repository acts as a *single point of failure*, where in case of any mistake, the entire work and the history of changes can be lost. Another drawback of centralized version control systems, is the need to be always connected for performing specific operations, such as *commit*.



(a) Centralized Version Control Systems (b) Distributed Version Control Systems

Figure 2.5: **Version Control Systems: Centralized vs Distributed**. Collaboration workflows: a) several users work together on the same central repository; and b) several users work on their local repositories and synchronize the changes with the others via a shared repository.

2.3.2 Distributed Version Control Systems

To address some limitations of the centralized version controls, a new generation of VCSs, the *Distributed Version Control Systems* has emerged. Instead of having only one central repository, users always retrieve the entire repository, including the complete history of changes. This allows to perform many operations, such as *commit*, even without being connected to a network.

As illustrated in the Figure 2.5(b), working offline is enabled using the *clone* command, where a *local repository* is mirrored in the user machine from the *remote repository*. User continues to edit files in the *working copy*, until it reaches a state where changes should be *committed*. During the *commit* process, a *message* is provided to describe the reason for the changes that have been realized. A new version of the repository is created comprising additional metadata for the user who performed the modifications and the timestamp information.

Synchronization

A VCS assists users to work collaboratively on shared artifacts, and helps them to prevent from overwriting changes made by each other. Basically, mechanisms to avoid change overwriting can be classified according to *pessimistic* and *optimistic* approaches [26]. The first ones are based on the *lock-modify-unlock* paradigm, which implies that modifications to an artifact are permitted only for one user at a time. The latter ones are based on the *copy-modify-merge* paradigm, where users work on personal copies, each reflecting the remote repository at a certain time. After the work is completed, the local changes are merged into the remote repository by an *update* command, comprising the phases: *comparison*, *conflict detection*, *conflict resolution*, and *merge*.

Users can perform many changes and commit them to the repository. Since the entire history of changes is logged, it is possible to trace it through unique identifiers and navigate to a specific version. However, in case that a *User A* wants to share her changes with other team members, she should publish them to the shared remote repository. In order to realize that, the *push* command is used. After the latest version is pushed to the remote repository, other users can retrieve it via the *pull* command.

Conflicts Different techniques, such as line-, tree-, and graph-based ones, can be employed to compare two versions of the same artifact [27]. The line-based technique, which achieved a wide applicability, compares artifacts line by line, with each line being treated as a single unit. This technique is also known as *textual* or *line-based comparison* [26]. Examples of VCSs that use the line-based approach are Subversion, CVS, Mercurial, and Git. Line-based comparisons are applicable on any kind of text artifact, as they do not consider syntactical information [27]. Accordingly, line-based approaches also neglect syntactical information of ontologies, which are commonly represented in some text-based OWL serialization.

Challenges arise when two ontology developers simultaneously modify the same artifact on their personal working copies. Changes might contradict each other, for instance, developers may both edit the name of an ontology concept. Such parallel and controversial modifications can result in conflicts during the merging operation of two ontology versions. In general, a conflict is defined as "a set of contradicting changes where at least one operation applied by the first developer does not commute with at least one operation applied by the second developer" [27]. Conflicts can be detected by identifying changed units (i.e., added, updated, deleted) in parallel. Conflict resolution can be done automatically or may require users to manually fix them by resolving the conflicting changes.

False-positive conflicts A distributed VCS follow the *optimistic* approach to enable the *concurrent* modification of the ontology artifacts, as well as conflict detection and resolution. From the ontology development point of view, the situation is exacerbated when different ontology authoring editors are used during the development process. This is due to the fact that these editors often produce different serializations of the same ontology, i.e., the ontology concepts are grouped and sorted differently in the files generated by the editors.[7] The ability of the VCS to detect the actual changes in ontologies is lowered. As a consequence, the VCS may detect a large number of *false-positive* conflicts, i.e., conflicts that do not result from ontology changes but from the fact that two ontology versions are differently serialized.

Hooks – An Event Triggered Mechanism

Hooks are special scripts triggered after occurrence of particular events, such as during commit or merge operations. They enable customization of actions and creating specific workflows to achieve a desired result or meet certain requirements. Thus, special mechanisms to automate tasks for development and deployment can be designed. According to the scope of the action coverage, Git hooks are divided in two groups: 1) *client-side*; and 2) *server-side* hooks.

Client-side hooks are invoked prior to occurrence of specific actions in the *local repository*. They are split into three categories: *committing-workflow hooks*, *email-workflow scripts*, and everything else.[8] Among the most important hook within the committing-workflow category is the *pre-commit* hook. It is invoked each time when a user runs the *git commit* command along with providing a commit message. This hook can be utilized for inspecting various aspects of the changes recently performed over the code. Additional test coverage techniques can be employed to ensure that changes do not break rules or constraints that are defined in advance.

[7] With "different serializations", we refer to two different ontology files that represent the same ontology using the same syntax (e.g., RDF/XML, Turtle, and Manchester) but use a different structure to list and group the ontology concepts.

[8] https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks. Date Accessed: 17 April 2018.

Server-side hooks reside in *remote hosting platforms*, i.e., GitHub, Bitbucket or Gitlab. Typically, these hooks are managed by administrators, who are responsible for maintaining the repository, monitoring activities and managing the user permissions. The most important hooks in this group are: *pre-receive*, *update*, and *post-receive*. They are invoked at different phases after running the *git push* command. *Pre-receive* and *update* are executed before the conducted changes are permanently written into repository. It is possible to prevent users from pushing changes if the commit or format of the commit message do not meet a certain criteria. *Post-receive* takes place after the push operation is successfully completed. Tasks, such as emailing the team involved into the development process or acting as a continuous integration service, are typical scenarios for this hook.

Branching and Merging

Branching and merging are fundamental features of a VCS, which allow parallel development within the same repository. Developers quite often want to test a new feature or to fix a bug in an isolated copy, without interfering the primary development line. The main reason is related with avoiding any negative impact that a new feature or bug fix might have to a mature version of the project. Typically, a repository comprises the following branches: *release*, often named as *master*, *development*, *testing* or branches associated with feature or issue name, respectively. There are two major *branching styles* that can be followed during the development process [28]. *Early Branching* is suitable for larger projects with fine-grained requirements for control and safety, but posing additional overhead for propagation of the changes. *Deferred Branching* fits better to scenarios that can tolerate a small portion of safety risks to gain a higher productivity.

On the other hand, merging operation should be carefully performed, since a wrongly conducted merge may cause the loss of changes or introduce new issues. An important aspect to be considered during multi-branch development, is doing regular synchronizations to avoid potential divergences between them. One way to tackle this, is using continuous integration services for allowing the harmonization of changes from different branches. However, this task should be regularly monitored and in many cases requires human intervention to reduce number of the possible issues that might arise. In the line with branching style, a team should also consider the *merging style* suitable for its use case [28]. *Restricted Merging Style* implies stricter policies and constraints for increasing the safety and avoiding any risk that may lead to non-intended consequences. *Relaxed Merging Style* sacrifices the safety for having less policies and constraints.

CHAPTER 3

Related Work

In this chapter, we discuss the related work to our research, according to the defined research questions. We initially describe the general classification of methodologies and their characteristics. We then dive more in deep in methodologies which are focused on the collaborative aspects of ontology development. Here, we elaborate the main aspects of these methodologies and how they are used in practice. We start the next section with a brief presentation of standalone tools used for creating ontologies. We then continue on revising the platforms dedicated for collaborative ontology development. Moreover, we look on how concurrently modified ontologies on version control systems are synchronized by preventing from the overlapping conflicts. In the end, we discuss the approaches provided in the literature to support test-driven development of the ontologies with a focus on efficient execution of a given set of test cases.

3.1 Methodologies for Collaborative Ontology Development

Ontology development is an active research area for many years in the Semantic Web community [29]. Several methodologies are proposed with the objective of facilitating the development process in various scenarios. Nowadays, teams involved in ontology creation are composed of many people using different communication channels to assist their collaboration and organize their assigned endeavors. Considering the collaborative aspect, methodologies are classified in two categories: 1) traditional methodologies, which target developing ontologies in organizations where a group of people work together in a closed environment; and 2) collaborative ontology development methodologies, with a strong focus on collaboration aspects, where a large number of people or a community is involved [5]. Another classification can be made according to the fact whether a methodology adhere to a specific workflow. In this regard, first group comprises methodologies that describe ontology development as set of activities that should be performed following a particular order. Second group comprises methodologies typically agile oriented and do not require to follow a strict workflow. These methodologies propose set of guidelines and practices covering a wide range of aspects of the development process.

3.1.1 Workflow-dependent Methodologies

In the following, some of the representative workflow-dependent methodologies for supporting collaboration during ontology development, are described.

Ontology life-cycle phases	Goals	Tasks
Specification	Define aim / scope/ requirements/ teams	▪ discuss requirements (S) ▪ produce documents (S) ▪ identify collaborators (S) ▪ specify the scope, aim of the ontology (S)
Conceptualisation	Acquire knowledge	▪ import from ontology libraries (P) ▪ consult generic top ontology (P) ▪ consult domain experts by discussion (S)
	Develop & Maintain Ontology	▪ improvise (P) ▪ manage conceptualisations (P) ▪ merge versions (P) ▪ compare own versions (P) ▪ generalize/specialize versions (P) ▪ add documentation (P)
Exploitation	Use ontology	▪ browse ontology (P) ▪ exploit in applications
	Evaluate ontology	▪ initiate arguments and criticism (S) ▪ compare others' versions (S) ▪ browse/exploit agreed ontologies (S) ▪ manage the recorded discussions upon an ontology (S) ▪ propose new ontology versions by incorporating suggested changes (S)

Figure 3.1: **The HCOME methodology**. Overview of the main phases of the HCOME methodology, their goals and respective tasks. (S) - denotes shared spaces, whereas (P) - denotes private spaces [30].

HCOME

The Human-Centered Ontology Engineering Methodology (HCOME) presented by Kotis et al. [30], is a methodology which has the main focus on creation of ontologies in distributed settings. The two major distinguished roles are: *knowledge workers*, who are experts from the field and *knowledge engineers*, responsible for modeling ontologies based on a particular formal language. HCOME suggests different spaces where team members can work and save the ontologies. The *Personal Space* is used by members to create and modify ontologies according to their personal opinions, as well as to manage various versions. Once the developed ontology evolves to a mature version, it can be uploaded into the *Shared Space*, where the other members can access and explore it. The *Shared Space* allows for the comparison of various versions of the same ontology and supports the discussion of modeling decisions. Authors identify four important issues that should be addressed by the methodology: 1) the work of the community members should be approach independent, allowing them to integrate different concepts, including those with informal definitions, reuse and refine existing ontologies; 2) lower the contributions barriers for knowledge workers by hiding implementation details and enable conceptualization in a way that is convenient for them; 3) workers should be able to communicate and shape the information synergistically by posting issues, proposing changes as well as avoiding deadlock situations; and 4) mapping the definition of the concepts to other external ontologies to ensure they reflect the intended meaning for the respective domain. Figure 3.1 depicts three phases of HCOME, where each of them have their goals and tasks associated with, respectively.

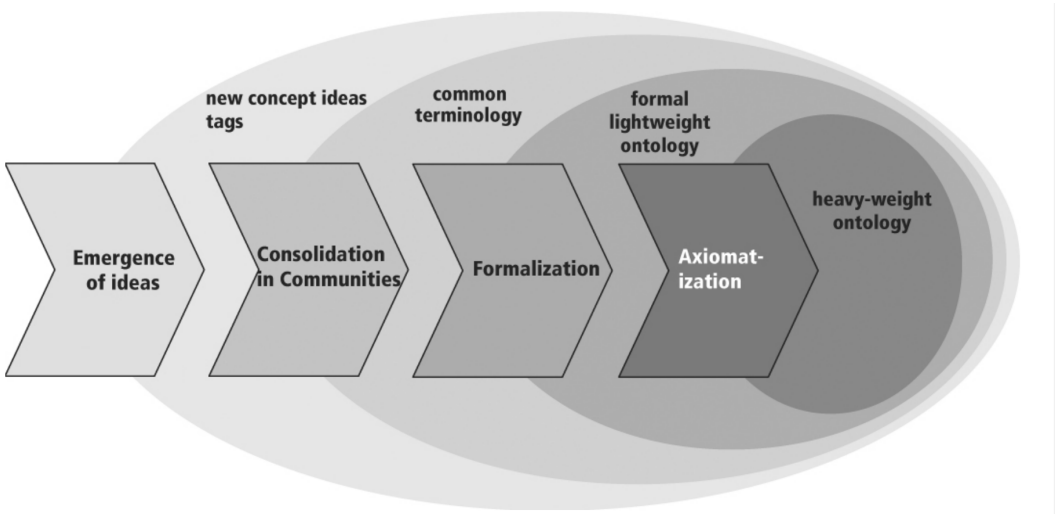


Figure 3.2: **The Ontology Maturing Process**. The four phases of the ontology maturing process: 1) emergence of ideas; 2) consolidation in communities; 3) formalization; and 4) axiomatization [31].

Specification - during this phase the domain is specified, including the requirements to restrict the ontology scope and the team comprising knowledge workers and knowledge engineers is established. The outcomes of this phase are recorded in specification documents to be used later on during the entire development process.

Conceptualization - after the definition of the domain, requirements and the team, the process continues with the conceptualization of the ontology. The team members start to review and import existing ontologies for a potential reuse. To better understand the domain, generic ontologies are consulted and the informal definitions of their concepts are analyzed. Next, the ontology is being developed in parallel by various knowledge engineers, where multiple versions should be mapped and merged in order to achieve a unified view and allow the reusability.

Exploitation - in this phase, the ontology is subject of a critical review process, where the involved workers provide their feedback. The ontology is investigated by various team members to identify and critique any wrong or incomplete concepts. The feedback and their comments are stored and published to later support and revise the design decisions.

Ontology Maturing

Braun et al. [31] observe the ontology engineering as an informal learning process, in which the involved team members enhance their comprehension and competency about a particular domain. Furthermore, as the definition of concepts should derive from a shared understanding, the collaborative work between ontology engineers and domain experts should be facilitated by reducing the level of formality and complexity. Therefore, the contribution of domain experts is increased whenever the methodology and tool support is lightweight, easy to use and understand. Another important aspect observed by the authors, is that ontology development is a continuous process where the ontology evolves over the time to reflect new ideas, changes on the real world and better understanding of the domain.

To accommodate the above observations, the authors defined the maturing model as illustrated in Figure 3.2, which is composed of the four following phases:

Emergence of ideas - is the initial phase, where the proposed ideas for concepts are rather

ad hoc and without a clear definition. Usually tags are used to annotate or look for particular resources which later can be replaced in case of previous incorrect definitions.

Consolidation in communities - a common terminology is created from the collaborative work of the engaged community. This is achieved by utilizing tags and refining them for the definition of the future concepts. Thus, the retrieval of resources is improved including additional synonyms that are used in the daily life.

Formalization - concepts are organized in a hierarchical fashion as well as their inter-relations are identified. As a result, a lightweight taxonomy is established, allowing users to navigate in a number of various broader and narrower levels.

Axiomatization - denotes the last phase of the maturing process, which transfer the outcome of the previous phases to a formal representation language. This enables capturing of the domain knowledge to improve inference processes, e.g., question answering systems. Knowledge engineers are highly involved in this phase since it requires the usage of complex axioms.

According to the authors, the proposed ontology maturing assumes that ontologies are not built from scratch but rather they evolve from set of informal tags to well defined taxonomies. Furthermore, this is seen as an iterative process over the four defined phases. As potential applications are identified to be from the areas of semantic annotation and resource retrieval.

DILIGENT

DILIGENT presented by Pinto et al. in [32], is another methodology for supporting collaborative ontology development in distributed settings. It considers the following important issues that the development process should have: 1) be realized in a *decentralized* fashion, where a number of stakeholders are not necessarily settled in the same geographic location; 2) enable users to have a *partial autonomy*, where they personalize their own local copy to reflect their needs; and 3) be *iterative*, by providing clear steps on how to iterate between different phases. The authors identifies four roles which participate in the development process: ontology engineers, knowledge engineers, domain experts and users.

The DILIGENT process depicted in Figure 3.3 comprises the following five activities in order to support the collaborative creation of a shared ontology:

Build - based on the predefined requirements, participants build an initial version of the ontology, which is not needed to be complete. Even the team is recommended to be small with the aim to avoid long discussions and achieve a consensus more easily for the specified terms.

Local adaptation - users are eligible to perform modifications of their local ontology copies, in order to accommodate the new business requirements or rules. However, these modifications are not directly represented in the shared ontology, rather they are collected by a responsible board which assesses for a potential adoption in the future.

Analysis - the collected modifications from the locally modified ontologies are investigated by the responsible board to find potential similarities, which will further be elaborated before introducing them in the new version of the shared ontology. In addition, new requirements that might come from users are considered and a balanced decision should be taken.

Revision - the shared ontology is revised on regular basis to prevent from a possible deviation between that and its local copies. Thus, the board is represented in a well-balanced way by various kinds of stakeholders involved in the development process.

Local updates - users are able to decide whether to apply the approved modifications of the shared ontology to their local copy. By applying changes, they benefit from the interoperability point of view with the respect to the jointly defined concepts.

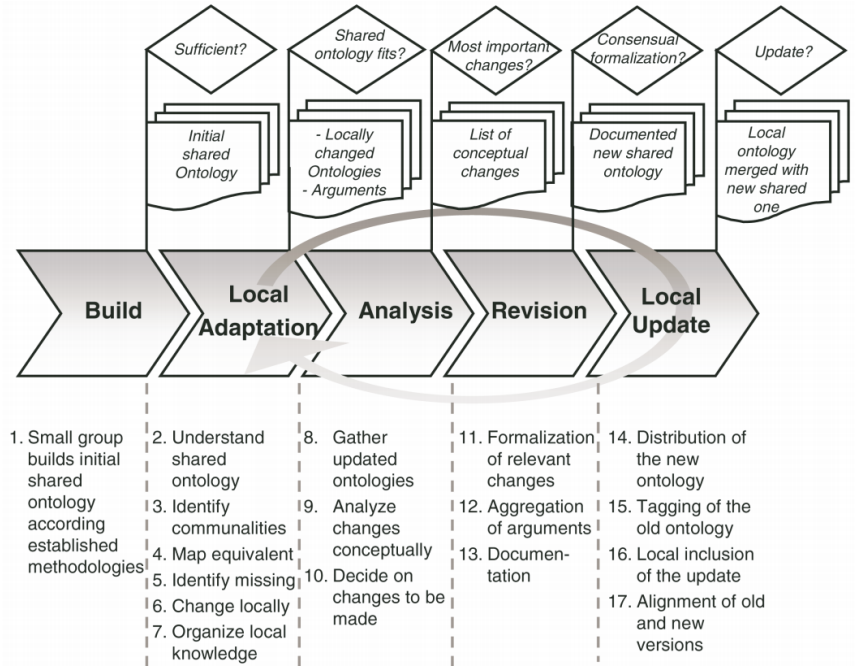


Figure 3.3: **The DILIGENT Life-cycle**. The different stages of the DILIGENT methodology: *Build*, *Local Adaption*, *Analysis*, *Revision*, *Local Update* [33].

3.1.2 Workflow-independent Methodologies

In the following, three well-known methodologies which do not enforce obeying to a particular workflow during the development process, are presented.

Guidelines for Constructing Reusable Domain Ontologies

Annamalai et al. [34] present *Constructing Reusable Domain Ontologies* (CRDO). It comprises guidelines to systematically construct domain ontologies based on a purpose-driven approach. A strategy to support developing reusable domain ontologies is outlined, starting first with identifying relevant domain knowledge to be encoded; next, consolidating the pieces of knowledge; finally, select the ones that can be reused and create isolated modules. The purpose of the authors is to facilitate the communication, integration and information sharing across the distributed environment of the scientific community, which performs experiments, gathers data and publishes various types of findings. Two types of ontologies are distinguished in this environment: *domain ontologies*, which capture more generic knowledge from a given domain and are loosely coupled with each other; and *purposive ontologies*, which encode specific domain knowledge by combining different reusable domain ontologies. Based on the fact whether existing ontologies will be used to develop domain and purposive ontologies, the proposed guidelines are divided in:

Ontology Development Guidelines I - groups the following steps: a) define the purpose and use of the ontology; b) identify the main concepts; c) define competency questions based on the given requirements; d) identify additional terms needed for competency questions and their answers; e) identify new concepts, attributes and their relations as well as structure them in a conceptual model; f) evaluate the conceptual model using competency questions; g) return to c), if more details should be included; h) create potential reusable submodules by clustering similar

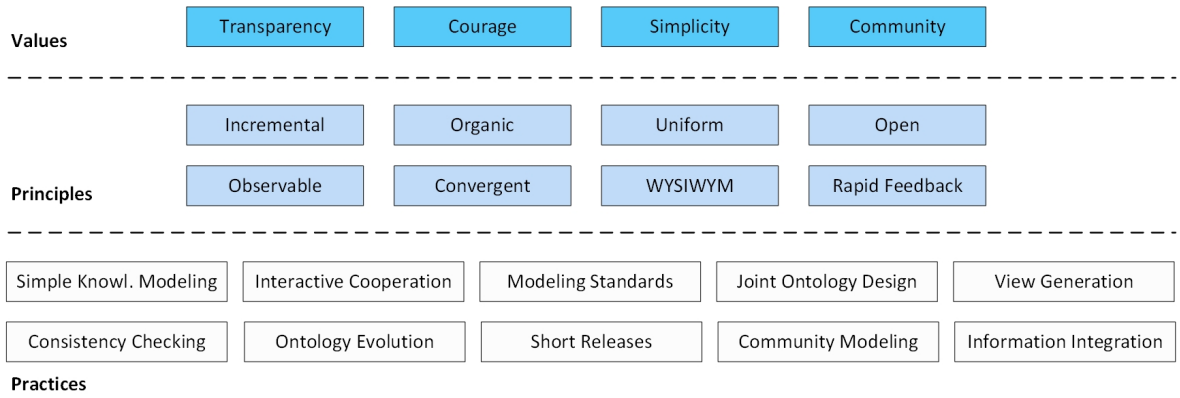


Figure 3.4: **Overview of the RapidOWL methodology**. It consists of three building blocks: *Values*; *Principles*; and *Practices*, each of them covering different aspects of ontology modeling [35].

concepts; i) link the submodules to the main model; and j) formalize reusable domain ontologies and purposive ontologies out of the main model and submodules.

Ontology Development Guidelines II - comprises the following steps for developing purposive and domain ontologies: a) define the ontology purpose; b) design the model of the purposive ontology; c) investigate for existing ontologies that can be reused; d) develop the unsupported part of the model according to the *Guidelines I steps b) - g)*; e) localize a possible region in the model that can be reused. If exists any, develop an independent and reusable submodule which contains the most relevant concepts. It is related to the *(Guidelines I, step h)*; f) reuse external ontologies (as identified in c), to accommodate specific application requirements; and g) recreate the model of the purposive ontology by inter-linking it with submodules, and proceed with the formalization of the designed ontologies.

RapidOWL

Inspired by eXtreme Programming (XP) of knowledge-based systems, Auer et al. [35] present RapidOWL, an adaptive and lightweight methodology for Collaborative Knowledge Engineering. Furthermore, RapidOWL adopts *Wiki Design Principles*[1], which are used for text editing in a collaborative fashion. The methodology does not provide any specific life-cycle, rather it aims supporting ontology development by performing rapid and small changes. A number of guidelines are proposed to facilitate the contribution of domain experts to the development process, by avoiding the necessity to rely on a considerable involvement of knowledge engineers. The building blocks of RapidOWL separated in three different categories are illustrated in Figure 3.4:

Values - RapidOWL brings the values from eXtreme Programming to represent the long-term goals. *Communication* and *Feedback* are combined into *Community*, to cover important aspects of communication and interaction across contributors including comments, critics, assessments and suggestions; *Simplicity*, to ease the maintenance of the ontology and data part of a knowledge base; *Courage*, to facilitate the progress and avoid possible modeling dead-ends. RapidOWL introduces *Transparency*, an additional new value, enabling the community to view the entire history of ontology changes as well as monitor and track contributions of the participants.

Principles - RapidOWL suggests various principles to guide the mid-term agile development of ontologies. These principles define characteristics that a single RapidOWL process should have.

[1] http://c2.com/cgi/wiki?WikiDesignPrinciples

Some of the relevant principles are: following a *uniform method* for modeling of the schema and data; *incremental* and *observable* development; and *rapid feedback* to facilitate the maintenance and accommodate diverse views of the community.

Practices - the third category comprises practices adopted from eXtreme Programming as well as from collaborative design of knowledge bases [36]. Among the most important ones are: *Joint Ontology Design*, to facilitate the work between domain experts, knowledge engineers and users; *Information Integration*, to ensure that the ontology captures adequate domain knowledge; *View Generation*, to provide various exploration views for a better understanding of the concepts hierarchy, their metadata and relationships across them.

Just Enough Ontology Engineering

Just Enough Ontology Engineering (JEOE) [37] is a set of aspects essential to ontology development based on principles of the ""just enough"" approaches. It indents to support experts of different fields, who may be part of the process with fundamental notions of ontology development, without forcing them to follow a specific methodology. JEOE comprises interdependent steps and activities to be adopted in an agile process and performed iteratively. Below are the steps that should be followed in JEOE:

1. Identify stakeholders - persons and their responsibilities during the development process and profile them according to common characteristics: goals, interests and requirements.

2. Define the purpose and goal - to have a clear understanding and expectations of the ontology to be developed as well as focusing the efforts to accomplish the identified requirements.

3. Requirements - definition serves as a mechanism for continuous verification of the ontology whether is able to solve the problem for which it is designed for.

4. Identify existing knowledge sources - and asses their qualitative and quantitative characteristics for a possible reuse and extend, in order to fill the missing gaps.

5. Scoping the ontology - by specifying the level of granularity and setting up the boundaries to help on reducing the complexity of modeling and avoid a never-ending process.

6. Quality assurance - using a “quality model” that is built on initial stages and comprises patterns to measure the qualitative and quantitative parameters of different aspects.

7. Competence - shows whether the developed ontology contains all necessary constructs to answer the given set of competency questions.

8. Define the ontology artifacts - including terminology and vocabulary, their meaning, the relationships between them, axioms and constraints to be used for consistency checking.

9. Implementation - by transferring the defined artifacts from the previous step into a formal knowledge representation language according to the goal, scope and the requirements.

10. Deployment - of the ontology or its modules as a compact and standalone artifact by specifying the access infrastructure and level of integration to be supported.

11. Testing and validation - of different aspects, such as syntax, consistency, integrity, redundancy of the ontology or its modules, as a part of a comprehensive quality assurance plan.

12. Publish - the ontology to provide access for the interested users by describing the mechanism and policies how to reach and use it, as well as specifying the license terms.

13. Maintenance and reuse - ensures that the ontology reflects the last state of the intended domain. This is realized by regularly updating and maintaining the things which are changed since the release of the last version.

Table 3.1: **Coverage of the ontology development aspects**. Comparison between methodologies with respect to coverage of the important aspects for ontology development.

	Reuse	Naming	Multilinguality	Documentation	Validation	Authoring
METHONTOLOGY[21]	Yes	No	Yes	Yes	Yes	No
CRDO [34]	Yes	No	No	No	No	No
DILIGENT [32]	Yes	Yes	No	No	No	No
On-To-Knowledge [40]	No	Yes	No	No	Yes	No
RapidOWL [35]	No	Yes	No	No	No	Yes
JEOE [37]	Yes	No	No	Yes	Yes	Yes
Linked Data Patterns [38]	Yes	No	Yes	Yes	No	Yes
NeOn Methodology [39]	Yes	No	Yes	No	Yes	No

Summary Generally, traditional and collaborative focused methodologies provide coverage to wide-range of aspects of the ontology development process. For instance, some of the above mentioned methodologies [21, 32, 35, 37–39] cover the main aspects for ontology development in a top-down approach. Various ontology life-cycle models comprising different phases are proposed and can be followed depending on the use case and team composition. In addition, a number of complementary guidelines and methods presented by lightweight oriented methodologies facilitate ontology creation. Despite this, we observe a lack of governance and best practices dedicated for distributed scenarios where the development process is centered around the version control. More specifically, guidelines to address ontology engineering from organizational point of view, such as: management of parallel development, version releases, communication and monitoring activities among team members and modularization are still missing in these scenarios. Furthermore, during the development process, providing a number of practices from the operational point of view, like: reuse, naming convention, multilinguality, documentation, and validation are crucial. Table 3.1 shows the coverage of the practical aspects from different methodologies.

A novel methodology for supporting the ontology development process in distributed and heterogeneous scenarios is needed. This methodology should facilitate a workflow-independent process in a dynamic environment, where the number of team members, development tools, and parallel ontology versions is constantly changing. One central characteristic is to actively support stakeholders when taking organizational and design decisions. Moreover, operational practices to facilitate ontology engineering process regarding the reusability, multilinguality, naming convention, validation and authoring for such scenarios are still missing.

3.2 Platforms for Collaborative Ontology Development

Over the years, much research has been conducted and several approaches has been presented to provide tool support for ontology development in different scenarios. One area of research is concerned with creation of web applications that lower the access barriers to the development process. Tools, such *SOBOLEO* [41] foster the collaborative editing of the SKOS thesauri with a specialized browser, to navigate and change the taxonomy as well as a semantic search engine for annotating web resources. SOBOLEO is used in the domain of social networks and offers tag recommendations for describing people based on existing ontologies. *TopBraid Enterprise Vocabulary Net* (TopBraid EVN)[2] is a proprietary tool to ease the collaborative creation of SKOS taxonomies and ontologies. It incorporates change audits, role management, search capabilities

[2] http://www.topquadrant.com/products/topbraid-enterprise-vocabulary-net/

as well as data quality rules to check for SKOS and OWL constraints. Moreover, it enables the creation of hierarchy reports through graphical user interfaces. *MoKi* [42] is a collaborative MediaWiki-based tool to support ontology modeling tailored for business processes. MoKi associates a wiki page, containing both unstructured and structured information, to each entity of the ontology and process model. However, a number of the proposed tools are not available anymore or the documentation about them is hardly accessible. Therefore, we look more in deep into well-known platforms for the ontology engineers community that are currently supported.

3.2.1 Integrated Environments with own Version Control

In the following, several well-known platforms which have their built-in version control for supporting collaborative ontology development, are described.

WebProtégé

One of the most prominent platforms in this area is *WebProtégé* [43], which is a lightweight version of the Protégé desktop editor. Its target, is supporting a wide range of users, starting from ontology engineers to domain experts, with the aim of lowering the threshold for a collaborative ontology development. As a result, WebProtégé is implemented based on a flexible and extensible infrastructure that can be utilized for various scenarios.

User Interface. WebProtégé is highly customizable and composed from a number of tabs with different functionalities. Each tab contains several *portlets*, which offer special views of ontology concepts, such as *Class Tree*, *Property Tree* and *Instance Table*.

Collaboration Support. It offers a variety of collaboration features to support the development of ontologies across team members. Among the most important ones are: tracking of changes, fine-grained access control and contextualized discussions. Changes realized to the ontologies are represented as instances of the CHAO ontology [44]. Respective portlets show and visualize notes and changes made by users at a particular time and concept.

Extensibility. WebProtégé is easily extended with additional functionalities using its plugin infrastructure. Developers can use the *Ontology Service* to add various backends, such as triple stores, OWL-API or Jena Library, thus avoiding the need to do any change in the user interface.

The first interaction of users is through client applications, such as *Desktop Protégé* or *WebProtégé*. These applications are connected with the *servlet container* via a *Remote Procedure Call* or *REST Services*. Finally, development is supported from the *collaboration framework* which enables tracking of the changes, administration of the access control, storing and facilitating discussions among team members. *OWL-API* allows for the concurrent modification and offline access of the ontologies, following principles of version control systems. Moreover, WebProtégé provides a chat service, as well as the functionality for annotating the vocabulary terms.

VocBench

VocBench [45] is a web application which targets editing of SKOS and SKOS-XL based thesauri. It supports the workflow management, validation, and publication of vocabularies, and provides a full history of changes as well as a dedicated SPARQL query endpoint. The main characteristics of VocBench are described in the following:

User Interface. Several tabs associated with particular functionalities enable a better exploration of the thesauri. Concepts are depicted using the *tree view* whereas the *details view* shows all information of a specific concept including multilingual labels and synonyms.

Collaboration Support. VocBench implements the separation of responsibilities through a role-based access control mechanism, checking user privileges for different tasks of the thesauri editing. The *editorial workflow management* facilitates the collaboration between the stakeholders by enabling monitoring and approving changes by responsible roles i.e., *content validators*.

Alignment. Since VocBench is dedicated to manage multiple thesauri at the same time, a dedicated feature is implemented to help on the creation of alignments between them. Mappings among the SKOS concepts in different thesauri can be manually created or assisted from a concept-tree feature associated with an advanced search mechanism.

In addition, VocBench allows for the *aggregation of concepts* that belongs to different schemas as well as filtering them based on a specific schema. External vocabularies and data can be imported and exported from and in various RDF serialization formats. Various statistical reports and metrics related to thesauri, such as hierarchy depth and width, level of uniformity can be generated. The integration of SPARQL 1.1, enables users to formulate customized queries to get more insights about the thesauri.

The implementation of VocBench is divided in several layers. The layer dealing with presentation and multiuser management is powered by *Google Web Toolkit*. It stores the user accounts and track their activity in a relational database, accessed via a JDBC connector. The communication with different triples stores is supported through RDF API abstraction layer which is based on *OWL ART*[3]. Data management layer is an extension of the Semantic Turkey RDF platform with new services necessary for the operation of VocBench. Thus, the management of RDFS and OWL ontologies is facilitated through a number of integrated layers and services.

PoolParty

PoolParty [46] provides a web interface for building and managing SKOS-based thesauri. A user-friendly GUI facilitates the participation of domain experts. It allows for the utilization of information from external Linked Open Data sources and performing a text-based analysis on documents to extract relevant concepts necessary for thesauri modeling.

User Interface. The GUI of PoolParty is developed by utilizing AJAX techniques to ease merging of concepts via drag & drop features as well as the auto-completion of labels while a user is typing. Moreover, a key-phrase extractor helps on semi-automatically expanding of thesauri by analyzing various documents, e.g. PDF files or web pages. A number of integrated visualization features, such as graph- and tree-based, are used to view concepts and their inter-relationships.

Collaboration Support. PoolParty has a strong focus on enabling collaborative development for people with no background knowledge on Semantic Web. *Taxonomists* are supported with a rich-feature set for creating and editing concepts and tracking the changes performed over the time. Any modification or deletion is captured using versioning mechanisms and can be compared in detail to find the potential differences. Workflows can be customized to address the project requirements and maintain the information flow across the involved stakeholders.

Publishing Capabilities. Thesauri may be published as Linked Open Data and linked to other external datasources, such as DBpedia lookup service. This enhances interlinking of the concepts and enriching them with additional information from outside. A SPARQL endpoint offers the capability for executing queries and getting more insights of the thesauri. Additionally, an HTML Wiki version of the thesauri enriched with RDFa metadata is published, as an easy alternative for browsing and editing of the concepts. Thesauri can be imported in various serialization formats, including RDF/XML or Turtle.

[3] http://art.uniroma2.it/owlart/

3.2.2 Integrated Environments based on Generic Version Control Systems

Three approaches for collaborative ontology development based on generic version control systems, such as *Apache Subversion SVN* or *Git*, are presented in the following:

SVoNt

Luczak at al. [47] proposes *SVoNt*, an approach for versioning of OWL ontologies based on SVN. Its objective is to enable the revision history of the ontology concepts similar to the way that SVN performs for documents. A separate server is used to store conceptual changes between different versions of ontologies. These versions are generated as a result of a *diff* operation between the modified and the base ontology. As a result, the backward compatibility of ontology changes made over the time is ensured. SVoNt supports conflict detection and resolution by comparing the structure and semantics of the ontologies.

Design and Implementation. SVoNt is implemented following the principles of a client-server architecture. The SVoNt server extends SVN and reuse its built-in features, like tracking, versioning and authentication. In addition, the SVoNt server includes functionalities dedicated for ontologies that are implemented in the pre-commit hook. During the commit procedure, the ontology is checked for consistency, changes are detected and logged to a metadata repository. An external mechanism is integrated to represent the structural or semantic changes that are realized in the ontology, in a concept-based level. As a result, a specific file is generated using Ontology Metadata Vocabulary (OMV) after each commit and used to store the ontological changes. On the other hand, the SVoNt client includes modules adopted to support the ontology versioning. Moreover, it is able to maintain the history of local changes of the ontology since the last modification. For this reason, special modules, such as *Change-Detection* and *Change-Selector* are used to detect and commit changes to the server, respectively.

OnToology

OnToology [48] is a tool for collaborative ontology development based on GitHub. The objective is to address several requirements related to the development process. These requirements cover various aspects of the ontology life-cycle, such as documentation, visual depiction, evaluation and publication of ontologies. Each repository should be registered to OnToology for enabling monitoring and generation of artifacts, each time when new changes are pushed.

Design and Implementation. OnToology comprises two main parts: 1) the *web interface*; and 2) the *integrator*. The web interface is responsible for handling the notifications delivered each time when the ontology is changed as well as enabling users to configure the platform according to their needs. The integrator executes an automatic workflow by communicating with three external components. It produces an HTML documentation of the ontology using *Widoco* [49], while a report for possible ontology pitfalls is provided based on the *OOPS* service [50]. AR2DTool[4] is used for creating class and taxonomy diagrams. Finally, all generated artifacts can be attached to the repository after a *pull request* is performed.

Ontohub

Ontohub [51] supports the development and management of heterogeneous ontologies in distributed environments. It enables sharing and exchanging of ideas as well as contributions among

[4] https://github.com/idafensp/ar2dtool

the involved communities. Ontologies written in heterogeneous languages can be integrated and easily retrieved. Complex inter-theory mappings of concepts and their relationships associated with formal semantics are allowed. Furthermore, alignment from a network of ontologies is possible and new ontologies can be derived as a result of that. The objective of Ontohub is to satisfy a subset of requirements defined in the Open Ontology Repository (OOR) initiative[5]. Among the most important features of Ontohub are related with supporting of: 1) distributed and modular development of ontologies; 2) alignment and combination of various ontologies; 3) different ontology languages, such as RDF, OWL and vice-versa translations; and 4) federated and collaborative ontology development repositories.

Design and Implementation. The Ontohub architecture is composed of several decentralized services performing specific tasks. Data exhange between Onthub and other similar platforms is allowed through a *federation API*. *Parsing and static analysis* service returns any symbol and sentence of a particular ontology in XML format. *Local inference* is a RESTful API which encapsulates batch-processing reasoners, such as Fact, Pellet and SPASS. Ontologies and their changes are stored and managed in the *persistence layer*, which is based on Git. Therefore, branching and merging as well local work on user machines are supported.

Summary The aforementioned platforms, such as *WebProtégé* [43], *VocBench* [45], *PoolParty* [46] provide a rich-feature set fulfilling many requirements for ontology development in collaborative scenarios. In order to manage change synchronization, conflict prevention and ontology evolution in general, these platforms utilize their integrated version control, which is tailored for processing semantics of ontology constructs. Thus, they are not focused on the direct inclusion of the generic version control systems as a core part of the ontology development process. On the other hand, *SVoNt*, *OnToology*, *Ontohub* are centered around generic version control systems, such as *SVN* and *Git*. *SVoNt* leverages SVN as a VCS for the versioning of ontologies. It stores conceptual changes between different versions of ontologies in a separate server. These versions are generated as a result of a *diff* operation between the modified and base ontology. SVoNt supports conflict detection and resolution by comparing the structure and semantics of the ontologies. *OnToology* [48] is a tool for ontology development based on Git. It generates a documentation using *Widoco*, while an ontology pitfalls report is provided based on the *OOPS* service. OnToology uses AR2DTool for creating class and taxonomy diagrams. *OnToology* is integrated on *GitHub* as hosting platform restricting users to only its basic features. Providing a user-friendly client which hides the complexity of the version control system is not in the focus of these works. Thus, these systems are rather suited for ontology development projects that involve users with a strong technical background. *Ontohub* platform has *Git* client integrated in its own architecture to manage changes. It provides a number of different views for documentation, visualization, and evolution. In addition, it uses the *Distributed Ontology, Modeling and Specification Language* (DOL) as a meta-level to allow for the representation and alignment of ontologies that are logically heterogeneous. *Ontohub* has a dedicated *hosting service* to store and manage ontologies from different domains. This platform is focused on supporting the development process after *submission* of changes to the remote repository by generating set of artifacts to enable further exploration. Furthermore, there is a lack of support for scenarios where various ontology authoring editors are used in the development process.

We foresee the necessity for an integrated environment which encapsulates a number of components to enable ontology development in distributed scenarios. Moreover, stakeholders

[5] http://www.oor.net/

using heterogeneous editors should be able to easily synchronize their changes. Apart from GitHub, this environment has to provide support for ontology projects managed in various hosting platforms, like *Gitlab* and *Bitbucket*. A number of additional views and services should allow third-party users and applications to explore and reuse ontologies in different formats.

3.3 Conflict Prevention during Change Synchronization

Synchronization of changes among different versions of the same artifact has attracted the attention of researches for several years. For example, enhancing VCSs with additional information related to the semantics of software code and re-factoring with the objective of improving the merging process has been proposed in a number of works [52–54]. Asenov et al. [55] and Protzenko et al. [56] propose to add *unique IDs* to source code elements in order to achieve a more precise conflict detection. Brun et al. [57] presents an approach called "speculative analysis", identifying the possible existence of conflicts in a continuous and precise way. Furthermore, they identify several classes of conflicts and provide detailed instructions how to address them. However, these approaches are focused on source code of software artifacts where the order of lines is important. Although, line order can also be important in our case, semantics encoded in the ontology is not necessarily affected by the position of the triples in the file.

Version Management for Model-based Development

We describe works whose main focus is on overcoming the problem of wrongly indicated conflicts in the field of *model-based development*, where *model* is the main artifact. *SMoVer*, a semantically enhanced Version Control System for models, is proposed by Altmanninger et al. [58, 59]. Using the semantic view concept to explain aspects of a modeling language, a better conflict detection can be achieved and the reason of conflicts is more easily determined.

Brosch [60] suggests to use a model checker for detecting semantic conflicts of an evolving UML sequence diagram. When an automatic merge is not possible due to conflicting changes, additional redundant information essential for the models is used to determine invalid solutions. With this technique, it is possible to assert concrete modifications realized in a sequence diagram.

A related technique uses domain-specific metamodels describing syntactic and semantic conflicts associated with their resolution [61]. This technique allows for the identification of different conflict patterns that occur during the modeling phase, which are frequently ignored by common structure-based algorithms.

Krusche et al. [62] tackle real-time model synchronization, and propose EMFStore, a peer-to-peer based solution for real-time synchronization of changes on model instances with all collaborators of a session. EMFStore borrows concepts of VCS's to synchronize models in real-time but does not include a VCS itself.

Zhang et al. [63] investigate composite-level conflict detection in UML Model Versioning, and propose a two-fold approach: in the preprocessing stage, redundant operations are removed from the originally recorded operation lists. During the conflict detection stage, a fragmentation procedure is performed to collect only potentially conflicted operations into the same fragment. Finally, a pattern-matching strategy is followed to solve the conflict detection problem.

Version Management for Ontologies

Approaches that focus on providing version management for ontologies in collaborative development processes are discussed in the following.

An ontology for unique identification of changes between two RDF graphs is presented by Lee et al. [64]. To recognize these changes (or deltas), a pretty-printed version of RDF graphs is utilized. The authors distinguish two types of deltas that can be applied as patches to RDF graphs. First, *weak* deltas, which are directly applied to the graph from where they are computed. Second, *strong* deltas, which specify the changes independently of the context. This approach focuses on the semantic representation of changes and its application to RDF graphs.

Vöelkel et al. [65] present *SemVersion*, an RDF-based system for ontology versioning. The approach is based on the two core components *data management* and *versioning functionality*. The first is responsible for the storage and retrieval of data chunks. The second deals with specific features of the ontology language, such as structural and semantic differences. To find semantic differences between two versions, e.g., whether a statement has been added or removed, SemVersion employs a simplified heuristic method for conflict detection.

Cassidy et al. [66] propose an approach for realizing a distributed version control system for RDF stores. The approach is based on a semi-formal theory of patches to allow for the manipulation of so-called RDF-patches with the objective of facilitating the revert and merge operations. An implementation of the approach has been realized on top of the VCS Darcs[6] which enables linguistic annotation on RDF stores among different users.

A holistic approach for collaborative ontology development based on ontology change management is described by Palma et al. [29]. The approach comprises different strategies and techniques to realize collaborative processes in inter-organizational settings, such as centralized, decentralized, and hybrid ones where multiple people are involved.

Edwards [67] proposes techniques for managing *high-level* application-defined conflicts. Consequently, the introduced mechanisms should be able to handle conflict resolutions. Further, certain types of conflicts can be tolerated and others forbidden according to the specified application requirements. Furthermore, a multi-editor environment for collaborative ontology editing is presented by Noy et al. [44]. The proposed framework is able to control and maintain ontologies, as well as support users throughout the whole ontology development process.

Summary The majority of the above mentioned approaches [29, 44, 65, 67] rely on their own version control mechanisms tailored for ontology development. Other approaches [64, 66] utilize a semi-formal theory of patches to find deltas among versions, thus enabling to revert or merge specific versions. Considering an increasing use of generic version control systems for ontology development, we identify a significant need to empower these systems for coping with different serializations. Moreover, the solution should be provided as an integrated service without forcing users to install on their local machines. As a result, a huge number of false-positive conflicts can be avoided and contributors can use heterogeneous ontology editors for modeling purposes.

3.4 Test-driven Approaches for Ontology Development

Many research approaches have been presented to support ontology engineers in developing ontologies according to domain requirements while at the same time avoiding constraints violation.

[6] http://darcs.net/

In this regard, Ren et al. [68] present an approach for enabling ontology engineers to formulate machine processable *Competency Questions (CQs)*. The implemented solution allows users to utilize the predefined CQs or to enter new CQs in a controlled natural language. As a result, appropriate *Authoring Tests* are generated and automatically executed to check whether an ontology meets predefined requirements. An approach for test-driven development of ontologies is described in [69]. It provides a number of predefined tests which help ontology developers to verify which axiom is missing and how to properly add it. Unit Tests for Ontologies [70] transfers principles of software engineering techniques to test-based development of ontologies. Two ontologies, *positive test ontology* T^+ and *negative test ontology* T^- are created to force the developed ontology to follow constraints defined in T^+ and avoid constraints defined in T^-. SHACL[71] is a language for validating RDF graphs against a set of conditions provided as *shapes* and other constructs expressed in the form of an RDF graph. [72] presents an approach for validating SHACL constraints by using two different algorithms. A methodology for assessing the quality of Linked Data is described in [73]. It uses the concepts of test-driven software development such as test cases to ensure a basic level of quality. Test cases defined as SPARQL templates later are materialized as concrete test queries for assessing the quality of the ontologies and knowledge bases. A parent-child relationship between test cases can be established, such that a failure of parent causes not executing of child test case. OntologyTest [74] is a tool that supports specification of different types of tests such as *instantiation tests*, *recovering tests*, or *satisfaction tests*. These tests check for the predefined ontology requirements, and can be run all at once, by grouping them into categories or individually.

In software development, *regression testing* is the process of retesting the software with a test suite after each modification. However, due to large amount of time needed for *regression testing*, much research has been conducted in areas such as: *minimization*, by eliminating redundant test cases; *selection*, by identifying relevant test cases based on recent changes; and *prioritization*, by ordering the test cases based on specific criteria [75]. Various techniques are developed in this regard. Particularly relevant for us are studies focused on so-called *Graph-Walk Approaches* such as Control Dependency Graphs *(CDGs)*. The *CDGs* utilizes a depth-first algorithm to traverse the graph for selecting a subset of test cases according to the modified code [76].

Summary Considering the above-mentioned approaches, we are focused on efficient test-driven development technique, where the ontology is the main artifact. In order to ensure an efficient evaluation of test cases, a dependency graph between child test cases which can have multiple parents should be established. Therefore, additional mechanisms are necessary to traverse the graph and store faulty test cases. As a result, the graph can be pruned with the objective of avoiding subsequent test cases from further evaluation. The approach should be able to perform a file-oriented evaluation of test cases, in order to respect the fact that an ontology can be composed of multiple files, representing submodules. As a specific functionality, such an approach should consider the fact that a team is composed by many members, causing the test case evaluation to be stakeholder-based.

Part II

Collaboratively Developing Ontologies

This part describes a methodology and a platform for enabling stakeholders to develop ontologies in a collaborative way. In Chapter 4, we start with collecting important requirements based on a round-trip development model, which comprises three fundamentals steps: *modeling*, *population* and *testing*. In addition, requirements are gathered based on the state of the art review as well as validated from a survey with ontology experts. These requirements cover methodological and technical perspectives of the ontology development process. In Chapter 5, we continue with presenting a lightweight methodology to support construction of ontologies in distributed scenarios. The presented methodology addresses in detail the organizational and operational aspects. Finally, in Chapter 6, an integrated environment for collaborative ontology development based on version control systems, is shown. A conceptual architecture and a concrete implementation of this environment is described in detail. It incorporates a number of different built-in and external components to address the technical requirements.

CHAPTER 4

Requirements for Collaborative Ontology Development

Ontologies are powerful means to realize the conceptualization of the knowledge for a particular domain. However, developing ontologies can require a significant investment, which is difficult to make by a single person or organization. Thus, a number of interested stakeholders come together and decide to define a new ontology for the chosen domain. They build a consortium which drives this process and in periodic meetings, representatives from the different stakeholders come together, communicate their specific needs, and try to find a consensus. If the outcome, which can be a vocabulary of terms, is at a satisfying maturity level, a specification document will be released. This process, which we refer to as *collaborative ontology development*, itself is a complex problem to be solved. In fact, the main challenge for the ontology engineers is to work collaboratively on a shared objective in a harmonic and efficient way while avoiding misunderstandings, uncertainty and ambiguity. The quality of the produced ontology is another challenge that must be tackled with adequate methods and concrete guidelines. Finding a suitable collaboration methodology is exacerbated by the number and diversity of the involved stakeholders as well as the complexity of the domains. Due to the open, distributed and participatory nature of the Web, a solution to this problem is of paramount interest.

In this chapter, we investigate the fundamental activities of the development process, i.e., *modeling*, *population* and *evaluation*. As a result, a *round-trip development model* is introduced which suggests an iterative and incremental fashion of moving between the fundamental activities. A number of methodological and technical requirements for enabling ontology development in distributed and heterogeneous environments according to the round-trip model are collected. In addition, to obtain insights for modeling practices, we perform an analysis of some of the widely used ontologies. We observe common modeling practices regarding to reuse, internationalization, documentation and naming present on these ontologies, which impact on better understanding for the defined concepts. For the methodological scope, we focus on workflow independent requirements needed for organizational and modeling purposes. Next, we derive a number of requirements necessary to provide technical support, and group them in four different categories: 1) *Change Management*; 2) *Quality Assurance*; 3) *User Experience*; and 4) *Ontology Deployment*.

Contributions of this chapter are summarized as follows:

- Conception of round-trip ontology development comprising of three fundamental activities: 1) *modeling*; 2) *population*; and 3) *testing*;

- Analyzing the modeling practices applied in some of the widely used ontologies;
- A set of requirements needed for the methodological support of the development process;
- A set of technical requirements necessary for designing and implementing an approach to enable distributed development of ontologies in heterogeneous scenarios.

This chapter is based on the following publications:

- **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Steffen Lohmann, Sören Auer. *Git4Voc: Collaborative Vocabulary Development Based on Git.* In International Journal of Semantic Computing (IJSC), 1-24, World Scientific. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, my contributions are related with collecting requirements for collaborative ontology development, the description of the approach, the revision of the related work and presentation of the use case evaluation;

- **Lavdim Halilaj**, Niklas Petersen, Irlán Grangel-González, Christoph Lange, Sören Auer, Gökhan Coskun, Steffen Lohmann. *VoCol: An Integrated Environment to Support Version-Controlled Vocabulary Development.* In 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW) 2016 Proceedings, 303-319, Springer. This article is a joint work with Niklas Petersen and Irlán Grangel-González, PhD students at the University of Bonn. In this article, I conducted the problem description, definition and implementation of the conceptual architecture, the revision of the state of the art approaches, the presentation of the use cases, and the realization of the user study evaluation;

- Irlan Grángel-González, **Lavdim Halilaj**, Gökhan Coskun, Sören Auer. *Towards Vocabulary Development by Convention.* In 7th Knowledge Engineering and Ontology Development (KEOD) 2015 Proceedings, 334-343, SciTePress. This article is a joint work with Irlan Grángel-González, a PhD student at the University of Bonn. My contributions focused on the review of state of the art approaches, devising, conducting and analyzing the user study and presentation of the outcomes.

This chapter is organized as follows: we initially discuss the fundamental phases of the ontology development process in Section 4.1. We then analyze several widely used ontologies and get insights on practices used to model them in Subsection 4.1.2. We start Section 4.2 with the foundation of requirements necessary for ontology construction. Subsection 4.2.1 lists requirements needed for organizational and modeling purposes. A number of technical requirements to be covered by the platform are described in Subsection 4.2.2. Finally, we summarize the chapter in Section 11.4.

4.1 Method

During a *collaborative ontology development* process a group of decentralized stakeholders with a shared objective work together to model a domain of interest in a harmonic and efficient way while avoiding misunderstandings, uncertainty and ambiguity. Deriving requirements for the envisioned methodology and development environment demands the clarification of our understanding of the most fundamental ontology development activities. An ontology comprises a terminology which is known as *TBox - Terminological Knowledge*. The creation of this

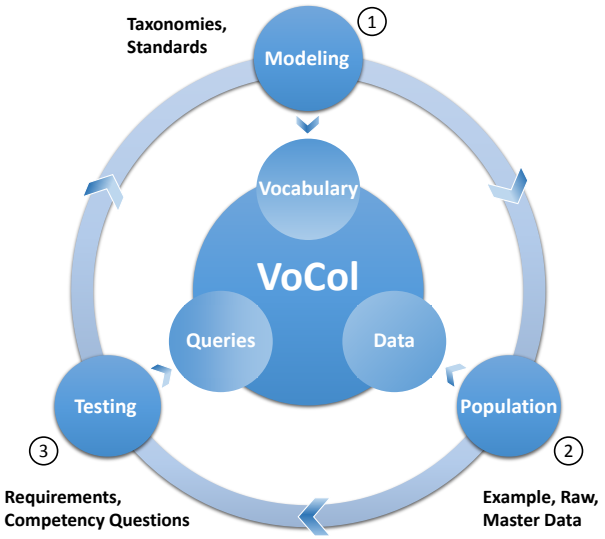


Figure 4.1: **Round-trip model**. A round-trip model covering the fundamental activities of the ontology development life-cycle: *modeling*, *population* and *evaluation*.

terminology is realized using a logical formalism during the modeling activities [4]. It comprises the analysis and conceptualization of the domain and the specification of the ontology terms, such as classes, properties, and the relationships between them. Once the ontology modeling has been completed, the next typical activity is the population. It includes the addition of actual data in line with the defined classes and properties, also known as *ABox - Assertional Knowledge* [77]. To verify whether the created ontology correctly represents the domain, a list of queries can be compiled from *competency questions* [68] and used for testing purposes. Ontology engineers may iterate in an incremental fashion between the modeling, population, and testing activities during the entire development life-cycle. In fact, these three core activities lead to the conception of the *round-trip* development, as illustrated in Figure 4.1. However, the order of these activities is not deterministic and can be performed in other way around, starting from testing, modeling, and population.

4.1.1 Important Roles

Ontologies represent a *shared conceptualization* of several people, i.e., a common understanding of a domain of interest. Usually, ontology development projects involve stakeholders that have various levels of expertise and interests. Traditional methodologies recognize three main distinctive roles: knowledge engineers, ontology engineers and domain experts [10]. Knowledge engineers are responsible to coordinate activities for extracting the knowledge of the domain by defining the requirements, constraints and relevant questions. The role of domain experts is to provide their knowledge about the domain, such as identifying important concepts, their attributes and the relationships between them. Moreover, domain experts can validate changes whether they are correctly modeled via issue reporting mechanisms or even perform changes in some organizations. As a result, a conceptual model is created which describes the elicited knowledge of the domain of interest. Next, ontology engineers transfer the conceptual model into a machine understandable format using a formal representation language. This is realized by finding the right axioms which impact the level of the expressiveness.

Table 4.1: **Roles and permissions**. Various active and passive roles and their permissions that are involved in the collaborative ontology development process.

Permission type	Contributors		End Users	
	Ontology engineers	Domain experts	Users	Machines
Propose changes	+	+		
Perform changes	+			
Report issues	+	+	+	
Consume	+	+	+	+

In collaborative ontology development projects, generally two roles are distinguished: ontology engineers and domain experts [5]. Domain experts or ontology contributors actively participate on the development process by providing feedback and proposing changes to the current concepts. Ontology engineers are responsible for encoding the knowledge of the domain using a specific formal language as well as ensuring the quality of the ontology being developed. They can perform changes in the ontology according to the new requirements or proposals that come from stakeholders. Normally, they can report issues or even develop a parallel version of the ontology according to different needs and scenarios, which in the end can be merged with the base version. Another role which occasionally is involved in ontology development, are the *end users*, who are mostly interested on the outcome in order to use the ontology for their particular use case. In addition, since ontologies are created with the objective to be comprehended by machines, intelligent agents can be another type of users. They are more affected by changes in the ontology, so it is very important that each version is validated for syntax and consistency before it is finally published. Table 4.1 lists common roles that actively contribute to the construction process or simply consume the ontology being developed as well as their respective permissions.

4.1.2 Analysis of Widely used Ontologies

With the objective of investigating common modeling practices, we compiled a list of the 20 most widely used ontologies, as represented in Table 4.2. The selection is based on the following criteria. Initially, a usage rate of more than 5% in all datasets of the Linked Data Cloud [78] is considered and as a result, 13 ontologies are chosen. Next, we look for recognized ontologies that contain best practices regarding documentation, dereferenceability and are used by independent data providers[1], which result on choosing three ontologies. Finally, *Linked Open Vocabularies* (LOV)[2] is observed for most reused ontologies. The outcome of this observation is four additional ontologies to be included on the list for further consideration. These ontologies are revised and used for many years, and also the community recognized that they are built following good practices [79]. For that reason, we believe that studying them will provide a better understanding of the common features and best practices of current ontology development. In this regard, we aim to understand important aspects of ontology creation, such as reuse, internationalization, documentation and naming as well as the implicit structure of these ontologies (e.g. use of logical axioms, property domain/range definitions).

With respect to **Reuse**, 80% of the ontologies use elements defined elsewhere and 57% of them reuse elements from at least two external ontologies. This shows a considerable presence

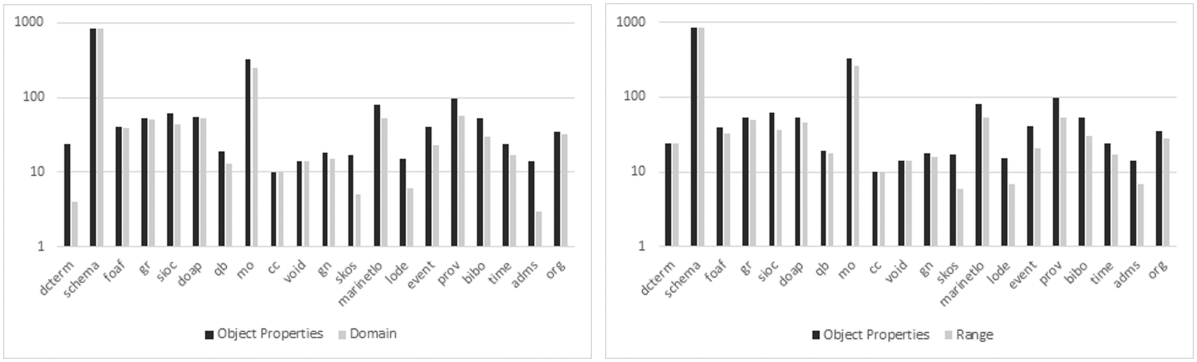
[1] http://www.w3.org/wiki/Good_Ontologies
[2] http://lov.okfn.org/dataset/lov/

Table 4.2: **Widely used ontologies**. The list of 20 ontologies, including *name*, *prefix* and *domain*, that are frequently used in different domains.

Name	Prefix	Domain
Friend Of A Friend http://xmlns.com/foaf/0.1/	*foaf*	Terms related to Persons (e.g., Agent, Document, and Organization).
Dublin Core ontology Terms http://purl.org/dc/terms/	*dcterms*	General metadata terms (e.g., Title, Creator, Date, and Subject).
WGS84 Geo Positioning http://www.w3.org/2003/01/geo/wgs84_pos#	geo	Represents longitude and altitude information in the WGS84 geodetic reference datum.
Socially Interconnected Online Communities ontology http://rdfs.org/sioc/ns#	*sioc*	Aspects of online community sites (e.g., Users, Posts, and Forums).
Simple Knowledge Organization System Namespace http://www.w3.org/2004/02/skos/core#	skos	Data model for sharing and linking knowledge organization systems.
Vocabulary of Interlinked Datasets http://rdfs.org/ns/void#	*void*	Metadata about RDF datasets (e.g., Dataset, and Linkset).
Biographical information http://vocab.org/bio/0.1/.html	*bio*	Biographical information about people, both living and dead.
Data Cube Vocabulary http://purl.org/linked-data/cube#	*qb*	Statistic data (e.g., Dimensions, Attributes, and Measures).
Vocabulary for Rich Site Summary http://purl.org/rss/1.0/	*rss*	Models the declaration for Rich Site Summary (RSS) 1.0.
Vocabulary for modeling abstracts things for people http://www.w3.org/2000/10/swap/pim/contact#	*w3con*	General concepts about people everyday life (e.g., Address, and Phone).
Description of a Project http://usefulinc.com/ns/doap#	*doap*	Terms for Open Source Projects (e.g., Version, and Repository).
Bibliographic Ontology http://purl.org/ontology/bibo/	bibo	Citations and bibliographic references (e.g., quotes, books, and articles).
Data Catalog Vocabulary http://www.w3.org/ns/dcat#	dcat	Facilitate interoperability between data catalogs published on the Web.
Schema.org http://schema.org	*schema*	Broad schema of concepts (e.g., Events, Organization, and Person).
GoodRelations http://purl.org/goodrelations/v1	gr	E-Commerce related terms (e.g., Products, Services, and Locations).
Music Ontology http://purl.org/ontology/mo/	*mo*	Terms related to music (e.g., Artists, Albums, and Tracks).
Creative Commons schema http://creativecommons.org/ns	*cc*	Describes copyright licenses (e.g., License Properties, and Work Properties).
GeoNames http://www.geonames.org/ontology	gn	Geospatial semantic information (e.g., Population, and PostalCode).
MarineTLO ontology http://www.ics.forth.gr/isl/ontology/MarineTLO/	marinetlo	Marine domain (e.g., Species, and Marine Animal).
Event Ontology http://purl.org/NET/c4dm/event.owl	event	Describes reified events (e.g., event, location, and time).

of the reuse aspect in the selected cases. An important aspect of **Internationalization** (I18n) is the support for multilinguality. This can be implemented by providing textual values for properties, such as `rdfs:label`, `rdfs:comment` in different languages, i.e., associating dedicated language tags for RDF string literals. In 70% of the ontologies, we encounter explicit literals in English language, i.e., *@en*. In 15% of the cases, there exist translations of the terms into other languages and in the remaining 15%, there were no explicit language tags used at all. Consequently, despite the fact that I18n is important for existing ontologies, we discover that the most common practice is to support only the English language.

Documentation refers to human readable labels and descriptions (e.g., using the `rdfs:label`, `rdfs:comment` properties) added to the ontology elements, i.e., classes, properties and individuals. We observe that `rdfs:label` or `rdfs:comment` are present in 86% of the cases. It is worth noting that the combination of the two above mentioned elements with `rdfs:isDefinedBy` is



(a) Object properties and domain axioms

(b) Object properties and range axioms

Figure 4.2: **Usage statistics of domain and range**. a) Relation of the amount of object properties and domain axioms; and b) Relation of the amount of object properties and range axioms.

used with a frequency of 57%. Only in one case (i.e., 5%), there is no any form of documentation. This shows that the documentation (i.e., `rdfs:label`, `rdfs:comment` for commenting, and `rdfs:isDefinedBy` for linking definitions) is widely used by the existing ontologies.

Another important practice in ontology creation is the convention for **naming** elements. The *CamelCase* notation is with 60% of the cases the most used one. In all other cases (i.e., 40%), no homogeneous naming convention could be identified. A combination of CamelCase notation, underscore or dash sign is commonly encountered instead.

We perform a statistical analysis regarding the inclusion of **domain and range axioms** for properties. Using the *Shapiro-Wilk* test over the observations of the object properties, domain and range axioms, we encounter that the data do not follow a normal distribution for these variables. Our hypothesis is that there is a correlation in the obtained data regarding the amount of object properties and the domain and range axioms. Therefore, the *Spearman* rank coefficient is computed, to check for any existing correlation. For the amount of object and domain properties as well as range axioms, a value of 0.91 and 0.95 is obtained, respectively. This indicates a strong correlation between object properties as well as domain and range axioms. The results for various ontologies are illustrated in Figure 4.2(a) and Figure 4.2(b), respectively. The y-axis is transformed to logarithmic scale for a better comprehension. We perform the same process between domain and datatype properties as well as range axioms. In this case, we obtain the value of 0.93 for both cases. These observations favor the conclusion that object properties and data properties should contain domain and range axioms. The percentage for **inverse properties** (60%) and **class disjointness** (50%) is calculated as well. This data indicate that the above mentioned axioms should be more carefully analyzed regarding the domain but are still important when building a ontology.

4.2 Requirements

In order to provide methodological and technical support for the round-trip development of ontologies, corresponding requirements have to be identified and addressed accordingly. Therefore, we identified requirements that are crucial for a successful adaptation of Git to ontology development. The process of gathering is realized by aggregating insights from the state of the art, analysis of the round-trip development, the outcome from the widely used ontologies,

and our own experiences with developing a number of the ontologies, such as *MobiVoc*, *SCORVoc* or *STO* using Git as a version control system.[3] These requirements are divided into two main groups, based on scope coverage: 1) methodological - dedicated to the governing and operational aspects of the development process; and 2) technical - related to the functionalities to be provided by an integrated environment that aims to support full-featured ontology development. Further, the technical requirements are split into four categories: 1) *Change Management*; 2) *Quality Assurance*; 3) *User Experience*; and 4) *Ontology Deployment*. Next, we describe in detail the requirements of the two main groups as well as categories of the second one.

4.2.1 Methodological Requirements

In the following, requirements that ease collaboration and governing aspects of the distributed development process are described.

`M1 Communication:` The collaborative development of ontologies is about finding consensus among different stakeholders. It is essential that they share ideas, make agreements, and discuss issues during the entire development life-cycle [44, 80], w.r.t. introducing new elements, extending or modifying the subsumption hierarchy [81]. An effective communication has a significant impact on the quality of the collaboration and its outcome.

`M2 Information Provenance:` Each change in ontology reflects the understanding of the domain by the respective stakeholder who conducted that. In case of disagreements, it is necessary to know which change has been made by whom at which time and for what reason. Hence, managing and documenting provenance of the information and design decisions is needed during the entire development process.

`M3 Workflow Independence:` The overall field of methodologies and workflows for collaborative ontology development is continuously changing [80]. Tools supporting collaboration should be generic and be able to adapt in highly dynamic context. Therefore, it is essential to facilitate scenarios with high flexibility that follow different methodologies and workflows.

`M4 Parallel Development:` The community should not be limited to a strict and linear path of the development process, rather it should be able to work simultaneously, considering their diverse interests or requirements. As a result, parallel versions might co-exist for purposes of *testing new features*, *bug fixing* or other issues. Therefore, support for *parallel development* is crucial to boost the productivity of the team in such scenarios [28].

`M5 Role Definition:` Stakeholders with different backgrounds and levels of expertise are involved in ontology development. Consequently, clear definition of roles and permissions is an important requirement that must be taken into consideration [5, 80].

`M6 Modularity:` Modularization is recognized as an important step in collaborative ontology building [82]. Reusability, the decrease of complexity, ownership and customization are some of the benefits of ontology modularization. Although there is no universal way to perform the modularization [83], guidelines based on the clustering or number of concepts are necessary to reduce the cost of maintenance and increase the applicability of ontologies.

`M7 Version Labeling:` Release versions of ontologies should be appropriately named using self-description labels. This ensures that users, either humans or machines have always the possibility to use specific version according to their needs, and not only the latest one.

[3] See https://github.com/vocol/mobivoc , https://github.com/vocol/scor and https://github.com/i40-Tools/StandardOntology.

`M8 Modeling Practices:` During the modeling phase, a number of practices should be persistently followed by all team members. These practices which comprise *naming conventions*, *reusability of existing ontologies*, *metadata definition* and *utilization of particular properties* help on increasing the quality of ontology and avoid later inconsistencies.

`M9 Multilinguality:` In order to have a wide range of applicability to different cultures and communities, the ontology terms *must* be translated into various languages [84]. The localization (and internationalization) process of the ontology should be continuously maintained and curated by dedicated team members.

4.2.2 Technical Requirements

The following requirements are crucial for supporting the development process from technical perspective. These requirements are divided in four categories: 1) *Change Management*; 2) *Quality Assurance*; 3) *User Experience*; and 4) *Ontology Deployment*.

Change Management This category comprises requirements related to the management of meta-information about changes as well as coping with heterogeneous ontology editors.

`P1 Conflict Detection:` Several users may simultaneously perform changes regarding to their needs or make corrections to existing terms in their local copies. Such changes must be tracked and conflicts must be detected during synchronization process.

`P2 Evolution Tracking:` Collaborative development of ontologies should respond to the evolution of the knowledge domain [5]. Therefore, support for identifying and documenting the semantic differences between versions is necessary to enable developers to understand the mentioned evolutions. This includes the modifications, the additions of new elements (i.e., classes, properties) as well as the removals of existing ones.

`P3 Editor Agnostic:` In contrast to software code, ontologies are abstract artifacts which can be serialized with different techniques without losing the encoded semantics. Since contributors may use different editors which style the syntax in different ways, the support of the collaboration must be editor agnostic and syntax independent.

Quality Assurance This category comprises requirements for the systematic checking of quality criteria that should be fulfilled by the ontology being developed.

`P4 Continuous Validation:` Syntactic and semantic correctness as well as the application of best practices on designing ontologies are relevant quality aspects. Providing tool support for these aspects is essential to help contributors in making fewer errors and ultimately increasing the quality of the ontology.

`P5 Testing:` Competency Questions, i.e., questions that the ontology must be able to answer, can be translated into queries and used as test cases in later phases [68]. An integrated ontology development environment should provide mechanisms to allow users for executing such queries efficiently.

User Experience This category groups requirements for enabling contributors to effectively achieve their objectives and in a user-centered manner.

`P6 Client Performance:` The development of ontologies is driven by contributors who might be affected from occasional network interruptions. Therefore, the contributors should be

able to work offline on their local machines as well as synchronize and distribute the ontology changes once they are again connected [85].

`P7 Documentation:` Domain experts are often team members with little technical expertise in knowledge representation and engineering tools. In order to enable them contributing to the development process, presenting the current state of the ontology in a human-friendly way is vital. Therefore, an automatic documentation generation feature is necessary.

`P8 Visualization:` Visualization is known to have a positive impact on the modeling, exploration, verification, and sense-making of ontologies [86]. It is particularly helpful for domain experts, but can also provide useful insights for knowledge engineers.

Ontology Deployment Finally, there are requirements concerning the deployment of the developed ontology that also need to be taken into account by an integrated environment.

`P9 Machine Accessibility:` An important requirement towards realizing the vision of the web as a global information space is to provide details about the ontology terms in various representation formats [87]. This enables machines to access and comprehend the ontology according to their specific use cases.

`P10 Querying:` In order to check whether the developed ontology is suitable for a certain use case, the environment should appropriate interfaces for execution of user-defined queries. Furthermore, a possibility to export the query results in various formats can help to achieve a better comprehension of the ontology.

4.3 Summary

In this chapter, we investigate the collaborative ontology development process in distributed and heterogeneous scenarios. This process involves a consortium of stakeholders which realize periodic meetings to communicate specific needs, and try to find a consensus for topics, such as ontology terms, their definitions, and the reuse of external ontologies. Next, we introduce the round-trip development model and explained its three fundamental steps that may be performed iteratively and in an incremental fashion. We then continued with analyzing several ontologies that are widely reused and presented results from this analysis regarding to their modeling practices and design decisions. We observed a number of commonalities in terms of the key aspects, such as reuse, documentation, multilinguality, naming, validation and practices regarding authoring. The main outcome of the above analyses is an exhaustive list with requirements that should be considered from the methodological perspective. Since software and ontologies are not the same, we study their differences by identifying the requirements that should be considered when construction of ontologies is centered around the version control systems. Next, from the technical perspective, we collected a number of requirements to be addressed by an integrated platform with the objective of supporting ontology construction in distributed scenarios.

CHAPTER 5

A Lightweight Methodology for Developing Ontologies in Distributed Environments

The dynamic World Wide Web demonstrates that the interoperability is also possible to some extend with a minimalistic standard set and flexible *de facto* standards. This is mainly enabled by focused applications and well documented specification pages. In some cases these *de facto* standards become real standards. However, the main idea is that they are not created in a top-down approach as in traditional standardization activities. Concretely, the implementation is not based on a predefined standard, but the standard is based on the adoption and the experience with existing implementations. That makes them more practical and avoid overly engineered standards like *Common Object Request Broker Architecture* (*CORBA*), or *Customised Applications for Mobile networks Enhanced Logic* (*CAMEL*). We consider this as a bottom-up approach for defining a standard.

Even within the Web context, the danger of overly engineered standards is actual. The vision of the Semantic Web for example, caused the enthusiastic creation of standards like the *Web Ontology Language* and the *Rule Interchange Format*, to represent knowledge and rules. However, these standards will be broadly adopted only if they are really practical enough to be used in various information systems. In contrast, positive examples likes *Schema.org*, clearly demonstrate that a practice-oriented approach is very effective. The definition, implementation and the usage of these ontologies is integrated pragmatically and not organized sequentially.

Our approach towards a practice-oriented ontology construction is to support the collaborative ontology development with a lightweight methodology for distributed version control in a domain-agnostic way. In this regard, we have chosen *Git* for the following two reasons: on the one hand, Git is a mature version control system supported by sophisticated tools and broadly used in software development projects. More than 85 million repositories were hosted by the GitHub service in April 2018.[1] On the other hand, well-known vocabularies and ontologies like *Schema.org*, *Description of a Project (DOAP)*, or the *Music Ontology*[2] publish their efforts on GitHub to leverage the contribution of the community. This indicates that the ontology development community is already familiar with Git.

In Chapter 4, a number of requirements from methodological and technical point of view are presented. This chapter addresses the requirements from Subsection 4.2.1, that are focused on the methodological aspects with the objective to ease the collaboration during development

[1] https://github.com/ten

[2] https://github.com/schemaorg, https://github.com/edumbill/doap, https://github.com/motools/musicontology

processes. We investigate the fundamental features of Git, and assess the potential exploitation for ontology construction. As a result, a number of guidelines covering information management, ontology structure, parallel development and deployment aspects, are proposed. Moreover, we analyze the applicability of the *Convention over Configuration* paradigm, a well-known paradigm and broadly adopted in software engineering, to ontology development. It aims at reducing the number of decisions that developers need to make, so they can focus on the mainly on development. Inspired by this paradigm and the broad adoption of lightweight ontologies like *Schema.org*, we propose a set of practices for facilitating ontology construction. We derive these practices from the study of widely used ontologies presented in Subsection 4.1.2 as well as our own experience in the ontology development process. The bottom-up and pragmatically best-practice oriented approach which we applied, is described in detail.

In this chapter, we address the following research question:

RQ1: Which best practices facilitate collaborative ontology development in distributed and heterogeneous scenarios?

Contributions of this chapter are summarized as follows:

- A set of guidelines to cover governing aspects of distributed ontology development;
- A number of best practices to be used for modeling and deployment purposes;
- Evaluation of the defined methodology with a concrete use case;
- A survey with ontology engineers to assess their opinion regarding the defined practices.

Some parts of this chapter are based on the following publications:

- **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Steffen Lohmann, Sören Auer. *Git4Voc: Collaborative Vocabulary Development Based on Git.* In International Journal of Semantic Computing (IJSC), 1-24, World Scientific. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, my contributions are related with collecting requirements for collaborative ontology development, the description of the approach, the revision of the related work and presentation of the use case evaluation;

- **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Sören Auer. *Git4Voc: Git-based Versioning for Collaborative Vocabulary Development.* In IEEE Tenth International Conference on Semantic Computing (ICSC) 2016 Proceedings, 285 - 292, IEEE. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I contributed to collecting requirements for collaborative development of ontologies, definition of the approach, the analysis of the related work and the presentation of the use cases in real world scenarios;

- Irlan Grángel-González, **Lavdim Halilaj**, Gökhan Coskun, Sören Auer. *Towards Vocabulary Development by Convention.* In 7th Knowledge Engineering and Ontology Development (KEOD) 2015 Proceedings, 334-343, SciTePress. This article is a joint work with Irlan Grángel-González, a PhD student at the University of Bonn. My contributions focused on the review of state of the art approaches, devising, conducting and analyzing the user study and presentation of the outcomes.

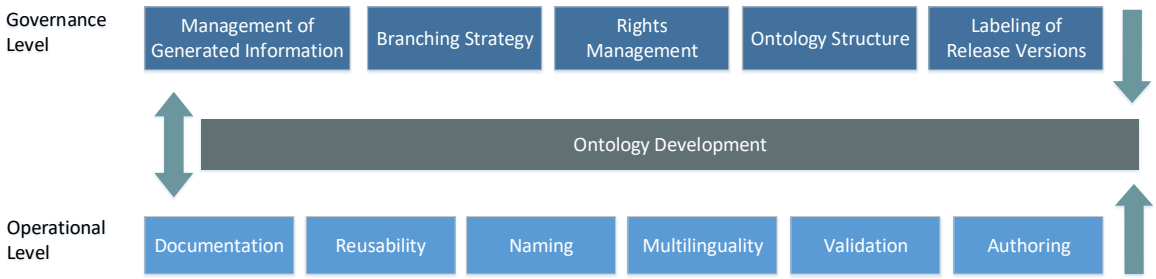


Figure 5.1: **The Git4Voc methodology**. *Governing* and *Operational* levels of Git4Voc, covering a number aspects and practices of the ontology development process, respectively.

This chapter starts with describing the approach that is followed to concept the Git4Voc in Section 5.1. In Subsection 5.1.1, we provide the guidelines on how to manage the generated information, strategies for parallel development, structuring the ontology according to its size, and labeling possibilities of release versions. We then provide a number of practices derived from analysis in Subsection 4.1.2 of the widely used ontologies. The presented approach is evaluated with a particular use case in Subsection 5.2.1 and a survey with ontology engineers is conducted in Subsection 5.2.2. Section 5.3 concludes the work in this chapter.

5.1 The Git4Voc Approach

Approaching a task can be done in two different ways. A top-down starts from the abstract and elaborates the concrete whereas A bottom-up starts from the concrete level and continues towards the abstract. From a logic perspective, the former corresponds to deductive reasoning. It starts with known facts that are considered as premises and seeks for conclusions. The latter, on the contrary, starts with a given set of statements and looks for premises that caused them.

In the context of defining a methodology for ontology development, a top-down approach starts with the facts that are known about the expected outcome, namely the ontology. From the different characteristics of it, a possible creation process is derived. In the next steps, a list of roles is established and a set of tools are developed or selected, which can be used among various steps of the overall process. In fact, most ontology engineering methodologies have been created by applying this approach. On the contrary, a bottom-up approach with the objective of deriving guidelines and best practices, is supposed to start from the current state of art and look for evidences that explain why people are doing what they are doing. The most common activities of the successful outcomes are then compiled as a set of best-practices. This is in fact the method we applied in this work. We advocate that there is no need for just another comprehensive methodology that is designed in detail in a top-down approach. Rather, we claim that in the meanwhile there are sufficient good examples to be analyzed and learnt from. For that reason, we empirically analyzed a number of popular ontology development efforts.

In typical scenarios, such as the cross-organizational or enterprise settings, there are different decision making bodies in various hierarchy levels involved in the entire ontology life-cycle. First, the *steering committee* is a board of people dealing with strategic decisions, such as which new ontologies should be created for what purpose, who will use them and who will be involved in the development process. Second, the *stakeholder committee* is responsible for governing aspects

related to the organization of the work, release versions, number of branches and types of roles including their permissions. Moreover, a *development team*, which can be part of the *stakeholder committee*, is responsible for concrete modeling aspects, including transferring of the domain knowledge into a formal language, by choosing the right axioms and handling technical issues.

We present Git4Voc, devised according to a bottom-up approach for covering aspects of the *stakeholder committee* which are presented in following. On the one hand, we show how the requirements listed in Section 4.2 can be addressed by exploiting the fundamental features of Git as a distributed version control. As a result, we investigated the following governing aspects as critical for development process: 1) management of generated information; 2) branching strategies; 3) rights management; 4) ontology modularization; and 5) labeling of release versions. On the other hand, from our analysis in Subsection 4.1.2 and examination of the state of the art, we defined a set of practices related to reuse, naming convention or multilinguality. Figure 5.1 illustrates governing aspects and development practices covered by the Git4Voc approach. The order of performing these aspects is irrelevant, stakeholders can start by agreeing on governing aspects and operational practices before development is started. They can also start constructing the ontology and then decide for aspects and practices following a *learn-by-doing* approach. However, a natural working process would be based on first approach, since it helps on reducing the number of corrections or misunderstandings in later phases.

5.1.1 Governing Aspects

In the following, we show in detail how requirements defined in Subsection 4.2.1 are addressed by our approach with the presented governing aspects.

Management of Generated Information

During the development process, a bunch of information is generated by different contributors. The capability to manage this information within the entire project life-cycle is essential. In fact, value added services like GitHub, GitLab or Bitbucket enrich Git functionality with powerful information management features. For instance, issues are a great way of tracking communications, reporting problems including bug fixes and announcements of the version releases. Communities like *Schema.org*, manage their discussions using GitHub. The above mentioned means support the requirement **M1**. Based on this fact, we propose that activities gathered in Table 5.1 should be documented. If possible, the name of issues should correspond to the name of the activities. Furthermore, the *Labeling*, as a built-in feature of the repository hosting platforms, should be used to organize the issues in different categories. In addition, all issues should contain the ontology version to which they belong to. Later on, this helps users to easily understand the version of the ontology which is affected by a particular issue.

Another important requirement in collaborative ontology development, is the ability to view the history of the changes according to the principles of the traceability in software engineering. This is conform to the requirement **M2**. Using commands *git log* and *git diff*, a user can explore the entire history of commits as well as the differences between them. Each commit should be realized based on the *Best Commit Practices*[3].

[3] http://www.git-tower.com/learn/git/ebook/command-line/appendix/best-practices

Table 5.1: **Common activities in collaborative ontology development**. A number of activities that are performed during ontology construction from different contributors.

Activity	Name	Description	Example
ACT1	Simple Addition/Deletion	Adding new or deleting existing elements like classes and properties	Adding a class in the last level of the taxonomy
ACT2	Complex Addition/Deletion	Adding new elements to be interconnected within the existing class or properties taxonomy	Adding a object property as a super property of two existing properties
ACT3	Modification	Modifying existing elements	Modifying the domain and range of an existing object property
ACT4	Reusing	Reusing elements of the Linked Data Cloud	Defining new local concepts by using external resources
ACT5	Alignment	Alignment of existing elements with equivalents in the cloud	Alignment of classes and instances with *owl:equivalentClass* and *owl:sameAs*
ACT6	Refactoring	Changing the name and metadata of an specific element and its connections	Renaming a class which is connected in many domain and range relation of properties and need to be renamed everywhere
ACT7	Common Metadata	Adding/Removing/Modifying predefine *RDFS* metadata	Adding metadata to a class with *rdfs:label, rdfs:comment*
ACT8	External Metadata	Adding/Removing/Modifying external metadata	Adding metadata to a class with *skos:prefLabel, dc:title*
ACT9	Translating	Adding/Removing/Modifying translation for the terms	Using *rdfs:label* to translate elements into different languages
ACT10	Modularization	Adding new modules to the existing ontology	Creating and integrating new modules due to new requirements
ACT11	Partitioning	Partitioning into different modules with existing elements	The ontology has grown in size and semantic complexity. Partitioning the existing ontology into different modules

Branching Strategy

Git is a very flexible tool, allowing teams to organize their work in different types of workflows[4], addressing the requirement **M3**. Branching strategies affect the quality in collaborative software development [88, 89]. Ontology development is mostly accepted to be a specific type of software development. Therefore, it is considered that the branching strategy impacts the quality of the ontologies. Well-known ontology initiatives, such as *Schema.org* use branches to organize their work. In order to design a branching model, it is important to understand the possible activities that a team can perform. Thus, we collected common activities of collaborative ontology development which are listed in Table 5.1. Aiming at producing an ontology with good quality, the entire team should be aware of these activities and how to face them in the development process. Due to their impact on the overall ontology, we have classified these activities into three categories: 1) basic activities (*ACT1, ACT7, ACT9*); 2) semantic issues (*ACT2, ACT3, ACT4, ACT5, ACT6, ACT8*); and 3) structural issues (*ACT10, ACT11*). This led us to the branching model that is depicted in Figure 5.2. We identified different branches according to the mentioned categories. Basic activities have to be performed in the *Develop Branch*. For the second category, we propose a dedicated branch called *Semantic Issues*. In case of the third category, a branch

[4] https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow

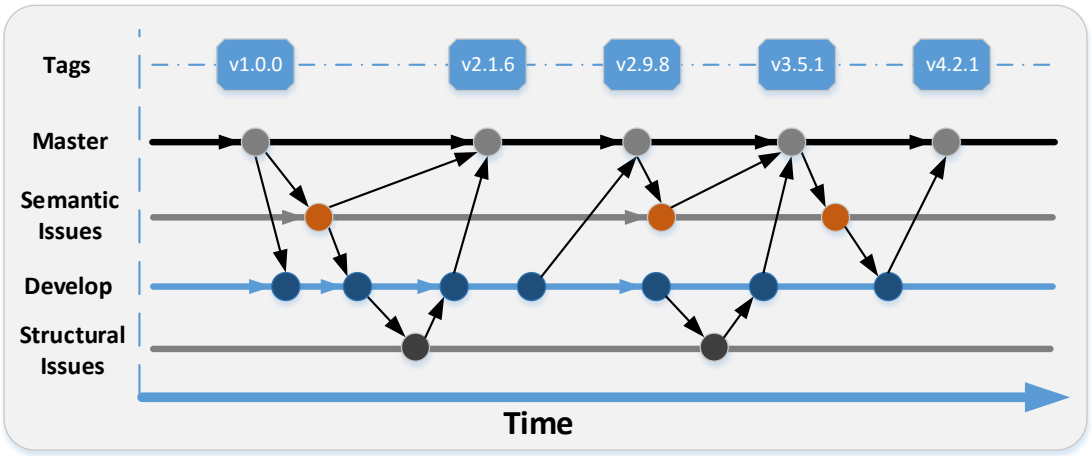


Figure 5.2: **Branching model for ontology development**. Parallel development can be facilitated and maintained using a number of dedicated branches for specific purposes, i.e., *master*, *semantic issues*, *develop*, and *structural issues*.

named *Structural Issues* has to be applied. It is important to bear in mind that we are not restricting the flexibility of Git regarding branches. On the contrary, other branches can be used as a complement of this model. Nevertheless, our approach of branching model will help developers because those branches are connected to specific activities in collaborative ontology development addressing the requirement **M4**. This model is built on top of the best practices for branching in software development.[5]

Rights Management

Definition of the roles is of paramount importance during any initiative for building ontologies in a collaborative fashion. Table 5.2 shows common roles and their permissions, with respect to the defined categories of activities. Standalone solutions, such as *GitLab*[6] and *Gitolite*[7] as well as third-party services like *Bitbucket*[8] and *GitHub*[9], offer basic options for the management of the user rights, like reading, writing, posting, adding new team members, and the definition of new tags. However, even with these solutions a high level of user management, i.e., restricting editing a specified number or type of classes, properties or instances, cannot be achieved with Git. In order to address requirement **M5**, we explore a combination of branching and hooks. With this combination a more fine grained access management can be achieved. Concretely, by using server-side hooks, administration of the rights on top of user roles is possible. For instance, an implementation of a *pre-push* hook can check for the user's role and permissions and deny if the necessary rights in the respective branch are not specified.

[5] http://nvie.com/posts/a-successful-git-branching-model
[6] https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/permissions/permissions.md
[7] https://github.com/sitaramc/gitolite
[8] https://confluence.atlassian.com/display/BITBUCKET/Add+Users,+Set+Permissions,+and+Review+Account+Plans
[9] https://help.github.com/articles/permission-levels-for-an-organization-repository

Table 5.2: **Roles and their primary activities**. Stakeholders team may have roles, such as ontology engineers, domain experts, translators, and common users which can work on different types of issues.

Roles	Basic Activities	Semantic Issues	Structural Issues
Ontology Engineers	+	+	+
Domain Experts	+	-	-
Translators	+	-	-
Common Users	-	-	-

Smart commits allow the repository contributors to control the execution of specific actions, such as *commenting on issues* or *record timestamp* information against issues.[10] These actions are encoded into the commit message based on the following pattern:

$$< ignored\ message\ text >< ISSUE\_KEY >< ignored\ message\ text > \#$$
$$< COMMAND >< optional\ COMMAND\_ARGUMENTS >$$

According to this pattern, we integrated the idea of the smart commits into Git4Voc. This introduces a flexibility to the user on choosing only specific tasks to be performed and facilitates obtaining and monitoring of the information for later contribution analysis, according to requirement **M2**. The following pattern presents a command which is encoded in the commit message:

$$< commit\ message > \# < COMMAND > \# < optional\ ARGUMENTS >$$

where:

- ⟨*commit message*⟩ is the message given by the user, expressed in a free text format;
- ⟨*COMMAND*⟩ is the action to be performed, expressed with predefined keywords or particular range of numbers;
- ⟨*ARGUMENTS*⟩ are optional variables a user can add to invoke specific actions.

The available commands are as follows:

- ⟨*All*⟩ or ⟨1⟩ – perform all tasks listed in the following (default behavior);
- ⟨*SyntaxValidation*⟩ or ⟨2⟩ – check the ontology for syntactic errors;
- ⟨*CheckBadPractices*⟩ or ⟨3⟩ – check the ontology for bad modeling practices;
- ⟨*Normalization*⟩ or ⟨4⟩ – normalize the structure of the ontology;
- ⟨*GenerateDocumentation*⟩ or ⟨5⟩ – generate a documentation of the ontology;
- ⟨*SemanticDiffs*⟩ or ⟨6⟩ – list semantic differences between ontology versions.

For example, to perform a default syntax validation (without specific arguments and no other additional task), the user can append the number 2 or the keyword *SyntaxValidation* to the commit message:

$$\text{“}Error\ correction\#SyntaxValidation\text{”}\ \ or\ \ \text{“}Error\ correction\#2\text{”}$$

The message is captured according to the above pattern and a specific action will be performed the based on the attached command.

[10] https://confluence.atlassian.com/fisheye/using-smart-commits-298976812.html

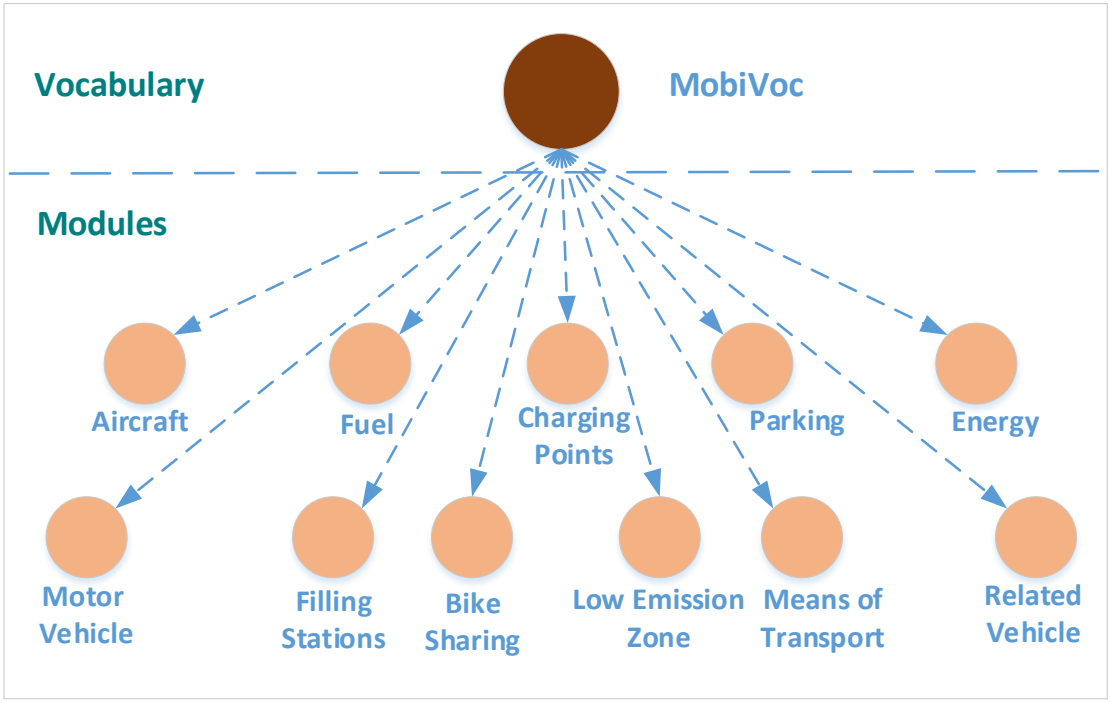


Figure 5.3: **The structure of MobiVoc**. The MobiVoc vocabulary comprises several modules divided according to their specific subdomain coverage.

Ontology Structure

The basic functionalities of Git do not support modularizing code or ontologies. Therefore, in order to address the requirement **M6**, we define several guidelines for organizing the ontology in files where each file represents a module. Considering the fact that each line should encode a triple and based on the insights of [90], we propose that files should not contain more than 300 triples. We highlight three possible forms for organization of the files. All of these cases are based on single Git repository to store the ontology files.

1. The complete ontology is contained in one single file. When the ontology is small (e.g. contains less than 300 lines of code) and represents a domain which cannot be divided in subdomains, it should be saved within one single file. If the number of contributors is relatively small and the domain of the ontology is very focused, organizing it into one single file might be possible, even if it exceeds 300 lines of code. However, if the comprehensibility is exacerbated, splitting it into different files should be considered.

2. The ontology is split in multiple files. If the ontology contains more than 300 lines of code or covers a complex domain, it should be organized into different subdomains or modules. In this regard, we mapped subdomains with modules. When the subdomains themselves are small enough, they should be represented by different files within the parent folder. There exists patterns for ontology modularization [91]. We developed the MobiVoc based on the pattern *n modules importing 1 module*. In this case, *1 module* was the ontology itself. The *n modules* like *Aircraft*, *Fuel* were saved in separate files. Each file represents a specific subdomain. By following this approach, domain experts can contribute independently to ontology development according

to their field of expertise. Figure 5.3 depicts the structure of MobiVoc and its modules.

3. Ontology modules are stored in files and folders. For huge ontologies that comprises complex domains, splitting it into files is not sufficient. This would lead to a large amount of files within a single folder. Therefore, if the subdomains are large enough to be split into files they should be represented by folders. Each folder contains files which are linked to modules. In this case, the folder and file structure should reflect the complex hierarchy of the overall domain. Splitting the ontology into files for specific purposes may facilitate requirements for role definition (M5) and multilinguality (M9). Associating roles to the specific files, such as *Translators*, facilitates the translation of the ontology terms into the required languages.

Labeling of Release Versions

Based on the requirement **M7**, proper labeling of release versions is vital, as it facilitates the reusability. One of the common ways to realize that, is to deploy each release version in a dedicated file. However, this could lead to the following problems as identified in [92]: 1) the number of files could increase rapidly; 2) choosing versions creates confusion; 3) maintenance needs additional resources; and 4) synchronizing with latest version from dependent applications requires additional effort. To avoid the above mentioned problems, we propose to keep versions of ontologies in the same file. These versions are separated by Git built-in functionality of tagging and are saved in the master branch, which is part of the branching model as illustrated in Figure 5.2. It is possible to create and filter tags at any time. Moreover, users can obtain a specific version of the ontology by just giving the tag name. Therefore, each released version of an ontology *must* have a version number. Based on the scheme from [93] and the mentioned categories of activities in Table 5.1, we propose tagging of different versions according to the following pattern: *v[StI.SeI.BaA]*, where *StI* stands for Structural Issues, *SeI* for Semantic Issues and *BaA* for Basic Activities. Each category is related with a number, in the respective position. Changes in the ontology regarding to the categories are commonly reflected by increasing the respective numbers. For instance, the difference between releases *v[1.0.0]* and *v[2.0.0]* indicates changes on the Structural Issues category (*StI*).

5.1.2 Development Practices

In this subsection, we provide a comprehensive list of practices for ontology development addressing the requirements: **M8** and **M9**. We derive this list from our own experience in creating ontologies, like *SCORVoc* and *MobiVoc*[11] in combination with the results of the analysis in Subsection 4.1.2 as well as with current state of the art practices. They will serve as guidelines focusing on the most important aspects of the ontology creation process. Moreover, these guidelines are independent of the concrete environment or tool used for development purposes.

Reuse

In the ontology construction field, the reuse of existing terms is of paramount importance [94, 95]. The main idea is to avoid creating new terms, but to utilize those that are present in the existing ontologies. Apart from saving time and investment costs, ontology reuse is expected to ensure a certain level of quality. The reason for this is that the longer an ontology exists and is reused, the more review processes it has gone through. Additionally, according to [87], the

[11] http://purl.org/eis/vocab/scor/, http://www.mobivoc.org/

reuse is considered to be a best-practice in ontology construction. Therefore, in the following we discuss important practices regarding reuse.

P-R1 Reuse of widely used ontologies We define authoritative ontologies as ontologies (cf. Table 4.2) which are: 1) published by renowned standardization organizations; 2) used widely in a large number of other ontologies; and 3) defined in a domain independent fashion, addressing more general concerns. Reusing authoritative ontologies increases the probability that data can be consumed by a wide range of applications [79].

P-R2 Reuse of non-authoritative ontologies Use ontology registries, like *LOV* and LODStats[12] or other more domain specific ontology registries, such as BioPortal[13] to find terms for reuse. For instance, by searching in LOV for a specific term the following information can be derived: 1) the number of datasets that use it; 2) the number of occurrences of the term in all datasets; and 3) the reuse frequency of the ontology to which the term belongs to [95]. Also, the semantic description and definition of the term should be checked to verify whether it fits the intended use. The above information supports the decision process regarding to which terms are positioned as better candidates for reusing.

P-R3 Avoid semantic clashes If the term has a *strong* semantic meaning for the domain, different from the existing ones, then a new element should be created.

P-R4 Reuse of individuals Especially elements from authoritative ontologies should be reused as individual ontology elements. For non-authoritative ontologies, a reuse of individual resources is less recommendable and the creation of own ontology elements with a possible alignment (cf. P-R6) or the reuse of larger modules (cf. P-R5) should be considered.

P-R5 Reuse of ontology module (Opposite of P-R4) Often ontologies require certain basic structures, such as addresses, persons, organizations, which are already defined in non-authoritative ontologies. Such structures comprise usually the definition of one or several classes and a number of properties. If the conceptualizations is matched, a whole module should be considered for reuse.

P-R6 Soft alignments with existing ontologies Instead of the strong semantic commitment of reusing identifiers from non-authoritative ontologies, additional alignments using axioms, such as `owl:sameAs`, `owl:equivalentClass`, `owl:equivalentProperty`, `rdfs:subClassOf`, `rdfs:subPropertyOf` can be established.

Utilization of SKOS Vocabulary The Simple Knowledge Organization System *SKOS* is a W3C recommendation for modeling taxonomies in the Web. *SKOS* is currently used by at least 478 ontologies [96]. The utilization of some *SKOS* constructs is considered as a best practice to declare and document the indexing terms (i.e., `skos:prefLabel`) and alternatives terms (i.e., `skos:altLabel`) [97, 98]. Both above mentioned properties are subproperties of `rdfs:label`. *SKOS* provides a more detailed notion of the labeling concept, which can be useful for a better descriptions of the ontology terms.

P-R7 Use `skos:prefLabel` to complement the labeling of concepts Add `skos:prefLabel` in combination with `rdfs:label` to provide a complementary semantic definition for each element. For instance, `skos:prefLabel` might be used to describe a shorter definition for a concept compared to `rdfs:label`.

[12] http://lov.okfn.org/dataset/lov/, http://lodstats.aksw.org/
[13] https://bioportal.bioontology.org/

P-R8 Use `skos:altLabel` to describe variations of the elements Add complementary descriptions for the elements, such as synonyms, acronyms, abbreviations, spelling variants, and irregular plural/singular forms by using `skos:altLabel`.

Authoring

In Subsection 4.1.2, we analyzed common practices followed by ontology engineers (i.e., the creation of object properties and their associated domain and range axioms). Those practices are always domain dependent, but still can serve as general guidelines to be followed in the process of designing ontologies.

P-A1 Domain and range definitions for properties When creating a property, consider to provide the associated domain and range definitions. This also means that in case of object properties, the corresponding classes should be defined. In case of datatype properties, the range should be a suitable datatype.

P-A2 Avoid inverse properties Create inverse properties only if it is strictly necessary to have bidirectional relations (i.e., `invalidates` and `isInvalidatedBy`). Inverse properties affect the size as well as the complexity of the ontology.

P-A3 Use of class disjointness Use class disjointness to logically avoid overlapping classes. Even though disjointness has been used in authoritative ontologies, it should be carefully examined since this can easily lead to semantic inconsistencies.

Documentation

Providing a user friendly view of ontologies for non-experts is crucial for integrating Semantic Web with everyday Web [99]. It facilitates contribution of domain experts during the development process. In addition, it helps the other interested parts for an easy adoption of ontology in the later phases as well. Basically, documentation generation tools require that following information should be present for each ontology resource to enable the representation in a user friendly view.

P-Do1 Use of rdfs:label and rdfs:comment Add an `rdfs:label` to every element, to define the main name of the concept that is being represented and an `rdfs:comment` to describe the context for which the element is created.

P-Do2 Generate human-readable documentation Easy-to-use documentation is critical for the wide adoption of the ontology. Use appropriate tools for documentation generation based on the types of URIs.

Naming Conventions

Following proper naming conventions has a high impact in ontology development [100]. Naming conventions help to avoid lexical inaccuracies and increase the robustness and exportability, particularly in cases when ontologies should be interlinked and aligned with each other [79]. Providing meaningful names increases the power of context-based text mining for automatic term recognition and facilitates manual and automated integration of terminological artifacts (i.e., comparison, checking, alignment and mapping) [100, 101].

Considering the literature on this topic [79, 102] and the results of Subsection 4.1.2, we define several practices to be followed in the process of naming elements in ontologies. For ontology construction, the use of the *CamelCase* notation is a considered as a best practice [103]. Our

study also indicates the presence of this notation in 62% of the cases. Therefore, we propose following notation forms to be used during ontology construction.

P-N1 Concepts as single nouns Name all ontology concepts as single nouns using CamelCase notation (i.e., *ChargingPoint*).

P-N2 Properties as verb senses Name all properties as verb senses in combination with CamelCase fashion. The name of a property should not be a plain noun phrase, in order to clearly distinct from class names (i.e., *hasProperty* or *isPropertyOf*).

P-N3 Short names Provide short and concise names for elements. When natural names contain more than three nouns, use the `rdfs:label` property for encoding long name while using a short name for the element. For instance, for *ManageSupplyChainBusinessRules* use *BusinessRules* and set the full name in the label. In order to explain the context (i.e., Supply Chain), complement this label with the `skos:altLabel`.

P-N4 Logical and short prefixes for namespaces Assign logical and short prefixes to namespaces, preferable no more than five letters (i.e., `foaf:XXX, skos:XXX`).

P-N5 Regular space as word delimiters for labeling elements For example, `rdfs:label "A Process that contains.."`.

P-N6 Avoid the use of conjunctions and words with ambiguous meanings Avoid names with "And", "Or", "Other", "Part", "Type", "Category", "Entity" and those related to datatypes like "Date" or "String".

P-N7 Use positive names Avoid using the negations. For instance, instead of *NoParkingAllowed*, use *ParkingForbidden*.

P-N8 Respect the names for registered products and company names In those cases is not recommended the use of CamelNotation. Instead, the name of the company or of a particular product should be used in the original form (i.e., SAP, Daimler AG).

Dereferenceability

One of the four rules to be followed during ontology development is naming things with HTTP URIs.[14] Adopting HTTP URIs for identifying things is appropriate due to the following reasons: 1) it is simple to create global unique keys in a decentralized fashion; and 2) the generated key is not used just as a name but also as an identifier.

By combining dereferenceability with content negotiation[15], it is possible to provide adequate content for a resource based on the type of request. There are three different strategies to make dereferenceable URIs of resources: 1) slash URIs; 2) hash URIs; and 3) a combination of both.[16]

P-D1 Use slash URIs If the client requests a resource from a server by specifying its URI, the server response will be *303 see other*. Slash URI should be used when dealing with large datasets to enable the server response with the only requested resource. For example, the ChargingPoint resource is identified as: *http://purl.org/net/mobivoc/ChargingPoint*. The URI of the turtle representation of above resource is *http://purl.org/net/mobivoc/ChargingPoint.ttl* and the URI of HTML representation is *http://purl.org/net/mobivoc/ChargingPoint.html*. To get information about ChargingPoint, the client gives its URI and specifies the request type. In turn, the server response will be *303 see other* by redirecting to the appropriate representation format.

[14] http://www.w3.org/DesignIssues/LinkedData.html
[15] http://www.w3.org/Protocols/rfc2616/rfc2616-sec12.html
[16] http://www.w3.org/TR/cooluris

P-D2 Use hash URIs This solution is formed by including a fragment to the URIs as in the following format *URI#resource*. Use hash URIs when dealing with small datasets to reduce number of HTTP round trips. For instance, the URI of the *ScorVoc* ontology is *http://purl.org/eis/vocab/scor* whereas the URI of the *Process* resource is *http://purl.org/eis/vocab/scor#Process*.

P-D3 Use combination between slash and hash URIs This allows a large dataset to be split into multiple fractions. It is appropriate for large datasets where it is not practical to serve all resources in single document(e.g., *http://purl.org/eis/vocab/scor/Process#this*).

Multilinguality

Providing multilingual ontologies is desirable but not a straightforward issue [84]. According to our empirical analysis in Subsection 4.1.2 and with the aim to keep things simple, we propose the following practices for multilinguality support.

P-M1 Use English as the main language Use English for every element and explicitly encode the resource definition using the *@en* notation.

P-M2 Encoding other languages In order to add a new language, use another triple according to the same format for every element. The following example illustrates this practice with translations for the *SupplyChain* class.

```
scor:SupplyChain rdf:type owl:Class ;
    rdfs:label   "SupplyChain"@en;
    rdfs:comment "A Supply Chain is a ..."@en ;
    rdfs:label   "Lieferkette"@de;
    rdfs:comment "Eine Lieferkette ist ..."@de.
```

This approach should be applied to all elements starting from the most basics ones, like `rdfs:label` and `rdfs:comment` as well as external annotation properties (e.g., `skos:prefLabel`).

Validation

Validation is an important aspect in the ontology development process [50]. It analyzes whether ontology correctly represents the knowledge domain in accordance to the user requirements and best practices for ontology modeling [10, 104]. Criteria used for validation activity are: 1) correctness; 2) completeness; and 3) consistency [39]. With the purpose of addressing the above mentioned criteria, we propose the following practices:

P-V1 Syntax validation When collaborating directly on the ontology source code, syntax checking is of paramount importance. Ideally, syntax checking is directly integrated into the editor and committing the code with errors is not possible. For example, tools like *Rapper*[17] or Web-based services, such as the *RDF Validation Service* or *OWL2 Validator*[18] can be used for finding common typos and syntax errors.

P-V2 Code-Smell checking Code smells are symptoms in the software source code that possibly indicate deeper problems. Similarly tools, such as *OOPS* can be used for ontology smell checking. OOPS is a Web-based tool for detecting common ontology pitfalls, such as: 1) missing relationships; 2) using incorrectly ontology elements; and 3) missing domain and range properties. The complete list of pitfalls detected by OOPS is presented in [50].

[17] http://librdf.org/raptor

[18] http://www.w3.org/RDF/Validator/, http://mowl-power.cs.man.ac.uk:8080/validator/

P-V3 Consistency checking Since we deal with lightweight ontologies it is not very likely to have axioms that produce semantic inconsistencies. Nevertheless, our analysis in Subsection 4.1.2 showed that in authoritative ontologies there are cases that lead to semantic inconsistencies (i.e., class disjointness). Handling inconsistencies impacts the quality of ontologies [105]. Tools, like *Pellet* or *Fact++*[19] should be used for consistency checking.

P-V4 Linked Data validation Tools, such as *Vapour* verify whether data are correctly published according Linked Data principles and the best publishing practices vocabularies.[20]

5.2 Evaluation

The main goal of the evaluation is to assess the applicability of the defined governance aspects to a well-know vocabulary. In addition, a survey with ontology engineers is conducted to analyze their opinion regarding to the presented operational practices for ontology modeling.

5.2.1 Schema.org Use Case

We evaluated our approach against *Schema.org*, which is one of the most well-known vocabulary at the moment. It is a collaborative initiative for creating, promoting and maintaining schemes for structured data on the internet, like web pages, and email messages. Among the companies involved in this initiative are Google, Microsoft, Yahoo and Yandex, which contribute to the vocabulary development process and apply the defined concepts to their web resources. According to the latest version from its repository[21], statistics of Schema.org are: 767 classes, 1190 properties and 273 instances, while these numbers are continuously increasing. The vocabulary is hosted and managed through Git and the GitHub platform. Basic Git functionalities, such as versioning, issue tracking, and branching, are used to organize the development process. Publication of the new versions and human-friendly representation is powered by a software application which is based on the *Google App Engine* and written in Python programming language. Since the development process is organized through Git and GitHub, we have identified Schema.org as an ideal case for analyzing how the application of our approach could improve the development and deployment process. In the following, we detail our analysis and show our proposed solution according to the governing aspects defined in the previous section.

Management of Generated Information

Using the *issue tracker* mechanism of GitHub, the community of Schema.org contributes actively in the development process by proposing new terms, or modifications to existing ones according to their needs. A total number of 1370 issues have been created so far, where 591 of them are already closed and 779 remain open. In order to make a clear grouping of issues based on their intention, 21 different labels, such as: 1) *schema.org vocab*; 2) *type:exact proposal*; 3) *site tools + python code*; and 4) *type:bug*, have been created. This categorization helps the users to easily find information related to the vocabulary changes or dedicated to the publishing software. However, 665 issues do not have any associated label, which makes it difficult to retrieve and categorize them for further analysis. Furthermore, bugs of the Schema.org software are announced using this mechanism as well. Other issues related to specific version can be found only on release

[19] http://clarkparsia.com/pellet, http://hermit-reasoner.com/
[20] http://validator.linkeddata.org/vapour, http://http://www.w3.org/TR/swbp-vocab-pub/
[21] https://github.com/schemaorg/schemaorg

notes of the Schema.org website[22], since no dedicated label exists for this purpose. Therefore, in order to find information related to the specific version which the issue pertains, we propose creating additional labels with specific purposes, such as: 1) *version dedicated*, represents issues according to the version number; 2) *module dedicated*, groups issues based on the module; 3) *extensions dedicated*, groups issues related to the extensions of the current concepts, etc.

Repository Organization Structure

The Schema.org repository contains both, the vocabulary and the publishing software. According to software development best practices, a project (which is comparable to a repository) should contain either software or data but not both at the same time. Thus, we propose creating two separate repositories, one for the vocabulary and the other one for the publishing software. By doing so, contributors and users of the vocabulary, which are our main target groups, can easily participate in the development process and any potential confusion can be avoided.

Further exploration of the repository, we see that the *data* folder comprises other folders and files related to different versions of the vocabulary. Within this folder, the main file is *schema.rdfa*, which contains a complete representation of the vocabulary in RDFa format. In order to simplify the contribution activity and avoid forcing users from opening each time the whole vocabulary, we propose splitting it in multiple files and multiple folders. We highlight three possible forms of organizing the Schema.org vocabulary:

1. Types of the concepts. The vocabulary may be split based on the types of the concepts, such as *classes*, *object properties*, *datatype properties* and *instances*. As a result, the repository will contain four different files, where each file represents one of the above concept types. In case that during the development process a new type of concept is introduced, a dedicated file associated with it should be created.

2. The number of lines/concepts per file. According to the above vocabulary statistics, the current version of the Schema.org comprises a total number of 2230 concepts. Since this number is already relatively high for a vocabulary and will increase over time, the vocabulary can be split based on the number of concepts. As a result, the repository could, for instance, consist of six files, where each file contains a bit more than 300 concepts.

3. Hierarchy of concepts. Another way of splitting Schema.org is based on the hierarchy of the concepts. Key concepts in the vocabulary hierarchy are: *Action*, *Creative Work*, *Event*, *Intangible*, *Medical Entity*, *Organization*, *Person*, *Place* and *Product*. These concepts can be saved into separate files. If the number of sub-concepts is high, the vocabulary can additionally be split into folders, where each folder represents one of the above concepts and the files within it represent the sub-concepts. In addition to the main concepts, Schema.org contains another concept called *Data Type*. This concept should be saved in a separate file. Figure 5.4 depicts a potential modularization structure of Schema.org and its concepts.

Branching Strategy and Release of the Versions

The development process on Schema.org is organized using different branches. The repository contains 114 branches named according to various styles: 1) from the combination of *sdo* prefix

[22] http://schema.org/docs/releases.html

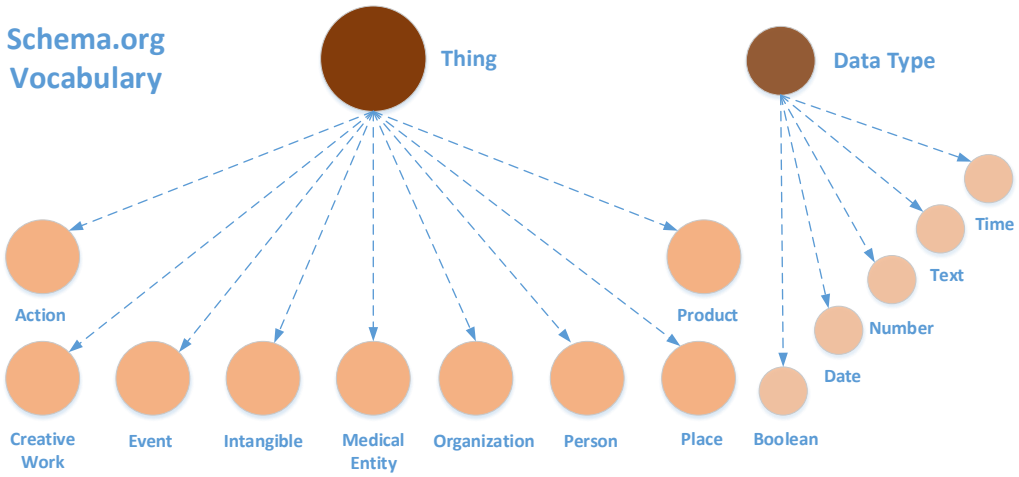


Figure 5.4: **Proposed modularization structure of Schema.org**. The structure of Schema.org can be organized into different sub-modules based on the key concepts as well as *Data Type* concept.

and one of the character's name in [23]; 2) the name of the contributor; 3) the number of issue that has been fixed; and 4) the name of the release version. In addition, several branches are named based on the name of the software feature, or new tools that have been experimented.

To align with our defined branching strategy and the activities presented in the Table 5.1, we propose the organization of the development process according to the following branches: 1) *Basic Activities*; 2) *Semantic Issues*; 3) *Structural Issues*; and 4) *Release*. This simplifies the involvement of the contributors with different roles, such as: 1) *ontology engineers*, which contribute in the major fixes and structural issues; and 2) *domain experts*, which contribute in adding, modifying or deleting the vocabulary concepts and fixing minor issues. Furthermore, temporal branches dedicated to the testing of specific versions of the vocabulary can be created. Periodically, all unnecessary branches should be removed from the repository. As a result, the development process will have a clear path and the confusion created by high number of branches and different conventions used for their names, will be avoided.

Following the approach for defining the release versions will result on a very convenient numbering pattern. Each change made on the branches results on increasing the number in the respective position of the following pattern: v[StI.SeI.BaA]. For example, if the current version of the vocabulary is v[1.2.4] and changes are made on the *Basic Activities* branch then *BaA* position is increased and the next version will be v[1.2.5].

5.2.2 Survey and Result Discussion

With the goal to validate the defined practices, we perform a survey with ontology engineers.[24] The experience in the selected group is as follows, 58% have up to two years and 41% from two to five years. The *Likert Scale* [106] was used to collect the opinions. Figure 5.5 depicts the results of the survey. Generally, all practices have received good evaluations regarding to the opinion of experts. The authoring aspect was the most controversial one. The practice **P-A2**, received some negative scores due to the existing debate regarding the use of inverse properties.[25] The

[23] https://en.wikipedia.org/wiki/Ghostbusters#Cast
[24] https://goo.gl/X8otxe
[25] https://lists.w3.org/Archives/Public/public-vocabs/2014Apr/0200.html

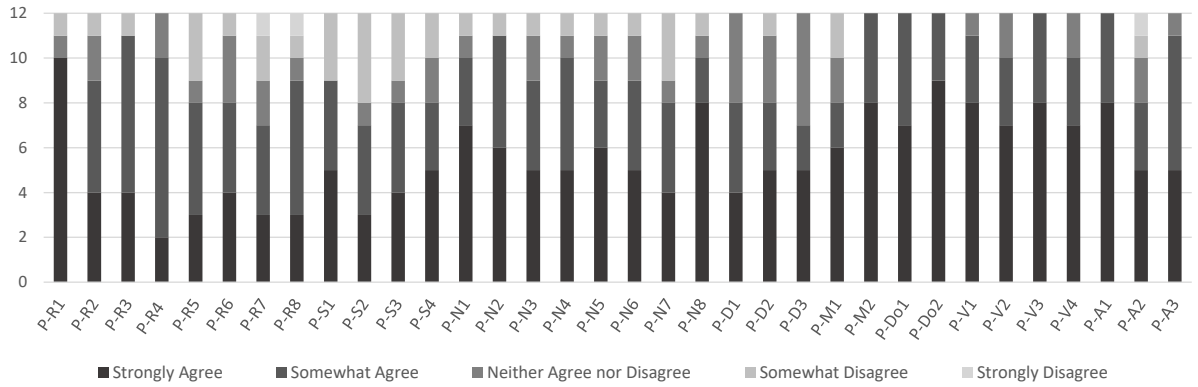


Figure 5.5: **Results of the survey with ontology engineers**. Horizontal axis lists proposed practices whereas vertical axis shows opinions of participants for each particular practice, respectively.

results of **P-A4, P-A5** show that even *SKOS* as a generally accepted standard still is not well received for a certain group of ontology engineers.

5.3 Summary

In this chapter, we present Git4Voc, a lightweight methodology for supporting ontology development in distributed environments. The overall goal is to pave the way for a new paradigm of ontology development similar to Software Development by Convention. It aims at reducing the number of decisions that developers need to make during the modeling process. We started with investigating the applicability of Git for collaborative ontology development. As a result, Git4Voc is conceived following a bottom-up approach by exploiting fundamental features of Git in combination with the state of the art practices. Several governing aspects are defined, such as: 1) management of generated information; 2) rights management; 3) branching and merging; 4) ontology modularization; and 5) labeling of release versions, to facilitate ontology construction using version control. Moreover, from our analysis in Subsection 4.1.2 and the examination of the state of the art, we define a set of practices related to reuse, naming convention, documentation, authoring, validation and multilinguality.

We evaluated our governing aspects against the Schema.org vocabulary, a community-driven initiative to enable creation, maintaining, and promoting structured data on the Web. As a result, the repository of Schema.org will have a clean structure, number of branches will be reduced, and issues will be easy reachable based on a clear categorization. Considering the vocabulary structure, Schema.org would have been divided into several submodules according to the basic concepts. In addition, the repository will reside only the vocabulary, whereas the tools that powers it, will be moved to another dedicated repository. On the other hand, to asses the relevance of the defined practices, we conduct a survey with experts. Generally, according to the opinion of ontology engineers, the majority of the practices received good evaluations. Certain practices, such as *P-A2*, *P-A4* and *P-A5* received some contradictory opinions due existing debate for the reuse of inverse properties or SKOS labeling axioms.

CHAPTER 6

An Integrated Environment for Collaborative Ontology Development

This chapter presents VoCol, an integrated environment that supports the development of ontologies centered version control systems. We designed *VoCol* as a comprehensive and flexible approach for realizing a fully-featured development platform. VoCol supports a fundamental round-trip model of ontology development, consisting of the three core activities *modeling*, *population*, and *testing*. In the spirit of test-driven software engineering, VoCol allows to formulate queries which represent competency questions for testing the expressivity and applicability of a ontology *a priori*. For *a posteriori* testing, it supports the automatic detection of "bad smells" in the ontology design by employing SPARQL patterns. For modeling, VoCol integrates a number of techniques facilitating the conceptual work, such as automatically generate the documentations and visualizations, thus providing different views on the ontology as well as an evolution timeline supporting traceability. For population, VoCol supports the integration of mappings between data sources (e.g., R2RML mappings to relational databases) and the ontology. The governance of distributed ontology development is supported by the access control as well as the branching and merging mechanisms of the underlying VCS.

VoCol bridges between the *conceptual* development of ontologies and the *operational* execution in a concrete IT landscape. It has been implemented following the principles for *extensibility*, *interoperability* and *customisability*. The VoCol platform is based on a loose coupling, leveraging the webhook mechanism provided by many VCSs with tools and techniques addressing particular aspects of ontology development. The modular architecture allows the platform to be extended by adding or exchanging components or services. New built-in components can be implemented or *off-the-shelf* components can be plugged-in to an orchestration layer. Users are able to *customize* VoCol by (de)activating a subset of components or services based on their specific needs or scenario. By bundling all tools and encapsulating dependencies into virtual environments, such as Vagrant and Docker containers, VoCol is an *interoperable* platform easy to be deployed or used as-a-service in conjunction with arbitrary VCS installations. Furthermore, an incorporated component ensures the *unique serialization* of ontology changes which can be modeled using heterogeneous editors to prevent false-positive conflicts. We demonstrate the applicability of VoCol with a real-world example from the industry domain and report on a user study that confirms its usability and usefulness.

In this chapter, we address the following research question:

> **RQ2**: How can ontology development workflows be mapped on and supported by distributed version control methods?

Contributions of this chapter are summarized as follows:

- An integrated and semi-automatic environment based on a modular architecture to enable ontology construction in distributed scenarios;
- An implementation of defined architecture composed of several services and components to address specific requirements;
- An evaluation of the approach with a use case from an industry partner, where the development process is driven using VoCol;
- A user study with ontology engineers to assess their opinion regarding the usefulness and usability of the integrated services.

This chapter is based on the following publication:

- **Lavdim Halilaj**, Niklas Petersen, Irlán Grangel-González, Christoph Lange, Sören Auer, Gökhan Coskun, Steffen Lohmann. *VoCol: An Integrated Environment to Support Version-Controlled Vocabulary Development.* In 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW) 2016 Proceedings, 303-319, Springer. This article is a joint work with Niklas Petersen and Irlán Grangel-González, PhD students at the University of Bonn. In this article, I conducted the problem description, definition and implementation of the conceptual architecture, the revision of the state of the art approaches, the presentation of the use cases, and the realization of the user study evaluation.

The remainder of this chapter is structured as follows: Section 6.1 describes the approach and a conceptual architecture composed of several services and components. In Section 6.2, we provide an implementation of the defined architecture centered around Git as the core component. We then demonstrate the practical application of VoCol in a real world scenario in Subsection 6.3.1. We conduct a survey with ontology engineers in Subsection 6.3.2 to asses the usefulness of the provided services. In the end, the work is summarized in Section 6.4.

6.1 The VoCol Approach

In order to implement VoCol as an integrated environment, we initially designed the system architecture, as illustrated in Figure 6.1. It follows the principles of *Component Based Software Development* (CBSD) [107], which promotes the reuse of components to develop large-scale systems. In other words, it advocates selecting the appropriate *off-the-shelf* components and assembling them into a well-defined software architecture. Following this idea, we compose VoCol from a set of smaller components according to the functionalities they provide. Each of these components is exchangeable and can be replaced by alternatives. In the following, these components and their characteristics are described in detail.

Version Control System A VCS component is required for the *management of the ontology changes*, such as *change capturing* and *change propagation*. According to [29], change manipulation includes supplementary methods and strategies that support the management of changes in

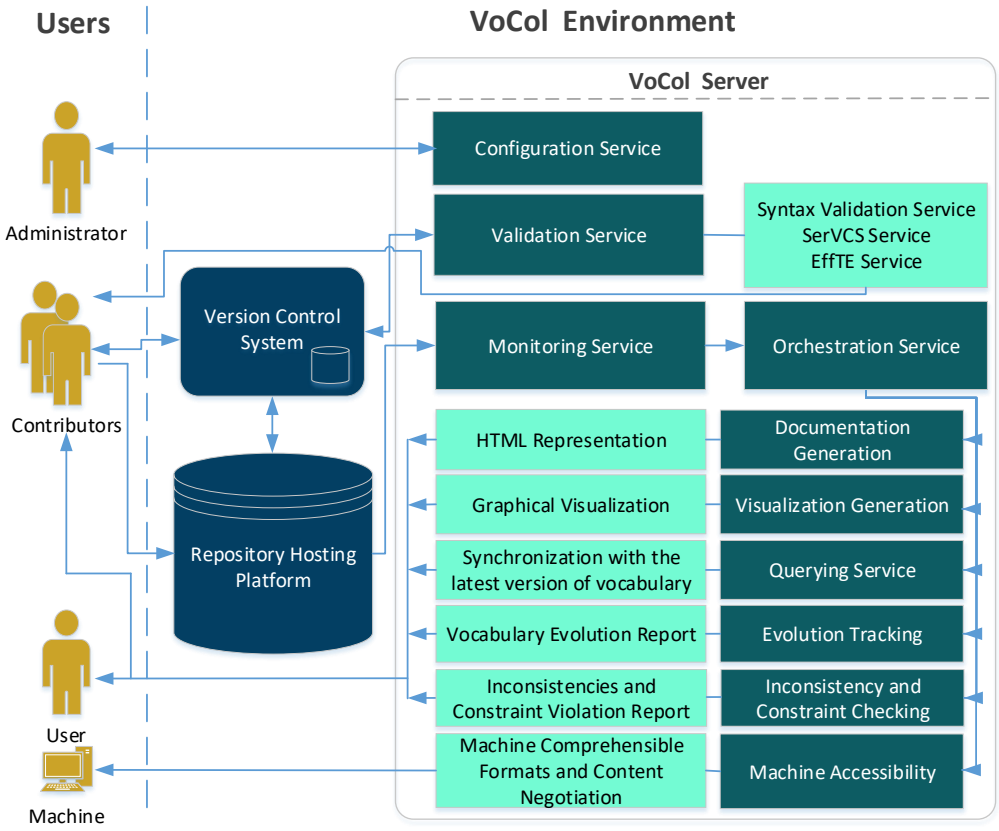


Figure 6.1: **The VoCol Architecture**. Main services of VoCol: *Configuration Service*, *Validation Service*, *Monitoring Service* and *Orchestration Service* with their respective components as well as the interactions between services and components.

distributed environments. By capturing and storing the changes, various revisions of the same ontology are created. Viewing the syntactic differences is realized by computing the changesets between different revisions of the ontology file. During change the synchronization process several conflicts may be detected as a result of syntactic differences, forcing the user to resolve conflict before submitting her local changes, thus addressing the requirement **P1**. To ensure a consistent development process, any change that is made should be propagated to all other contributors. A distributed VCS enable contributors to work collaboratively, without the need of sharing a common network or the necessity of being always online addressing the requirement **P6**. In addition, conflicts inevitably arise in environments where multiple contributors are working simultaneously and changing ontology terms. The VCS detects conflicting changes by employing built-in algorithms and ensures conflict resolution. It allows for the integration of these conflicts in an effective and easy way, according to the requirement **P1**.

Since the VCS is the first component that is aware of changes, we declared it to be the core component of the overall VoCol system. Each additional component that is necessary to support ontology development is triggered by the VCS. We also integrated a hosting platform into the VoCol environment (cf. Figure 6.1), as it provides a low-threshold access for involved stakeholders. It acts as the repository storage where the ontology files are saved and accessed.

Its feature for *Access Control* authenticates users and outputs a permit or a deny message according to the set of permissions. Furthermore, using the *Issue Tracker* of the repository hosting platform, contributors are able to discuss about the ontology by proposing new terms or alternatives for existing ones. In cases where sensitive information should be transmitted, the repository hosting platform can deliver *email notifications* to private user accounts.

Integrated Validation Service With the objective of addressing the requirement **P4**, such that any change submitted to the repository is correct with respect to syntax, consistency and do not violate pre-defined constraints, a dedicated service is integrated into VoCol. This service comprises the following components:

Syntax Validation To ensure that the latest revision of the ontology in the remote repository is always syntactically correct, VoCol integrates a syntax validation component. In principle, syntax validation could be executed at different stages of the overall workflow. However, with the aim to keep the requirements on the client side at a minimum level, we integrated the syntax validation as a service in the backend. It rejects syntactically incorrect commits and provides a detailed error report in those cases.

Unique Serialization In a distributed environment, contributors use different editors during the development process which may produce various structures of ontology files. To avoid this problem, a component integrated into VoCol creates a unique serialization of ontology terms before the changes are pushed to the remote repository. Thus, the VCS is prevented from indicating *false-positive* conflicts, making it editor agnostic as required in **P3**.

Inconsistency and Constraint Checking After the changes have been pushed to the remote repository, validations of semantic inconsistencies and constraints violation are performed. As a result, two reports with detailed information on respective findings are generated and can be used for corrections addressing the requirement **P5**.

Monitoring Service Repository hosting platforms typically expose most of their functionality via web service APIs, so that it can be controlled programmatically. Any change to the repository is delivered as a payload event to this service which is listening on VoCol server. As a consequence, the other services for performing specific tasks are automatically invoked.

Orchestration Service Is responsible for coordinating the execution of particular components according to a pre-defined workflow. The number of the components to be invoked may vary depending on their activation status. This service is triggered by *Monitoring Service* after occurrence of a push event. Following components are integral part of this service:

Documentation Generation With the objective of addressing the requirement **P7**, a documentation generation component creates an HTML representation of the ontology. This permits contributors to easily navigate through the ontology by providing a human-friendly overview of it. Moreover, contributors are able to explore annotations of the modeled concepts into a number of different languages.

Visualization Generation The integrated visualization component depicts the ontology terms and their connections in a graphical way, and allows an interactive navigation over the modeled concepts. It complements the generated documentation by particularly representing the structure, distribution, and relationships within the ontology. By graphically

depicting terms, such as classes, properties and their connections, a better view of general structure is provided. Consequently, users are empowered with a coherent view of the ontology according to the requirement **P8**.

Evolution Tracking The VCS takes care of maintaining the revision history of the files. To detect *semantic* differences between ontology versions, an *evolution tracking* component is integrated into VoCol. It shows which classes and properties have been added, removed, or modified, enabling users to see the evolution history over the time as required in **P2**.

Querying Component VoCol integrates a SPARQL endpoint synchronized with the latest version of the ontology to address the requirement **P10**. Queries derived from competency questions can be used to verify whether the ontology meet the domain requirements. These queries are stored in the repository and are pre-loaded in the query interface.

Machine Accessibility Using *content negotiation* and dereferenceable URIs, VoCol delivers various machine-comprehensible representations. By specifying the content type in the HTTP header along with the resource URI, the ontology can be accessed by different software agents compliant with Linked Data principles[1] addressing the requirement **P9**.

Configuration Service This service provides a *graphical user interface* to facilitate the configuration of VoCol. The system administrator can choose from various tools for syntax validation and documentation generation. Furthermore, the other components can be activated or deactivated simply by selecting the corresponding checkboxes.

6.1.1 Contributor Workflow

The interaction of the contributors with the VoCol environment starts with cloning the repository in her local machine. Next, the user is able to add, modify or delete ontology concepts according to her assignments. To store her work, she commits changes to local repository. During the commit phase, changes are validated for the syntax and inconsistency errors as well as the violation of pre-defined constraints. If the validation process is passed, a unique serialization of the ontology is generated and changes are successfully committed. To make her changes available, the user continue with synchronization of the local ontology with the *Remote Hosting Platform*. After invoking the *push* command, the hosting platform delivers a notification to the *Monitoring Service*. This activates the *Validation Service* to perform the server-side checking of the recent changes. Next, the *Orchestration Service* is triggered and a number of specific tasks, such as documentation generation, and visualization are initiated for the execution. The generated artifacts are served to the user through the *Presentation Service*. Figure 6.2 illustrates interaction of a contributor with VoCol and sequence of the steps that are performed.

6.2 Implementation

We implemented *VoCol backend* using *Node.js*[2], as an interoperable cross-platform programming language along with *Express.js*[3], which is a flexible web application framework dedicated for *Node.js*. It manages any request that comes from the user or software application and based on that, specific actions are performed. VoCol backend comprises a set four main services: *Configuration Service*, *Monitoring Service*, *Orchestration Service* and *Validation Service*. These services

[1] https://www.w3.org/DesignIssues/LinkedData.html
[2] https://nodejs.org/
[3] https://expressjs.com/

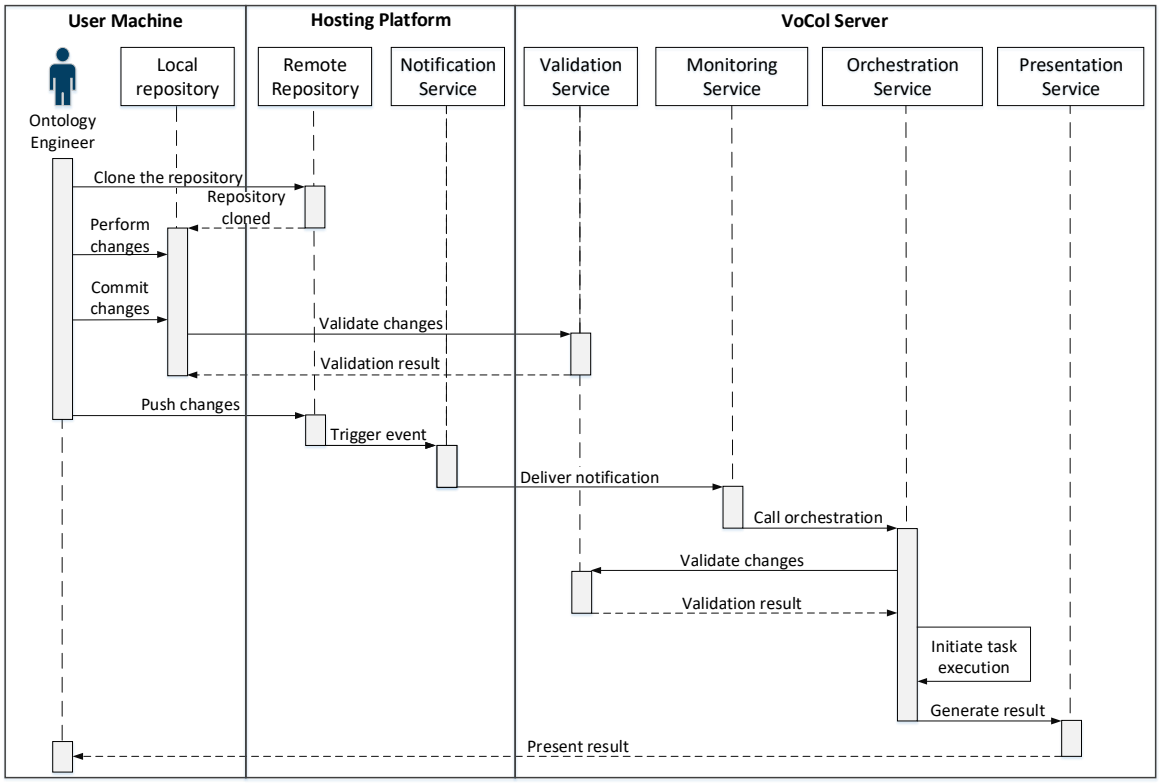


Figure 6.2: **Contributor Workflow**. Sequence of steps and events occurring in a typical development workflow of a contributor during the interaction with the VoCol environment.

are responsible for providing functionalities for syntax validation, visualization, documentation generation, evolution depiction, and querying.

6.2.1 Configuration

This service is developed to allow the utilization of VoCol for different application scenarios. It enables the system administrator to configure VoCol by entering the details of the ontology repository (e.g., repository URL, and user credentials) in a graphical user interface (GUI) (cf. Figure 6.3, *General Info* and *Repository Info* sections). The administrator defines the main branch of the repository by entering the value in the *Branch Name* field. For this branch, all selected services will be provided by VoCol. Via checkboxes within the *Additional Services* section, services for visualization, evolution report, and querying, can be selected for automatic execution by VoCol. Furthermore, by checking the *Turtle Editor* option, a tool that allow online editing of *Turtle* files and has direct synchronization with GitHub repository [108], is added into the VoCol environment. The option *Predefined Queries* indicates that queries defined in files with the extension *.rq* are automatically loaded into the SPARQL interface. Next, VoCol can be configured to run on *public mode* where everyone who has the link can access and explore the provided content or it can be restricted to *private mode*, which allows access only to the people with the given credentials. Finally, all serialization formats that VoCol should deliver via content negotiation, can be selected.

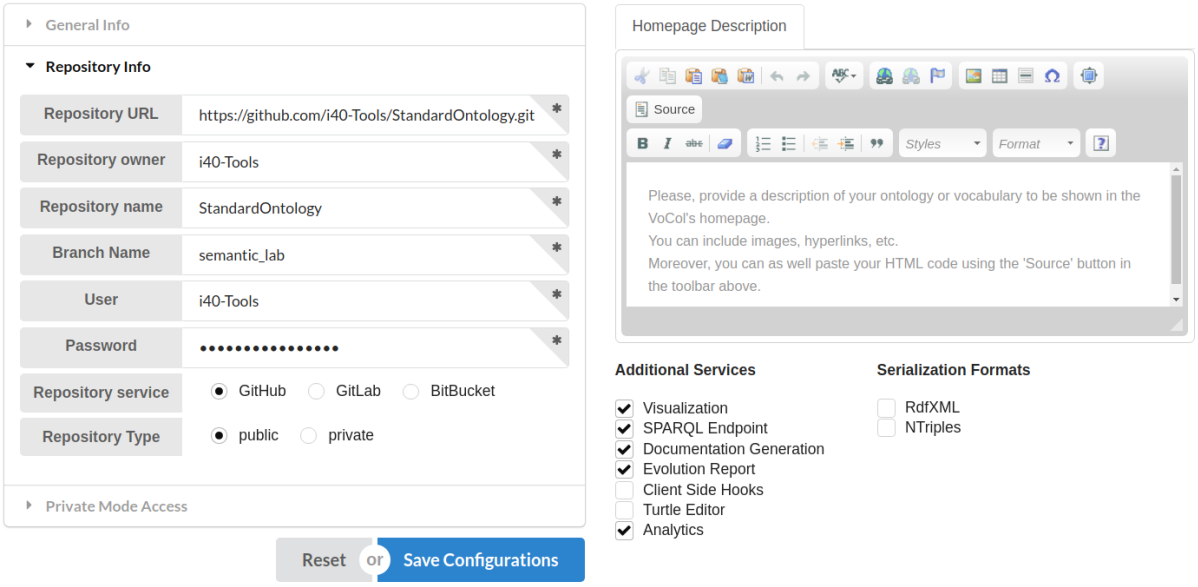


Figure 6.3: **The VoCol Configuration Page**. The configuration page contains several sections: 1) *General Info*; 2) *Repository Info*; 3) *Private Mode Access*; 4) *Additional Services*; and 5) *Serialization Formats*. Details about the ontology project can be provided in the *Homepage Description* section. This page allows for the administrating of the VoCol platform for different scenarios.

VoCol is able to recognize the used hosting platform (e.g., GitHub, Bitbucket) based on the URL entered for the ontology repository, and automatically access the respective API to create a *webhook*. This hook contains the address of the VoCol server to which the repository hosting platform will henceforth send information about any push event.

6.2.2 Client-side Tasks

Client-side tasks refer to the tasks performed before pushing to the remote repository, including a number of specific tasks realized after invocation of the push command.

Integrated Validation Service is responsible for validating the syntax, checking for fulfillment of requirements via SPARQL queries and generating a unique serialization.

To reduce the efforts needed for the subsequent corrections, VoCol validates the syntax before pushing the changed files to the repository. An adapted *pre-commit* hook (cf. Listing 6.1) posts the ontology files that have been changed with tools like *Protégé* or *TopBraid Composer*[4], from the user machine to the VoCol server. First, the server validates the ontology files for syntactic errors. If the validation fails, the user receives a detailed error description, including the file name, the affected lines in the files, and the type of the error. If the syntax validation succeeds, a unique serialization of the ontology files is created using the SerVCS component that we implemented on top of the RDF serialization tools, like Rdf-toolkit or Rapper[5]. As a result, the ontology elements are serialized in an alphabetic order, which reduces the number of false-positive conflicts indicated by the VCS during the merging process. Additionally, the integrated *TurtleEditor*

[4] http://protege.stanford.edu , http://www.topquadrant.com/composer/.
[5] https://github.com/edmcouncil/rdf-toolkit, http://librdf.org/raptor/

can be used to edit the ontology files directly on the repository hosting platform. Following the idea of a *just-in-time debugger*, this editor implements an instant validator that immediately reports on all found syntax errors. Furthermore, it provides auto-completion of ontology terms according to the declared namespaces.

```
#!/bin/sh
#
  files=$(git diff --cached --name-only --diff-filter=ACM | grep ".ttl$")
  if [ "$files" = "" ]; then
      exit 0
  fi
  for file in ${files}; do
            fileContent=`cat ${file}`
            fileHeader="${file}StringToSplit"
            fileContent="${fileHeader}${fileContent}"
            echo " ${fileContent} " > tempFileCurl.ttl

      res=$(curl -X POST --data-binary @tempFile.ttl http://vocol/client)
            code="$?"
            if [ "$code" = "6" ] || [ "$code" = "7" ]; then
                echo "Connection refused or service not available";
                pass=false
            elif [ "$code" = "0" ]; then
              if echo $res | grep -q "Service Temporarily Unavailable"; then
                  echo "Connection refused or service not available";
                  pass=false
              elif echo $res | grep -q "Undefined error"; then
                  message="See the above messages!"; pass=false
              elif echo $res | grep -q "Error"; then
                  echo "Validation Failed! ${file}"
                  echo $res; pass=false
              elif echo $res | grep -q "Syntax Validation"; then
               echo "Validation Passed! ${file}"
              elif echo $res | grep -q "prefix"; then
                  echo "Validation passed and the ${file} file reformatted"
                  echo "$res" > ${file}
              fi
            else
                echo "Undefined error. Contact the administrator!"; pass=false
            fi

            rm -f tempFileCurl.ttl
  done
        git update-index --again

  echo "\Validation complete\n"

  if ! $pass; then
      echo "COMMIT FAILED: Syntactic errors found.\n"
      echo ${file}
      exit 1
  else
      echo "COMMIT SUCCEEDED"
  fi
```

Listing 6.1: **Pre-commit hook**. A customized pre-commit hook to submit the local changes to the VoCol server for syntax validation, checking for bad modeling practices and generating a unified serialization.

6.2.3 Server-side Tasks

Server-side tasks refer to the tasks related to the validation and publication of artifacts in human and machine-comprehensible formats, performed after occurrence of each Git push event.

Triggering Changes on the Repository Using the *PubSubHubbub* protocol[6], after each push event, the repository hosting platform delivers a payload with information about the last commit(s) to a server subscribed to it. The *Monitoring Service*, implemented in VoCol, receives the payload and pulls the ontology from the remote repository.

Validation and Error Reporting Next, the IVS is triggered to validate each file for syntax errors using tools like *Rapper* or *Jena Riot*[7]. This task is rerun on the server side to avoid further processing of ontologies with syntax errors, which can happen if users do not validate the syntax during the commit phase. If the validation fails, an HTML document is created with detailed information about the found errors.

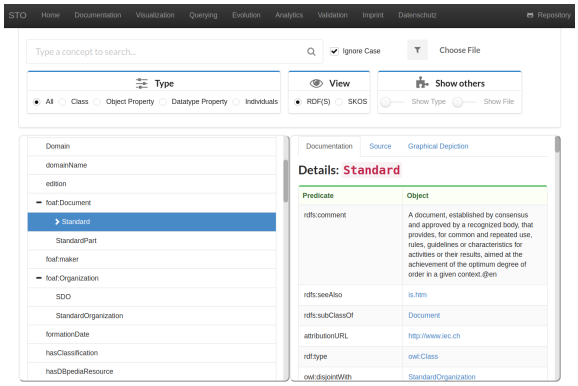
Publishing the Artifacts for Humans and Machines If the syntax validation process is passed successfully, all ontology files are merged into a single file. After that, the following tasks are performed automatically to generate updated artifacts for the evolution report, documentation, and visualization.

Documentation Generation: A dedicated component is responsible for generating human-friendly documentation of the ontology. It comprises rich features to facilitate exploration and understanding of the ontology, its taxonomy and details of each concept. Users can quickly reach any concept by typing its name in the *search bar*, which causes the tree to be updated with a list of concepts containing the given letters. A number of different filtering capabilities enable users to show concepts that belong to a specific file and filter them according to the type, e.g. *classes*, *properties* or *individuals* as depicted in Figure 6.4(a). After selecting a particular concept, all associated metadata, object and datatype properties as well as its instances are displayed in a tabular format of *key-value* pairs. Figure 6.4(b) illustrates various representation formats, such as *RDF/XML*, *Turtle*, *N3* and *JSON*, which are provided in the *Source* tab of the selected concept. In addition, users may view its *graphical depiction* as shown in Figure 6.4(c), where the selected concept is the *root* node surrounded by many other nodes connected through different properties. The *single page per concept* feature, as illustrated in Figure 6.4(d), enables deferenceability for each term defined in the ontology. Furthermore, following the principle of *RDF Molecule Template* [109], defined as a set of triples sharing the same subject, a comprehensive view of the selected concept including its metadata, relationships with other concepts and its instances is provided. Statistics of the ontology comprising number of classes, properties, individuals can be illustrated using different *graphical charts* as shown in Figure 6.5(c).

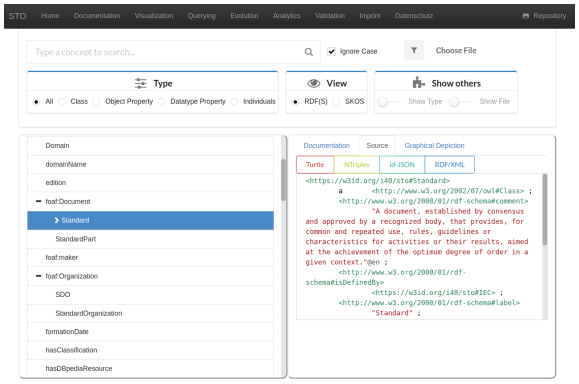
Visualization Generation: The visual depiction of the entire ontology is realized using the *WebVOWL* [110] tool. WebVOWL is integrated as a component to allow an interactive visualization of ontologies. It implements the *Visual Notation for OWL Ontologies* (VOWL) to graphically represent the ontology concepts and their relations in a dynamic layout according to a force-directed graph fashion. An excerpt of a generated visualization is shown in Figure 6.5(a). WebVOWL has many built-in features, enabling users to *search* for concepts, filter based on the concept type, e.g., *class disjointness*, or *object or datatype properties*. The layout can be exported in various formats, such as *JSON* or *SVG*.

[6] https://pubsubhubbub.appspot.com

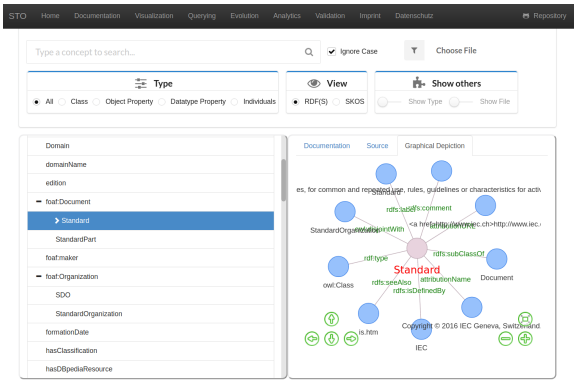
[7] http://librdf.org/raptor/, https://jena.apache.org/documentation/io/



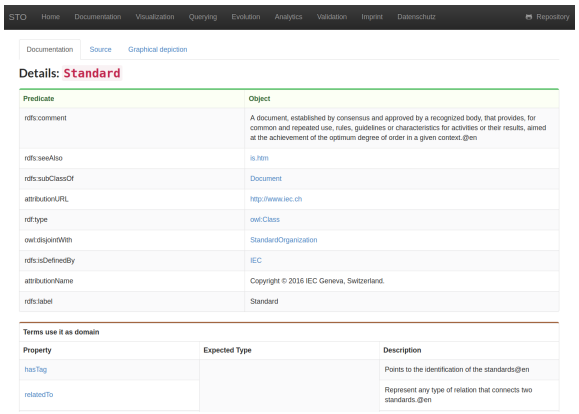
(a) Tree View



(b) Source View



(c) Graphical Depiction



(d) Single Page per Concept

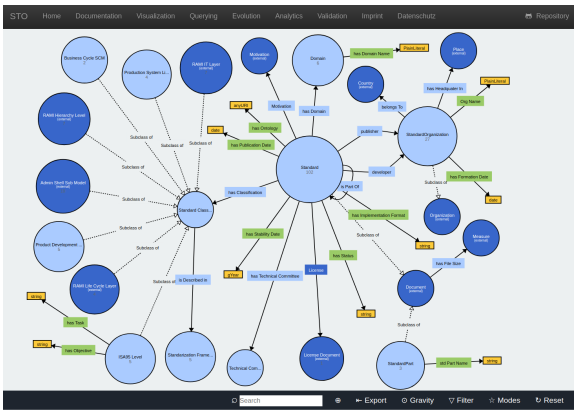
Figure 6.4: **Human-friendly Documentation**. Information about an ontology concept represented using various views: *Tree View*, *Source View*, *Graphical Depiction*, and *Single Page per Concept View*.

Evolution Tracking: When the semantic differences between ontology versions exist, an evolution report is generated using the Owl2vcs [111] tool. It utilizes algorithms for structural diffs and three-way merge tools along with OWL 2 direct semantics. The application of direct semantics eliminates problems with blank nodes and allows for comparing ontologies axiom by axiom. This report lists each point in time, i.e., in a timeline fashion, when a new ontology revision has been pushed, as shown in Figure 6.5(d). After selecting a node in the timeline window, details related to semantic changes like *addition*, *removal*, or *modification* of elements are represented in a tabular format. Users are able to filter tabular entries based on the change type, person who submitted, submission time or commit message.

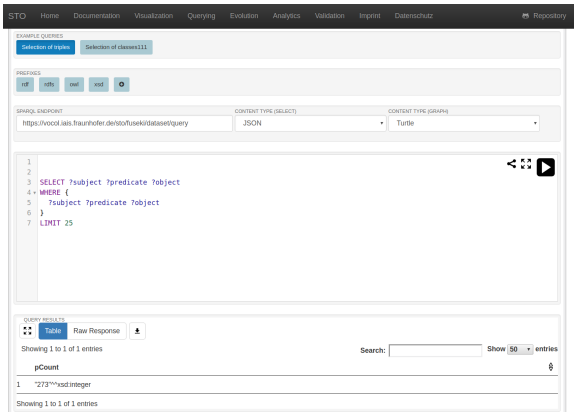
Machine Accessibility: Various machine-comprehensible formats of the ontology being developed, such as *Turtle*, *JSON* and *RDF/XML*, are delivered to an agent through the *content negotiation* mechanism. This mechanism which is an integral part of VoCol backend, is implemented based on the *best practices for publishing vocabularies*[8]. Following the principles of the *server-driven*[9], an agent specifies the *accept parameter* in the *HTTP header*

[8] http://www.w3.org/TR/swbp-vocab-pub/. Date Accessed: 25 April 2018.

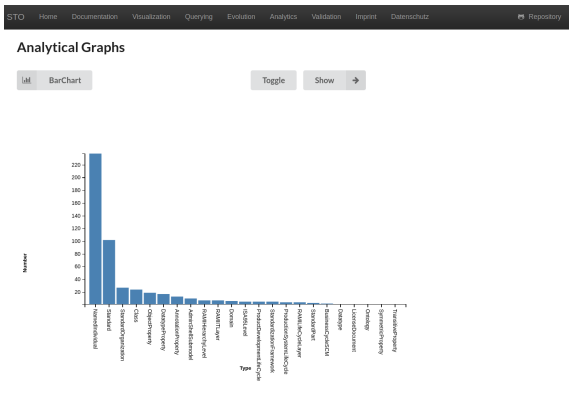
[9] https://www.w3.org/blog/2006/02/content-negotiation. Date Accessed: 25 April 2018.



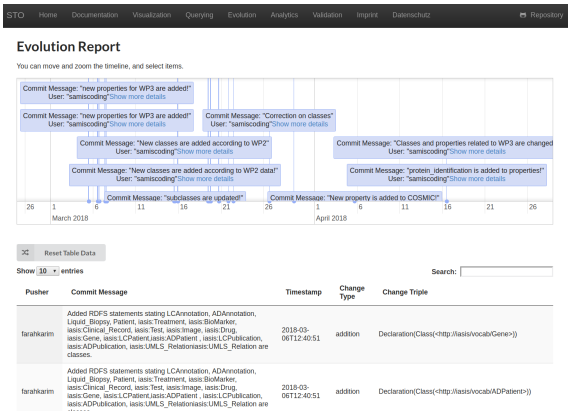
(a) Visualization



(b) Querying



(c) Statistics



(d) Evolution

Figure 6.5: **Additional VoCol Views**. Other views, such as *Visualization*, *Querying*, *Statistics* and *Evolution* provide possibilities for exploring and better understanding of the ontology being developed.

request, whereas the server deliver the requested format. As a result, agents are provided with the latest version of the entire ontology at any time. Furthermore, the combination of the *content negotiation* with the *dereferenceability* enable agents to perform more granular requests, such as asking only for a particular concept in different representation formats.

Querying Component: An integrated SPARQL endpoint component using *Jena Fuseki* allows to perform queries and exporting the results in different formats. Users can test whether the ontology being developed meets their requirements. Additionally, this component checks for the existence of files within the repository that has extension *.rq*, commonly dedicated to SPARQL queries. All found files are uploaded to this component considering the file name as the query name, and the content of the file as the query. Table 6.1 lists some examples of the predefined queries, that can be easily changed or extended. An indication for constraint violation is the case when the returned value after executing the corresponding SPARQL query does not match the value in the "Expected Value" column. The results of the validation process is reported in HTML format. Listing 6.2 depicts the SPARQL query that checks for missing *rdfs:label* and *rdfs:comment* in the English language.

Table 6.1: **Predefined Queries**. Examples of the predefined queries for constraint checking, such as finding duplicate labels or comments, checking for forbidden words, etc.

Query	Expected Value	Required
At least one *owl:Ontology* needs to be defined	isNotEmpty	Mandatory
Two resources should not have the same *rdfs:label*	isEmpty	Mandatory
Two resources should not have the same *rdfs:comment*	isEmpty	Mandatory
All resources should have *rdfs:label* and *rdfs:comment* in English	isEmpty	Optional
All resources must not have literals with "foo bar", "lorem" or "ipsum"	isEmpty	Mandatory
All resources should have *rdfs:label* different from *rdfs:comment*	isEmpty	Optional
All resources should have *rdfs:comment* in different languages	isEmpty	Optional
All skos:ConceptSchemes should have a *skos:definition* in english	isEmpty	Mandatory
All skos:Concepts should be *skos:inScheme*	isEmpty	Mandatory
All skos:Concepts should have a *skos:broader* statement	isEmpty	Optional

```
SELECT DISTINCT ?r WHERE
{  ?r rdf:type ?type .
   MINUS { ?r rdf:type skos:Concept. }
   MINUS { ?r rdf:type skos:ConceptScheme. }
   OPTIONAL { ?r rdfs:label ?label .
   FILTER( (STRLEN(?label) > 0)  && langMatches( lang(?label), 'en' ))}

   OPTIONAL { ?r rdfs:comment ?comment .
   FILTER((STRLEN(?comment) > 0) && langMatches( lang(?comment), 'en'))}

   FILTER ( !bound(?label) || !bound(?comment) )
} ORDER BY ?r
```

Listing 6.2: **Check for documentation properties**. A SPARQL query to check whether resources have at least one English *rdfs:label* or *rdfs:comment*, which are used for documentation purposes.

6.2.4 Deployment

With the objective of facilitating the installation and configuration, VoCol is deployed in virtual environments, such as VirtualBox and Docker. As a result, the process of setting up such an image is reproducible and documented at the same time. The VoCol environment, thus works as an isolated platform without affecting the rest of the physical machine. This ensures high portability, allowing the administrator to easily start, stop, move, or share it as well as making VoCol as a an interoperable and cross-platform solution applicable in various operating systems. With a few additional steps, the VoCol environment can be installed and configured on a clean web server. All implementation details are available on the VoCol repository[10].

6.3 Evaluation

We show the applicability of VoCol in an industry use case to evaluate its usefulness and effectiveness in a real-world setting. Furthermore, we conduct a qualitative user study to get additional insights into the usefulness and usability of VoCol.

[10] https://github.com/vocol/vocol

6.3.1 Industry Application

In this scenario, VoCol is applied in a specific industry use case to develop ontologies for formally describing the assets of an enterprise, including how they are relate to each other. All of these ontologies are the intellectual property of the industry partner. We are restricted in the information we can provide here, but we share at least some experiences and insights that we gathered during project implementation.

A group of seven people contributed in parallel to the development of the ontologies. While the ontology engineers conducted most of the formalization, the domain experts participated by creating issues. Overall, 46 issues of different types were created ranging from proposals to add, modify, or remove ontology concepts. Additionally, other issues, such as bug fixes and feature requests are registered in various phases of the project. In total, the developed ontologies comprise 151 classes, 93 object properties, 225 datatype properties, and 79 instances.

The loose coupling characteristic of VoCol allowed us to integrate a new component for defining and establishing R2RML mappings between the developed ontologies and legacy systems of the industry partner. By doing so, users were able to execute queries against these systems and receive the results in various representation formats, such as tabular, pie, and bar charts.

According to the informal feedback of the involved stakeholders, VoCol provides a very useful and effective support in this use case. In particular, the different views are considered to be very helpful in getting a better understanding and exploring the current status of ontologies. The easy and comfortable access to all services via one integrated web interface was essential for stakeholders to optimize their efforts with respect to contribution and use of ontologies.

Despite the above mentioned benefits of VoCol for this use case scenario, one of the drawbacks that we experienced is the lack of a simple form-based editing of ontology terms. This prevented domain experts from contributing with their ideas directly to the development process, enforcing the continuous involvement of ontology engineers.

6.3.2 User Study

We conduct a qualitative user study of VoCol under controlled conditions using the *Concurrent Think Aloud* (CTA) method: Participants are observed and asked to verbalize their thoughts while performing the given tasks [112]. At the beginning of each session, the interviewer gave a general introduction into VoCol. The interaction with the system as well as comments and suggestions are recorded for a later analysis. After completing the given tasks, participants had a discussion with the interviewer about their experiences and any difficulties they faced while performing the experiment. To measure the usability and ease of use, participants are asked to fill a questionnaire at the end of the interview.

Participants To ensure that participants represent as closely as possible the targeted ontology engineers group of the VoCol system, we chose twelve users with different levels of expertise. Their expertise range from basic ontology modeling experience to a more advanced expertise in knowledge conceptualization and representation.

Tasks and Questionnaire We designed a set of tasks that comprise all activities of the round-trip development described in the Section 4.1: Starting from the modeling activity, the first task was to define several classes with various numbers of properties. The next task was concerned with the population of the ontology, in which users had to create instances based on the defined classes. The tasks are first performed on the user machine by committing all changes to the local repository, and later pushing those changes to the remote repository. In

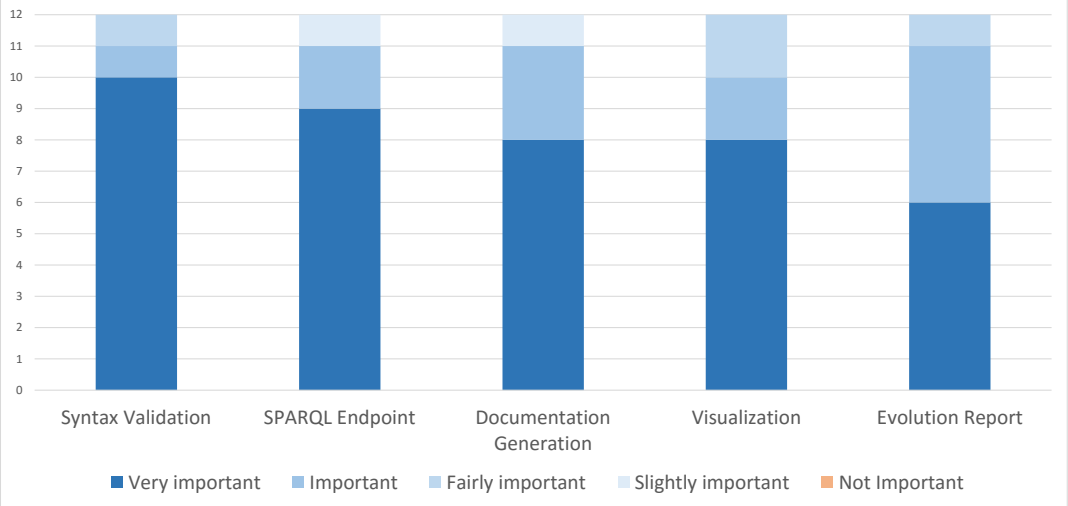


Figure 6.6: **Scores given for VoCol services**. The given scores for the usefulness of the VoCol services, such as *Syntax Validation*, *SPARQL Endpoint*, *Documentation Generation*, *Visualization* and *Evolution Report* according to the participants of the study.

addition, to test the functionality of the TurtleEditor, users were asked to perform the same tasks using it. The SPARQL endpoint was used to execute test queries verifying whether the developed ontology met certain criteria. All functionalities provided by VoCol, including the syntax validation before commit and after push events to the remote repository, documentation generation, and visualization, are covered in the user study.

In addition, we asked the participants to fill an electronic post-study questionnaire composed of two main sections. The first section contains the USE Questionnaire[11], which uses five-point Likert scales for rating, ranging from 1 (strongly disagree) to 5 (strongly agree). We evaluated four usability dimensions: 1) usefulness; 2) ease of use; 3) ease of learning; and 4) satisfaction. To get more insights into specific areas, we defined three additional questions in the second section of the questionnaire. With these questions, we aim to get the participants' opinion about: 1) the importance of the individual services integrated into VoCol; 2) negative and positive aspects of the system through an open response question; and 3) potential services to be integrated in the future. The evaluation material is available online[12].

Results We obtained the evaluation results by observation, discussions at the end of each session, and the post-study questionnaires. The following are some of the findings that we derived from the analysis of the observation notes and discussions:

- Participants with prior knowledge about VCS, especially with Git, find VoCol very easy to learn and use;
- A few participants expect to see the provenance metadata in the browser, i.e., the date and author for each term added to the ontology;
- The instant syntax-checking and auto-completion feature of the TurtleEditor is considered very helpful by the majority of the participants.

[11] http://hcibib.org/perlman/question.cgi?form=USE
[12] https://figshare.com/articles/VoCol_Evaluation_Material/3438371

The results from the USE questionnaire showed that the responders rated their experience with VoCol very high. The average scores received by each dimension are as follows: *usefulness = 4.34*, *ease of use = 3.97*, *ease of learning = 4.35*, and *satisfaction = 4.31*. These scores indicate a high usability of VoCol (nearly all scores are > 4) and correlate with the oral feedback of the participants that VoCol is "easy to learn and use", as well as the informal feedback of the stakeholders from the industry use case that VoCol provides "very useful and effective support".

Figure 6.6 shows that each of the services provided by VoCol is of high relevance to the study participants. For instance, 10 of the 12 participants consider *syntax validation* a *very important* service, while the scores for the other services are only slightly lower due to different level of expertise of participant where for more experienced ones, the provided services are less impacting. Some interesting suggestions made by the participants are: 1) creating a possibility for dynamically adding and removing tools from the user interface; and 2) automatic recommendation of similar ontologies (e.g., using the LOV API[13]).

6.4 Summary

This chapter presents VoCol, an integrated environment for distributed development of ontologies based on version control systems. We argue that the development of an effective and efficient environment for distributed collaboration is the main challenge in the context *collaborative ontology development* in distributed environments. A conceptual architecture is introduced with the objective of addressing the requirements identified in the Subsection 4.2.2. The fundamental principle of this approach is to enable developing of VoCol according to the given architecture by extending the functionality of plain version control systems with additional modules to cover specific aspects of the ontology construction.

VoCol is implemented on the basis of the widely used Git as version control and leveraging the *webhook* mechanism of the repository hosting platforms. Tasks such as content negotiation, documentation and visualization generation, as well as evolution tracking are performed in a fully automated way. In addition, a querying service, synchronized with the latest version of the ontology, enables users to execute customized SPARQL queries. The VoCol environment is easily expandable with other tools to provide additional functionalities. The current implementation of VoCol is tailored to small and medium size ontologies. However, it can be adopted and extended for supporting various scenarios by replacing its components with adequate alternatives. The applicability of VoCol is demonstrated with a real-world example for an industry partner. Additionally, a user study to asses the its usability and usefulness is conducted with a group of twelve participants having different level of experience. The results from the practical application and the user study provide evidences that VoCol effectively support the entire ontology development life-cycle centered around versions control systems along with a set of useful and usable services to cover specific modeling aspects.

[13] http://lov.okfn.org/dataset/lov/vocabs

Part III

Quality Assurance for Ontology Development

In Part II, we described a number of collected requirements along with the defined methodology and the platform to support collaborative ontology construction in distributed scenarios. This part is focused in synchronization of changes and quality aspects of the development process. In particular, an approach for preventing false-positive conflicts during the synchronization of parallel changes performed by heterogeneous authoring editors is presented in Chapter 7. As a result, changes to the ontology are serialized according to a unique criteria, thus reducing a huge number of conflicts detected by generic version control systems. We then continue with Chapter 8, which describes an approach to efficiently ensure the development of qualitative ontologies conform pre-defined domain requirements. A suite of test cases built after the requirement definition phase is used to prevent any non-indented modification to the ontology. To efficiently evaluate this suite, test cases are organized in a dependency graph, which is traversed using the *breadth-first search algorithm*.

CHAPTER 7

Serialization Agnostic Ontology Development in Distributed Settings

This chapter presents SerVCS, an approach for enabling VCSs to deal with various serializations of the same ontology in a multi-editor scenario in order to address the requirement **P3** defined in Subsection 4.2.2. The presented approach pave the way to built a dedicated component, which then is integrated into VoCol platform. It facilitates the development process which requires significant efforts and knowledge, and the participation of different stakeholders who are geographically distributed [29]. In this process, it is crucial to track, propagate and synchronize ontology changes to all contributors. Thus, supporting change management is indispensable for a successful ontology development in distributed settings.

A *Version Control System* (VCS) assists users to collaboratively work on shared artifacts, and helps them to prevent from overwriting each other changes. Basically, mechanisms to avoid change overwriting can be classified in *pessimistic* and *optimistic* approaches [26]. The first ones are based on the *lock-modify-unlock* paradigm, which implies that modifications to an artifact are permitted only for one user at a time. The latter ones follow the *copy-modify-merge* paradigm, where users work on personal copies, each reflecting the remote repository at a certain time. After completing the work, the local changes are merged into the remote repository by an *update* command, comprising the phases *comparison*, *conflict detection*, *conflict resolution*, and *merge*.

Different techniques, such as line-, tree-, and graph-based ones, can be employed to compare two versions of the same artifact [27]. The line-based technique, which achieved wide applicability, compares artifacts line by line, where each line is treated as a single unit. This technique is also known as *textual* or *line-based comparison* [26]. Examples of VCSs that use the line-based approach are Subversion, CVS, Mercurial, and Git. Line-based comparisons are applicable on any kind of text artifact, as they do not consider syntactical information [27]. Accordingly, line-based techniques also neglect syntactical information of ontologies, which is commonly represented in some text-based RDF serialization.

Challenges arise when two ontology engineers modify in parallel the same artifacts on their personal working copies. These changes might contradict each other, for instance, engineers may both edit the name of an ontology concept simultaneously. Such parallel and controversial modifications can result in conflicts during the merging of two ontology versions. In general, a conflict is defined as "a set of contradicting changes where at least one operation applied by the first developer does not commute with at least one operation applied by the second developer" [27]. Conflicts can be detected by identifying changed units (i.e., added, updated,

deleted) that have been performed in parallel. Conflict resolution can be done automatically or may require user intervention to manually fix them by resolving the conflicting changes.

From the ontology development point of view, the situation is exacerbated when various ontology editors are used during the development process. This is due to the fact that these editors often produce different serializations of the same ontology, i.e., the ontology concepts are grouped and sorted differently in the files generated by the editors.[1] As a result, the ability of VCSs to detect the actual changes in ontologies is lowered, since they find a number of conflicts that are actually not given but are a result of different serializations of the ontology files. In order to increase the accuracy of conflict detection in VCSs, the problem of different groupings and orderings must be tackled.

We present *SerVCS* with the objective to enhance VCSs for coping with different serializations of the same ontology, following the principle of *prevention is better than cure*. *SerVCS* resorts on unique ontology serializations and minimizes the number of false-positive conflicts. It is implemented on top of Git, utilizing tools such as Rapper and RDF-toolkit for syntax validation and unique serialization, respectively. We conducted an empirical evaluation to determine the conflict detection accuracy of *SerVCS* whenever simultaneous changes to an ontology are performed using different ontology editors. Experimental results suggest that *SerVCS* allows VCSs to conduct more effective synchronization processes by preventing false-positive conflicts.

In this chapter, we address the following research question:

> **RQ3**: How can concurrent changes from heterogeneous ontology authoring editors be effectively synchronized?

Contributions of this chapter are summarized as follows:

- A formal definition of an approach for enabling ontology development in distributed environments using heterogeneous editors;
- An implementation of the defined approach on top of Git and VoCol to prevent contributors from introducing false-positive conflicts;
- An empirical evaluation to asses the accuracy of *SerVCS* with regard to conflict detection in two different scenarios, by changing: 1) the ontology size; and 2) the sorting criteria.

This chapter is based on the following publications:

- **Lavdim Halilaj**, Irlan Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *SerVCS: Serialization Agnostic Ontology Development in Distributed Settings.* In Communications in Computer and Information Science (CCIS) 914 - Revised Selected Papers from 8th International Joint Conference, IC3K 2016, Porto, Portugal, 213-232, Springer. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I conducted the formalization of the problem, implementation of the approach, the revision of the state of the art approaches, the presentation of the use cases, as well as the analysis of the results;

- **Lavdim Halilaj**, Irlan Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *Proactive Prevention of False-Positive Conflicts in Distributed Ontology Development.*

[1] With "different serializations", we refer to two different ontology files that represent the same ontology using the same syntax (e.g., RDF/XML, Turtle, Manchester) but use a different structure to list and group the ontology concepts.

In 8th International Conference on Knowledge Engineering and Ontology Development Proceedings (KEOD), 43-51, SciTePress. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I led the formalization of the problem, implementation of the approach, the revision of the related work, the presentation of the use cases, as well as the analysis of the results.

The remainder of this chapter is organized as follows: It starts with an overview of the motivating scenario, which is presented in Section 7.1. The problem is defined in Section 7.2 and in Section 7.3, we describe the *SerVCS* approach. The implementation is described in Section 7.4, whereas the approach is evaluated against a number of different scenarios in Section 7.5, before the chapter is summarized in Section 7.6.

7.1 Motivating Example

As a motivating example, we consider two users working together in developing an ontology for a specific domain. In order to ease the collaboration and maintain different versions of the developed ontology that result from changes, they decide to use Git. They proceed by setting up the working environment and creating an initial ontology repository which contains several files. Together, users define the ontology structure with the most fundamental concepts and upload the ontology file F to the *remote repository*. After that, they decide to proceed with their tasks by separately working on their local machines.

The users start synchronizing their local working copies with the remote repository, as illustrated in *Scene 1* of Figure 7.1. *Scene 2* depicts simultaneous changes performed on different copies of the same ontology file, such as adding new concepts, modifying existing ones, or deleting concepts. For realizing this task, different ontology editors are used. In our case, *User 1* works with *Desktop Protégé*[2], whereas *User 2* prefers to edit the ontology with *TopBraid Composer*[3].

After finishing the task, *User 1* uploads her personal working copy (F^*) to the *remote repository*, as shown in *Scene 3*. Next, *User 2* completes his task and starts uploading the changes he made on his local copy to the *remote repository*. While trying to trigger this action, he receives a rejection message from the VCS, listing all changes which result in conflicts, as depicted in *Scene 4*. These conflicts need to be resolved in order for the VCS to allow the user to successfully upload his version (F^{**}) to the *remote repository*. *User 2* starts resolving the conflicts manually by comparing his version of the ontology with the one of *User 1* that has already been uploaded to the remote repository.

Since the users are working with different ontology editors where each use its own serialization during saving the ontology file, the files are differently organized. For instance, while the concepts in one of the files are grouped into categories, such as *Classes* and *Properties*, they are ordered alphabetically in the other case, without any grouping. Consequently, the information about actual changes, i.e., concrete changes on the ontology performed by each user, can no longer be detected by the line-based comparison of the VCS, but a huge number of conflicts result, which are due to the different organization of the ontology files. This prevents *User 2* from merging his changes, and his version of the ontology cannot be uploaded to the remote repository.

This scenario illustrates that, despite the various benefits provided by a VCS for collaborative ontology development, it has not been possible so far to effectively use a VCS in cases where different editors and ontology serializations are used.

[2] http://protege.stanford.edu
[3] http://www.topquadrant.com/composer/

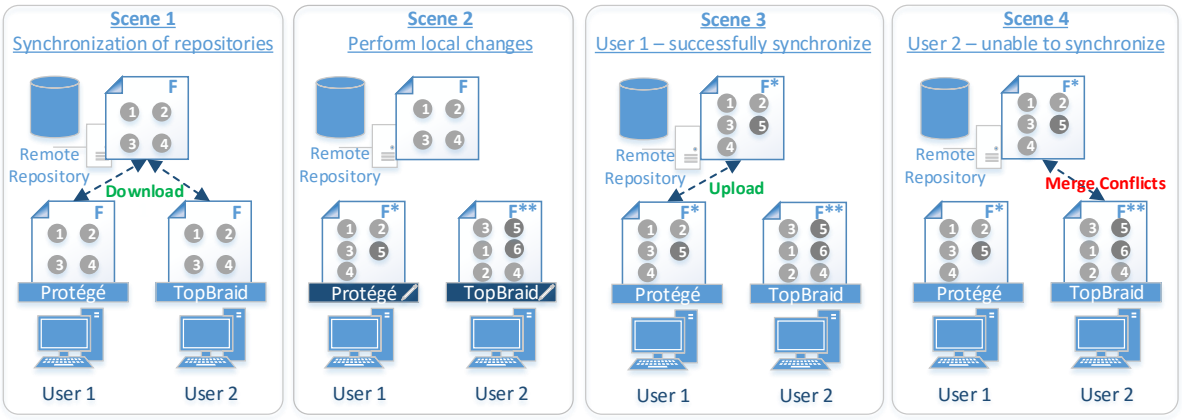


Figure 7.1: **Motivating Example**. A distributed environment illustrating an ontology development process. Different ontology editors, e.g., Editors X and Y, are used for defining ontology F by Users 1 and 2, respectively. F* and F** represent local versions of F. If F* is uploaded first, changes in F* can be synchronized. Changes in F** cannot be merged whenever F* and F** serializations are different.

7.2 Problem Definition

This section provides basic terminology and the of formal definition of the *SerVCS* approach. An ontology is represented in an RDF document, which A is formally defined as $A \subset (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$, where $\mathbf{I}$, $\mathbf{B}$, and $\mathbf{L}$ correspond to sets of *IRI*s, *blank nodes*, and *literals* (typed and untyped), respectively [113].

Definition 1 (Changeset): *Given two RDF documents A and A^*, a changeset of A^* with respect to A is defined as follows:*

$$ChangeSet(A^*/A) = (\delta^+(A^*/A), \delta^-(A^*/A), <), \text{ where}$$

- $\delta^+(A^*/A) = \{t \mid t \in A^* \wedge t \notin A\}$,
- $\delta^-(A^*/A) = \{t \mid t \in A \wedge t \notin A^*\}$, *and*
- $<$ *is a partial order between the RDF triples in* $\delta^+(A^*/A) \cup \delta^-(A^*/A)$.

Example 1: *Consider two RDF documents $A = \{t_1, t_2, t_3\}$ and $A^* = \{t_1, t_2, t_4\}$ such that A^* is a new version of A where the RDF triple t_4 was added and the triple t_3 was deleted. Then, the changeset of A with respect to A^*, $ChangeSet(A^*/A)$, is as follows:*

- $\delta^+(A^*/A) = \{t_4\}$,
- $\delta^-(A^*/A) = \{t_3\}$, *and*
- $<= \{(t_4, t_3)\}$.

Definition 2 (Syntactic Conflicts): *Given two RDF documents A and A^*, and the changeset of A^* with respect to A, $ChangeSet(A^*/A) = (\delta^+(A^*/A)$, $\delta^-(A^*/A), <)$, there is a syntactical conflict between A and A^* iff there are RDF triples t_i and t_j such that:*

- $t_i \in \delta^-(A^*/A)$,
- $t_j \in \delta^+(A^*/A)$,

- $(t_i, t_j) \in <$, *and*
- $t_i = (s, p, o_i)$, $t_j = (s, p, o_j)$, *and* $o_i \neq o_j$.

Example 2: *Consider two RDF documents A and* A^* *with triples* t_3*=(*`:Train, rdfs:label, "Trai"@en`*) and* t_4*=(*`:Train, rdfs:label, "Trainn"@en`*). Since the object value of the property* `rdfs:label` *of the subject* `:Train` *has been changed, there is a syntactic conflict between the RDF documents A and* A^*.

Definition 3 (RDF Document Serialization): *Given an RDF document A and an ordering criteria* η*, a* serialization *of A according to* η , $\Gamma(A, \eta)$ *corresponds to an ordering of the triples in A according to* η*:*

$$\Gamma(A, \eta) =< t_1, t_2, ..., t_n >$$

Example 3: *Suppose three RDF triples* t_1*,* t_2*, and* t_3 *are defined as follows in an RDF document A:* t_1*=(*`:Car, rdfs:label, "Car"@en`*),* t_2*=(*`:Truck, rdfs:label, "Truck"@en`*), and* t_3*=(*`:Bus, rdfs:label, "Bus"@en`*), respectively. A serialization* $\Gamma(A, \eta)$ *of A listing the triples by their labels in alphabetical order* η *would be:*

$$\Gamma(A, \eta) =< t_3, t_1, t_2 >$$

Definition 4 (False-Positive Conflicts): *Given two RDF documents A and* A^* *such that* F_1 *and* F_2 *are serializations of A and* A^* *according to some ordering criteria* η_1 *and* η_2*, respectively. There is a* false-positive conflict *between* F_1 *and* F_2*, iff there exist* η *ordering criteria such that:*

$$\Gamma(A, \eta) = \Gamma(A^*, \eta) \text{ and } F_1 \neq F_2$$

Example 4: *Consider serializations* $F_1 =< t_1, t_3, t_2 >$ *and* $F_2 =< t_2, t_1, t_3 >$ *both representing two identical RDF documents* $A = A^*$*, respectively, such that* $A = \{t_1, t_2, t_3\}$*. Then, there are three false-positive conflicts between* F_1 *and* F_2*, because there exist ordering criteria* η*,* $\Gamma(A, \eta) = \Gamma(A^*, \eta)$.

7.3 The SerVCS Approach

With the objective of enabling ontology development in distributed environments, where sets of changes are performed (cf. Definition 1) using different editors, the indication of *False-Positive Conflicts* (cf. Definition 4) by the VCS must be avoided. For this reason, ontologies should have a *unique serialization* (see Definition 3). In order to realize that, we developed *SerVCS*, which generates a unique serialization of ontologies regardless of the used editing tool. The modeled concepts (triples) are ordered alphabetically in this unique serialization, first according to the *subject* name, then by *property* name. That way, ontologies (represented as text-based RDF documents) have always a consistent serialization in the remote repository. As a result, a high accuracy of conflict detection can be achieved and the identified conflicts are reduced to those caused by overlapping changes, *Syntactical Conflicts* (cf. Definition 2). This enables a VCS to automatically resolve most conflicts using its built-in algorithms. In the worst case, a user is confronted with conflicting changes and has to manually resolve them by providing a valid and consistent ontology. Since all ontologies have a unified serialization in the remote repository, the user is able to see the differences between any two versions of the ontology. Figure 7.2 illustrates the *SerVCS* approach, which consists of five main steps: 1) input: RDF documents serialized by

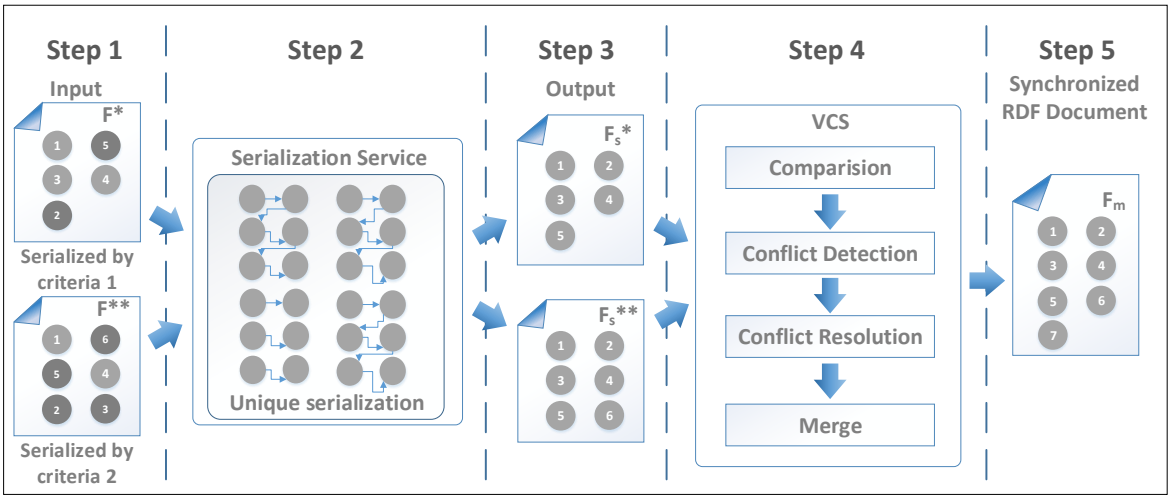


Figure 7.2: **The SerVCS Approach**. SerVCS receives RDF documents serialized by different sorting criteria (Step 1), and generates a synchronized RDF document (Step 5). In Step 2, a unique serialization is produced. RDF documents are sorted with same criteria (Step 3). Finally, a VCS synchronization process is performed (Step 4), i.e., comparison, conflict detection, conflict resolution, and merge.

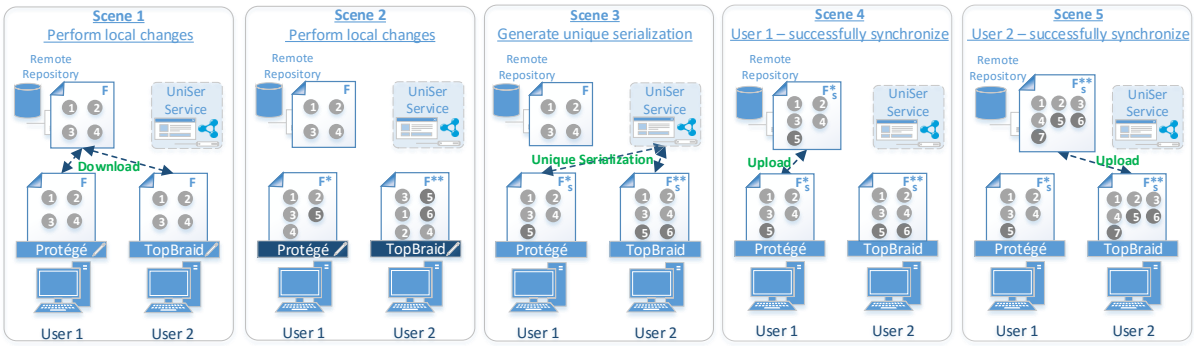


Figure 7.3: **The SerVCS Development Workflow**. A distributed environment illustrating an ontology development process using SerVCS. Different ontology editors, e.g., Editors X and Y, are used for defining ontology F by Users 1 and 2, respectively. F* and F** represent local versions of F. Before synchronization with remote version, a unique serialization is created for F* and F**. F* is uploaded first. Next, changes in F** are successfully synchronized with F* since they have a unique serialization and any possible conflict is easy to be detected and resolved.

different sorting criteria; 2) generate unique serialization; 3) output: RDF documents sorted with same criteria; 4) synchronization process from the point of view of VCS; and 5) final outcome: synchronized RDF document.

Figure 7.3 depicts the ontology development workflow using the *SerVCS* approach. After personal working copies are synchronized with the remote repository (cf. *Scene 1* of Figure 7.1), users start performing their tasks with various ontology editors. When making any changes, such as adding, removing, or modifying existing concepts, the updated ontology is saved locally on the machine of the user, as illustrated in Figure 7.2, *Scene 2* (which is still identical to *Scene 2* of Figure 7.1). Next, these changes are uploaded to the remote repository. *Scene 3* shows that a unique serialization of the ontology is created as intermediate step. As a result, the concepts are

organized using a common ordering criteria. In *Scene 4*, *User 1* uploads her changes successfully to the remote repository. Lastly, as illustrated in *Scene 5*, *User 2* starts uploading his changes to the remote repository. Since the ontology has a unified serialization, the VCS can merge both versions. In case of overlapping changes, the VCS shows exactly the lines which resulted in conflicts. Formally, a list of conflicts LC identified by *SerVCS* is defined as follows:

Definition 5 (List of Conflicts): *Given two RDF documents A and A^* such that F_1 and F_2 are serializations of A and A^* according to ordering criteria η_1 and η_2, a list $LC =< c_1, \ldots, c_n >$ of conflicts between F_1 and F_2, identified by* SerVCS*, comprises triples $c_i = (i, entry_{i1}, entry_{i2})$:*

- $i \in [1, MIN(size(F_1), size(F_2))]$,
- $entry_{i1} = (s_{i1}, p_{i1}, o_{i1})$ *and* $entry_{i2} = (s_{i2}, p_{i2}, o_{i2})$ *are RDF triples at the position i in F_1 and F_2, respectively,*
- $entry_{i1}$ *and* $entry_{i2}$ *are different, i.e.,* $s_{i1} \neq s_{i2}$ *or* $p_{i1} \neq p_{i2}$ *or* $o_{i1} \neq o_{i2}$.

Theorem 1: *Given serializations F_1 and F_2 according to ordering criteria η of RDF documents A and A^*, respectively. Consider $LC =< c_1, \ldots, c_n >$ the list of conflicts between F_1 and F_2 identified by* SerVCS*. If there are only syntactical conflicts between A and A^*[4], then for all $c_i = (i, entry_{i1}, entry_{i2}) \in LC$*

- $entry_{i1} = (s, p, o_{i1})$ *and* $entry_{i2} = (s, p, o_{i2})$, *and*
- $o_{i1} \neq o_{i2}$.

Proof 1: *We proceed with a proof by contradiction. Assume that there are only syntactical conflicts between A and A^*, and there is a conflict c_i in LC, such that $c_i = (i, (s_{i1}, p_{i1}, o_{i1}), (s_{i2}, p_{i2}, o_{i2}))$, and $s_{i1} \neq s_{i2}$ or $p_{i1} \neq p_{i2}$. Since F_1 and F_2 are serializations according to the same ordering criteria η, $entry_{i1} \in \delta^-(A^*/A)$ and $entry_{i2} \in \delta^+(A^*/A)$. However, the statement $s_{i1} \neq s_{i2}$ or $p_{i1} \neq p_{i2}$ contradicts the fact that only syntactical conflicts exist between A and A^*.*

7.4 Implementation

The architecture depicted in Figure 7.4 has been implemented to empower VCSs to prevent wrongly indicated conflicts. The architecture consists of three main components: 1) a VCS, which handles different ontology versions via changesets; 2) a UniSer component, which generates unique serializations for the RDF documents; and 3) a repository hosting platform, which stores the RDF documents and propagates the changes.

7.4.1 Version Control System

Git is used as a Version Control System (VCS), i.e., Git is responsible for managing different versions of the ontologies. Furthermore, the Git *hook* mechanism is utilized to automatize the process of generating the unique serialization of the ontologies before they are pushed to the remote repository. Once the modification of the ontology is finished, it is added to the Git *stage* phase. The next step proceeds with committing the current state to the personal working copy.

[4] Given two RDF-documents A and A^*, and $ChangeSet(A^*/A) = (\delta^+(A^*/A), \delta^-(A^*/A), <)$, there are only syntactical conflicts between A and A^*, iff $size(A^*) = size(A)$, and for each RDF triples t_i and t_j:

- $t_i \in \delta^-(A^*/A)$ and $t_j \in \delta^+(A^*/A)$,

then, there is a pair $(t_i, t_j) \in <$, and $t_i = (s, p, o_i)$, $t_j = (s, p, o_j)$, and $o_i \neq o_j$.

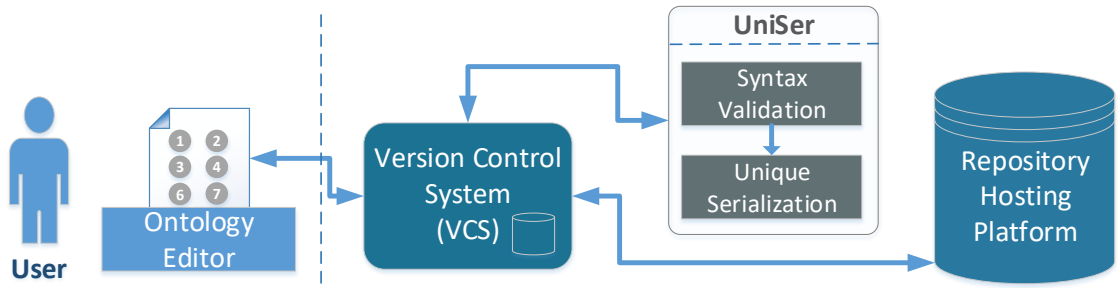


Figure 7.4: **The SerVCS Architecture**. Users interact with ontology editors, e.g., Protégé: 1) a VCS handles different ontology versions via changesets, e.g., Git; 2) The UniSer component performs syntax validation and generates unique serializations, e.g., Rapper or RDF-Toolkit; and 3) A Repository Hosting Platform stores the ontologies and propagates the changes (GitHub).

The initialization of the commit event triggers a hook named *pre-commit*. This hook is adapted with a new workflow to handle the process of automatically generating a unique serialization, apart from the default one provided by Git.

SerVCS uses *Curl*[5] as command-line HTTP client to send the modified files to the *UniSer* service. In case that ontologies fail to pass the validation process, the commit is aborted and a corresponding error message is shown to the user. Otherwise, the files are organized according to the unique serialization. Subsequently, newly generated content overwrites the current content of the files by replacing the old serialization created by the ontology editor with the new unique serialization created by *UniSer*. When no error occurs during the entire process, the *pre-commit* hook event is completed and the commit is applied successfully. As a result, a new revision of the modified ontologies is created and the user is able to further proceed with successfully pushing her version to the remote repository. In addition, *GitHub* is used as hosting platform for the repository to ease the collaborative development among several contributors.

7.4.2 UniSer

Furthermore, we implemented a stand-alone service, *UniSer*, using the cross-platform JavaScript runtime environment *Node.js*. Other tools are integrated to realize the tasks required for this service, e.g., syntax validation and unique serialization. The service accepts the ontology files as input through an HTTP interface and returns to the client either the error message from the validation process or the unique serialization of the file.

Once the input is received, *UniSer* validates the ontology, since a prerequisite for the unique serialization process is that ontology files are free of syntactic errors. The syntax validation is performed by *Rapper*. In case of errors, a detailed report comprising the file name, error type, and error line is returned to the client. Otherwise, the process continues with creating a unique serialization using RDF-toolkit[6] or *Rapper*, according to the user preference for the sorting criteria to be used. During this task, a unified serialization of the ontology file is created by 1) grouping elements into categories, such as classes, properties, and instances, and 2) alphabetically ordering elements within the categories. The unique serialization of the ontology is send back to

[5] https://curl.haxx.se
[6] https://github.com/edmcouncil/rdf-toolkit

the client as final outcome and the respective file is updated accordingly.

In the following, we present several examples of a simple ontology serialized in *Turtle* format comprising three main concepts: *Train* and *UrbanTrain* of type *owl:Class* and a *UrbanTrain01* instance of the *Train* class.

```
--------------------------------------------------------------------------------
@prefix     : <http://example.com/> .
@prefix owl : <http://www.w3.org/2002/07/owl#> .
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:Train        rdf:type   owl:Class ;
              rdfs:comment "Train Concept"^^xs:string ;
              rdfs:label "Train"^^xs:string ;
              rdfs:subClassOf :Vehicle .

:UrbanTrain   rdf:type   owl:Class ;
              rdfs:comment "Train Concept"@en ;
              rdfs:label "Train"@en ;
              rdfs:subClassOf :Train .

:UrbanTrain01 rdf:type :UrbanTrain ;
              rdfs:comment "UrbanTrain01 operates in zone B"@en ;
              rdfs:label "UrbanTrain01"@en .
--------------------------------------------------------------------------------
```

The below excerpt serialized using Protégé tool is shown as follows:

```
--------------------------------------------------------------------------------
@prefix     : <http://example.com/> .
@prefix owl : <http://www.w3.org/2002/07/owl#> .
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

#################################################
#        Classes
#################################################
###      http://example.com/Train
:Train        rdf:type   owl:Class ;
              rdfs:comment "Train Concept"^^xs:string ;
              rdfs:label "Train"^^xs:string ;
              rdfs:subClassOf :Vehicle .

###      http://example.com/UrbanTrain
:UrbanTrain   rdf:type   owl:Class ;
              rdfs:comment "Train Concept"@en ;
              rdfs:label "Train"@en ;
              rdfs:subClassOf :Train .

#################################################
#        Individuals
#################################################
###      http://example.com/UrbanTrain01
:UrbanTrain01 rdf:type :UrbanTrain ;
```

```
              rdfs:comment "UrbanTrain01 operates in zone B"@en ;
              rdfs:label "UrbanTrain01"@en .
```

The same ontology serialized with the TopBraid Composer is as follows:

```
@prefix     : <http://example.com/> .
@prefix owl : <http://www.w3.org/2002/07/owl#> .
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:Train
     a   owl:Class ;
     rdfs:comment "Train Concept"^^xs:string ;
     rdfs:label "Train"^^xs:string ;
     rdfs:subClassOf :Vehicle .

:UrbanTrain
     a   owl:Class ;
     rdfs:comment "Train Concept"@en ;
     rdfs:label "Train"@en ;
     rdfs:subClassOf :Train .

:UrbanTrain01
     a :UrbanTrain ;
     rdfs:comment "UrbanTrain01 operates in zone B"@en ;
     rdfs:label "UrbanTrain01"@en .
```

Using the *UniSer* service, the excerpt of the ontology is generated according to a unique serialization. The following listing depicts the result after the serialization by UniSer (which is nearly identical to the TopBraid Composer serialization).

```
@prefix     : <http://example.com/> .
@prefix owl : <http://www.w3.org/2002/07/owl#> .
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:Train        rdf:type   owl:Class ;
              rdfs:comment "Train Concept"^^xs:string ;
              rdfs:label "Train"^^xs:string ;
              rdfs:subClassOf :Vehicle .

:UrbanTrain   rdf:type   owl:Class ;
              rdfs:comment "Train Concept"@en ;
              rdfs:label "Train"@en ;
              rdfs:subClassOf :Train .

:UrbanTrain01 rdf:type :UrbanTrain ;
              rdfs:comment "UrbanTrain01 operates in zone B"@en ;
              rdfs:label "UrbanTrain01"@en .
```

Table 7.1: **Ontology Description**. Ontologies of different sizes, described in terms of number of triples, subjects, properties, and objects.

Ontology	# triples	# subjects	# properties	# objects
Synthetic ontology	16	6	4	10
DBpedia Ontology	30,793	3,986	23	16,807
Gene Ontology	1,540,109	266,919	49	473,227

7.5 Empirical Evaluation

In this section, we present the results of an experimental study investigating the effectiveness of the *SerVCS* approach. The goal of the experiment is to analyze the impact of the ontology size, type of ontology changes, and sorting criteria on the behavior of *SerVCS*. For this purpose, we assess the following research questions:

- **RQ1** Does the size of the ontology have an impact on the behavior of *SerVCS*?
- **RQ2** Is the effectiveness of *SerVCS* affected by the different sorting criteria?

The experimental configuration to evaluate these research questions is as follows:

Ontologies: We compare the behavior of *SerVCS* using three ontologies of different sizes. Table 7.1 describes these ontologies w.r.t. number of triples, subjects, properties, and objects.

- **Synthetic ontology**: Synthetically generated small-size ontology composed of 16 RDF triples with six different subjects, four properties, and ten objects.
- **DBpedia Ontology**[7]: Medium-size ontology composed of 30,793 RDF triples. This ontology is used to describe Wikipedia infoboxes in DBpedia.
- **Gene Ontology(GO)**[8]: Large-size ontology composed of 1,540,109 RDF triples. GO describes molecular activities and relationships among genes.

Ontology Change Generation: The number of ontology changes are randomly generated following a *Poisson distribution*, i.e., we simulate ontology changes performed by users assuming that these changes obey a *Poisson distribution*. The parameter λ indicates the average number of ontology changes per time interval, i.e., $\lambda = 2$ simulates that in average two ontology changes are performed per hour. Figure 7.5 illustrates the number and types of ontology changes performed by two users during eight hours. To ensure that our evaluation represents as much as possible a real scenario, a list of basic changes typically performed in ontology development is utilized (cf. Table 7.2). These changes are randomly chosen following a *uniform distribution* with replacement. We consider the change type *Modification* to be a combination of *Deletion* and *Addition*.

Metrics: We report on the number of conflicting lines (**NCL**). It is computed as the number of conflicts indicated by Git during the merge process of two versions of the ontology after each hour, and corresponds to the cardinality of the list of conflicts LC (cf. Definition 5).

Gold Standard: We compute the gold standard by summing up the number of conflicting

[7] http://wiki.dbpedia.org/services-resources/ontology/
[8] http://www.geneontology.org/

Table 7.2: **Ontology Changes**. Basic changes of type *Addition*, *Modification* and *Deletion*performed during ontology development process as well as their respective examples.

ID	Change Type	Description	Example
CH1	Addition	Adding new elements like classes and properties	Add a new class, e.g., the class *Train* with properties *rdfs:label* and *rdfs:comment*
CH2	Modification	Modifying existing elements	Modify a property value, e.g., *rdfs:label* of *UrbanTrain* class
CH3	Deletion	Deleting existing elements	Delete an instance, e.g., the *UrbanTrain01* instance if it exists

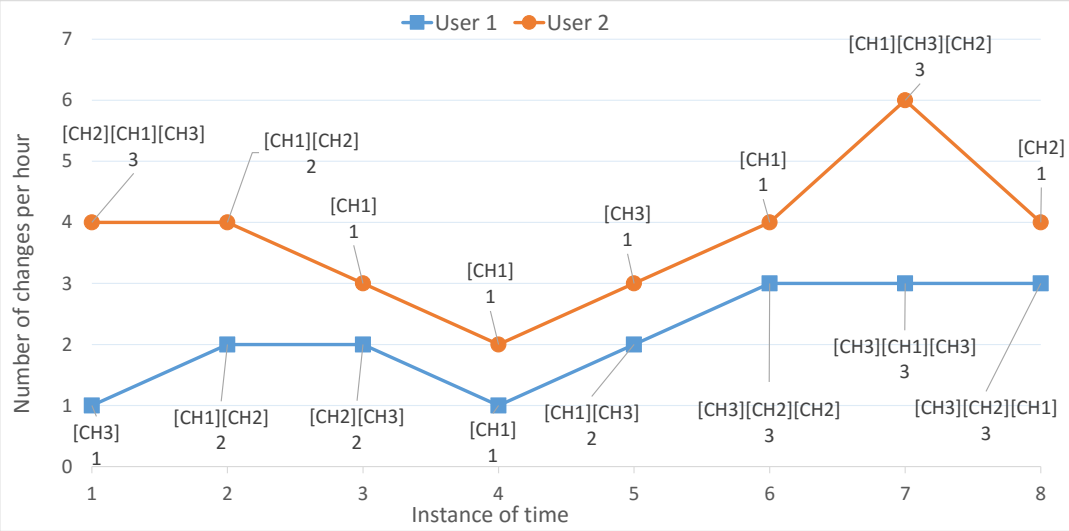


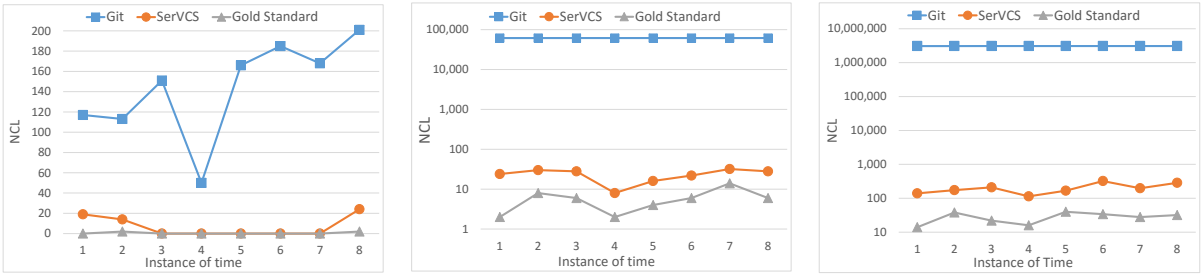
Figure 7.5: **Ontology Change Distribution**. Number and types of ontology changes (CH) per user in an interval of 8 hours. A Poisson distribution with $\lambda = 2$ models an average of two changes per hour. A uniform distribution with replacement is followed to sample the type of ontology changes (CH).

lines (**NCL**), which corresponds to the cardinality of overlapping changes made by users in a particular hour (cf. Definition 2).

Implementation: Experiments were run on a Linux Ubuntu 14.04 machine with a 4th Gen Intel Core i5-4300U CPU, 3MB Cache, 2.90GHz with 8GB RAM 1333MHz DDR3. *SerVCS* is implemented using Node.js version 4.4.5. The *syntax validation* is realized using Rapper version 2.0.15 whereas the *unique serialization* is performed using RDF-toolkit version 1.4.0.1 and Rapper respectively. The used Git version is 1.9.1. The ontology change generator is implemented using RStudio version 0.99.902[9].

Method: In order to answer the research questions, ontology changes of two users are simulated; two different ontology editors are assumed. *User 1* works with TopBraid, whereas *User 2* works with Protégé. Two scenarios are evaluated: 1) Users work purely with the functionalities of Git. 2) *SerVCS* along with Git as VCS is used. Log of ontology changes is kept during the

[9] https://www.rstudio.com/products/RStudio/



(a) Synthetically Generated Ontology (b) DBpedia Ontology (log. scale base 10) (c) Gene Ontology (log. scale base 1,000)

Figure 7.6: **Impact of Ontology Size on SerVCS**. Number of conflicting lines (**NCL**) detected by Git and *SerVCS* compared to the Gold Standard based on the Ontology Change Distribution in Figure 7.5. (a) *SerVCS* detects the same NCLs as the Gold Standard in five instances of time in the Synthetic Ontology; (b) *SerVCS* indicates up to three orders of magnitude less NCLs than Git in the DBpedia Ontology; (c) *SerVCS* indicates up to four orders of magnitude less NCLs than Git in the Gene Ontology. *SerVCS* is not equally affected as Git.

experiment. In total, users make 30 changes: 11 additions, 9 modifications, and 10 deletions. The distribution of ontology changes per user is simulated with the Poisson distribution shown in Figure 7.5. A log of ontology changes is available on GitHub[10].

7.5.1 Impact of the Ontology Size

For answering research question **RQ1**, we follow the above described method to evaluate the behavior of plain Git and *SerVCS* with three different ontologies (cf. Table 7.1). Figure 7.6 shows the number of conflicting lines (NCL) in the *Gold Standard*, as well as the ones detected by Git and *SerVCS*. In the three ontologies, NCL values are significantly less in *SerVCS* than in Git. Moreover, in the Synthetic Ontology (small-size ontology), the NCL values are the same in *SerVCS* and *Gold Standard* for five time instances. In the DBpedia Ontology (medium-size ontology), *SerVCS* indicates up to three orders of magnitude less NCLs than Git. Finally, *SerVCS* reports up to four orders of magnitude less NCLs than Git in the Gene Ontology (large-size ontology). Git utilizes a line-based algorithm for the comparison of ontology changes conducted by users. As the size of an ontology increases and modified ontologies are sorted differently, the number of compared ontology lines also increases. Therefore, Git performance is deteriorated as shown in Figure 7.6. On the other hand, *SerVCS* compares pairs of changes ontologies also line-wise, but both documents are sorted using the same criteria and the space of potential conflicting lines in *SerVCS* is smaller. Thus, as shown in Figure 7.6, *SerVCS* is not equally affected as Git from different ontology sizes. However, *SerVCS* may also wrongly identify conflicts, i.e., NCL values are not the same in *SerVCS* and *Gold Standard*. This behavior of *SerVCS* happens when users concurrently modify the subject or predicate of an RDF triple, i.e., a non-syntactical conflict is generated and Theorem 1 is not satisfied.

7.5.2 Impact of the Sorting Criteria

With the goal of answering research question **RQ2**, the experimental method is also followed when *SerVCS* utilizes different sorting criteria, i.e., RDF-toolkit and Rapper are used to

[10] https://github.com/lavdim/unistruct

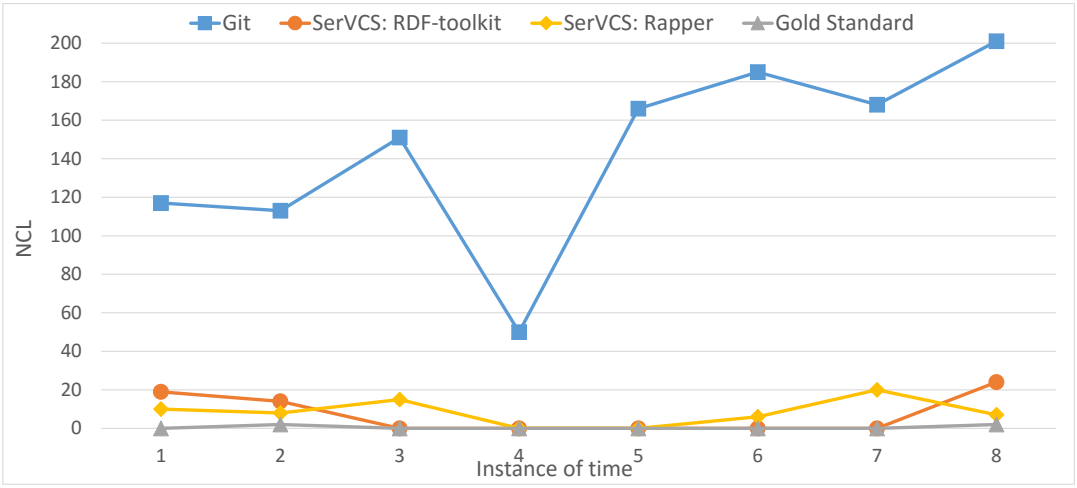


Figure 7.7: **Impact of Sorting Criteria**. Number of conflicting lines (**NCL**) detected by Git and *SerVCS* compared to Gold Standard. Synthetic Ontology is modified according to the Ontology Change Distribution in Figure 7.5. *SerVCS* follows two different sorting criteria produced by RDF-toolkit and Rapper. *SerVCS* exhibits similar behavior in both sorting criteria.

generate unique serializations. RDF-toolkit sorts triples based on ontological concepts; RDF triples of classes, properties, and instances are categorized, and then, RDF triples are ordered alphabetically within category. Rapper simply sorts RDF triples in alphabetical order. Results in Figure 7.7 show that *SerVCS* exhibits similar behavior in both sorting criteria and is able to identify the same NCL values as the *Gold Standard* in several instance times. Moreover, *SerVCS* also outperforms Git independently of the sorting criteria.

7.6 Summary

In this chapter, we present *SerVCS*, a generic approach for the realization of optimistic and tool-independent ontology development on the basis of Version Control Systems. As a result, VCSs become *editor agnostic*, i.e., capable to detect actual changes and automatically resolve conflicts using the *built-in* merging algorithms. We implemented and applied the *SerVCS* approach on the basis of the widely used Git as version control. In addition, we developed a middleware service to generate a unique serialization of ontologies before they are pushed to the remote repository. The unique serialization ensures that ontologies have always the same serialization in the remote repository, regardless of the used ontology editor. Thus, we avoid incompatibility problems with regard to wrongly detected conflicts resulting from the use of different ontology editors, and assist ontology developers to collaborate more efficiently in distributed environments.

To study the effectiveness of *SerVCS* compared to Git, we perform an empirical evaluation. The results suggest that *SerVCS* reduces the number of false-positive conflicts when different ontology editors are utilized concurrently during the development process. Additional experiments are conducted to evaluate the impact of the ontology size, as well as different sorting criteria. Based on the achieved results, we conclude that the effectiveness of *SerVCS* is less impacted compared to Git whenever the size of the ontologies is increased or different sorting criteria are used to generate unique serializations.

CHAPTER 8

A Dependency-aware Approach for Test-driven Ontology Development

In this chapter, we present *EffTE*, an approach for efficient test-driven ontology development. This approach is designed to address the requirement **P5** defined in Subsection 4.2.2 to efficiently ensure that ontology changes performed by different stakeholders are conform to domain requirements and do not have any non-intended consequence.

The development of domain-specific ontologies requires joint efforts among different groups of stakeholders, such as ontology engineers and domain experts. Functional requirements can be expressed through *Competency Questions*, which are questions that the underlying ontology should be able to answer [4]. However, the concurrent definition of ontology concepts often results in a violation of the defined requirements, or creates design issues like duplicate entries or missing documentation. For example, in the large and monolithic DBpedia ontology, version 2016-04[1], only 556 from a total of 2,849 properties have an associated label description or a comment [114]. To ensure that any ontology modification has only the expected effects, a set of test cases can be defined based on *Competency Questions*. This is similar to the principles of *test-driven software development*, where test cases (which represent requirements) are defined before the code is actually written [23].

On the contrary, the ontology as the main artifact is subject of change due to number of reasons, such as 1) the evolution of the domain; 2) changes in the user perception of the domain; or 3) fixing design issues [115, 116]. Since much effort is invested in creating and extending ontologies, it is crucial to make ontology development and maintenance cost-effective [117]. However, with a naive approach, test cases are exhaustively evaluated to identify any non-intended modification or design issue introduced during the concurrent development of the ontology. While the number of test cases can be large and their evaluation time may be high, the ontology development process can be impacted negatively. The evaluation should be user based, such that for a number of test cases associated to a specific user of group of users, the evaluation is instantiated only for them. Moreover, the evaluation of test cases has to be file or module dedicated considering the fact that ontology may comprise several files or modules.

We present *EffTE*, an approach for efficient test-driven ontology development. *EffTE* relies on a dependency graph defined by the stakeholders, i.e., ontology engineers, according to their needs; it enables prioritization and selection of test cases to be evaluated. Traversing the dependency graph is performed using *breadth-first* search; *tabu* test cases are tracked and ignored for further

[1] http://wiki.dbpedia.org/services-resources/ontology/

evaluation because of faulty parents. As a result, the number of test cases that are evaluated is minimized, thus reducing the time required for ontology validation after each modification. We conduct an empirical evaluation to determine the efficiency of our approach under different conditions, such as changing the number of predefined test cases, ontology sizes, and dependency relationships of test cases. Experimental results suggest that our approach is more efficient than a naive one, in particular with an increasing ontology size and number of test cases.

In this chapter, we address the following research question:

> **RQ4**: How can the quality and efficiency in distributed and heterogeneous ontology development be ensured?

Contributions of this chapter are summarized as follows:

- A formal definition for an approach to efficiently evaluate of a set of test cases derived from the domain requirements;
- An implementation of the *EffTE* approach on top of Git and VoCol to prevent contributors from submitting non-indented changes;
- An empirical evaluation to asses the efficiency of *EffTE* with regard to time needed and the number of executed test cases in three different scenarios, by changing: 1) the number of test cases; 2) the ontology size; and 3) the dependency between test cases.

This chapter is based on the following publications:

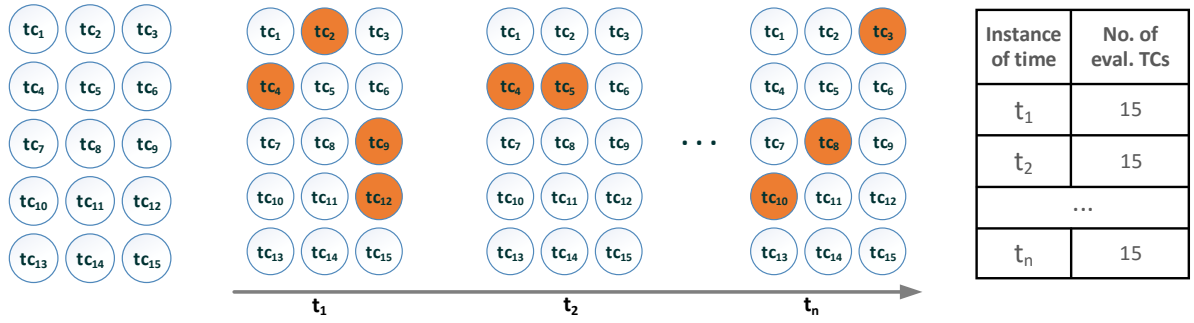
- **Lavdim Halilaj**, Irlán Grangel-González, Steffen Lohmann, Maria-Esther Vidal, Sören Auer. *EffTE: A Dependency-aware Approach for Test-Driven Ontology Development.* In 33rd ACM/SIGAPP Symposium On Applied Computing (ACM SAC) 2018 Proceedings, ACM. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I devised the formalization of the problem, led the definition and implementation of the proposed approach, reviewed related work, and prepared of the experiments and analysis of the obtained results;

- **Lavdim Halilaj**, Irlán Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *DemoEffTE: A Demonstrator of Dependency-aware Evaluation of Test Cases over Ontology.* In 13th International Conference on Semantic Systems (Semantics) - Posters and Demo Track, 2017. This demonstration article is joint work with Irlán Grangel-González, a PhD student at the University of Bonn. In this article, I conducted the description of the architecture, implementation and demonstration of the prototype.

The remainder of this chapter is organized as follows: Initially, the motivating example is presented in Section 8.1. The problem is described in Section 8.2 whereas Section 8.3 defines the *EffTE* approach. In Section 8.4, we outline the implementation of *EffTE*. The approach is evaluated in different scenarios in Section 8.5 and the Section 8.6 summarizes the work.

8.1 Motivating Example

Suppose a team composed of ontology engineers and domain experts work together to develop an ontology for a specific domain. After the *requirement gathering* phase, they define questions that the ontology should be able to answer, e.g., *List attribute types of particular concept where*



(a) A set of TCs (b) Faulty TCs over time (c) Evaluated TCs

Figure 8.1: **Motivating Example**. a) Test Cases (TCs) to check if an ontology satisfies the design requirements following an entailment regime; b) faulty test cases (orange) over time after each ontology modification; and c) number of evaluated test cases per instance of time. Every instance of time, the set of TCs is completely evaluated.

a condition is fulfilled or *there should be at least 20 classes that represent the most basic concepts of the domain* and individually enumerate them. Each class should have at least two subclasses, providing further details about the intended domain. Furthermore, to achieve a basic quality of the ontology, they define several guidelines to be followed by each team member. These guidelines include generic constraints that should not be violated during ontology development, e.g., "*all concepts must have at least one* `rdfs:label` *in English*" or "*all concepts must not share the same English* `rdfs:label`". As initial step, ontology engineers define a set of 15 test cases as depicted in Figure 8.1(a). This set ensures that each change made to the ontology is in line with the requirements and has only the intended effects according to a given entailment regime. Moreover, it prevents team members from violating the already defined constraints. Figure 8.1(b) shows several faulty test cases after each modification of the ontology at different instances of time t_i. For example, in the following time instances t_1: tc_2, tc_4, tc_9, tc_{12}; in t_2: tc_4, tc_5; and t_n: tc_3, tc_8, tc_{10} have failed, respectively. Since there exists no specified dependency relationship between the test cases, all of them will be evaluated at every instance of time t_i. This process is time-consuming, in particular when test cases have a natural dependency relationship. For instance, checking for duplicate labels should not be performed if *none* of the concepts defined so far do not have labels associated to them; or checking for the existence of subclasses should not be performed before the respective classes are defined. Additionally, there can be other cases where the dependency between test cases is more project oriented, such as one test case checks for a particular concept or resource; in case of satisfaction, another test case verifies its properties or relations to another concept or resource. Moreover, a number of test cases can be valid to only a contributor or group of contributors, therefore the evaluation process should be associated with them. In addition, since ontology may comprise several files, it should be possible to assign test cases to be evaluated only for particular files.

8.2 Problem Definition

Test Suite (TS): A test suite *TS* represents a set of test requirements $\{r_1, \ldots, r_n\}$ that need to be satisfied during the development of an ontology O according to a given *entailment regime*. A test-driven development approach allows for an efficient validation of test cases that comprise a test suite *TS*. Our test-driven development approach *EffTE* is able to identify the minimal

Table 8.1: **EffTE Notations**. Symbols and their respective definitions used to describe the test-driven development approach *EffTE*.

Symbols	Description
TS	Test suite with a set of requirements $\{r_1, \ldots, r_n\}$
O	Ontology
tc	A Test Case represented as a SPARQL query
ϕ	Entailment Regime
$\sigma(r_i \mid STC, O, \phi)$	Set of test cases in STC associated with a requirement r_i over O using ϕ
$[[tc]]^{\phi}_{\mathrm{O}}$	Set of mappings resulting of evaluating tc over O following ϕ
Fault($TS \mid STC, O, \phi$)	Requirements in TS not achieved in O using ϕ because of faulty test cases in STC
$\mathcal{F}(STC' \mid STC, O, \phi)$	Test cases in STC - STC' that fail as a consequence of failures of test cases in STC'
$\mathcal{T}(STC \mid O, \phi)$	Faulty test cases in STC over O following ϕ
$\mathbb{F}(STC' \mid STC, O, \phi)$	Fault Detection Effectiveness of STC' given O using ϕ and STC; is a *higher is better measure* that takes values in [0.0;1.0]
$TCG^{\phi}_{O} = (STC, E)$	A dependency graph in STC over O using ϕ; dependencies between test cases are represented in set of edges E

number of test cases required to be evaluated in order to determine if an ontology does not satisfy a test suite TS based on the given *entailment regime*. The following definitions state the core concepts required to formulate the problem tackled by *EffTE*; Table 8.1 summarizes symbols used for representing *EffTE* notations and definitions.

Test Cases (TCs): A test case corresponds to the formal specification of checks required to validate if the requirements in a TS are satisfied in O according to a *entailment regime*. Test cases can be represented using any rule language for ontologies, such as SWRL; however, we focus on test cases expressed as SPARQL queries over concepts defined in O respecting the test suite entailment regime. Given a test suite $TS = \{r_1, \ldots, r_n\}$ over an ontology O, an entailment regime ϕ, and a set of test cases STC, the function $\sigma(r_i \mid STC, O, \phi)$ represents the subset of test cases in STC that need to evaluated against O to achieve the requirement r_i following the entailment regime ϕ. The evaluation of a test case tc in STC is denoted by $[[tc]]^{\phi}_{\mathrm{O}}$. The ontology O meets the test case tc following ϕ, iff $[[tc]]^{\phi}_{\mathrm{O}}$ is different from the empty mapping $\mu_{\emptyset}$ [19]. Otherwise, tc is a faulty test case in O w.r.t. the requirement r_i such that $tc \in \sigma(r_i \mid STC, O, \phi)$ is not achieved by O following ϕ.

Faulty Test Case Identification: Given a set STC of test cases implementing a test suite TS, Fault($TS \mid STC, O, \phi$) is a set of requirements in TS associated with at least one test case tc in T whose evaluation fails in an ontology O following an entailment regime ϕ.

$$\mathrm{Fault}(TS \mid STC, O, \phi) = \{r_i \mid r_i \in TS \ \wedge \ tc \in \sigma(r_i \mid STC, O, \phi) \ \wedge \ [[tc]]^{\phi}_{\mathrm{O}} = \mu_{\emptyset}\} \quad (8.1)$$

Determining the faulty test cases from STC requires a considerable amount of time, since all the test cases need to be evaluated over the ontology O according to ϕ. Nevertheless, dependencies between test cases as well as subsumption relationships can be exploited to identify a subset of test cases whose failure implies the failure of the test cases in STC.

Given an ontology O, an entailment regime ϕ, and a set STC' of test cases in STC, the fault detection effectiveness of STC' given O, ϕ, and STC, denoted as $\mathbb{F}(STC' \mid STC, O, \phi)$, corresponds to the fraction of the number of faulty test cases from STC' divided by the number of faulty test cases in $STÇ$:

$$\mathbb{F}(STC' \mid STC, O, \phi) = \frac{\mid \mathcal{F}(STC' \mid STC, O, \phi) \mid}{\mid \mathcal{T}(STC \mid O, \phi) \mid + 1}, \text{ where} \tag{8.2}$$

- $\mathcal{F}(STC' \mid STC, O, \phi)$ is the set of faulty test cases in STC not included in STC' that fail as a consequence of failures of test cases in STC' (grey nodes in Figure 8.2(b)).
- $\mathcal{T}(STC \mid O, \phi)$ is the set of faulty test cases in STC over O following ϕ (orange and grey nodes in Figure 8.2(b)).
- $\mathbb{F}(STC' \mid STC, O, \phi)$ is a *"higher is better measure"* that takes values in [0.0;1.0]. A value close to 0.0 indicates that a small number of faulty test cases in STC can be detected from STC', whilst a value close to 1.0 expresses that the number of faulty test cases in STC' and STC are almost the same.

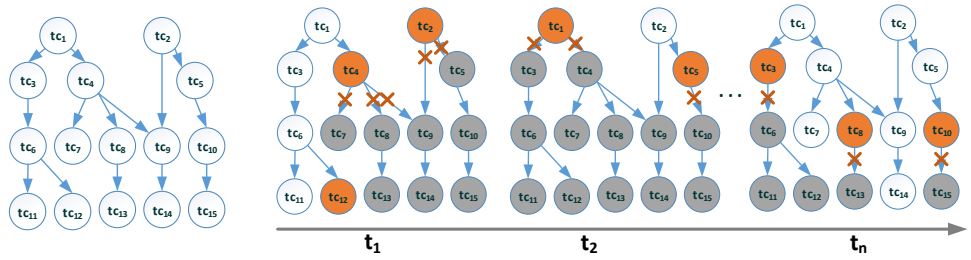
Test Case Prioritization Problem: The *EffTE* approach selects from a set of test cases STC that implement a test suite TS for an ontology O using an entailment regime ϕ, a subset STC' with maximal fault detection effectiveness $\mathbb{F}(STC' \mid STC, O, \phi)$. The problem of prioritizing test cases in STC over O and ϕ is defined as follows:

$$\underset{STC' \subseteq STC}{\text{argmax}}\ \mathbb{F}(STC' \mid STC, O, \phi) = \{tc \mid tc \in STC' \wedge \forall STC'' \subseteq STC, \mathbb{F}(STC'' \mid STC, O, \phi) \leqslant \mathbb{F}(STC' \mid STC, O, \phi)\} \tag{8.3}$$

The *set cover problem* [118] can be reduced to the problem of prioritizing test cases, thus demonstrating the NP-completeness of the test case prioritizing problem. *EffTE* implements an approach that relies on information about the dependencies between test cases to provide an approximate solution to the problem of prioritizing test cases.

8.3 The EffTE Approach

To solve the problem of prioritizing test cases in STC over an ontology O given an entailment regime ϕ, the *EffTE* approach tracks information about the dependencies between test cases in STC. The *EffTE* approach implements a graph traversal algorithm that explores the test cases STC' whose failure imply the failure of the test cases that depend on them. Furthermore, each test case contains information about to which user or group of users or for which ontology file it be evaluated. Thus, the fault detection effectiveness $\mathbb{F}(STC' \mid STC, O, \phi)$ can be maximized.



(a) A TCG_O^{ϕ} graph (b) Faulty TCs over time

Instance of time	No. of eval. TCs	F(STC'| O, STC)
t_1	7	0.66
t_2	3	0.8
...		
t_n	10	0.55

(c) Eval. TCs

Figure 8.2: **The *EffTE* Approach**. a) A test case dependency graph TCG_O^{ϕ} with 15 test cases (TCs); b) faulty test cases over time (orange nodes are faulty test cases *(STC')*, gray nodes represent TCs ignored for evaluation $F(STC' \mid STC, O, \phi)$; and c) number of evaluated TCs and $F(STC' \mid STC, O, \phi)$ values per instance of time. Dependencies enable to ignore faulty test cases.

A dependency graph of test cases: Given an ontology O following ϕ, a set of test cases STC that implement a test suite TS for the ontology O, a test case dependency graph $TCG_O^{\phi} = (STC, E)$ is a directed acyclic graph (DAG), where:

- $E \subseteq STC \times STC$ is a set of directed edges representing dependencies between test cases. If $(tc_i, tc_j) \in$ E, then the test case tc_j depends on the test case tc_i. Furthermore, tc_j fails whenever the evaluation results of tc_i or tc_j over O following ϕ return the empty mapping $\mu_\emptyset$, i.e., tc_j fails whenever $[[tc_i]]_O^{\phi}=\mu_\emptyset$ or $[[tc_j]]_O^{\phi}=\mu_\emptyset$.

The approach: Figure 8.2 illustrates the behavior of the *EffTE* approach, contrasting it to Figure 8.1. *EffTE* relies on modeling the relationships between test cases as a dependency graph $TCG_O^{\phi} = (STC, E)$ to solve the problem of prioritizing test cases. Thus, *EffTE* identifies a faulty subset *STC'* of *STC*, and removes the subadjacent graph from further consideration. The user can define different TCG_O^{ϕ} according to a given test suite *TS*. Figure 8.2(a) depicts an exemplary TCG_O^{ϕ} where tc_5 and tc_9 depend on the outcome of tc_2. If tc_2 is identified as faulty, such that $[[tc]]_O^{\phi}$ is equal to *empty mapping* $\mu_\emptyset$, the evaluation of the test cases tc_5, tc_9 will not be performed. As depicted in Figure 8.2(b), in the time instance t_1, the faulty test cases are tc_2, tc_4, and tc_{12}. This enables pruning the graph traversal, and excluding all test cases that depend on the faulty ones from further evaluation. Similarly, in subsequent time instances $t_2, \ldots, t_n$, for any faulty tc, the dependent test cases will not be considered for evaluation. As a result, the number of evaluated test cases is reduced. Figure 8.2(c) shows the number of test cases that are evaluated per invocation in the given example, e.g., seven test cases are evaluated in t_1, only three test cases in t_2, while in t_n, ten of them are evaluated.

Traversing a test case dependency graph: We used *Breadth First Search (BFS)* algorithm for traversing a TCG_O^{ϕ} to efficiently find the nodes in TCG_O^{ϕ} that are closer to the root and correspond to faulty test cases. Given a set of root test cases S and their dependencies in E, TCG_O^{ϕ} is systematically explored by the *BFS* algorithm to evaluate every reachable tc starting from S. As shown in Algorithm 1, after the initial phase of defining S, the procedure continues by recursively iterating on each tc of the TCG_O^{ϕ}. On each iteration, a tc is retrieved from the queue defined as a *FIFO* list and is set as *current_tc*. Three preconditions must be met before evaluating *current_tc*: (1) the *current_tc* must not have been already visited; (2) all of its

Algorithmus 1 : Breath-First Search with Tabu mechanism for traversing a TCG_O^{ϕ}

```
1  create lists: visitedTCs, nextTCs, tabuTCs, currentTCs;
2  add rootTCs to currentTCs;
3  while currentTCs is not empty do
4      current_tc = currentTCs.dequeue();
5      if current_tc not in visitedTCs or current_tc.Parents not in tabuTCs or
         current_tc.Parents are in visitedTCs then
6          if [[tc_i]]_O^φ ≠ μ_∅ then
7              foreach child in current_tc.Children do
8                  if child is already defined as parent then
9                      STOP the procedure to avoid a cyclic possible graph;
10                 end
11                 add child to nextTCs
12             end
13         else
14             add current_tc to tabuTCs; continue;;
15         end
16         if current_tc.FileName = OntologyFileName and current_tc.User = UserName
             then
17             evaluate current_tc;
18             add current_tc to visitedTCs;
19     else if current_tc.Parents in tabuTCs then
20         add current_tc to tabuTCs;
21     else if current_tc.Parents not in visitedTCs then
22         add current_tc to pendingTCs;
23     if currentTCs.dequeue() is null AND (nextTCs is not empty or pendingTCs is not
         empty) then
24         add nextTCs to currentTCs;
25         pendingTCs to currentTCs;
26         nextNodes.clear(); pendingNodes.clear();
27     end
28 end
```

parents have already been visited; and (3) none of its parents has already been classified as *tabuTC*. In case that $[[tc]]_{\mathrm{O}}^{\phi}$ is equal to *empty mapping* $\mu_{\emptyset}$, the *current_tc* is added to the *tabuTCs* list which represents $\mathcal{F}(STC \mid O, \phi)$. Additionally, the *current_tc* is checked whether the user who is invoking with it and the ontology file are associated with it. If these preconditions are met, the algorithm continues with the evaluation of *current_tc*. To avoid any possibility on entering in a endless cycle, the children of *current_tc* are checked whether they are previously classified as *parents* which will cause immediate stop of the procedure. Otherwise, its children are added to the *nextTCs* list to be considered in the next iterations. The iteration over the *currentTCs* list continues until there is no *tc* left. Next, all nodes that exist in the *nextTCs* or *pendingTCs* lists, will be added to the *currentTCs* list for future evaluation. The execution of the algorithm terminates when there is no *tc* left in *currentTCs*, *nextTCs*, or *pendingTCs*.

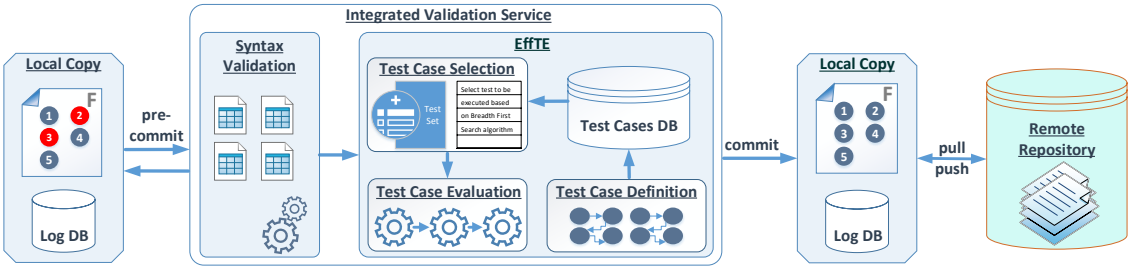


Figure 8.3: **Implementation of *EffTE***. A locally modified ontology file is received, as well as a set of test cases and their dependencies; the result of the validation process is returned as output. An *Integrated Validation Service* validates syntactic errors in an ontology using a *Syntax Validation* component. A *Test Case Validation* component checks the ontology against a set of test cases. The ontology is synchronized with a *Remote Repository* for the distribution of changes.

8.4 Implementation

The *EffTE* architecture depicted in Figure 8.3 consists of three main components: 1) the *Version Control System (VCS)* manages ontology versions via change logs; 2) the *Integrated Validation Service* performs a syntax validation of the ontology and a validation of the ontology against the *STC*; and 3) the *Repository Hosting Platform* stores and propagates ontology changes to other users. *EffTE* is part of VoCol architecture described in Figure 6.1.

8.4.1 Version Control System

Managing versions of the ontology files is realized using *Git* as a Version Contro System (VCS). The Git *hook* mechanism is utilized to prevent users from committing changes to an ontology file without successfully passing a complete set *STC*. During the committing phase, the *pre-commit* hook is triggered which transfers the modified ontology files to the validation service. The commit is terminated if the modified ontology files are unable to pass the validation process, and the user is notified with an error message comprising the failure details. On the other hand, the commit is successfully completed if no error occurs during the validation process. As a result, a new revision of the modified ontology files is created and the user is able to further proceed with pushing the new version into the remote repository.

Remote hosting platform: *GitHub* is used as remote hosting platform for saving different versions of the ontology as well as tracking and propagating the changes among users. This facilitates the synchronization of various replicas of the ontology files.

8.4.2 Integrated Validation Service

We implemented *EffTE* as a component within Integrated Validation Service (IVS). The *IVS* accepts the ontology files as input through an HTTP interface and returns to the client either an error message from the validation process or a successful passing message. Next, the *EffTE* component validates the ontology file against a set of test cases. The TCG_O^{ϕ} is defined by the user as an *Adjacency List* collection and stored in the *Test Cases DB* module. The *Test Case Selection* is responsible for prioritization and selection of the test cases to be evaluated. Each test case is evaluated in the *Test Case Evaluation* module on a given entailment regime considering

Table 8.2: **Ontology Description**. Different ontology sizes in terms of number of triples, subjects, properties, and objects.

Ontology	# triples	# subjects	# properties	# objects
FOAF	631	86	15	192
Schema.org	8,103	1,569	13	3,545
DBpedia	30,793	3,986	23	16,807

the information whether test case is associated with the user who is conducting changes or the file on which changes are performed. In case of failure a flag is added to the response message, which prevents users from realizing the commit. Additionally, users receive details regarding the faulty test cases. Otherwise, a validation message notifying the user for correctness of the recent changes is provided and the commit is successfully applied.

8.5 Empirical Evaluation

We present the results of an experimental study realized with the objective to investigate the efficiency of the *EffTE* approach. The goal of the experiment is to analyze the impact of: 1) ontology size; 2) various dependency graph TCG_O^{ϕ}; and 3) number of test cases on the efficiency of *EffTE*. We assess the following research questions:

- **RQ1** How does the ontology size impact *EffTE* efficiency?
- **RQ2** How do different dependency graphs TCG_O^{ϕ} affect *EffTE* efficiency?
- **RQ3** How does the number of test cases affect the *EffTE* efficiency?

The experimental configuration is defined as follows:

Ontologies: We compare the behavior of *EffTE* using three ontologies of different sizes. Table 8.2 describes information about these ontologies with regard to number of triples, different subjects, properties, and objects.

- **FOAF**[2]: used to describe persons, activities, and relationships between them and other objects. Further, *social networks* can be described even without a central database.
- **Schema.org**[3]: describes entities, actions, and their relations with the objective of promoting the usage of the structured data in Web.
- **DBpedia**[4]: is a manually created cross-domain ontology which allows for the description of the information extracted from Wikipedia infoboxes.

```
PREFIX rdf: <http://.../22-rdf-syntax-ns#>
PREFIX rdfs: <http://.../rdf-schema#>

SELECT * WHERE { ?s ?p ?o .
            FILTER(?o = <some-resource>) }
```

Listing 8.1: **A simple Test Case**. A SPARQL SELECT query, where <some-resource> is replaced to generate a faulty test case according to the configuration settings.

[2] http://xmlns.com/foaf/spec/
[3] http://schema.org/
[4] https://wiki.dbpedia.org/services-resources/ontology

Test case definition: With the objective of achieving a fair comparison, and avoiding any impact on the evaluation time that arises from the complexity of the query itself, we use the query depicted in Listing 8.1. It represents a simple *SPARQL SELECT* query where the variable *<some-resource>* is replaced with a particular non-existing concept or instance of some concept to artificially simulate its failure for experimental purposes.

Dependency graph generation: Different dependency graphs TCG_O^{ϕ} are randomly generated using *randomDAG* function of the *RStudio* platform. The parameter *prop* denotes the probability of connecting a test case to another test case with higher topological ordering i.e., $prop = 0.2$, generates TCG_O^{ϕ} where test cases have an average of two parent test cases.

Test case failure generation: We randomly generate the number of faulty test cases following a *Poisson distribution*, i.e., test cases that fail after each change realized by users presuming that these failures occur according to a *Poisson distribution*. The parameter λ indicates the average number of faulty test cases after each ontology change, i.e., $\lambda = 2$ simulates that in average two faulty test cases after each change. Following a *Uniform distribution* with replacement, various test cases are randomly chosen to fail.

Metrics: We report on the following metrics: 1) evaluation time (**ET**), which corresponds to the time required for evaluation of a *STC*; and 2) number of evaluated test cases (**ETC**).

Baseline: Represents: 1) time required for evaluation of a *STC* by a naive approach; and 2) the maximum number of evaluated test cases, which corresponds to the number of test cases evaluated in a naive based approach. The naive approach corresponds to the exhaustive evaluation of all the test cases, where neither order nor dependency information is considered.

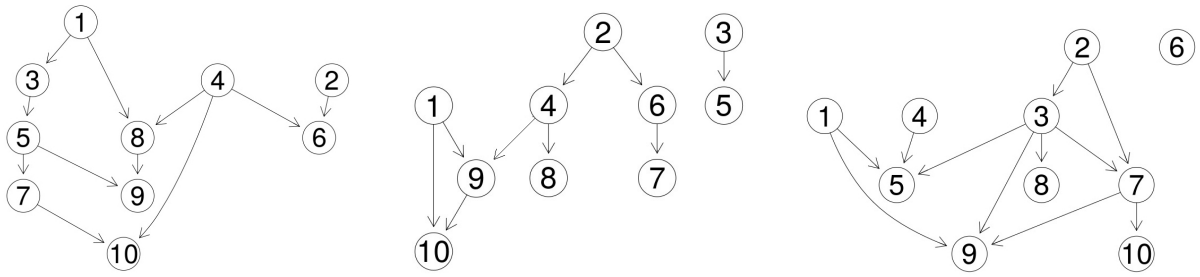
Implementation: Experiments are run on a Linux Ubuntu 16.04 machine, with a 6th Gen Intel Core i7-6820HQ, 2.70GHz, and 16GB RAM 2133MHz DDR4. *EffTE* is implemented using Node.js version 4.4.5. The *Syntax Validation* is implemented using Rapper version 2.0.15 whereas the *Test Case Validation* component using Java Apache Jena libraries version 3.0.1. Git version is 2.7.4 is used as the *VCS*. Generation of the faulty test cases for different scenarios is realized with functions for *Poisson Distribution* and *Uniform Distribution* in RStudio version 1.0.136[5].

Method: Three different scenarios are defined to answer the above mentioned research questions by changing: 1) the size of the ontology; 2) the topology of TCG_O^{ϕ}; and 3) the number of test cases. On each scenario, experiments are run twice in each time instance *(TI)*, using the naive and *EffTE* approach. In time instances *TI-1:4*, different test cases are randomly chosen to fail following a *Uniform distribution* with replacement. We enforce a *zero-faulty tc* in *TI-5*, to compare the efficiency of the naive approach and *EffTE* in similar conditions. Table 8.3 illustrates the configuration of each scenario. For first scenario, as a dependency graphs TCG_O^{ϕ} is the one depicted in Figure 8.4(a). For second scenario, three different dependency graphs TCG_O^{ϕ} with 10 test cases each are generated as depicted in Figure 8.4. Finally, for third scenario, three different dependency graphs TCG_O^{ϕ} with: a) 10; b) 20; and c) 30 test cases, respectively are generated (cf. Figure 8.5). The *RDFS* entailment regime is used in all the scenarios.

[5] https://www.rstudio.com/products/RStudio/

Table 8.3: **Experimental Set-Up**. Three scenarios varying: Ontology sizes, dependency graph topology, and number of test cases (TCs). Different faulty TCs are generated for first four instances of time (TI), whereas in TI-5, zero faulty TCs are enforced.

Scen.	Ontology	No. ofTCs	Faulty TCs per TI				
			TI-1	TI-2	TI-3	TI-4	TI-5
1	Foaf	10	3, 4, 7	3	1, 5, 7, 9	2, 7, 5, 8	-
	Schema.org						
	DBpedia						
2	DBpedia	10	3, 4, 7	3	1, 5, 7, 9	2, 7, 5, 8	-
3	DBpedia	10	3, 4, 7	3	1, 5, 7, 9	2, 7, 5, 8	-
		20	1, 6, 12, 16, 19	2, 4, 8, 9	2, 39, 14, 20	1, 4, 19	-
		30	3, 18	7, 9, 13, 22, 28	6, 7, 10, 14, 23	2, 8, 18	-



(a) TCG_O^{ϕ} No#1 (b) TCG_O^{ϕ} No#2 (c) TCG_O^{ϕ} No#3

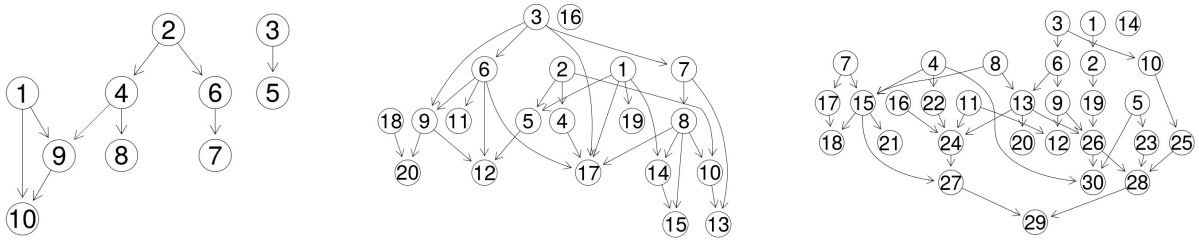
Figure 8.4: **Dependency Graph Topologies**. Different topologies of dependency graphs between, each of them comprising ten test cases.

8.5.1 Impact of the Ontology Size

To answer **RQ1**, we follow the above described method for assessing the efficiency of the naive approach and *EffTE* with three different ontologies (cf. Table 8.2). Figure 8.6(a) shows the evaluation time for a *STC* required by the naive approach and *EffTE*. In the three ontologies, the *ET* values are lower for *EffTE* compared to the naive approach in time instances *TI-1,2,3,4*. We observe that *ET* values of *EffTE* and the naive approach, in case of the *FOAF* ontology are not different. The reason for this is that *FOAF* is a small-size ontology and query execution time is relatively low. In the *Schema.org ontology* case, which is a medium-size ontology, an increased difference of *ET* values between two approaches is recognized. Finally, *EffTE* requires lower *ET* values in the first four instance of time compared to the naive approach for the *DBpedia Ontology* which more than 30K triples.

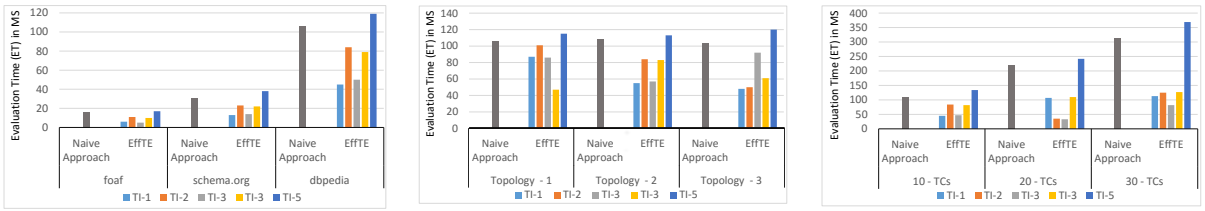
8.5.2 Impact of the Topology of TCG_O^{ϕ}

With the aim of answering the research question **RQ2**, various TCG_O^{ϕ} are defined, i.e., test cases have different dependencies between each other. As shown in Table 8.3, *Scenario 2*, three different TCG_O^{ϕ} with 10 test cases each are generated. For every TCG_O^{ϕ}, a particular number of test cases are chosen to fail. Results in Figure 8.6(b) shows that *EffTE* exhibits better performance on



(a) TCG_O^{ϕ} with 10 TCs (b) TCG_O^{ϕ} with 20 TCs (c) TCG_O^{ϕ} with 30 TCs

Figure 8.5: **Different number of test cases**. Dependency graphs having various number of test cases: a) 10 ; b) 20; and c) 30 test cases, respectively.



(a) Different ontology size (b) Different TCG_O^{ϕ} (c) Different number of TCs

Figure 8.6: **Evaluation Time (ET): Naive Approach vs *EffTE***. Impact of different scenarios on Execution Time (ET): a) Ontology size; b) dependency graph topology; and c) and number of test cases. Results suggest that *EffTE* exhibits better *ET* values in four instances of time *TI-1,2,3,4* whereas in *TI-5* it performs worse, since there is no faulty test case.

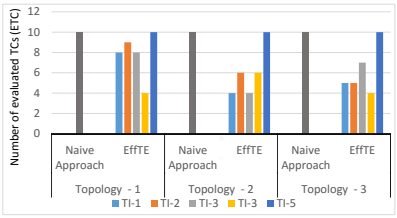
time needed to evaluate all the dependency graphs on four instances of time. The *EffTE* will require additional time in *TI-5* because of no faulty *tc*. As shown in Figure 8.7(a), the *ETC* values are less in *EffTE* compared to the naive approach in the first four instances of time.

8.5.3 Impact of the Number of the Test Cases

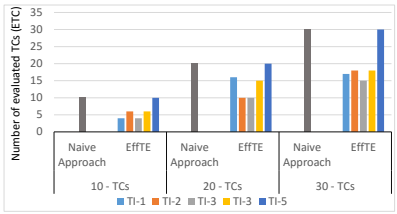
We define *Scenario 3* (cf. Table 8.3), to answer **RQ3**. This scenario comprises three TCG_O^{ϕ} composed of 10, 20, and 30 test cases, respectively. Similar to other scenarios, for each TCG_O^{ϕ}, a number of test cases are chosen to fail. From the Figure 8.6(c), we can observe that with the increase number of test cases, *EffTE* exhibits better performance on the time needed for the evaluation of a TCG_O^{ϕ} in time instances *TI-1,2,3,4*. As expected, *EffTE* requires a bit more time in *TI-5* than the naive approach in case of absence of faulty test cases. Additionally, Figure 8.7(b) depicts the *ETC* values of two approaches for different number of predefined test cases. We note that *ETC* values are considerably lower in *EffTE*, in first four instances of time where faulty test cases exists.

8.5.4 Discussion

As we can observe, in the first four instances of time of each scenario, *EffTE* is performing better than the naive approach. However, as shown in *TI-5*, *EffTE* may also need more time for evaluation of test cases, i.e., *ET* values are higher in *EffTE*. This behavior occurs when there are no faulty test cases and additional workload is only used for traversing a TCG_O^{ϕ}. Although the test case to be evaluated is intentionally chosen to be the "simplest" one, the reported time from



(a) Different topology of dependency graphs



(b) Different number of Test Cases (TCSs)

Figure 8.7: **Number of Evaluated Test Cases (ETC): Naive Approach vs *EffTE***. Impact of different scenarios on *ETC*: a) Dependency graph topology; and b) number of test cases. Results suggest that *EffTE* exhibits better *ETC* values in four instances of time *TI-1,2,3,4* whereas in *TI-5* it has same *ETC* values as a naive approach, since there is no faulty test case.

the experiments shows an improvement of hundreds milliseconds. This improvement becomes more evident in cases when the size of the ontology is not small and increases progressively in time as well in cases when the ontology is defined by a team composed of several members which are contributing in parallel during the development process. Moreover, if the complexity of the test case increases, i.e., *checking for duplicate labels* particularly in the ontologies with a large number of concepts, avoiding such kind of test cases from further evaluation if *none of the defined concepts has label associated* becomes a crucial aspect in the development process. Figure 8.7 shows that *EffTE* reports lower *ETC* values, i.e., less number of evaluated test cases in time instances *TI-1,2,3,4*, whereas in *TI-5*, it has same *ETC* values with the naive approach. As a conclusion, whenever exists a faulty test case that has other test cases depending on its outcome, the overall number of evaluated test cases is considerably lower using the *EffTE* approach. In worst case, the number of evaluated test cases in *EffTE* is equal with the number of evaluated test cases in a naive based approach.

8.6 Summary

This chapter presents *EffTE*, an approach for efficient test-driven development of ontologies. *EffTE* relies on a dependency graph TCG_O^{ϕ} modeled by users according to their requirements on a given entailment regime; dependency graphs enable prioritization of the test cases. Traversing a TCG_O^{ϕ} is realized by a *BFS* algorithm along with an additional mechanism that stores *tabu* test cases, i.e., test cases to be ignored for further evaluation because of faulty parents. As a result, the number of evaluated test cases is minimized, thus the fault detection effectiveness $\mathbb{F}(STC' \mid STC, O, \phi)$ is increased and the test case validation time is reduced. We perform an empirical evaluation to study the efficiency of *EffTE* compared to a naive approach on various scenarios with different ontology size, dependency graph topologies, and number of test cases. The results suggest that in all cases where a faulty test case exists, the efficiency of *EffTE* is higher compared to a naive approach. Furthermore, additional parameters are considered, such as users who perform changes and modified ontology files (if an ontology is developed in multiple files). Thus, a user- and file- based selection of test cases is achieved which enhances the ontology development process on collaborative environments.

Part IV

Applications and Conclusions

In this part, we start with presenting of the applicability of our holistic approach to build a number of ontologies for different domains. Chapter 9 describes two approaches which aim to support the semantic interoperability in the context of Industry 4.0. Next, in Chapter 10, an approach based on ontologies for allowing data integration across heterogeneous data sources, is presented. Chapter 11 provides general information about 19 ontologies from various domains which are built using our integrated development environment. Further, this chapter describes more in detail the methodological and technical aspects of nine ontologies currently being developed and managed on the top of the VoCol platform.

This thesis is concluded in Chapter 12, by revisiting the research questions. Finally, we look into future extensions of this work from both, research and technical perspectives.

CHAPTER 9

Establishing Semantic Interoperability between Industry 4.0 Models

This chapter presents two approaches for solving interoperability issues in the respective domains. They are based on ontologies as main artifacts used to model the knowledge from domain as well as to enable interlinking between concepts and different granularity levels. The ontologies are developed using the Git4Voc methodology, in particular practices for naming convention and reuse of external ontologies. In addition, VoCol is used as an integrated environment to support the development and deployment of these ontologies.

We initially start with engineering and manufacturing domain, where currently there is an atmosphere of departure to a new era of digitized production. In different regions, ongoings initiatives are known under various names, such as *industrie du futur* in France, *industrial internet* in the US or *Industrie 4.0* in Germany. Within the German Industry 4.0 initiative, the concept of an Administrative Shell was devised to respond to these requirements. The Administrative Shell is planned to provide a digital representation of all information being available about and from an object which can be a hardware system or a software platform. Therefore, we present an approach to develop such a digital representation based on semantic knowledge representation formalisms, such as RDF, RDF Schema and OWL. Our concept of a *Semantic I4.0 Component* addresses the communication and comprehension challenges in Industry 4.0 scenarios using semantic technologies. The approach is illustrated with a concrete example, showing its benefits in a real-world use case.

We continue with addressing the interoperability from perspective of standards interlinking. A number of different standards and reference architectures have been proposed to empower interoperability in *Smart Factories*. Standards allow for the description of components, systems, and processes, as well as interactions among them. Reference architectures classify, align, and integrate industry standards according to their purposes and features. To tackle the problem of interoperability between standards, we survey the landscape of Industry 4.0 standards from a semantic perspective and develop the *STO*, an ontology for describing standards and their relations. It allows to describe characteristics of I4.0 standards and exploiting them for further classification from various perspectives according to the reference architectures. Moreover, the semantics encoded in *STO* enable relations discovery between I4.0 standards as well as mapping them across reference architectures proposed by different industry communities.

Contributions of this chapter are summarized as follows:

- An approach to digitally represent *Administrative Shell* concept based on semantic knowledge representation formalism's, such as RDF, RDF-Schema and OWL;
- An ontology to describe Industry 4.0 standards and their detailed characteristics as well as their interrelations.

This chapter is based on the following publications:

- Irlán Grangel-González, **Lavdim Halilaj**, Sören Auer, Steffen Lohmann, Christoph Lange, Diego Collarana. *An RDF-based Approach for Implementing Industry 4.0 Components with Administration Shells.* In IEEE Emerging Technologies and Factory Automation (ETFA) 2016 Proceedings, 1-8, IEEE. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn. My contributions to this article are related to the revision of the state of the art approaches, implementation of the proposed approach, the presentation of the use cases, as well as the analysis of the results;

- Irlán Grangel-González, Paul Baptista, **Lavdim Halilaj**, Steffen Lohmann, Maria-Esther Vidal, Christian Mader, Sören Auer. *The Industry 4.0 Standards Landscape from a Semantic Integration Perspective.* In 22nd IEEE International Conference on Emerging Technologies And Factory Automation (ETFA) 2017 Proceedings. This article is a joint work with Irlán Grangel-González, a PhD student at the University of Bonn and Paul Baptista, a student assistant at Fraunhofer IAIS. In this article, I contributed to the implementation of the proposed approach, reviewing of related work and analysis of the obtained results.

The remainder of this chapter is organized as follows: First, Section 9.1 presents an approach for semantification of Administrative Shells used as intermediate layer for Industry 4.0 components. Second, an ontology for describing standards, their metadata and interrelations as well as allowing interlinking with external data sources is presented in Section 9.2. Finally, Section 9.3 summarizes the ideas and contributions presented in this chapter.

9.1 A Semantic Administrative Shell for Industry 4.0 Components

The dynamic of today's world imposes new challenges to the enterprises. The globalization, the ubiquitous presence of the internet and the development of hardware systems are some of the technological improvements that provoke changes everywhere. Industry 4.0 (I4.0) is a term coined in Germany to refer to the fourth industrial revolution. This is understood as the application of concepts such as Internet of Things (IoS), Cyber-physical Systems (CPS), the Internet of Services (IoS) and data-driven architectures in the real industry. With approximately the similar meaning, in North America, the term Industrial Internet has been created. This term is very similar to I4.0, but the application is broader than industrial production. Other areas are included as well, for instance, smart electrical grids [119].

With the goal to develop the Industry 4.0 vision, CPS are of paramount importance. CPS integrate physical and software processes [120]. In order to do so, they use various types of available data, digital communication facilities, and services [121].

While the vision of digitizing production and manufacturing gained much traction lately, it is still relatively unclear how this vision can actually be implemented with concrete standards

and technologies. The physical network connection problem is meanwhile largely solved using technologies, such as *Profibus/Profinet* [122] and *OPC-UA* [123]. However, the more challenging problem is to make smart industrial devices able to communicate and *understand* each other as a prerequisite for cooperation scenarios. To address this problem, we need techniques and standards for representing and exchanging information, data and knowledge between devices participating in manufacturing and production processes. Such standards must be flexible to accommodate new features, usage scenarios, cover multiple domains, device categories, and bridge organizational boundaries. Most importantly, they must be able to evolve seamlessly over time to facilitate the swift realization of new features and scenarios as they become apparent.

Within the Industry 4.0 initiative, the concept of an *Administrative Shell* was devised to respond to these requirements. The Administrative Shell is planned to provide a digital representation of all information (and services) being available about and from a physical manufacturing component. We present an approach to develop such a digital representation based on semantic knowledge representation formalisms, such as RDF, RDF-Schema and OWL. The advantages of such an RDF-based approach are:

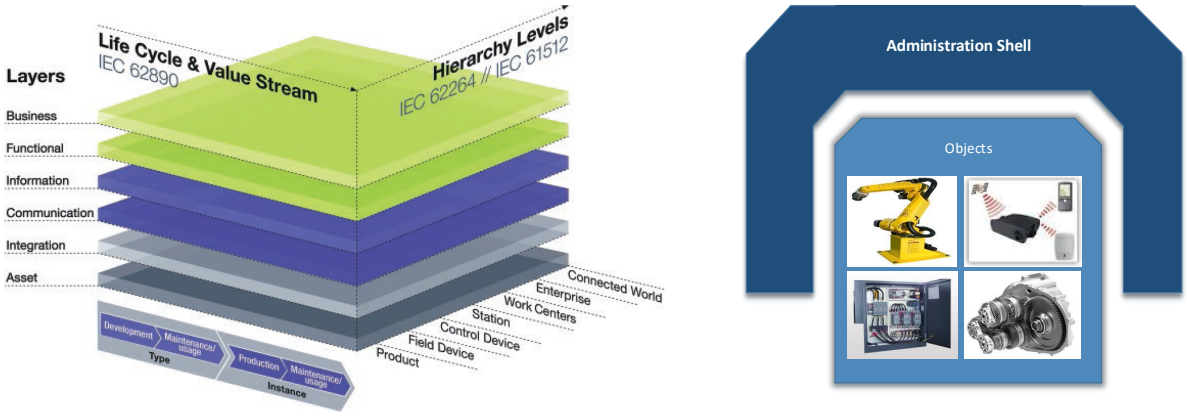
- *Identification.* The use of URI/IRIs provides a decentralized, holistic, extensible global identification scheme for all relevant entities either physical or abstract;
- *Integration.* The simple, but at the same time expressive statement-centric RDF data model enables the representation of facts, raw data, schema, provenance and meta-data information in a unified manner;
- *Coherence.* Existing and new taxonomies, vocabularies, and ontologies can be mixed and meshed to represent information about various domains, application scenarios, or organizations, in a natural way.

9.1.1 Background

The Reference Architecture Model for Industry 4.0 (RAMI 4.0) encompasses the core aspects of Industry 4.0 in a three-dimensional model [124, 125]. It illustrates the connection between IT, manufacturers/plants and the product life-cycle in a three-dimensional space. Each dimension shows a particular part of these domains divided into different layers, as depicted in Figure 9.1(a). The model extends the hierarchy levels defined in *IEC 62264/61512* by adding the concepts *Product* on the lowest level and *Connected World* at the top level, which goes beyond the boundaries of an individual factory.

The vertical axis on the left hand side of Figure 9.1(a) represents the IT perspective, comprising layers ranging from the physical device (*asset*) to complex functions, as they are available in ERP systems (*functional*). These layers correspond to the IT way of thinking, where complex projects are decomposed into smaller manageable parts. The horizontal axis on the left hand side indicates the product life-cycle where *Type* and *Instance* are distinguished as two main concepts. The RAMI 4.0 model enables the representation of data gathered during the entire life-cycle. The horizontal axis on the right hand side organizes the locations of the functionalities and responsibilities in a hierarchical structure.

Industry 4.0 Component A component is a core concept in the Industry 4.0 context. As defined in [124], an I4.0 component constitutes a specific case of a Cyber-Physical System (CPS). It is used as a model to represent the properties of a CPS, for instance, real objects in a



(a) RAMI Model. (b) Industry 4.0 component.

Figure 9.1: **Industry 4.0 Concepts**. a) Reference Architecture Model for Industry 4.0 (RAMI 4.0), comprising the three dimensions: *layers*, *life-cycle* and *hierarchy levels* (source [124]); b) An Industry 4.0 component object wrapped by an Administration Shell (adapted from [124]).

production environment connected with virtual objects and processes. An I4.0 component can be a production system, an individual machine, or an assembly inside a machine. It is comprised of two foundational elements: an object and its Administration Shell. Every object or entity that is wrapped by an Administration Shell becomes an I4.0 component, as illustrated in Figure 9.1(b). In the following, the different parts of I4.0 components are presented in more detail.

Object In [124], the term *object* is used to refer to an individual physical or non-physical entity. An object can be an entire machine, an automation component, or a software platform; it can be a legacy system or a new system. The industry should be able to integrate and benefit from these objects in I4.0 contexts, independently of their type and age.

Administration Shell The Administration Shell is used to store all important data of an object. Its goal is to create benefits for each participant in a networked manufacturing [124], including:

Data Management The Administration Shell provides mechanisms to manage large amounts of data and information generated by manufacturers and other stakeholders. This includes storing and administrating information about configuration, maintenance, and connectivity.

Functions Different functions, such as operations, maintenance tasks, or complex algorithms implementing business logic, can be provided by the Administration Shell. These functions facilitate the interaction between I4.0 components and other actors.

Services Although the information of a component is stored only once, it can be used beyond the boundaries of the component, within enterprise networks or in a cloud. The information can be made available to different users and can be accessed in various use cases.

Integration The Administration Shell, in combination with communication protocols, offers the possibility of an easy integration of I4.0 components.

Modularity Each specific part of an object should be able to store information in the Administration Shell. This ensures that all information is saved and ready to be used for subsequent analysis or various application scenarios.

9.1.2 Challenges

Developing the vision of Industry 4.0 raises new challenges which become even harder when participants in a networked manufacturing apply different policies. Some of the critical challenges of the I4.0 in general and the I4.0 component in particular, are presented in the following:

Interoperability (Ch1) On the one hand, I4.0 vision includes new ways of managing data, machines and components. On the other hand, the enterprises need to maintain a huge amount of legacy systems with their corresponding existing data. Commonly, this data is in different formats (e.g., plain text, DBMS, and XML). The new data and new formats have to coexist with the old ones.

Global unique identification (Ch2) Enabling intercommunication among I4.0 components over the Internet is a big challenge. In addition to this, there should be a linking mechanism between the I4.0 components and the generated information [126]. Therefore, addressing this challenge is of paramount importance in order to realize the vision of I4.0.

Data availability (Ch3) Another challenge is the availability of the data beyond the boundaries of the manufacturers and across different hierarchy levels. This challenge becomes even harder when various policy rules from manufacturers are applied. I4.0 components will communicate with each other and interact with the environment through exchanging the data generated from different sensors and react to the events by triggering actions with the aim of controlling the physical world [127]. Therefore, sharing the generated data between participants [128] is a key factor in the Industry 4.0.

Standardization compliance (Ch4) The Standardization process is an important step toward the realization of I4.0. Several standards to deal with different layers in the enterprises exist nowadays. For instance, *AutomationML* [129], *Profibus* [130] and *OPC-UA* [123, 131] are just some of the examples of the mentioned standards. The core idea of this effort is to provide a detailed description of the components in the manufacturing process. The production process constantly generates different components and the standards need to reflect this dynamically. As a result, the standards grow in size and number, making interoperability between them a problem to solve.

Integration (Ch5) Highly dynamic environment is one of the key obstacles to the establishment of the vision of I4.0. The complexity of horizontal and vertical integration of the I4.0 components is drastically increased with the fluctuating number of the participants. Self-sense, self-configuration and self-integration are some concepts used to describe autonomous interaction of a component with the environment in a networked manufacturing. Following the principles from Reconfigurable Manufacturing Systems (RMS), adding, removing, replacing or rearranging the components must not affect the production process [132]. Thus, developing a consistent data model is a crucial factor that facilitates the integration of I4.0 components in the changing environments.

Multilinguality (Ch6) In order to achieve a wide range of applicability to different cultures and communities [133], the I4.0 should be able to support localization (and internationalization) of the generated information. This will decrease the learning curve and allow easier and faster adoption of the Industry 4.0 in real production environment.

9.1.3 An RDF-based Approach for Semantifying I4.0 Components

Semantic technologies play a crucial role with regard to the description and management of things, devices, and services [134, 135]. Moreover, it has been recognized that I4.0 components

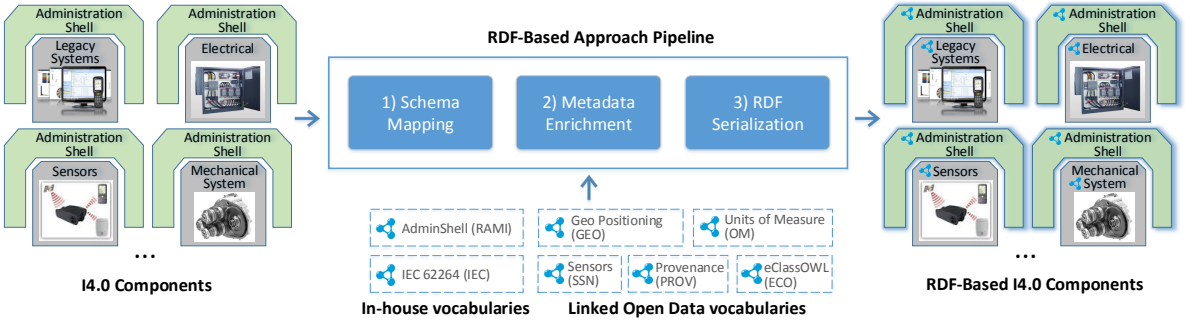


Figure 9.2: **The RDF-based Pipeline**. The pipeline for semantifying I4.0 components comprising RDF vocabularies of relevant standards to represent information about a wide range of components.

and their content should follow a common semantic model [124]. Therefore, we propose an RDF-based approach to pave the way towards a common semantic model for Industry 4.0 components, as illustrated in Figure 9.2. For representing the hierarchy levels of the RAMI 4.0 model, we created an RDF-based vocabulary conform the IEC 62464:2013 standard.

Interoperability To meet the interoperability demand, RDF and Linked Data have proven to be successful for integrating various types of data [136–138]. Embedded in the semantics for Administrative Shell, we propose RDF as a middle layer to support the interoperability between the data of the legacy systems and the data generated by the I4.0 component. We aim to establish RDF as a *lingua franca* for data interoperability in the I4.0 landscape.

Global unique identification Identification of each I4.0 component using global unique identifiers ensures entity disambiguation and retrievable [139]. According to Linked Data principles [140], HTTP URIs should be used for naming things. Following the above-mentioned principle, we propose that each I4.0 component should be identified by an HTTP URI. By doing so, a decentralized, holistic and extensible global unique identification scheme for I4.0 components is established. As a consequence, we will have derefenceable I4.0 components, which are able to self-locate and communicate with each other.

Data availability The benefits of employing RDF as the standard for representation of the data are twofold. Firstly, various data serialization formats are easy to be generated and transmitted over the network. Secondly, using SPARQL, as a query language and protocol, it is possible to make data available through a standard interface. RDF representation of the data can be created on the fly, even if they are stored in relational databases or other data formats [141]. By doing so, our approach enables data sharing between legacy systems and other participants in a networked manufacturing as well.

Standardization compliance Following the idea of employing RDF as a lingua franca for data integration, we propose to translate existing standards into RDF vocabularies and SKOS thesauri. The interoperability between standards can thus be managed through integration of the respective vocabularies. In addition, these vocabularies are also connected with the Administrative Shell data (cf. Figure 9.2). As an example, we created an RDF vocabulary for the *IEC 61360 - Common Data Dictionary* (IEC CDD)[1]. IEC CDD is a common repository of concepts for all electrotechnical domains based on the methodology and the information model of IEC 61360. It provides a widely accepted terminology and definitions based on sources, such as IEC standards as well as other industry standards. It

[1] http://std.iec.ch/cdd/iec61360/iec61360.nsf/

contains four major concepts: `Component`, `Material`, `Feature` and `Geometry`. `Component` describes an industrial product which serves a specific function. Moreover, considering a given context it should not be decomposable or physically divisible and is intended for use in a higher-order assembled product. `Component` is represented by an `Object` in RAMI Model, and when it is surrounded by the Administrative Shell forms an I4.0 component.

Integration Running on completely unified and consistent data model facilitates the integration of I4.0 components. Newly added components need a shorter time for the integration process. Other peers will be aware of new peer and the way of communication with it by simply synchronizing with the latest version of vocabulary which contains all necessary information for interaction and data exchanging between peers in a networked manufacturing.

Multilinguality Since various communities across the world will interact with I4.0 components, it is very important that they will receive terms in their own language. The semantic web technologies enable implementation of multilinguality in a very straightforward manner. This will remain valid even for the newly introduced languages or concepts.

The *RAMI* Vocabulary

Our approach defines a semantic vocabulary for the Administration Shell concept by providing an ontological formalization of the elements that describe I4.0 components. Since the Administration Shell is a key concept of the RAMI 4.0 model, we decided to use the namespace *rami* for the vocabulary also. The core classes of the vocabulary are `rami:BasicData`, `rami:AdminShell` and `rami:Object`. The class `rami:AdminShell` represents the Administration Shell concept and its properties. The objects in the RAMI 4.0 model are described by the `rami:Object` class. In addition, properties like `rami:name`, `rami:isPartOf` and `rami:description` are created to represent the characteristics and features of the object. The basic data associated with the object are represented by the `rami:BasicData` class. Different types of data, such as sensor, mechanical, electrical, or physical data, as subclasses to the `rami:BasicData` class can be further added. Further, it permits incorporation of existing models, such as the *Object Memory Model* (OMM), which supports the creation of digital memories for manufacturing objects [142]. OMM provides *blocks* for grouping data on a certain aspect of an object. These blocks contain metadata for describing the object, for instance, its ID, description, or format. In the *RAMI* vocabulary, we included some of the OMM concepts, such as identification and description type, and organize them in the `rami:BasicData` class. In this way, different types of data associated with the object, for instance, `rami:EngineeringData`, inherit attributes that have been defined for `rami:BasicData`. Additionally, they inherit attributes specifically defined for `rami:EngineeringData`, in this case, `standard name`, and `version`. The `rami:AdminShell` class is used to connect the object with its basic data. Figure 9.3 depicts the main classes and properties of the *RAMI* vocabulary.

In manufacturing processes, provenance is crucial aspect [143]. For instance, authenticating a specific product w.r.t. its manufacturer, or the date when it was manufactured, are critical information to record within the manufacturing context. For this reason, we reused the W3C *Provenance Ontology*[2] to track the creator and contributors of an object in the *RAMI* vocabulary.

With the goal of aligning the RAMI 4.0 model with the IEC 62264 hierarchy levels, we define the class `rami:RAMIHierarchyLevel`. Instances of this class represent the RAMI 4.0 hierarchy levels (e.g., `rami:Station`, and `rami:WorkCenter`). This allows to link concepts, such as the IEC 62264 *Storage Unit*, which is a type of *Work Center*, as shown in Listing 9.1.

[2] https://www.w3.org/TR/prov-o/

```
@prefix iec62264: <https://w3id.org/i40/iec/62264/> .
@prefix rami:     <https://w3id.org/i40/rami/> .

iec62264:StorageUnit  rami:RAMIHierarchyLevel  rami:WorkCenter .
```

Listing 9.1: **Alignment RAMI 4.0 and IEC 62264 concepts**. RDF representation of alignments between concepts of RAMI vocabulary and those of IEC 62264.

Units of measurement are of paramount importance in manufacturing environments to ensure correct functioning and coordination of processes. Units are required for the specification of products as well as for representing the data produced by measuring devices (e.g. sensors). Often, units are represented as simple strings, e.g., °C, mm, and kg. This has the drawback that the semantics of the units are not machine-readable and sometimes unknown or ambiguous. For example, both "18 in" and "45,72 cm" are referring to the same length.

For properly representing units, we aligned the *RAMI* vocabulary with the *Ontology of Units of Measure (OM)* [144]. This ontology provides global identifications and definitions for units of measurement, including quantities, measurements, and dimensions. By using the *in* and *cm* concepts from the OM ontology, the semantics of the units can be *understood* by a machine because their formal definitions can be looked up in the ontology via the URIs of the concepts as well as processed and interpreted by a software agent. For example, "centimetre" is defined as a unit in the dimension of length, amounting to 1/100 of the "metre" unit. Listing 9.2 illustrates how data values can be represented using the OM ontology.

```
@prefix om:   <http://www.wurvoc.org/vocabularies/om-1.8/> .
@prefix rami: <http://w3id.org/i40/rami/> .
@prefix eco:  <http://www.ebusiness-unibw.org/ontologies/eclass/5.1.4/#>
@prefix ex:   <http://example.org/data/> .

ex:object1        eco:P_BAA018001     ex:lengthOfObj1 .
ex:lengthOfObj1   om:numerical_value  "42.72" .
ex:lengthOfObj1   om:units_of_measure_or_measurement_scale
                                      om:centimetre .
rami:object2      eco:P_BAA018001     ex:lengthOfObj2 .
ex:lengthOfObj2   om:numerical_value  "18" .
ex:lengthOfObj2   om:units_of_measure_or_measurement_scale
                                      om:inch-international .
```

Listing 9.2: **Reuse of the OM ontology**. Representing length units using the OM ontology.

We consider the alignment with eCl@ss [145]. eCl@ss is a cross-industry classification system to describe products and services using unique identifiers. In the context of Industry 4.0, eCl@ss performs a crucial function by providing common definitions of a vast amount of products and services. eCl@ss is available as an RDF-based vocabulary[3] and can therefore be easily reused and aligned with our approach. To describe units of measurement, the eCl@ss vocabulary incorporates the *GoodRelations* vocabulary[4]. Since the OM ontology contains more specialized and rich descriptions for units, we propose to use both (i.e., the eCl@ss and *GoodRelations* vocabularies) jointly. Based on this, we recommend to align the eCl@ss concepts to our definitions in the *RAMI* vocabulary. Figure 9.3 illustrates the core classes and their relationships between each other in the *RAMI* vocabulary.

[3] http://www.heppnetz.de/projects/eclassowl/
[4] http://www.heppnetz.de/projects/goodrelations/

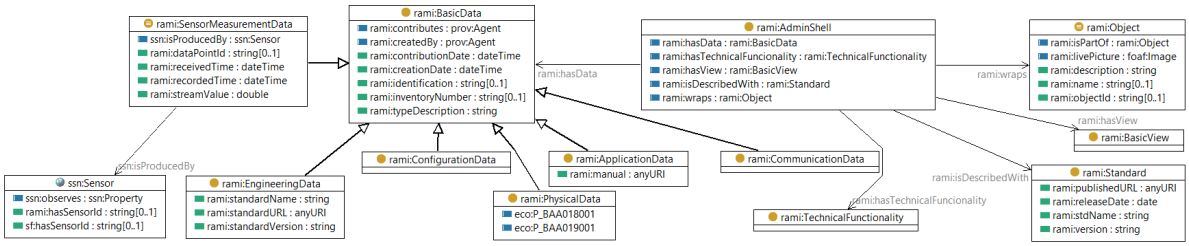


Figure 9.3: **RAMI Vocabulary**. Overview of the core classes and relationships of the *RAMI* vocabulary.

IEC 62264 Vocabulary

The RAMI 4.0 model is centered around the *IEC 62264* standard to define hierarchy levels for the manufacturing domain. Next to the hierarchy levels, this standard specifies core concepts for the development of manufacturing companies, such as work centers, production lines, and storage zones. Based on these definitions, we developed an RDF-based vocabulary that models the structure as well as the concrete semantics of these concepts.

Our RDF-based approach allows to align the information models of different companies with the proposed standard. For example, the term *Plant* is commonly used in the manufacturing world. The meaning of this term is equivalent to the term *Site* according to the standard.

Following the same idea, also other cases, i.e., expressing that one concept is broader or narrower than some other, can be addressed by reusing specialized vocabularies, such as the Simple Knowledge Organization System (SKOS) vocabulary.

9.1.4 Use Case

The objective of Industry 4.0 vision is to enable a decentralized production using smart objects that are autonomous with respect to decision-making. To accomplish this goal, object metadata, data, and relations with other objects, need to be semantically described within the Administrative Shell. By doing so, the information provided by one object can be understood and exploited by other smart objects in the production chain. To illustrate the applicability of our approach, a use case using semantic Administrative Shell to describe an I4.0 component and some of its basic relations is presented in the following. Listing 9.3 shows the semantic representation of the Administrative Shell for the *Motor controller CMMP-AS-C2-3A-M3* object (a product of Festo AG[5]). For brevity, we describe here the most relevant data of the resources. This example contains four instances of respective types. An `AdminShell1` surrounds `Object1` and connects it with the majority of the concepts in the domain, as the `Platform1` in this case. Also, `Object1` has its technical data defined on the resource `TechnicalData1` (cf. Figure 9.4).

One of the main advantages of the Semantic Administrative Shell is the uniform data representation according to the RDF model, which enables efficient integration and querying the data comprised in the shell. In order to illustrate the data retrieval, we have designed simple SPARQL queries. For example, it is relevant to know the technical characteristics of a component in its various phases (e.g. *Single, Three*). In the following query (cf. Listing 9.4), we construct an RDF graph that contains a description of the technical feature of an object with *Single-phase*.

[5] https://www.festo.com/cat/en-gb_gb/products_CMMP_AS

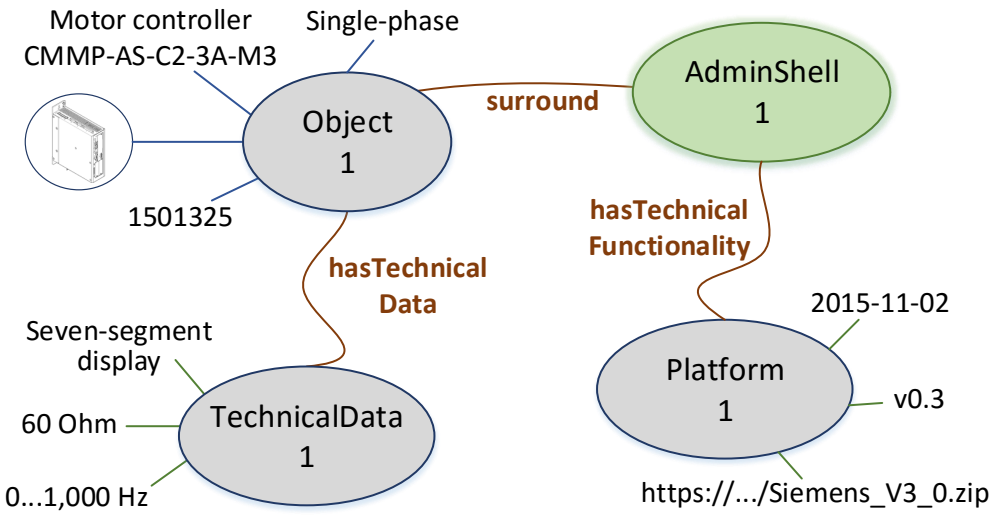


Figure 9.4: **An example of Industry 4.0 graph.** Several concepts of Industry 4.0 interlinked with each other as well as their respective attributes.

```
@prefix i40c: <http://purl.org/eis/i40c/>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

i40c:AdminShell1        a       i40c:AdministrativeShell  ;
  rdfs:label                    "AdminShell1"^^xsd:string ;
  i40c:surround                 i40c:Object1 ;
  i40c:hasTechFuncionality      i40c:Platform1 .

i40c:Object1            a       i40c:Object ;
  rdfs:label                    "Motor control..."@en ;
  i40c:hasId                    "1501325"^^xsd:string ;
  i40c:hasPhase                 "Single-phase"@en ;
  i40c:hasTechnicalData         i40c:TechnicalData1 ;
  i40c:image                    "<http://...b4dc.jpg>" .

i40c:TechnicalData1   a         i40c:TechnicalData ;
  rdfs:label                    "TechnicalData1"@en ;
  i40c:BrakingResistance        "60 Ohm";
  i40c:Outputfrequency          "0...1,000 Hz" ;
  i40c:display                  "Seven-segment display"@en .

i40c:Platform1          a       i40c:Platform ;
  rdfs:label                    "Function blocks ..."@en ;
  i40c:functionBlockUrl         "https://.../Siemens_V3_0.zip";
  i40c:hasDate                  "2015-11-02"^^xsd:date ;
  i40c:hasVersion               "3.0"^^xsd:string .
```

Listing 9.3: **The RDF representation of Administrative Shell**. The Semantic Administrative Shell for the servo motor controller CMMP-AS-C2-3A-M3.

```
@prefix i40c: <http://purl.org/eis/i40c/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

SELECT {
  ?object          rdfs:label              ?name.
  ?technicalData   i40c:brakingResistance  ?resistance .
  ?technicalData   i40c:outputFrequency    ?frequency .
} WHERE {
  ?object          i40c:hasPhase          "Single-phase"@en .
  ?adminShell      i40c:surround           ?object .
  ?object          rdfs:label              ?name.
  ?object          i40c:hasTechnicalData   ?technicalData .
  ?technicalData   i40c:brakingResistance  ?resistance .
  ?technicalData   i40c:outputFrequency    ?frequency .
}
```

Listing 9.4: **Obtaining information about a particular phase**. SPARQL query to retrieve the electrical data where the phase variable has value of *Single-phase*.

Another example is retrieving the metadata of the I4.0 component platform, during the maintenance cycle. The platform entities are referring to functional library elements, which are specific to a certain automation system. The query modeled in Listing 9.5 obtains the details of the platform like the name, version and the software URL that supports the I4.0 object.

```
@prefix i40c: <http://purl.org/eis/i40c/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

SELECT {
  ?object       i40c:hasId                    ?objectId.
  ?platform     rdfs:label                    ?name .
  ?platform     i40c:hasVersion               ?version .
  ?platform     i40c:hasDate                  ?date .
  ?platform     i40c:functionBlockUrl         ?url .
} WHERE {
  ?adminShell   i40c:surround                 ?object .
  ?adminShell   i40c:hasTechnicalFuncionality ?platform .
  ?object       i40c:hasId                    ?objectId.
  ?platform     i40c:hasVersion               ?version .
  ?platform     i40c:hasDate                  ?date .
  ?platform     i40c:functionBlockUrl         ?url .
  ?platform     rdfs:label                    ?name .
}
```

Listing 9.5: **Obtaining metadata about the platform**. SPARQL query to retrieve the version, date and URL of the software of the component's platform.

The above use case shows how the Semantic Administrative Shell provides a flexible data model. This semantic representation helps to overcome the challenges that I4.0 is facing.

9.2 A Semantic Integration Perspective for Industry 4.0 Standards

For the realization of the I4.0 vision, many organizations and stakeholders need to collaborate and interchange data and information [146]. Standardization efforts enable the adoption of production and manufacturing technologies across different organizations and stakeholders. Standards define terms, components, procedures, dimensions, and materials along with many other aspects of relevance for production and manufacturing. Standards are of paramount importance in the manufacturing and automation domain and for the realization of the I4.0 vision where different enterprises and stakeholders interact in an interoperable manner [147].

Although standards are regionally extended or adapted, they represent a common understanding of terms for a wide range of practitioners from the industry. There exist standards that describe the enterprise as a whole, while others have been created to deal with specific problems, such as PLC programming or automation design. Relevant examples to I4.0 are *AutomationML* for mechatronic data modeling and *OPC UA* for machine-to-machine communication.

Different organizations have developed reference architectures to align standards in the context of I4.0. In Germany, the *Deutsches Institut für Normung* (DIN) along with other organizations, published the "Reference Architecture Model for Industry 4.0 (RAMI 4.0)"[6]. In the United States, the *National Institute of Standards and Technology* (NIST) published a "Standards Landscape for Smart Manufacturing Systems"[7]. In China, the *Ministry of Industry and Information Technology* (MIIT) and the *Standardization Administration of China* (SAC) published the "National Smart Manufacturing Standards Architecture Construction Guidance". All these reference architectures pursue the common objective of providing a roadmap for the use of standards for smart factories.

Emphasis is put on the interoperability of the standards and the alignment with the processes in the factories. Despite the classification of the existing standards, there is still no structured and systematic approach to describe these standards and the relationships among them. In addition, the knowledge of the standards, the domain they cover, and the overlap that might exist among them is still not formalized. In this work, we devise a semantic-based landscape for I4.0 and present the *STO* ontology for describing I4.0 standards. The semantics encoded in *STO* is exploited for discovering implicit relations between standards. An evaluation allows for the tracking known properties of existing I4.0 standards and uncovering relations that were not initially modeled, e.g., a relation between AutomationML (AML) and IEC 61499 [148]. Thus, this landscape provides a building block for the implementation of a knowledge graph for Smart Factory standards, as well as for their mapping and semantic integration.

In particular, we make the following contributions:

- The Standards Ontology (*STO*) to describe characteristics of standards for Smart Factories and their relations;
- A description of more than 60 relevant standards to I4.0 and 20 standard organizations using *STO* and the classification of standards with regard to their RAMI layers and dimensions and NISTIR criteria, such as Product Life-cycle Management (PLM);
- A real-world use case showing the applicability and benefits of the standardization landscape following a semantic-based approach.

[6] https://bit.ly/2M3G7od/

[7] https://www.nist.gov/publications/current-standards-landscape-smart-manufacturing-systems

9.2.1 Background

In the following, relevant architectures of standardization landscapes for Smart Manufacturing, in addition to the RAMI model (cf. Subsection 9.1.1), are presented. It is important to note that some standards are classified at different layers these architectures.

Standards Landscape of the National Institute of Standards and Technology The *National Institute of Standards and Technology* (NIST) has defined a standards landscape focusing Smart Manufacturing Systems [149]. The standards landscape classifies standards with respect to their functions. First, standards are aligned with regard to the levels of the manufacturing pyramid of the *ISA 95* standard, i.e., from the device to the enterprise level. Second, standards are organized taking into account different phases of the product life-cycle, such as Modeling Practice, Product Model and Data Exchange, Manufacturing Model Data, Product Category Data, and Product Life-cycle Data Management. Finally, standards are classified with regard to the life-cycle of production systems. In this case, they include categories, such as Production System Model Data, Production System Engineering and Production System Operation and Maintenance. Similarly, *ISA 95* classifies standards into various layers. However, *ISA 95* is not a multi-dimensional model, and the classification of standards is more general than in RAMI. For instance, OPC UA is classified in the SCADA layer providing an uniform information model framework. However, in this classification, OPC UA focuses only on the SCADA systems and not on the use of the data, e.g., data management and analytic processes.

National Smart Manufacturing Standards Architecture The Ministry of Industry and Information Technology of China in a joint effort with the Standardization Administration created a report for defining a National Smart Manufacturing Standards Architecture [150, 151]. The architecture comprises three dimensions: 1) smart functions: from resource elements, system integration, interconnect, and information convergence to new business models; 2) life-cycle: from design, production, logistics, marketing and sales to service; and 3) hierarchy levels: from device, control, plant, and enterprise to inter-enterprise collaboration. Their objective is to classify standards into different architectural levels according to their functions.

9.2.2 Methodology

Here, we present the methodology followed during the creation of the semantic-based landscape for I4.0 standards. We first looked into the different initiatives related to standards classifications, i.e., the RAMI model, the NIST standard landscape, and the Chinese approach. These initiatives are focused on I4.0 and the related Smart Factory concept.

The RAMI model comprises the *Administration Shell* concept. It describes how standards are linked to certain submodels, e.g., *identification*, *communication*, or *engineering* [152]. For example, the identification submodel is aligned with the *ISO 29005*, whereas the communication submodel with the *IEC 61784 Fieldbus Profiles*. The aim of the current specification of the RAMI model is not to provide a complete description of the standards. However, it provides a starting point to further investigate the relevance for the Administration Shell concept.

Next, we considered the submodels, e.g., communication, engineering, and found other standards particularly relevant for them. For instance, the engineering domain is aligned with standards, such as *IEC 61360*, *IEC 61987*, and *eCl@ss*. In this regard, we extended our classification to include the Automation Markup Language *AutomationML* (AML) [153], also

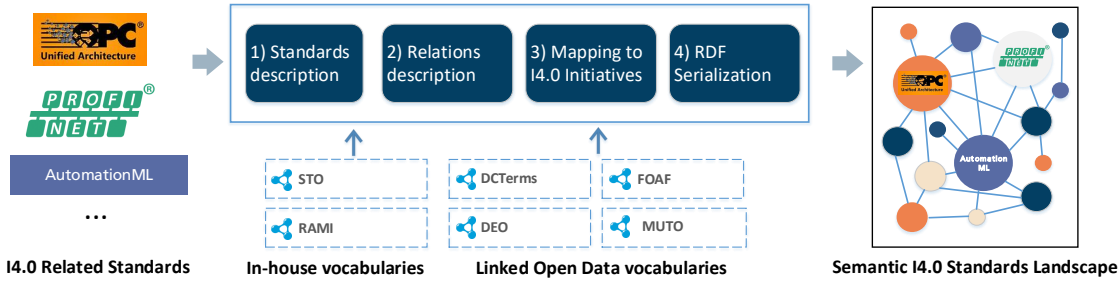


Figure 9.5: **The Semantic Standard Landscape Pipeline**. I4.0 standards are received as input and the output is a graph representing relations between standards. *STO* and existing vocabularies are utilized to describe known relations among standards. A reasoning process exploits the semantics encoded in *STO* to infer new relations between standards.

known as IEC 62714, since it is recognized as an important standard for engineering in the I4.0 realization [154–156]. The RAMI model also classifies standards according to its layers. We analyzed the *IT* layer of RAMI and classified standards for further mappings. For this, we considered all layers, from the *Asset* to the *Business* layer. Further, we investigated specific synergies of standards which are commonly described in scientific or white papers. For example, the relationship between *OPC UA* and AML is relevant for the realization of the I4.0 vision [154, 155]. As a result, these two standards have been combined into a new standard: *DIN SPEC 16592*. Provenance and other characteristics of I4.0 standards, e.g., relations between them, correspond to the properties modeled in the I4.0 landscape. For instance, the following relations are described in the literature: OPC UA with IEC 61499 [157], and OPC UA and IEC 61850 [158]. Further, we extend our view on current standards and their use in the Smart Manufacturing domain by analyzing related initiatives, e.g., NISTIR 8107 [149]. In this work, standards are classified according to different areas of importance for Smart Factories, e.g., PLM. We considered PLM-related areas, such as Product Life-cycle Data Management and Product Catalog Data, and categorized standards regarding their functions in these areas of PLM, e.g., *ISO 10303* and *ISO 13584*, respectively. Figure 9.5 depicts the pipeline that implements the methodology we followed during the creation of the I4.0 Standards Landscape.

9.2.3 An RDF-based Approach for the I4.0 Standards Landscape

The Standards Ontology (*STO*) is designed to semantically describe standards related to I4.0 as well as their relations. We used a knowledge-driven approach to model the most relevant concepts in order to represent all metadata of the I4.0 standards. The *STO* development follows best practices for ontology building provided in the Git4Voc methodology, e.g., reusing existing concepts of well-known ontologies, such as *MUTO* for tagging [159], *FOAF*[8] for representing and linking documents and agents (e.g., persons, organizations), *DCTERMS*[9] for document metadata, including licenses as well as the *RAMI* vocabulary for linking Standards with *RAMI* concepts. *VoCol* has been used as an integrated environment by providing various views of the ontology, such as a human-friendly documentation, graphical visualization, and statistical charts. As a result, apart from ontology engineers, other groups of users, in particular domain experts, are able to actively contribute to the further development of *STO*. Following best practices for

[8] http://purl.org/muto/core#, http://xmlns.com/foaf/spec/
[9] http://dublincore.org/documents/dcmi-terms/

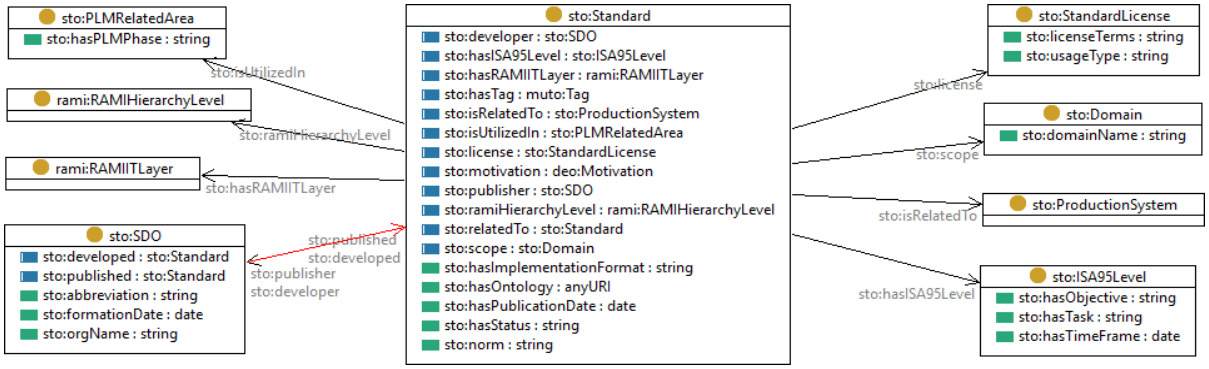


Figure 9.6: **The Standards Ontology (*STO*)**. *STO* classes and properties describe the meaning and relations of I4.0 standards. The *RAMI* vocabulary models a layer where a standard is classified in the RAMI architecture. Data type and Object properties are represented by green and blue squares, respectively; red arrows represent inverse functional properties.

ontology publishing[10], the *STO* ontology is available via a *W3ID* permanent URL as well as registered in the LOV service[11].

Class description In the following, the main *STO* classes are described.

sto:Standard: describes the concept of a *standard*. Since standards are usually published as documents, this class is a specialization of the `foaf:Document` class to represent the publication format of the standards.
sto:SDO: models common organizations that develop standards, such as *ISO*, *IEC*. This class specializes `foaf:Organization`, allowing to link its instances to *FOAF* defined resources.
sto:Domain: specifies relevant domains to the standards, e.g., Manufacturing Operation Management, Functional Safety, Industry Automation.
sto:ISA95Level: describes levels of the ISA95 standard. It enables linking of standards to the NIST initiative along with the linking them to the ISA95 levels.
sto:ProductionSystem: represents standards for modeling of complex systems, automation engineering as well as operation and maintenance perspectives of production systems.

Futhermore, the *STO* ontology reuses other classes from the *MUTO* and *RAMI* vocabularies, such as `muto:Tag`, `rami:RAMIHierarchyLevel`, `rami:RAMIITLayer`, respectively. Figure 9.6 illustrates the class diagram of *STO* concepts and their connections between each other.

Properties description We describe some of the core properties of *STO* as follows:

sto:license: links a standard with its correspondent license document. This information shows whether the standard is free to use or has to be purchased first.
sto:isPartOf: links a standard document with the specific parts of the standard.
sto:published: links a standard with the organization that published the standard.
rami:ramiHierarchyLevel and **rami:RAMIITLayer**: align standards with their hierarchy level and ITLayer in the RAMI model, respectively.

[10] https://www.w3.org/TR/swbp-vocab-pub/
[11] https://w3id.org/i40/sto, http://lov.okfn.org

sto:relatedTo: represents links between I4.0 standards. This property is symmetric and transitive. The inference model based on `sto:relatedTo` allows for the uncovering new relations between standards. For example, the standard *ISO 13849* is directly related to *IEC 61511*. Likewise, *IEC 61511* is directly related to *IEC 61508*. By utilizing this transitivity, a relation of *ISO 13849* to *IEC 61508* can be inferred.

sto:scope: describes the scope of a standard by defining its specific goals and limitations.

sto:hasOntology: refers to the ontology of a standard, if it has been already defined. For instance, the *ISO 15926* standard, used in the integration of life-cycle data for process plants, is available as an ontology[12] that can be linked accordingly.

Additionally, standards can be linked with other external resources using the following properties: `sto:hasWikipediaArticle`, `sto:hasWikidataEntity`, `sto:hasDBpediaResource`, and `sto:hasOfficialResource`.

Population

Based on the *STO* ontology, we created the *STO* dataset. It contains the descriptions for a set of existing standards, along with their metadata, e.g., licenses and relations. It also includes descriptions for the organizations that published the standards. The dataset can be expanded with further information by the community, through directly accessing it on GitHub[13]. Further, a public VoCol instance[14] is provided, where users can easily view and explore the described standards and their metadata, to enable a better comprehension for non-ontology engineers. In the following, we exemplify some of the most important concepts as instances described in *STO*.

Listing 9.6 depicts the semantic description of the OPC UA standard defined by the OPC Foundation. The overall dataset comprises more than 60 standards and more than 20 standard organizations at the moment. Moreover, we document more than 20 direct relations between the standards, with the possibility of inferring more than 50 new ones.

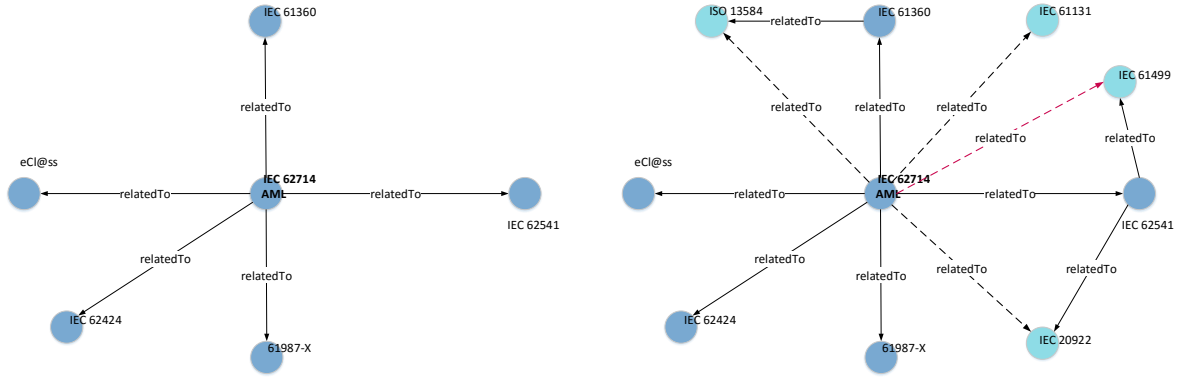
```
@prefix sto: <https://w3id.org/i40/sto#> .
@prefix rami: <https://w3id.org/i40/rami#> .
@prefix dc:  <http://purl.org/dc/terms/> .
sto:IEC_62541        rdf:type sto:Standard;
 dc:description        "OPC Unified Architecture (OPC UA) is an ..."@en;
 rami:hasRAMIHierarchyLevel rami:ControlDevice;
 rami:hasRAMILifeCycleLayer rami:Communication;
 sto:hasISA95Level          sto:SCADALevel;
 sto:developer              sto:OPC_Foundation;
 sto:hasDBpediaResource     <http://dbpedia.org/.../OPC_Unified_Architecture>;
 sto:hasOfficialResource    <https://opcfoundation.org/.../opc-ua/>;
 sto:hasTag                 "OPC UA"@en;
 sto:hasWikidataEntity      <https://www.wikidata.org/entity/Q623244>;
 sto:hasWikipediaArticle    <https://wikipedia.org/..OPC_Unified_Architecture>;
 sto:norm                   "62541";
 sto:publisher              sto:IEC;
 sto:relatedTo              sto:IEC_62714, sto:IEC_61499;
 sto:scope                  sto:Industrial_Automation.
```

Listing 9.6: **The RDF representation of an Industry 4.0 standard**. Semantic description of the OPC UA standard with the STO ontology and its metadata.

[12] https://www.posccaesar.org/wiki/ISO15926inOWL

[13] https://github.com/i40-Tools/StandardsVocabulary

[14] http://vocol.iais.fraunhofer.de/sto/



(a) Explicit relations between AML and other I4.0 standards

(b) Explicit and inferred relations between AML and I4.0 standards

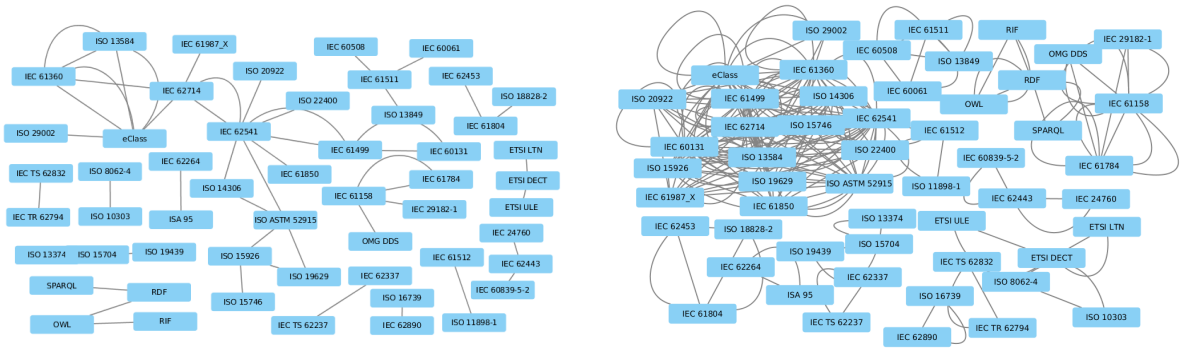
Figure 9.7: **I4.0 Standards related to AML**. Relations between I4.0 standards are visualized using graphs; continuous and dashed directed arrows represent explicit and inferred relations, respectively. The inference model relies on the transitive and symmetric properties of `sto:relatedTo`. (a) Known relations between AML and I4.0 standards are explicitly described using the *STO* object property `sto:relatedTo`. (b) Relations between I4.0 standards connected to AML with dashed directed arrows and colored in a different color, are inferred. The relation between AML (IEC 62714) and the I4.0 standard named Measurement and Control Devices (IEC 61499) has been validated [148].

```
PREFIX rami: <https://w3id.org/i40/rami#>
PREFIX sto: <https://w3id.org/i40/sto#>
SELECT DISTINCT
?name ?RAMIITLayer ?ISA95Level ?PLM ?AdminShellSubmodel ?license ?publisher
WHERE {
    ?std  a                          sto:Standard ; sto:norm   ?norm ;
    (sto:publisher/sto:name)         ?publisher ;
    (sto:publisher/sto:abbreviation) ?publisherAbbrv ;
    rami:hasRAMIITLayer              ?RAMIITLayer .
    FILTER(LANGMATCHES(LANG(?publisher), "en"))
    OPTIONAL { ?std sto:hasTag                    ?tag .}
    OPTIONAL { ?std sto:license                   ?license ;
    sto:hasOfficialResource                       ?officialResource . }
    OPTIONAL { ?std rami:hasAdminShellSubmodel    ?AdminShellSubmodel . }
    OPTIONAL { ?std sto:hasISA95Level             ?ISA95Level . }
    OPTIONAL { ?std sto:isUtilizedIn              ?PLM . }
    BIND(CONCAT(?publisherAbbrv," ",?norm," - ",?tag) as ?name)
}
```

Listing 9.7: SPARQL query for retrieving the metadata of the encoded standards.

9.2.4 Use Case

One of the main motivations to create the standards dataset is to describe I4.0 standards using *STO*, as well as to study existing relations among them. Figure 9.7(a) depicts explicit relations, which are currently annotated in the dataset; Figure 9.7(b) shows inferred relations, which are obtained after executing the aforementioned query and running the inference process based on the symmetric and transitivity properties of `sto:relatedTo`. These queries can be evaluated as



(a) Graph representing explicit relations between I4.0 standards

(b) Graph representing explicit and inferred relations between I4.0 standards

Figure 9.8: **Relations between I4.0 Standards**. Relations between I4.0 standards are visualized using graphs; continuous and dashed lines represent explicit and inferred relations, respectively. The inference model relies on the transitive and symmetric properties of sto:relatedTo. For readability only symmetric relations are represented using undirected line. (a) Known relations between I4.0 standards are explicitly described using the *STO* object property sto:relatedTo. (b) Relations between I4.0 standards are inferred; the graph comprises 74 edges: 50 are inferred while 24 are explicit.

well on the *STO* VoCol instance in the *Analytics Menu*. To this end, the *ForceGraph* option should be chosen as the graph type, and the *Direct Standard Relations* or *Direct and Indirect Standard Relations* options as queries to be executed.

This feature allows for the retrieving not only relations that have been explicitly defined in the *STO* dataset, but also those that are inferred. For instance, relations of AML (IEC 62714) and

Table 9.1: **Exemplar Descriptions of I4.0 Standards in the I4.0 Standard Landscape**. An I4.0 standard is described in terms of the classification level in the reference architectures, e.g., RAMI and ISA95; as well as basic properties like license and publisher. The same I4.0 standard can be classified by two reference architectures, e.g., IEC 62264.

Standard	*RAMIITLayer*	*ISA95Level*	*PLM*	*Submodel*	*license*	*Publisher*
IEC 62714	Information	-	ProductModelDataExchange	Engineering	Open	IEC
ISO 19439	Business	Enterprise	-	-	Propietary	ISO
IEC 62264	Functional	Enterprise	-	-	Propietary	IEC
SE ModBus	Integration	SCADA	ProductionEngineering	-	Propietary	SE
IEC 61360	Asset	-	ManufacturingModelData	Engineering	Propietary	IEC
IEC 62541	Communication	SCADA	-	-	Open	IEC

OPC UA (IEC 62541) as well as OPC UA (IEC 62541) and IEC 61499 were explicitly defined in the dataset. Based on this fact, the relation between AML and IEC 61499 (cf. Figure 9.7) is inferred. We validated that this relation exist and is important according to the literature [148]. Furthermore, all relations between the I4.0 standards can be retrieved at once (cf. Figure 9.8). The result of this query reports on 24 relations between the standards (cf. Figure 9.8(a)) on the dataset, and 50 new relations inferred when the symmetric and transitive properties are considered by the inference process (cf. Figure 9.8(b)).

Finally, the query in Listing 9.7 is executed over STO dataset to retrieve description of standards. The results, particularly help to understand how standards are linked with important I4.0 concepts, such as the Administration Shell submodel, the IT RAMI layer as well as with

PLM. Additional metadata used to describe I4.0 standards in the I4.0 standards landscape, e.g., name and tag, the license, and the publishing organization, are obtained (cf. Table 9.1).

9.3 Summary

This chapter presents two RDF-based approaches where ontologies are modeled to support interoperability in exchanging information in the different scenarios. The Git4Voc methodology along with the VoCol platform are used to manage and facilitate the collaborative and distributed development among different stakeholders.

Administrative Shell Vocabulary First, we describe an approach for semantically representing information about smart I4.0 devices with an Administrative Shell. The approach is based on structuring the information using an extensible and lightweight vocabulary aiming to capture all relevant information. Compared to prior approaches, the RDF-based Semantic Administrative Shell has a number of advantages. The URI/IRI based identification scheme provides a unified way to identify all types of relevant entities, from physical objects, abstract concepts, properties, concrete raw and derived data. Existing standards (e.g., eClass, IEC device characteristics or AutomationML) are more easily integrated and referenced. Information about and from different objects can be combined (since a basic integration can be achieved by merging sets of triples). Accessing the information in a unified way is established by using SPARQL, as a standard protocol and query language.

Standards Ontology Second, we develop both a landscape of Industry 4.0 related standards and the *Standard Ontology* (*STO*) for the semantic description of standards and their relations as well as respective organization responsible for publication and maintenance. Further, we investigate the existing reference models like RAMI and NIST, and populated the STO with descriptions of more than 60 standards, more than 20 standardization organizations, and 74 relations between the standards. Next, we illustrate the benefits of our semantic-based approach with a concrete use case.

We consider this work as a first step in a larger research and development agenda aiming at equipping manufacturing devices with semantics-based means to ease communication and data exchange. In the medium to long term, we aim to bring more intelligence to the edge of production facilities thus promoting self-organization and resilience of these devices. Furthermore, with the ontology-based representation of standards and their relations, our objective is to facilitate the structuring, discovery, selection, and integration of standards on a conceptual as well as operational and implementation level.

CHAPTER 10

Establishing Semantic Interoperability between Industry 4.0 Data

This chapter focuses on describing an *information model* realized for a global manufacturing company to describe its assets and information sources. In addition, it enables the semantic integration and exchange of data beyond manufacturer boundaries.

Although the vision of digitizing production and manufacturing has gained much traction lately (viz. Industry 4.0), it is still not clear how it can actually be *implemented* in an interoperable way using concrete standards and technologies [160]. A key challenge is to enable devices to communicate and to *understand* each other as a prerequisite for cooperation scenarios [161]. Various standards, such as those from the ISO and IEC series, are used to describe information about manufacturing, security, identification and communication, among other areas. For instance, machines produce assets following the description given by work orders. Work orders drive the production process for a given asset, and these information are managed in a different software platforms. Further important information for decision making, such as energy consumption for each produced asset by a particular work order, is not easily managed in the current setting. Such issues reduce the efficiency of the production process and therefore hamper the realization of the Industry 4.0 vision in actual working environments.

Integrating all relevant information and automating as many production steps as possible is the central goal of the Industry 4.0 vision [162]. Instead of envisioning one monolithic system or database, we pursue a decentralized semantic integration, i.e., the formal description and linking of all relevant assets and data sources based on an aligned set of RDF vocabularies – the *information model*. This avoids unnecessary data redundancy and allows structured querying and analyses across individual assets and data sources. The information model serves as a crystallization point and reference for data structures and semantics emerging from the data sources and value chains. Further, it is aligned to important industry standards, such as *RAMI* [124] and *IEC 62264* [163], to additionally foster data exchange and semantic interoperability.

The information model is centered around machine data and describes all relevant assets, key terms and relations in a structured way, making use of existing as well as newly developed RDF vocabularies. In addition, it comprises numerous RML mappings that link different data sources required for integrated data access and querying via SPARQL. The technical infrastructure and methodology used to develop and maintain the information model is based on a Git repository and utilizes our VoCol platform, as the development environment as well as the Ontop framework for Ontology Based Data Access.

Contributions of this chapter are summarized as follows:

- A methodology for the development, governance and interlinking of vocabularies for the domain of interest;
- A technical infrastructure to implement the *Information Model* for establishing data integration across heterogeneous data sources;
- The application of the defined approach to concrete uses cases, demonstrating the benefits and opportunities provided by the information model;
- A survey with stakeholders to asses their option on the benefits of the *Information Model* and benefits of semantic technologies in general as well as a report on lessons learned during project implementation.

This chapter is based on the following publication:

- Niklas Petersen, **Lavdim Halilaj**, Irlán Grangel-González, Steffen Lohmann, Christoph Lange, Sören Auer. *Realizing an RDF-based Information Model for a Manufacturing Company – A Case Study.* In 16th International Semantic Web Conference (ISWC) 2017 Proceedings, 350-366, Springer. This article is a joint work with Niklas Petersen and Irlán Grangel-González, PhD students at the University of Bonn. My contributions focused on the implementation of the proposed approach, the preparation and presentation of the use cases, and analysis of different strategies for data integration for the given scenario.

This chapter is organized as follows: In Section 10.1, we describe the motivation of this work. The core contributions, the information model, modeling approach and its implementation are presented in Section 10.2 and Section 10.3, respectively. In Section 10.4, we apply the information model to two use case scenarios, demonstrating its benefits and opportunities. Section 10.5 reports findings and lessons learned derived from the stakeholder interview. Section 10.6 summarizes the work that has been realized in this chapter.

10.1 Motivating Example

The information modeling project involved employees from different departments and hierarchical levels of the manufacturing company, external consultants and a third party IT provider. The company itself realized that their IT infrastructure has reached a level of complexity making difficult to manage and effectively use their existing systems and data. While adding new sensors to production lines is straightforward, using the sensor data effectively to improve the production process and decision-making can be cumbersome. The need to share production data with clients led them to evaluate the fitness of semantic technologies. For example, the production of bearing tools is fairly quality-driven depending on the customer specifications. Sharing the production details in a more processable format (compared to non-machine-comprehensible formats) aroused interest. Further goals were to gain a *bigger picture of the company's assets* (physical and non-physical) and to capture as much expert knowledge as possible.

The concrete use cases are based upon a machine newly introduced into the production lines of the company, a so-called *machine tool*. It is a machine that requires the mounting of tools to assemble specific metal or rigid products. Compared to older generations, the new machine is heavily equipped with embedded sensors that monitor the production process.

Tool management Possible tools to be mounted into the machine are cutters, drillers or polishers. A tool usually consists of multiple parts. The number of parts depends on the

manufacturer of the tool, which is not necessarily the same as the manufacturer of the machine. Mounting tools into a machine is a time-consuming task for the machine operator. Uncertain variables of the tools, such as location, availability and utilization rate, play a major role in the efficiency of a work shift and of a machine in particular. The production of certain goods may wear a tool out quickly, thus decreasing its overall lifetime and forcing the machine operator to stop the machine and replace it with a new tool. Reducing the idle time for remounting the machine by clearly describing its configuration, location and weariness was therefore one concrete goal to be addressed by the information model.

Energy consumption Producing goods with the machine tool is an energy-intensive process. Before we started the information modeling project, only the energy costs per factory were known. Sensors were added to track the energy consumption per machine and processed work order. For the cost calculation, data from the added sensors and the work orders, which resides in different data sources, needs to be linked and jointly queried. Therefore, integrating this data to be able to retrieve the information at run-time was another concrete goal to be achieved.

Three types of data were of particular interest in the project: i) Sensor Data (SD), ii) the Bill of Materials (BOM), and iii) data from the Manufacturing Execution System (MES). The SD comprises sensor measurements of the machine tool. These measurements record parameters needed for the continuous monitoring of the machine, such as energy, power, temperature, force, and vibration. The MES contains information about work orders, shifts, material numbers, etc. The machine produces assets based on the work order details, which provide the necessary information for the production of a given asset. The BOM contains information about the general structure of the company, such as work centers, work units, associated production processes, as well as information related to the work orders and the materials needed for a specific production.

10.2 Realizing an RDF-based Information Model

The information model aims at a holistic description of the company, its assets and information sources. The core of the model is based on a factory ontology we developed in a previous project [164], which describes real world objects from the factory domain, including factories, employees, machines, their locations and relations to each other. In addition, the information model comprises the mappings between ontologies that represent the data sources (i.e., SD, BOM, MES) and their corresponding schemes.

10.2.1 Development Methodology

Our development methodology was based on the approach proposed by Uschold et al. [165]. We first defined the purpose and scope of the information model; then, we captured the domain knowledge, conceptualized and formalized the ontologies and aligned them with existing ontologies, vocabularies and various standards. Finally, we created the mappings between the data sources and ontological entities. In line with best practices, we followed the iterative and incremental *round-trip model* based on the Git4Voc methodology, i.e., with an increased understanding of the domain, the information model was continuously improved.

All artifacts were hosted and maintained by VoCol, which we adapted with additional features to allow querying and retrieving data from heterogeneous sources. VoCol supports the requirements of the stakeholders: i) version-control of the ontology; ii) online and offline editing; and iii) support for different ontology editors (by generating a unique serialization before changes are merged to avoid false-positive conflicts).

In addition, it offers different web-based views on the ontology, including a human-readable documentation, a visualization and graphical charts generated from customized queries applied to the ontology and instance data. These views are designated to ease the collaboration of domain experts in the development process, i.e., enabling them to participate via single access point and without having to set up and maintain a proper infrastructure themselves.

Purpose and scope The information model comprises i) a formal description of the physical assets of the company, ii) mappings to database schemas of existing production systems, and iii) a formalization of domain-related knowledge of experienced employees about certain tasks and processes within the company. The heart of the information model represents the aforementioned *machine tool*, including its sensor data, usage processes and human interaction. Therefore, the majority of concepts are defined by their relation to this machine.

The scope is set by the motivating examples *energy consumption* and *tool management* introduced in Section 10.1. The objective of the management is to gain a clearer picture of all assets of the company. For example: What local knowledge exists in the factory? What kind of data exists for which machine? Where is that data? Who has access to it? Discussions on fully automated order-driven production sites are ongoing. The management hopes for this to be supported by the information model, and we aim to provide the basis towards that goal.

Capturing domain knowledge The knowledge from domain is captured in different ways:

1. The company provided descriptive materials of the domain, including maps of the factories, descriptions of machines and work orders, information about processes, sensor data and tool knowledge. The types of input material ranged from formatted and unformatted text documents to spreadsheets and SQL dumps.
2. An on-site demonstration of the machine within the factory was given during the project kick-off, including a discussion of further contextual information missing in the material. In subsequent meetings, open questions were clarified and concrete use cases for the information model were discussed.
3. We reviewed relevant existing ontologies and industry standards, intending to build on available domain conceptualizations and formalizations.
4. We created customized document templates to enable easy participation of domain experts by collecting input on the ontology classes and properties in a structured way. We collected names and descriptions of all properties having a given class as their domain in one table with one row per property; additional details about the domains and ranges of properties were collected in a separate table. These documents were continuously handed over to the domain experts to be reviewed and completed.
5. We trained IT-affine employees of the company to model ontologies with editors, such as Protégé, TopBraid Composer and the Turtle editor integrated into the VoCol environment.[1]

Conceptualizing and formalizing Since `Machine`(s) are the main assets of the manufacturing company, they have been used as a starting point for creating the ontology. Each machine contains a geo-location (property with range `Geometry`) and is part of a certain `Section`, which is in a certain `Hall`. Certain tools can be mounted directly onto the machine holder. Tools are the parts that wear out over time and need to be replaced. The geo-location of machines enables

[1] http://protege.stanford.edu, http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition

showing their position on a map. The tool ontology part reflects the configuration options for tools of multiple tool manufacturers. Specialized classes and properties are created to detail the core concepts and their relationships. In total, the developed ontologies comprise of 148 classes, 4662 instances, 89 object and 207 datatype properties.

Aligning with existing ontologies & standards The developed information model consists of concepts from existing vocabularies and industry standards formalized during the project. Particularly, concepts from the VIVO (`vivo:Building`), NeoGeo (`ngeo:Geometry`), FOAF (`foaf:Person`) and Semantic Sensor Network `ssn:Sensor`) ontologies are reused.[2] We aligned the ontologies of the information model to the RDF vocabularies of the industry standards, such as RAMI (cf. Subsection 9.1.3) and IEC 62264 [163]. RAMI is used to structure the interrelations between IT, manufacturing and product life-cycles, whereas IEC 62264 defines the hierarchical levels within a company. RAMI includes the IEC concepts and adds the "connected world" as an additional level, to align with the basic idea and motivation of this work.

10.2.2 Information Model Governance

Introducing new technologies is often a challenge for companies. It has to be well-aligned with the organizational structure of the company to balance the added value produced for the information model to the business and the maintenance costs of the technology. Thus, in parallel with the introduction of the information model, we defined a procedure to support the *governance* of information to ensure the maintenance of the model and uniform decision-making processes.

Since the core of the information model is a network of ontologies and vocabularies with a clear hierarchical and modular structure, there are boards of experts assigned to each part, which are responsible for its maintenance. Design decisions cover, for instance, including new terms or removing existing ones, reusing and aligning with external vocabularies, and the continuous alignment with Industry 4.0 standards, e.g., RAMI, IEC, ISO. Additionally, we provided concrete guidelines for maintaining the information model along with the use of VoCol, for example[3]:

- detailed documentation of all terms defined in the vocabulary by `skos:prefLabel`, `skos:altLabel`, and `skos:definition`;
- multilingual definition of labels, i.e., in English and German;
- definition of `rdfs:domain` and `rdfs:range` for all properties;
- inclusion of provenance metadata, licenses, attributions, etc.

10.3 Architecture and Implementation

With the objective to provide a uniform interface for accessing heterogeneous and distributed data sources, we designed and implemented the architecture illustrated in Figure 10.1. It is extensible and able to accommodate additional components for accessing other types of data sources as well as supporting federated query engines. The architecture distinguishes the following four main layers, some of which are orthogonally located across different components:

The **ontology layer** consists of several ontologies that have been created to conceptualize a unified view of the data. Wache et al. [166] distinguish three main approaches of using ontologies to explicitly describe data sources: i) *Global Ontology Approach* - all data sources are described

[2] `http://vivoweb.org`, `http://geovocab.org`, `http://xmlns.com/foaf/spec`, `https://www.w3.org/TR/vocab-ssn`
[3] These are based on the W3C "Data on the Web Best Practices" Recommendation, `https://www.w3.org/TR/dwbp/`

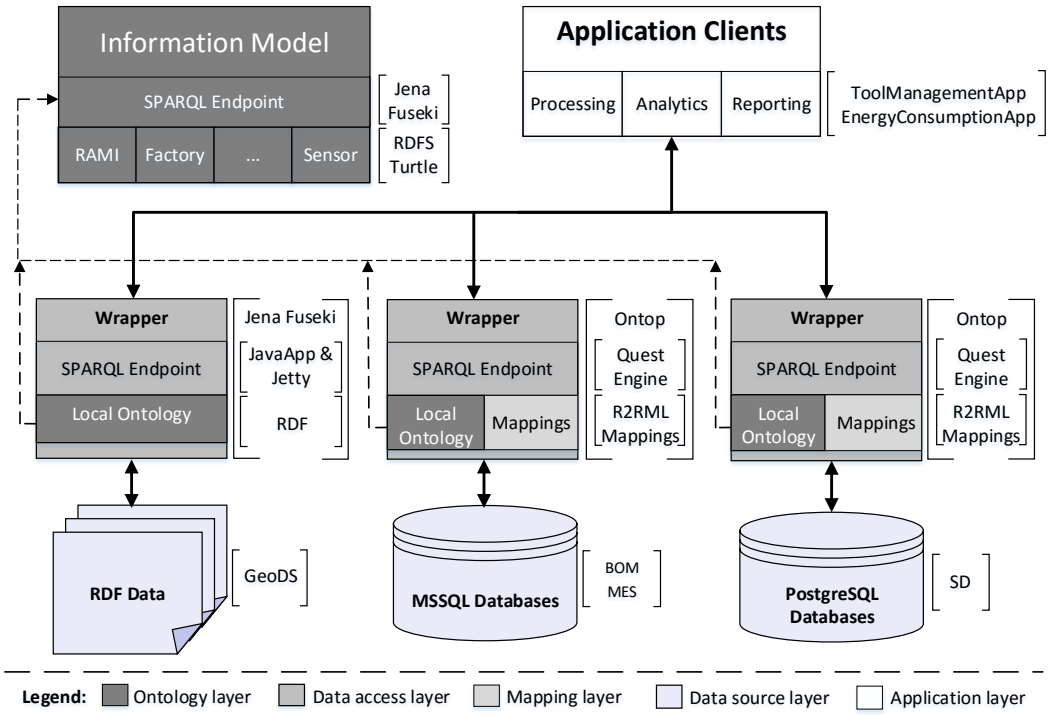


Figure 10.1: **The implemented architecture**. It comprises several layers hosting various components: a) ontology Layer; b) data access layer; c) mapping layer; d) data source layer; and e) application layer.

in an integrated, global ontology; ii) *Multiple Ontology Approach* - separate local ontologies represent the respective data sources, and mappings between them are established; the iii) *Hybrid Ontology Approach* - a combination of the two previous approaches to overcome the drawbacks of maintaining a global shared ontology and mappings between local ontologies.

We followed the third approach, which enables new data sources to be easily added, avoiding the need for modifying the mappings or the shared ontology. Accordingly, our ontologies are organized in two groups: i) a shared ontology to represent the highest level of abstraction of concepts and mappings with external ontologies; and ii) local ontologies representing the schemas of the respective data sources. This makes our architecture quite flexible with respect to the addition of diverse types of data sources [167].

The **data access layer** consists of various wrappers acting as bridges between client applications and heterogeneous data sources. It receives user requests in the form of SPARQL queries, which are translated into the query languages of the respective data sources, and returns the results after query execution. Relational databases are accessed via the *Ontology-Based Data Access* (OBDA) paradigm, where ontologies serve as a conceptualization of the unified view of the data, and mappings to connect the defined ontology with the data sources [168]. Particularly, the Ontop [169] framework is used to access the relational data sources, i.e., the BOM, MES, and SD data, which exposes them as virtual RDF graphs, thus eliminating the requirement to materialize data into RDF triples. Jena Fuseki is used as in-memory triple store for loading GeoDS, since the information about the geo-locations of the machines are less than 20K triples.

The **mapping layer** deals with the mappings between the data stored in the data sources and the local ontologies. For the definition of the mappings, we used *R2RML*[4], the W3C standard

[4] http://www.w3.org/TR/r2rml/

RDB-to-RDF mapping language. As a result, it is possible to view and access the existing relational databases in the RDF data model.

The **data source layer** comprises the external data sources, i.e., databases and RDF datasets. Due to the high dynamicity and the huge amount of incoming data, the data sources are replicated and synchronized periodically. As a result, any performance and safety degradation of the production systems is avoided. Additional types of data sources can be easily integrated in the overall architecture by defining local ontologies, mappings with the global ontology and data sources as well as choosing an appropriate wrapper.

The **application layer** contains client applications that benefit from the unified access interface to the heterogeneous data sources. These applications can be machine agents or human interface applications able to query, explore and produce human-friendly presentations.

10.4 Use Cases

We applied the developed information model to the use cases introduced in Section 10.1 to demonstrate the possibilities resulting from semantically integrated data access.

10.4.1 Tool Management

Figure 10.2 displays different views on the assets of the company. On a world map (cf. Figure 10.2(a)), the sites of the company are highlighted based on their geo-location given in the information model. By zooming in, the different locations can be investigated w.r.t. their functionality, address, or on-site buildings up to the level of machines. By clicking on the objects on the map, static production data is displayed. As an example, Figure 10.2(b) shows all tools stored in a certain paternoster system, grouped by drawer. Figure 10.2(c) provides an example of a machine with its properties: production name, current status, self-visualization, mounted basic holder, and tool with its diameter. Further, it contains links to existing external analytical web pages. A "Determine Tool Availability" function is offered for locating the tools to be assembled in the closest paternoster storage system based on the location of the machines.

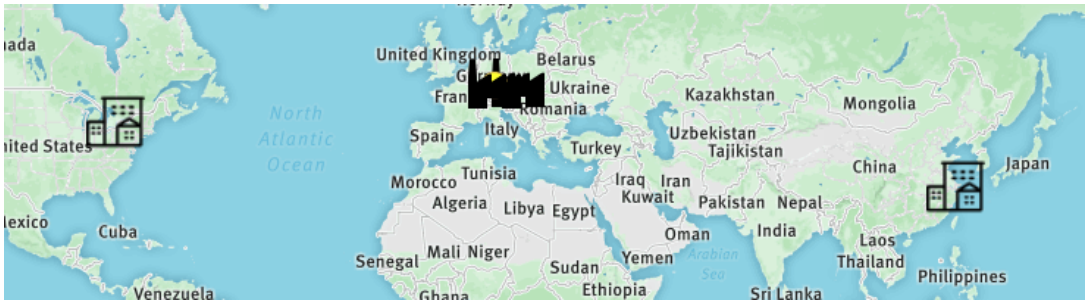
Each time a certain view is opened, a SPARQL query retrieves the required data in the information model. Geo-locations are drawn to the world map view using Leaflet JavaScript[5].

10.4.2 Energy Consumption

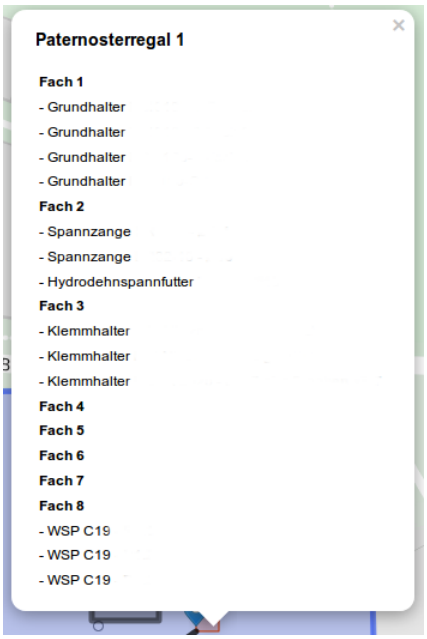
Information about the energy, power or temperature are critical for the company to forecast the production process, expenses and maintenance. In the second use case, we asked the following question: what is the *energy consumption* of a given machine for a given day for a particular *work order*? To answer this question, data from sources (SD, MES, BOM) are taken into account. Since the SD lacks the *work order* definitions, we used time intervals to access the required energy stream data. Next, we linked the work order IDs in the BOM and MES databases. Among others, it includes its material number, total execution time and target production amount.

As a part of the information model, we created R2RML mappings for all data sources: 1) mappings from the work orders table to the ontology (cf. Listing 10.1); and 2) mappings from the energy data to the ontology (cf. Listing10.2). Based on the defined mappings, we created several queries for retrieving information about work orders and energy consumption. For example,

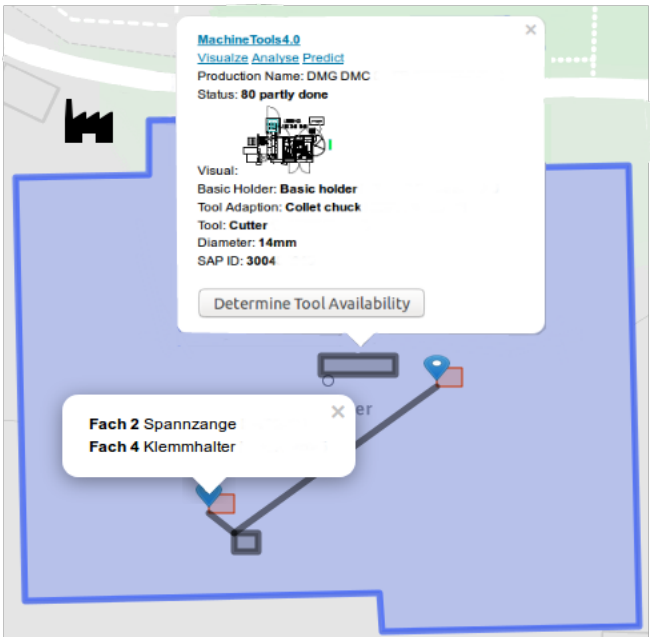
[5] http://leafletjs.com/



(a) Global view of the company sites



(b) Paternoster view



(c) Factory view

Figure 10.2: **Various views of the tool management application**. a) Geographical depiction of the places where the company is located; b) List of tools stored in a paternoster system; and c) Detailed information about a machine located within a factory.

Listing 10.3 illustrates a specific query used to retrieve the energy consumption values for a work order in a specific time interval.

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix im: <http://iais.fraunhofer.de/infomodel#> .
<WorkOrderMap> a rr:TriplesMap ;
  rr:logicalTable [ rr:tableName "WorkOrders" ];
  rr:subjectMap   [ rr:template
    "http://.../infomodel/WorkOrder/{WorkOrderId}"; rr:class im:WorkOrder];
  rr:predicateObjectMap
    [rr:predicate im:workOrderId;    rr:objectMap [rr:column "WorkdOrderId"]],
    [rr:predicate im:targetAmount;   rr:objectMap [rr:column "TargetAmount"]],
    [rr:predicate im:totalExecTime; rr:objectMap [rr:column "TotalExecTime"]],
    [rr:predicate im:matDesc;        rr:objectMap [rr:column "MatDesc"]].
    ...
```

Listing 10.1: **Work orders mappings**. R2RML mappings between the local ontology and work orders data in MES database.

```
@prefix rami: <https://w3id.org/i40/rami/#> .
<EnergyConsumptionMap> a rr:TriplesMap;
  rr:logicalTable [rr:tableName "View_GetEnergyConsumption" ];
  rr:subjectMap   [rr:template
     "http://iais.fraunhofer.de/infomodel/{DateOfMeasure}/{HourOfMeasure}";
                   rr:class rami:SensorMeasurementData ];
  rr:predicateObjectMap
    [rr:predicate im:dateOfMeasure; rr:objectMap [rr:column "DateOfMeasure"]],
    [rr:predicate im:hourOfMeasure; rr:objectMap [rr:column "HourOfMeasure"]],
    [rr:predicate im:measurementValue; rr:objectMap [rr:column "Value"]].
    ...
```

Listing 10.2: **Energy consumption mappings**. R2RML mappings between the local ontology and energy consumption in SD database.

```
PREFIX im: <http://iais.fraunhofer.de/vocabs/infomodel#>
SELECT ?hour ((?latest - ?earliest) AS ?measurementByHour) {
  { SELECT ?hour (MIN(?measurement) AS ?earliest)
    WHERE {
             ?machineSensor im:energyTime ?time;
                            im:energyValue ?measurement.
    }
    GROUP BY (HOURS(?time) AS ?hour) }
  { SELECT ?hour (MAX(?measurement) AS ?latest)
    WHERE {
             ?machineSensor im:energyTime ?time;
                            im:energyValue ?measurement.
    }
    GROUP BY (HOURS(?time) AS ?hour) }
  FILTER (?hour>= timeFrom && ?hour <= timeTo) } ORDER BY ?hour
```

Listing 10.3: **Retrieving energy consumption**. A SPARQL query to retrieve the energy consumption per machine per hour.

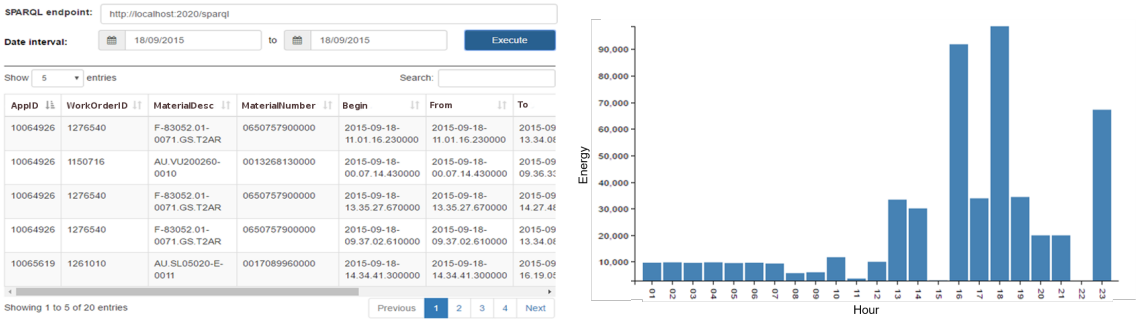
This allowed us to integrate the information from the three data sources using the information model and SPARQL queries. Figure 10.3(a) depicts the information of work orders for a given machine, and Figure 10.3(b) shows the energy consumption per hour for that machine for a given day. Overall the performance of the implemented approach was satisfactory, i.e., time to retrieve the energy consumption of particular work order was less than five seconds.

10.5 Evaluation and Lessons Learned

To gain feedback from the stakeholders involved in the information modeling project, we designed a questionnaire and sent it to the stakeholders, asking for anonymous feedback. Table 10.1 lists the questions and results of the questionnaire. We were interested in how the stakeholders evaluate the developed information model and semantic technologies in general, based on the experience they gained in the project.

10.5.1 Stakeholder Feedback

Five employees of the manufacturing company (three IT experts, one analyst, and one consultant) who were actively involved in the project, answered the questionnaire. The results varied across the stakeholders: While some regarded the information model and future potential of semantic



(a) Work orders (b) Energy consumption

Figure 10.3: **Various views of the analytics application**. a) Work order data for a given machine in a time interval of one day; and b) Energy consumption of a given work order within a day.

Table 10.1: **Survey questions and given scores**. Likert scale of 1 to 5, 1 = not at all, 5 = very much is used to categorize the given scores from stakeholders. **M** represents the *mean value* and **SD** represnts the *standard deviation*.

Question	**M**	**SD**
1. Did the developed RDF-based information model meet your expectations?	2.4	0.9
2. Do you think investing in semantic technologies can result in a fast ROI?	3.0	1.4
3. Do you consider semantic technologies fit for usage in the manufacturing domain?	3.6	0.9
4. Are you satisfied with the software for semantic technologies available on the market?	2.8	1.3
5. Is it easy to hire personnel with knowledge in semantic technologies?	1.8	0.4
Free-text questions:		
6. What do you expect from semantic technologies in manufacturing contexts?		
7. What is the biggest bottleneck in using semantic technologies in manufacturing contexts?		

technologies as promising, others remained skeptical about its impact within the company. Question 6 asking for the expectations towards semantic technologies (cf. Table 10.1) was answered by nearly all as an "enabler for autonomous systems" and by one as a "potential technology to reduce the number of interfaces". One stakeholder praised the "integration and adaption" capabilities of semantic technologies. Question 7 asking for the biggest bottleneck yielded the following subjective answers: "lack of standardized upper ontologies", "lack of field-proven commercial products", "lack of support for M2M communication standards", "skepticism of the existing IT personnel". While the stakeholders find the advantages of semantic technologies appealing and suited to solve the underlying essential problems of interoperability and integration, a successful application requires the provision of tools suited for different user groups.

10.5.2 Lessons Learned

In the following, we reflect on the lessons learned during implementation of this work including opinions from the company staff about current state of the semantic technologies.

Technology awareness within the company After all, the majority of the stakeholders were enthusiastic and committed to develop an integrated information model and applications on top of it. Nevertheless, reservations on the fitness of the technology and methodology existed from the start. A few stakeholders preferred a bottom-up approach by first gathering and generating internally an overview of the existing schemas and models before involving external parties (such

as our research institute). However, the management preferred an *outside view* and put a focus on quick results. Instead of spending time on finding an agreement on how to proceed, speed was the major driving force. Thus, they preferred to try out a (for them) "new" technology and methodology, which does not yet have the reputation of strong industry maturity.

Perceived maturity of semantic technologies While semantic technologies are already widely used in some domains (e.g., life sciences, e-commerce or cultural heritage), there is a lack of success stories, technology readiness and show-case applications in most industrial areas. With regard to smaller and innovative products, the penetration of semantic technologies is still relatively small. A typical question when pitching semantic technologies within companies is "Who else in our domain is using them already?". Therefore, it is important to point to successful business projects, even if details on them are usually rare.

Lack of semantic web professionals on the job market Enabling the employees of the manufacturer to extend the information model by themselves is crucial for the success of the project. Consequently, it is necessary to teach selected stakeholders the relevant concepts and semantic technologies. Hiring new staff experienced with semantic technologies is not necessarily an easy alternative. Compared to relational data management and XML technologies, there is still a gap between the supply of skilled semantic technology staff and the demand of the market.[6]

Importance of information model governance Of major importance for the company is a clear governance concept around the information model, answering questions, such as who or which department is allowed to access, modify and delete parts of the information model. An RDF-based information model has advantages in this regard: i) it enables people across all sites of the company to obtain a holistic view of company data; ii) current data source schemes are enriched with further semantic information, enabling the creation of mappings between similar concepts; and iii) developers can follow a defined and documented process for further evolving and maintaining the information model.

Building on top of existing systems Accessing data from the existing infrastructure as a virtual RDF graph was a crucial requirement of the manufacturer. It avoids the costs of materializing the data into RDF triples and the redundant maintenance in a triple store, and at the same time, benefits from mature mechanisms for querying, transaction processing, and security of the relational database systems. Three different data access strategies were considered:

DB in Dumps Relational data to be analyzed is dumped in an isolated place away from the production systems, as not to affect their safety and performance. This strategy is used in cases where the amount of data is small and most likely static or updated very rarely.

DB in Replication All data is replicated, allowing direct access from both production systems and new analytic platforms. This approach was considered in cases where data changes frequently and the amount of data is relatively high. It requires allocation of additional resources to achieve a "real-time" synchronization and to avoid performance degradation of the systems in production. We used this strategy to implement our solution, since it

[6] For the related field of data science, the European Data Science Academy has conducted extensive studies highlighting such a skill/demand gap all over Europe; cf. Deliverables D1.2 and D1.4 ("Study Evaluation Report 1/2") downloadable from http://edsa-project.eu.

allows to access the data sources as a virtual RDF graph and benefit from the maturity of relational database systems.

DB in Production The strategy of accessing data in real-time systems does not require allocating additional resources, such as investment in new hardware or software. Since this strategy exposes a high risk for performance degradation of the real-time systems, whereas sensitive information requires high availability and not providing it on time can have hazardous consequences, we did not apply it in our scenario.

10.6 Summary

This chapter presents a case study on realizing an RDF-based information model for a global manufacturing company using semantic technologies. The information model is centered around machine data and describes all relevant assets, concepts and relations in a structured way, making use of existing as well as specifically developed ontologies. The objective of the information model is to describe business units, their processes and assets for the entire organization. Furthermore, it contains a set of RML mappings that link different data sources required for integrated data access and SPARQL querying. Finally, it is aligned to relevant industry standards, such as *RAMI* [124] and *IEC 62264* [163], to additionally foster data exchange and semantic interoperability. We described the used methodology to develop the information model, its technical implementation and reported on the gained results and feedback. Additionally, we reflected on the lessons learned from the case study. The guidelines and practices defined in the Git4Voc methodology are used to organize the development process and facilitate design decisions regarding to the naming conventions, reuse and multilinguality. On the other hand, all artifacts generated over the time are hosted and managed into the VoCol platform.

The use of data-centric approaches in engineering, manufacturing and production is currently a widely discussed topic (cf. the related initiatives to Industry 4.0 or Smart Manufacturing). The challenges and complexity of data integration are perceived as a major bottleneck for the comprehensive digitization and automation in these domains. A key issue is to efficiently and effectively integrate data from different sources to ease the management of individual factories and production processes up to complete companies. The presented information model is envisioned to serve as a crystallization point and semantic reference in this context.

CHAPTER 11

Collaborative Development of Ontologies in Real-world Scenarios

In this chapter, we present VoColReg, a registry to support development and exploration of ontologies for various domains in the same time. Additionally, statistics and useful insights for a number of ontologies hosted in VoColReg are described in detail.

The number of ontologies used for different purposes, such as data integration, information retrieval or search optimizations is constantly increasing. An ontology is comprised of classes, properties and instances, to represent as good as possible the intended domain. In case that an ontology is not any more accessible or even not published, humans and dependent machines that were potentially using it before, will suffer from the possibility to explore and use its defined concepts. This is against the semantic interoperability, one of the fundamental aspects for enabling the Semantic Web vision [170]. Furthermore, it negatively impacts the reusability of the concepts by imposing extra efforts on capturing and modeling the knowledge of a domain, which might cause duplication and inconsistencies across terminologies and their semantic descriptions.

Several platforms and mechanisms are developed to enable organization of ontologies into repositories. They facilitate searching and maintenance, which are important factors for a better comprehension and reuse of ontologies [51]. For instance, BioPortal [171] supports a number of technical requirements related to the assessment of biomedical ontologies via Web browsers or other Web services. In addition, it allows the community to evaluate the ontology content and contribute on its evolution process. Linked Open Vocabularies (LOV) [170] offers mechanisms for searching over vocabularies using different criteria, such as metadata and ranking values, as well as allows accessing them via APIs and a SPARQL endpoint. The LOV initiative collects a number of indicators, including incoming and outgoing links of the vocabularies, history of versions and other metadata. OntoHub [51] supports the development of modular and distributed ontologies along with the definitions of logical relations between them. Ontologies can be aligned and translated into other languages as well as offered as linked data.

We created VoCoReg, a registry of ontologies which is built on top of the VoCol platform. It operates as an integrated ecosystem where the community can browse, discuss, reuse and improve ontologies in a collaborative fashion. In addition to help the community on discovering ontologies, it provides a suite of tools and services, including ontology documentations, term search, and ontology evolution. Incorporation of sophisticated tools for visual representation facilitates cognitive exploration and navigation over defined concepts. Currently, the VoColReg registry hosts 19 ontologies from various domains, such as *manufacturing*, *health care* and *education*. In

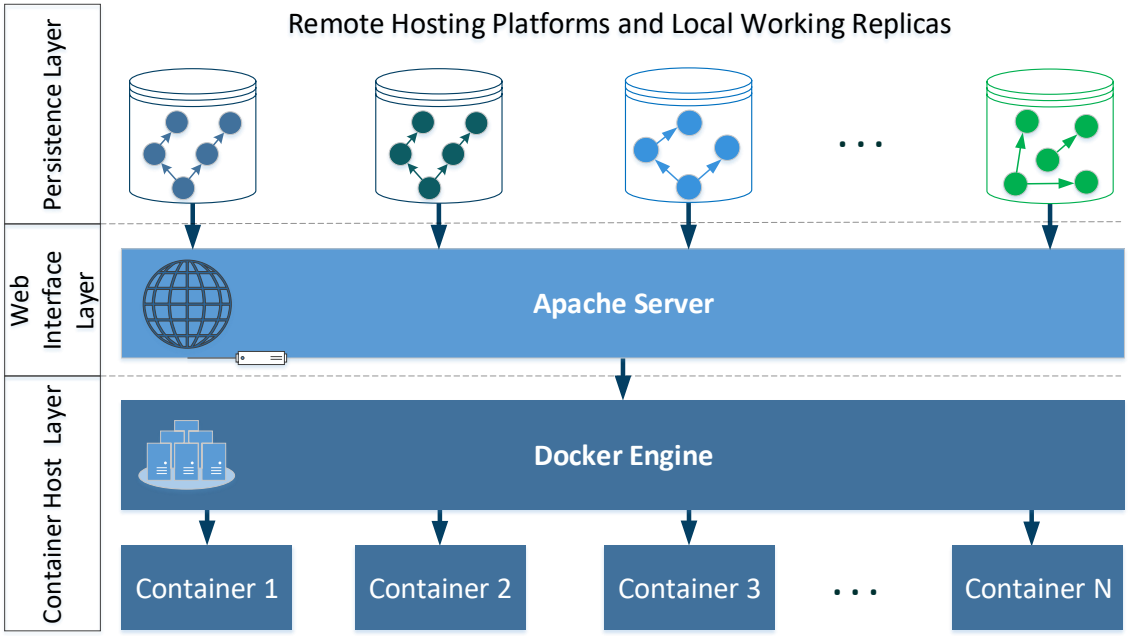


Figure 11.1: **The VoColReg Architecture**. It is composed of several layers: a) persistence layer; b) web interface layer; and c) container host layer. Each layer consists a number of different components.

this chapter, we describe in detail nine ontologies that are currently being developed. Various information, including the number of classes, properties and instances as well as details related to the development process, such as number of commits, contributors, branches, etc., are collected. We then analyze these statistics and present the findings along with a discussion of potential mechanisms to achieve a better quality for ontologies and ease the collaboration process.

Contributions of this chapter are summarized as follows:

- An architecture and implementation of VoCoReg to enable searching and development of ontologies in distributed environments;
- A detailed description of ontologies that are hosted in VoColReg, including metrics and statistics related to development aspects;
- Analysis of findings and a discussion of potential mechanisms to improve the development process according the practices and guidelines of the Git4Voc methodology.

This chapter is organized as follows: It starts with an overview of the VoColReg architecture in Section 11.1, including its layers and their main characteristics. In Section 11.2, we present details of the ontologies currently hosted and being developed in VoColReg. Section 11.3 reports findings and insights from a deeper analysis of these ontologies. The work that has been realized in this chapter is summarized in Section 11.4.

11.1 Architecture

VoColReg has an extensible and flexible architecture able to accommodate many instances of VoCol as well as additional components for accessing various types of data sources. It consists

of three main layers (cf. Figure 11.1): a) *Persistence Layer*; b) *Web Interface Layer*; and c) *Container Host Layer*. The details of each layer are described as follows:

Persistence Layer: stores and maintains ontologies and their versions. By managing the atomicity and durability of the changes, it is possible to easily backup and restore a particular ontology version. This layer is composed of two components: 1) *Remote Hosting Platforms*, which allows for change distribution and synchronization as well as administering with role permissions of the stakeholders involved in the development process; and 2) *Local Working Replicas*, where team members directly contribute on ontology construction.

Web Interface Layer: is responsible for two main tasks: 1) accepts notifications from *Remote Hosting Platforms* after occurrences of each push event via *PubSubHubbub* protocol, according to the principles of *publish-subscribe* for a distributed communication; and 2) it presents various views of the ontology by exposing them via a web-based *graphical interface*, helping users to explore and understand the ontology.

Container Host Layer: consists of a *Docker Engine* responsible for managing a number of *docker containers*. These containers act as *isolated node workers*, where each node has a VoCol installation inside connected with a repository hosted in *Persistence Layer*. Therefore, nodes can be restarted, stopped or even moved to another server, without affecting the work that is being realized in the *Persistence Layer*.

VoColReg provides an infrastructure to perform *federated queries* over external data sources using SPARQL endpoints. Another important feature, is offering metadata and other information for enabling users to have a better overview for ontologies. With the objective of facilitating the quality assessment of ontologies, the following information are provided:

Ontology Metrics: are provided for each ontology, including number of axioms, classes, object properties, datatype properties, individuals as well as the expressivity profile.

Validation Report: shows the results of the validation process. It consists of two parts: 1) syntax checking that shows detailed information about syntactic errors, if any; and 2) consistency checking, listing details of any inconsistency that has been found.

Evolution Details: are generated after the comparisons of each new version pushed to the *Remote Repository* with the previous one. Information such as, affected axioms, user who pushed the changes or date and time, are easily navigated via user interface.

Ontology accessibility A customized sharing mechanism implemented into VoColReg allows administrators to manage access permissions to their ontology. They can decide for the visibility of the ontology, whether it will be *public*, *shared with a link*, or *private*. Choosing the public option, means that the ontology link will be added to the VoColReg homepage, so everyone can see and access it. If the option *shared with a link* is chosen, then only users who know the link can access the ontology. Finally, by selecting the *private* option, administrators have to specify credentials which later should be provided by users in order to be able to access the ontology.

Ontology availability Users can browse and consume the content of the ontology via various forms. Apart from the *graphical user interface*, ontologies are exploited through two additional mechanisms. First, using a SPARQL endpoint, users or applications can post specific queries and retrieve customized results to be used for further analyses. Second, the *Content Negotiation* mechanism enables software agents to receive the appropriate format, simply by specifying the *content-type* in the HTTP request.

Table 11.1: **Ontologies being developed in VoColReg**. Overview of the ontologies representing various domains in terms of accessibility, hosting platform, expressivity and the number of triples.

Ontology	Domain	Access	Hosting Platform	Expressivity	# triples
STO	Industry 4.0	/sto/	GitHub	SHOI(D)	2135
MobiVoc	Mobility	/mobivoc/	GitHub	ALUO(D)	12134
SCORVoc	Supply Chain	/scorvoc/	GitHub	ALCHO(D)	8496
IDS	Industry 4.0	/ids/	GitHub	ALH(D)	1727
MatOnto	Supply Chain	/matonto/	GitHub	ALCHO(D)	359
IASIS	Health	/iasis/	GitHub	AL(D)	727
SARO	Education	/saro/	GitHub	AL(D)	338
Onto. #1	Accounting	private	Bitbucket	ALUO(D)	7515
Onto. #2	Health	private	Bitbucket	SRIF(D)	3210

Ontology exploration Relationships and other important information of concepts can be modeled using a number of different properties and attributes. For instance, representing the taxonomy between concepts is realized utilizing properties, such as *rdfs:subClassOf* from RDFS vocabulary or *skos:narrower* and/or *skos:broader* from SKOS vocabulary. Users can browse the generated class hierarchy in the *Documentation* view along with further details related to other important attributes. A comprehensive picture of the relations among concepts is provided in *Visualization* view, where concepts are depicted in a node-link interactive diagram.

11.2 Ontologies in VoColReg

A total number of 19 ontologies in various sizes are currently hosted in VoColReg. Overall, these ontologies comprise 191K triples, where 18K are distinct subjects, 839 are properties and 47K are objects. Hosted ontologies represent different domains, ranging from: 1) *Industry 4.0* - with four ontologies; 2) *Education* - three ontologies; 3) *Supply Chain*, *Health Care* and *Web Interoperability* - with two ontologies each; and 4) *Accounting*, *Law*, *Research*, *Manufacturing* and *IT* - each of them with one ontology. Regarding to the accessibility of ontologies: 11 of them are publicly reachable and listed in the VoColReg homepage, so people can chose from the list for further exploration, four of them are accessible from people who know the link, and the last four can only be accessed via given credentials. Repositories including version history for these ontologies are managed using *hosting platforms* as follows: 13 are in GitHub, four on Bitbucket and two of them are stored in Gitlab.

In the following, we focus our analysis on nine ontologies that are currently being developed by various teams. Table 11.1 presents an overview of these ontologies regarding to the name, domain, accessibility, hosting platform, level of expressiveness and number of triples. More details about ontologies in terms of number of classes, different types of properties, such as object properties, datatype properties, annotation properties from OWL schema and properties from RDF schema, are described in Table 11.2.

Table 11.3 provides statistics about the development process, such as number of commits, contributors, issues, branches, modules and languages used to define concepts for each of the selected ontologies. Numbers provided in the tables are frequently changing considering that the development process of these ontologies is currently ongoing.

Table 11.2: **Ontology statistics**. Metrics of the ontologies currently being developed in terms of number of classes, types of properties, such as object properties, datatype properties, annotation properties from OWL schema and properties from RDF schema.

Ontology	# classes	# total prop.	# object prop.	# datatype prop.	# ann. prop.	# rdf prop.	# individ.
STO	11	34	16	17	1	0	239
MobiVoc	27	147	22	51	1	73	80
SCORVoc	279	256	1	249	2	4	224
IDS	104	135	86	49	0	0	95
MatOnto	11	35	7	28	0	0	12
IASIS	90	120	58	62	0	0	2
SARO	29	47	0	0	0	47	0
Onto. #1	92	100	15	11	74	0	1075
Onto. #2	72	96	63	32	1	0	27

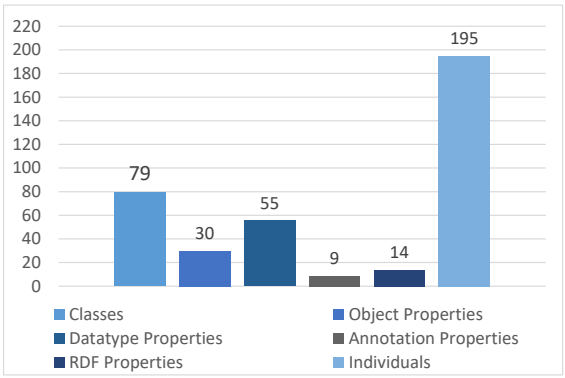
Table 11.3: **Development statistics**. Overview of the ontologies currently being developed in terms of number of commits, contributors, issues, branches, modules and languages.

Ontology	# commits	# contributors	# issues	# branches	# modules	# languages
STO	395	9	37	3	1	1
MobiVoc	319	13	40	4	4	2
SCORVoc	165	5	27	1	1	1
IDS	125	3	11	7	22	2
MatOnto	76	4	8	1	1	1
IASIS	349	3	0	1	1	0
SARO	31	2	0	1	1	1
Onto. #1	116	4	6	1	4	1
Onto. #2	208	2	5	1	1	7

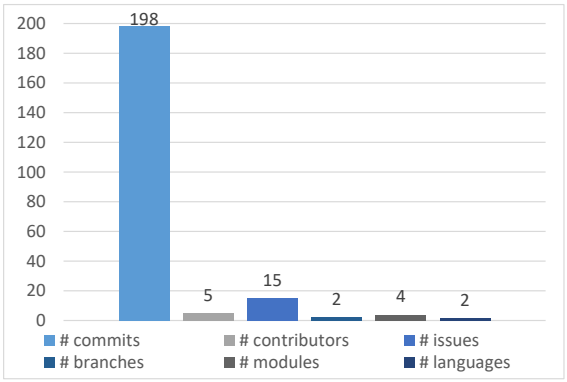
11.3 Analysis and Discussions

By analyzing the ontologies currently being developed, we can observe some interesting facts about them. These ontologies contain a total number of 715 classes, 970 properties and 1754 individuals. Splitting properties more in detail, 124 are properties from RDF schema and the rest are from OWL schema, where: 268 are *object properties*, 499 *datatype properties* and 79 *annotation properties*. Figure 11.2(a) depicts the average number of classes, properties and individuals per ontology. Overall, ontologies have an average number of 143 classes, 30 object properties, 55 datatype properties, 9 annotation properties, 14 RDF properties, and 195 individuals. Figure 11.2(b) illustrates average number of commits, contributors, issues, branches, modules and languages per ontology, respectively.

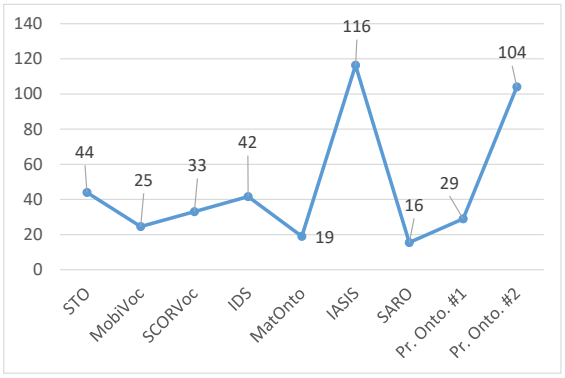
Ontology modularity As we can see from Figure 11.2(b), the average number of modules per ontology is four. The *Industrial Data Space* (IDS) ontology has the highest modularity rate with a total number of 22 modules, representing various functionalities of the IDS reference architecture. If we omit the IDS ontology, the average number of modules per ontology is decreased to two. Figure 11.2(d) presents more details related to number of classes and properties per module



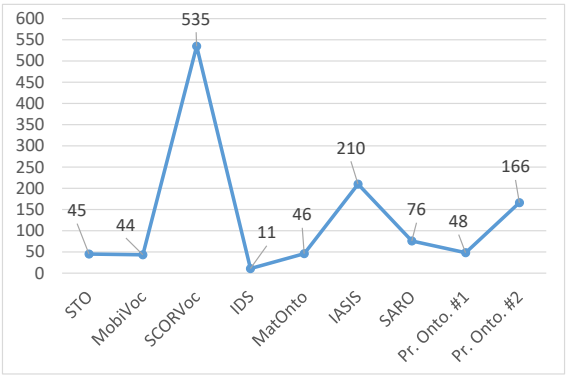
(a) Average metrics about ontologies



(b) Average development statistics



(c) Average number of commits per user for each ontology, respectively



(d) Average number of classes and properties per module per ontology

Figure 11.2: **Various statistics**. Four different statistics about the nine ontologies: a) Average number of classes, properties and individuals per ontology; b) Average number of commits, contributors, issues, branches, modules, and languages per ontology; c) Average number of commits per user per ontology; and d) Average number of classes and properties per module per ontology.

for each ontology. We can observe that majority of ontologies do not follow any pattern for modularization as presented in Subsection 5.1.1, such as number of triples or number of concepts per module. They are comprised of only one module, thus causing potential reusability difficulties or maintenance problems. To increase the modularity of the ontology, teams can create rules for defining new modules. Furthermore, ensuring the applicability of these rules can be realized using SPARQL queries to check whenever the ontology or a module reaches a particular number of concepts or triples. As a result, the team will be notified for the necessity of splitting into a new module in order to be able to continue with development process.

Ontology multilinguality Out of nine ontologies, eight of them have *labels* or *comments* in English language, three of them are also translated in German language, and one has translations in seven different languages: French, Italian, Japan, Polish and Spanish in addition to English and German. One ontology does not use any language tag at all, although the meaning of its concepts are explained using *rdfs:comment*. We can observe that the majority of ontologies are only translated in one language, which burden the objective of achieving a wide range of reusability. There might be several reasons for not providing translations in other languages

rather than English, such as: 1) the ontology is developed for general purposes; 2) is applicable in only a particular country; or 3) there is a lack of local experts who can translate terms in their own language. A suite of test cases along with their dependency relationships, can be evaluated using the *EffTE* approach forcing the definition of each concept in at least one language.

Ontology expressivity The ontologies in VoColReg have different levels of expressivity: $\mathcal{AL}(\mathcal{D})$ and $\mathcal{ALU}\mathcal{O}(\mathcal{D})$, each of them with two ontologies, whereas $\mathcal{SHOI}(\mathcal{D})$, $\mathcal{ALCHO}(\mathcal{D})$, $\mathcal{ALH}(\mathcal{D})$, $\mathcal{ALCHO}(\mathcal{D})$, $\mathcal{SRIF}(\mathcal{D})$, each with one ontology, respectively. We can observe that the ontologies are mainly expressed using the base language $\mathcal{AL}$ or attributive language, which allows for the definition of: *atomic concepts*, *conjunctions*, *value restrictions* and *limited existential quantification*. Two ontologies use an additional $\mathcal{S}$, which is the abbreviation for $\mathcal{ALC}$ or attributive language with transitive roles. $\mathcal{ALC}$ includes full negation and disjunction, boolean values (*and*, *or*, *not*) as well as restrictions on role values. More details about the meaning for the above mentioned letters are explained as below [172]:

- $\mathcal{U}$ Concept union;
- $\mathcal{C}$ Complex concept negation;
- $\mathcal{H}$ Role hierarchy with *rdfs:subPropertyOf*;
- $\mathcal{F}$ Functional properties, a particular case of uniqueness quantification;
- $\mathcal{R}$ Axioms for complex role inclusion and disjointness; reflexivity and irreflexivity;
- $\mathcal{O}$ Nominals, such as enumerated classes of object value restrictions: *owl:oneOf*, *owl:hasValue*;
- $\mathcal{I}$ Inverse properties;
- $(\mathcal{D})$ Usage of datatype properties or data values.

11.4 Summary

In this chapter, we describe VoColReg, a registry which offers searching and publishing capabilities for ontologies. The incorporation of a number of rich interfaces, allows VoColReg to perform as a generic infrastructure, thus enabling a community-based ontology development through a peer-review and collaborative process. Users can easily browse the hosted ontologies, explore and asses them for a potential reuse while navigating over defined concepts and understanding their relationships between each other. Moreover, using a standard tree view, the hierarchy of concepts modeled either through RDFS or SKOS vocabularies, can be displayed. In addition to the user interface, a variety of access mechanisms are provided, with the objective of facilitating the reuse of ontologies from humans and software agents. These mechanisms include: dedicated SPARQL endpoints per ontology, content negotiation and dereferenceability. We gave an overview of ontologies hosted in VoColReg with a focus on those that are still in the development process. Details about number of triples, classes, properties and instances as well as the number of commits, branches, contributors, are presented. Finally, we describe the outcome of the analysis over these ontologies. From our findings, we observe that some of them do not follow any strategy for branching, so the entire development process is driven using only one branch. Few ontologies follow a modularization approach, splitting concepts according to specific subdomains. Regarding multilinguality, the majority of them have defined *rdfs:label* and *rdfs:comment* in only one language, concretely in English language. An ontology does not associate any language tag to *rdfs:label* and *rdfs:comment* to define the meaning of its concepts. In addition to these aspects, we discuss several features to be used or implemented in order to enforce the stakeholders for following specific guidelines and best practices or avoid constraint violations.

CHAPTER 12

Conclusions and Future Direction

In this thesis, we study the problem of supporting collaborative ontology development in distributed and heterogeneous environments. We devise a lightweight methodology and a platform to enable stakeholders working together in such environments. We discuss the problem and challenges to be addressed in Chapter 1. The background with fundamental terminology essential for this thesis and an overview of the related work are presented in Chapter 2 and Chapter 3, respectively. In Part II, we collect a number of requirements that guide the approach followed in this thesis. The next chapters describe the defined approach from the methodology and technical perspectives. Part III dives more into detail to the problem of ensuring quality of ontologies in terms of effective synchronization and efficient evaluation in distributed scenarios. Part IV initially describes several approaches to support the semantic interoperability and data integration using ontologies. Furthermore, we provide details about the application of our approach in concrete uses cases. Finally, in this chapter, we conclude the thesis by assessing the results with respect to the research questions. Moreover, Section 12.2 presents future work that can expand the work of this thesis in both directions: research and technical.

12.1 Revisiting the Research Questions

As we stated in the introduction, the main goal of this thesis is to advance the field of collaborative ontology development by providing methodological and technical support. In this regard, we split the core problem of collaborative ontology development in distributed environments into four research questions. The first research question is related to the methodological support for facilitating stakeholders to organize and manage their collaborative work along with an analysis of best practices for ontology development. The second research question aims at transferring ontology development workflows to version control methods. The third research questions deals with the problem of synchronization of changes performed with various ontology authoring editors. Finally, the objective of the fourth research question is investigating the possibilities to efficiently ensure quality of ontologies considering predefined domain requirements.

In order to answer these research questions, we initially investigated collaborative ontology development as the process of identifying the main terms for a domain of interest by finding a consensus between the involved stakeholders. Next, we analyzed the fundamental steps of this process: *modeling*, *population* and *testing*, usually performed in an iterative and incremental fashion. Moreover, a number of widely reused ontologies are studied for their commonalities in terms of important development aspects, such as reuse, documentation, multilinguality, naming,

validation and authoring. Based on the findings, we collected a number of crucial requirements for the process of developing ontologies in collaborative fashion in Chapter 4. In this section, we go through the defined research questions and summarize the achieved contributions, respectively.

RQ1: Which guidelines and best practices facilitate collaborative ontology development in distributed and heterogeneous scenarios?

While answering this research question, we have observed that applying guidelines and best practices foster ontology development in collaborative scenarios. As a result, activities and design decisions taken over the time are clearly organized and managed by responsible stakeholders. Several exiting methodologies outline a predefined number of activities to be performed in a systematic way, while other methodologies describe best practices for an agile-oriented ontology construction. In the context of this research question, Chapter 5 presents *Git4Voc*, a lightweight methodology comprising an exhaustive list of guidelines and best practices for collaboratively developing ontologies. Git4Voc is conceived following a bottom-up approach by exploiting fundamental features of Git, applicable for ontologies along with state of the art practices. Its governing guidelines covers aspects, such as: 1) administration of generated information; 2) rights management; 3) branching and merging; 4) ontology modularization; and 5) labeling of release versions, to facilitate ontology construction using a version control. Moreover, from our analysis of widely used ontologies and examination of the state of the art, we defined a set of best practices related to reuse, naming convention, documentation, authoring, validation and multilinguality. The presented methodology is evaluated against a concrete use case as well as with a survey where ontology experts are asked for their opinion regarding the defined practices. By applying Git4Voc to the *Schema.org* use case, a clear organization of the ontology structure, communication issues and special branches is achieved, thus helping team members to easily collaborate and contribute in various development phases. Finally, from the survey with ontology experts, we observed that the majority of the defined practices received good evaluation results. Therefore, we can conclude that Git4Voc is indeed beneficial and provides comprehensive support for collaboratively developing ontologies in distributed and heterogeneous environments.

RQ2: How can ontology development workflows be mapped on and supported by distributed version control methods?

By investigating the actual trend of building ontologies in collaborative scenarios, which implies: 1) organizing the entire development process centered around the version control systems; 2) involving a huge community with diverse views and different requirements; and 3) reusing existing ontologies as much as possible, instead of constructing them from scratch, we perceived a high demand for a flexible and easily adaptable solution to be used in various cases. Therefore, the ability to map development workflows with distributed version control methods is a crucial factor that allows ontology modeling in such scenarios. Although current state of the art approaches cover a wide range of the development activities, they lack of support for dynamically changing and heterogeneous environments. In this thesis, concretely in Chapter 6, we present an approach and its realization based on technical requirements collected in Chapter 4 to answer **RQ2**. *VoCol* is an integrated environment for the distributed development of ontologies based on version control systems. It has a modular architecture where an orchestration service is able to invoke different components in an automatic fashion. VoCol has been implemented following principles for *extensibility*, *interoperability* and *customisability*. The modularity enables the

platform to be extended by adding new components or replacing existing ones. The principle of *interoperability* is addressed in two directions. First, VoCol is an interoperable platform that can be deployed in different operation systems. Second, it enables modeling of ontologies with heterogeneous editors. Users are granted with the possibility to customize the services or components to be automatically invoked according to their use case. The VoCol platform has a rich interface to assist reusability of ontologies being developed whereas it offers various access mechanisms for external users and machines. Furthermore, Chapter 11 presents *VoColReg*, a registry of ontologies developed on top of the VoCol platform. Currently, it hosts 19 ontologies from various domains that are supported during entire development process. Considering the evidences from its applicability in a number of concrete use cases, we argue that VoCol enables accommodation of diverse ontology development workflows with distributed version control methods while providing essential support for constructing ontologies in collaborative scenarios.

RQ3: How can concurrent changes from heterogeneous ontology authoring editors be effectively synchronized?

By analyzing scenarios where multiple stakeholders simultaneously perform changes in ontology replicas, we observed that employing mechanisms to effectively realize the synchronization process avoids from a potential loss of changes and introducing of syntactic and semantic errors. Distributed version control systems follow *optimistic* approaches to allow *parallel* modification of the ontology artifacts. These systems usually utilize line-by-line comparison techniques to detect conflicts of different versions of the same file. Since the position of the triple within an RDF file is semantically irrelevant, ontologies can be serialized using different ordering criteria. Consequently, a large number of *false-positive* conflicts, i.e., conflicts that do not result from ontology changes but from the fact that two ontology versions are differently serialized can be detected. To answer the research question **RQ3**, we describe an approach that tackles the problem of change synchronization performed with heterogeneous ontology authoring editors. Following the principle *prevention is better than cure*, Chapter 7 presents *SerVCS*, an approach to enhance version control for coping with different serializations of the same ontology. As a result, version control systems become *editor agnostic*, i.e., capable to detect actual changes and resolve syntactical conflicts using *built-in* merging algorithms. Thus, incompatibility problems with regard to wrongly detected conflicts caused from the use of different authoring editors are avoided. From empirical results, we can conclude that SerVCS is significantly more effective compared to plain Git on reducing the number of false-positive conflicts, including scenarios when the ontology size or sorting criteria is changed.

RQ4: How can the quality and efficiency in distributed and heterogeneous ontology development be ensured?

While answering this research question, we observed that ensuring compliance with respect to domain requirements is one of the critical aspects to achieve qualitative ontologies. A set of test cases derived from domain requirements are used to prevent from non-intended ontology changes and violation of constraints. Moreover, we perceived that having methods to efficiently evaluate a given set of test cases has a positive impact on the quality. As the state of the art approaches allow only for an exhaustive evaluation of the set of test cases, in Chapter 8 we present *EffTE*, an approach for ensuring quality and efficiency during ontology construction based on principles of the *test-driven development* technique. EffTE relies on a dependency graph between test cases

modeled by users to enable priority evaluation. The *breadth first search* algorithm is used to traverse the given dependency graph. An additional mechanism stores test cases to be excluded from further evaluation due to faulty parent tests. This leads to a minimal evaluation of the number of test cases, increasing the fault detection effectiveness and decreasing the test case validation time. Based on an empirical evaluation, the efficiency of EffTE is significantly higher compared to a naive approach. This is valid in various scenarios with different ontology sizes, dependency graph topologies, and number of test cases, whenever a faulty test case exists. EffTE is able to cope with additional parameters, such as users who perform changes and modify ontology files (if an ontology comprises multiple files). Therefore, the selection of test cases to be evaluated is user- and file-specific. By utilizing EffTE as a an integrated service in the VoCol platform to cover *client-side* and *server-side* activities performed before and after occurrences of *push* events, respectively, we argue that stakeholders are enforced to efficiently model qualitative ontologies according to the given domain requirements.

The main outcome of this thesis is a holistic approach bridging together methodological and technical dimensions of the core problem of *collaborative ontology development in distributed and heterogeneous environments*. We conclude that this approach provides a comprehensive support for a wide range of aspects and activities, such as design decisions, change synchronization, quality assurance, maintainability, publication and exploration. Hence, stakeholders can effectively and efficiently build ontologies in collaborative scenarios whereas third party users can exploit and consume the results via a number of integrated components and services.

12.2 Future Work

There exist various directions for future work as extension of this thesis, as well as advancements of actual limitations. In this section, we explore a number of potential research and technical directions, while keeping the same motivation and objectives presented in the thesis.

12.2.1 Research Perspective

In the following, we discuss several possible work streams that can be realized from the research perspective, including new complementary areas to the objectives of the thesis.

Improving ontologies via crowdsourcing Our approach is focusing on supporting stakeholders for collaboratively developing ontologies. However, ontologies are dynamic artifacts that evolve over the time, to represent the knowledge from a domain of interest. Although, much effort is invested in the development process, there exists always the possibility that ontologies suffer from missing concepts or relationships between them. Facilitating the process of continuous ontology evolution, can be done via crowdsourcing, i.e., involving a wider and unidentified crowd in the process. Questions Answering (QA) systems are a new technique for answering natural language questions in an automatic fashion by exploiting facts that are encoded in a Knowledge Graph (KG) [173]. Ontologies can be the backbone of these QA systems, and are commonly open to the crowd. Thus, any question that cannot be answered by the systems is a potential indication for missing concepts or relationships. Our methodology and platform can be extended to accommodate the integration of other systems, such as QA, to foster the evolution of ontologies. The automatically recommended changes are later assessed by the team, thus causing ontology enrichment or correction, if necessary. The new version is applied to the QA system, so the previously not answered questions can now be answered.

Machine-supported ontology modularization Modularization of the large monolithic ontologies is crucial for avoiding the reusability, scalability and maintenance problems [83]. Although in Git4Voc and VoCol, we provide the respective guidelines and mechanisms on how to perform the modularization, the human intervention to accomplish this task is still required. Combining the *network analysis* and *unsupervised classification* methods to automatically generate standalone modules would lead to a significant improvement towards dealing with the aforementioned problems. The decomposition can be realized based on the hierarchy, structure or the existence of subdomains within a large ontology. Creating domain specific ontologies by composing a number of standalone modules, is another interesting topic is this regard. Leveraging a *conceptual clustering* paradigm along with *similarity measure* techniques can lead to an effective way towards machine-aided ontology construction.

Detecting and resolving semantic conflicts during change synchronization The presented approach for change synchronization is currently limited to syntactic conflicts. In the future, we envision to develop new techniques for automatically detecting and resolving semantic conflicts. Incorporation of probabilistic models and machine learning mechanisms would empower the ability of SerVCS to achieve a more accurate conflict detection and resolution. A semantic layer can be added to prevent inconsistencies caused as a result of merging two different versions of the same ontology. Another crucial factor during synchronization process is the amount of time needed for conflict detection and resolution. Efficient techniques should be investigated and developed to avoid additional overload during ontology modeling. As a result, effectiveness and efficiency of VoCol will be improved on resolving both, syntactic and semantic conflicts.

Smart and continuous test cases evaluation To ensure qualitative ontologies, users are forced to manually create test cases and the dependency relationships between them. This might require large efforts, in particular for teams which lack ontology engineers. Adding features for automatic generation of test cases based on domain requirements as well as creation of their relationships in a dependency graph may lead to a significant improvement of the ontology quality. Moreover, the approach can be advanced with mechanisms for dynamic prioritization and selection of the test cases and limit their evaluation only to the recent ontology modifications. Thus, a new version can then be assessed with regard to only modified parts and a full ontology checking is avoided. The challenge to be considered here is that in many cases changes may affect multiple concepts and relationships in different ways.

12.2.2 Technical Perspective

The technical dimension is the other area in which our work can be extended and improved. In the following, we describe in detail three important aspects for increasing flexibility and lowering barriers of using our holistic approach in different scenarios.

VoCol as a Service In order to use VoCol, the team should install it either on its own server or on the cloud and configure it to monitor their repository where the ontology is hosted. Providing *VoCol as a Service* (VaaS), where users can simply subscribe the repositories and benefit from all its functionalities will enable a wide range of usage in various domains. The envisioned architecture of VaaS is illustrated in Figure 12.1. It is composed of five layers: 1) *persistence* - enable modeling and storing ontologies; 2) *orchestration* - invokes and coordinates automatic execution of a number of components; 3) *validation and synchronization* - responsible

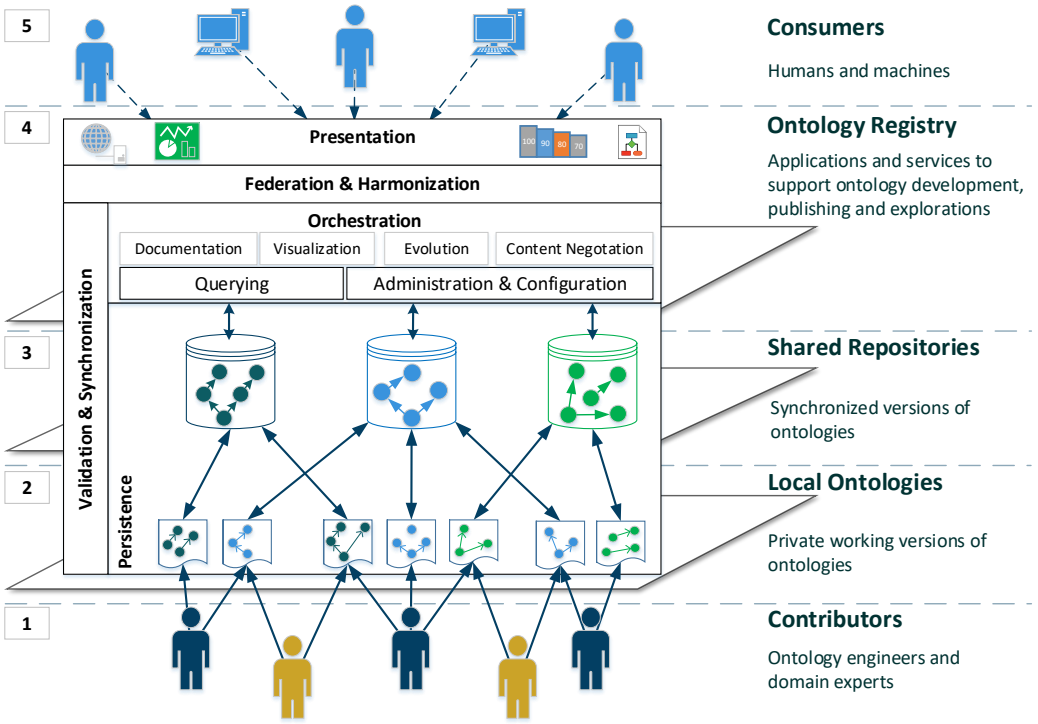


Figure 12.1: **The architecture of VoCol-as-a-Service from a technical perspective**. It is composed of five layers containing a number of different components and services: 1) persistence; 2) orchestration; 3) validation and synchronization; 4) federation; and 5) presentation.

for quality checking as well as detecting and resolving conflicts; 4) *federation* - allows federated querying over multiple internal and external ontologies; and 5) *presentation* - offers rich forms for facilitating exploration and navigation of ontologies. VaaS will act as a hosting registry of many ontologies where a number of different analysis and studies can be performed over them.

Plugin-based architecture and workflow VoCol has a modular architecture where different components can be easily exchanged. However, to realized that, a small portion of code is required for allowing the *orchestration* service to invoke them and present the outcome in an appropriate fashion. This hinders the ability of users to employ other components necessary for dealing with specific tasks during ontology modeling. The implementation of a plugin-based mechanism along with a metalanguage to describe the functionality of components, their invocation interface, and the way to present the results, would facilitate adoption of VoCol for various use cases.

Synchronous development The organization of work into VoCol is inherited from Git, which uses the *optimistic* approach, i.e., *clone-modify-merge*. After cloning the repository, stakeholders can asynchronously perform ontology modifications on their local copies and later merge with changes from the others. However, further optimization of this platform for domain experts requires providing a web interface based on the *pessimistic* approach, i.e., *lock-modify-unlock*. This would enable a synchronous development in scenarios where stakeholders are aware in real-time for actual changes and prevented from overwriting modifications of each other.

Bibliography

[1] D. L. Rubin, N. Shah and N. F. Noy, *Biomedical ontologies: a functional perspective*, Briefings in Bioinformatics **9** (2008) 75, URL: https://doi.org/10.1093/bib/bbm059 (cit. on p. 3).

[2] *Vocabularies*, W3C Report, World Wide Web Consortium (W3C), 2015, URL: https://www.w3.org/standards/semanticweb/ontology (visited on 12/03/2018) (cit. on p. 3).

[3] J. M. Juran, *Juran's Quality Control Handbook*, 4th, Mcgraw-Hill (Tx), 1974, ISBN: 0070331766, URL: https://www.amazon.com/gp/product/0070331766 (cit. on p. 3).

[4] M. Grüninger and M. S. Fox, "Methodology for the design and evaluation of ontologies", *IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995 (cit. on pp. 4, 51, 109).

[5] E. Simperl and M. Luczak-Rösch, *Collaborative ontology engineering: a survey*, Knowledge Engineering Review **29** (2014) 101, URL: https://doi.org/10.1017/S0269888913000192 (cit. on pp. 6, 25, 31, 52, 55, 56).

[6] R. Angles and C. Gutiérrez, *Survey of graph database models*, ACM Computing Surveys (CSUR) **40** (2008) 1, URL: http://doi.acm.org/10.1145/1322432.1322433 (cit. on p. 15).

[7] S. M. Cohen, "Aristotle's Metaphysics", *The Stanford Encyclopedia of Philosophy*, Winter, Metaphysics Research Lab, Stanford University, 2016 (cit. on p. 15).

[8] T. R. Gruber, *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*, International Journal of Human-Computer Studies **43** (1995) 907, ISSN: 1071-5819, URL: http://dx.doi.org/10.1006/ijhc.1995.1081 (cit. on p. 15).

[9] R. Studer, V. R. Benjamins and D. Fensel, *Knowledge Engineering: Principles and Methods*, Data & Knowledge Engineering **25** (1998) 161, URL: https://doi.org/10.1016/S0169-023X(97)00056-6 (cit. on p. 15).

[10] A. Gómez-Pérez, M. Fernández-López and Ó. Corcho, *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*, Advanced Information and Knowledge Processing, Springer, 2004, ISBN: 978-1-85233-551-9, URL: https://doi.org/10.1007/b97353 (cit. on pp. 15, 23, 51, 71).

[11] M. R. Genesereth and N. J. Nilsson, *Logical foundations of artificial intelligence*, Morgan Kaufmann, 1988, ISBN: 978-0-934613-31-6 (cit. on p. 15).

[12] F. Manola, M. Eric and B. McBride, *RDF Primer*, W3C Recommendation, World Wide Web Consortium (W3C), 2014, URL: https://www.w3.org/TR/rdf-primer/ (visited on 12/02/2018) (cit. on p. 16).

[13] M. Arenas, C. Gutiérrez and J. Pérez, "Foundations of RDF Databases", *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School, Brixen-Bressanone, Italy, Tutorial Lectures*, 2009 158, URL: https://doi.org/10.1007/978-3-642-03754-2_4 (cit. on p. 16).

[14] D. Brickley and R. Guha, *RDF Schema 1.1*, W3C Recommendation, World Wide Web Consortium (W3C), 2014, URL: https://www.w3.org/TR/rdf-schema/ (visited on 26/02/2018) (cit. on p. 20).

[15] S. Staab and R. Studer, eds., *Handbook on Ontologies*, International Handbooks on Information Systems, Springer, 2004, ISBN: 3-540-40834-7 (cit. on p. 20).

[16] M. K. Smith, C. Welty and D. L. McGuinness, *OWL Web Ontology Language Guide*, W3C Recommendation, World Wide Web Consortium (W3C), 2018, URL: http://www.w3.org/TR/owl-guide/ (visited on 05/04/2018) (cit. on p. 20).

[17] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue and C. Lutz, *OWL 2 Web Ontology Language Profiles*, W3C Recommendation, World Wide Web Consortium (W3C), 2018, URL: https://www.w3.org/TR/owl-profiles/ (visited on 05/04/2018) (cit. on p. 21).

[18] S. Harris and A. Seaborne, *SPARQL 1.1 Query Language*, W3C Recommendation, World Wide Web Consortium (W3C), 2013, URL: http://www.w3.org/TR/sparql11-query/ (visited on 26/02/2018) (cit. on p. 21).

[19] J. Pérez, M. Arenas and C. Gutiérrez, *Semantics and complexity of SPARQL*, ACM Trans. Database Syst. **34** (2009) 16:1, URL: http://doi.acm.org/10.1145/1567274.1567278 (cit. on pp. 22, 112).

[20] T. Berners-Lee, J. Hendler and O. Lassila, *The Semantic Web*, Scientific American **284** (2001) 34 (cit. on p. 22).

[21] M. Fernández-López, A. Gómez-Pérez and N. Juristo, "METHONTOLOGY: From Ontological Art Towards Ontological Engineering", *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*, Ontology Engineering Group (OEG), American Asociation for Artificial Intelligence, 1997 (cit. on pp. 23, 38).

[22] H. S. A. N. P. Pinto and J. P. Martins, *Ontologies: How can They be Built?*, Knowledge and Information Systems **6** (2004) 441 (cit. on p. 24).

[23] S. Fraser, K. L. Beck, B. Caputo, T. Mackinnon, J. Newkirk and C. Poole, "Test Driven Development (TDD)", *Extreme Programming and Agile Processes in Software Engineering, 4th International Conference, XP, Genova, Italy. Proceedings*, 2003 459, URL: https://doi.org/10.1007/3-540-44870-5_84 (cit. on pp. 25, 109).

[24] S. Peroni, "A Simplified Agile Methodology for Ontology Development", *OWL: - Experiences and Directions - Reasoner Evaluation - 13th International Workshop, (OWLED), and 5th International Workshop, ORE, Bologna, Italy, Revised Selected Papers*, 2016 55 (cit. on pp. 25, 26).

[25] B. de Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?",
Proceedings of the ICSE Workshop on Cooperative and Human Aspects on Software Engineering, (CHASE), Vancouver, BC, Canada, 2009 36,
URL: https://doi.org/10.1109/CHASE.2009.5071408 (cit. on p. 26).

[26] T. Mens, *A State-of-the-Art Survey on Software Merging*,
IEEE Transactions on Software Engineering **28** (2002) 449 (cit. on pp. 27, 28, 95).

[27] K. Altmanninger, M. Seidl and M. Wimmer, *A survey on model versioning approaches*,
International Journal of Web Information Systems **5** (2009) 271,
URL: https://doi.org/10.1108/17440080910983556 (cit. on pp. 28, 95).

[28] B. Appleton, S. Berczuk, R. Cabrera and R. Orenstein,
"Streamed lines: Branching patterns for parallel software development",
5th Annual Conference on Pattern Languages of Program Design, (PLoP), Proceedings,
vol. 98, 1998 (cit. on pp. 29, 55).

[29] R. Palma, Ó. Corcho, A. Gómez-Pérez and P. Haase,
A holistic approach to collaborative ontology development based on change management,
Journal of Web Semantics **9** (2011) 299,
URL: https://doi.org/10.1016/j.websem.2011.06.007 (cit. on pp. 31, 44, 78, 95).

[30] K. Kotis and G. A. Vouros,
Human-centered ontology engineering: The HCOME methodology,
Knowledge and Information Systems **10** (2006) 109,
URL: https://doi.org/10.1007/s10115-005-0227-4 (cit. on p. 32).

[31] S. Braun, A. P. Schmidt, A. Walter, G. Nagypál and V. Zacharias,
"Ontology Maturing: a Collaborative Web 2.0 Approach to Ontology Engineering",
Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC) at the 16th International World Wide Web Conference (WWW) Banff, Canada, 2007 (cit. on p. 33).

[32] H. S. Pinto, S. Staab and C. Tempich, "DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolving Engineering of oNTologies", *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI), Valencia, Spain*, 2004 393 (cit. on pp. 34, 38).

[33] C. Tempich, H. S. Pinto and S. Staab,
"Ontology Engineering Revisited: An Iterative Case Study", *3rd European Semantic Web Conference, (ESWC), Research and Applications, Budva, Montenegro, Proceedings*, 2006 110, URL: https://doi.org/10.1007/11762256_11 (cit. on p. 35).

[34] M. Annamalai and L. Sterling,
"Guidelines for Constructing Reusable Domain Ontologies",
Ontologies in Agent Systems, Proceedings of the Workshop on Ontologies in Agent Systems (OAS) at the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, Melbourne, Australia, 2003 71 (cit. on pp. 35, 38).

[35] S. Auer and H. Herre, "RapidOWL - An Agile Knowledge Engineering Methodology",
Perspectives of Systems Informatics, 6th International Andrei Ershov Memorial Conference, PSI, Novosibirsk, Russia. Revised Papers, 2006 424,
URL: https://doi.org/10.1007/978-3-540-70881-0%5C_36 (cit. on pp. 36, 38).

[36] H. Knublauch, *An agile development methodology for knowledge-based systems including a Java framework for knowledge modeling and appropriate tool support*,
PhD thesis: University of Ulm, Germany, 2002,
URL: http://vts.uni-ulm.de/docs/2002/2101/vts_2101.pdf (cit. on p. 37).

[37] P. D. Maio, "'Just enough' ontology engineering", *Proceedings of the International Conference on Web Intelligence, Mining and Semantics, (WIMS), Sogndal, Norway*, 2011 8, URL: http://doi.acm.org/10.1145/1988688.1988698 (cit. on pp. 37, 38).

[38] L. Dodds and I. Davis, *Linked data patterns*,
Online: http://patterns.dataincubator.org/book (2011) (cit. on p. 38).

[39] M. C. Suárez-Figueroa,
NeOn methodology for building ontology networks: specification, scheduling and reuse,
PhD thesis: Technical University of Madrid, 2012, ISBN: 978-3-89838-338-7,
URL: http://d-nb.info/1029370028 (cit. on pp. 38, 71).

[40] Y. Sure, S. Staab and R. Studer, "On-To-Knowledge Methodology (OTKM)",
Handbook on Ontologies, ed. by S. Staab and R. Studer,
International Handbooks on Information Systems, Springer, 2004 117,
ISBN: 3-540-40834-7 (cit. on p. 38).

[41] V. Zacharias and S. Braun,
"SOBOLEO – Social Bookmarking and Lighweight Engineering of Ontologies",
Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC) at the 16th International World Wide Web Conference (WWW) Banff, Canada, 2007 (cit. on p. 38).

[42] C. Ghidini, M. Rospocher and L. Serafini,
"MoKi: a Wiki-Based Conceptual Modeling Tool",
Proceedings of the nternational Semantic Web Conference, (ISWC), Posters & Demonstrations Track: Collected Abstracts, Shanghai, China, 2010 (cit. on p. 39).

[43] T. Tudorache, C. Nyulas, N. F. Noy and M. A. Musen,
WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web.,
Semantic Web **4** (2013) 89, URL:
http://dblp.uni-trier.de/db/journals/semweb/semweb4.html#TudoracheNNM13
(cit. on pp. 39, 42).

[44] N. F. Noy, A. Chugh, W. Liu and M. A. Musen,
"A Framework for Ontology Evolution in Collaborative Environments",
5th International Semantic Web Conference, (ISWC), Athens, GA, USA, Proceedings,
2006 544, URL: https://doi.org/10.1007/11926078_39 (cit. on pp. 39, 44, 55).

[45] A. Stellato, S. Rajbhandari, A. Turbati, M. Fiorelli, C. Caracciolo, T. Lorenzetti,
J. Keizer and M. T. Pazienza,
"VocBench: A Web Application for Collaborative Development of Multilingual Thesauri",
12th European Semantic Web Conference, (ESWC), Latest Advances and New Domains, Portoroz, Slovenia. Proceedings, 2015 38 (cit. on pp. 39, 42).

[46] T. Schandl and A. Blumauer,
"PoolParty: SKOS Thesaurus Management Utilizing Linked Data",
7th Extended Semantic Web Conference, (ESWC), Research and Applications, Heraklion, Crete, Greece, Proceedings, Part II, 2010 421 (cit. on pp. 40, 42).

[47] M. Luczak-Rösch, G. Coskun, A. Paschke, M. Rothe and R. Tolksdorf, "SVoNt - Version Control of OWL Ontologies on the Concept Level", *Informatik: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2, Leipzig, Deutschland*, 2010 79 (cit. on p. 41).

[48] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. S. Pérez and Ó. Corcho, "OnToology, a tool for collaborative development of ontologies", *Proceedings of the International Conference on Biomedical Ontology, (ICBO), Lisbon, Portugal.* 2015, URL: http://ceur-ws.org/Vol-1515/demo3.pdf (cit. on pp. 41, 42).

[49] D. Garijo, "WIDOCO: A Wizard for Documenting Ontologies", *16th International Semantic Web Conference, (ISWC), Vienna, Austria, Proceedings, Part II*, 2017 94, URL: https://doi.org/10.1007/978-3-319-68204-4%5C_9 (cit. on p. 41).

[50] M. Poveda-Villalón, M. C. Suárez-Figueroa and A. Gómez-Pérez, "Validating Ontologies with OOPS!", *Knowledge Engineering and Knowledge Management - 18th International Conference, (EKAW), Galway City, Ireland. Proceedings*, 2012 267 (cit. on pp. 41, 71).

[51] M. Codescu, E. Kuksa, O. Kutz, T. Mossakowski and F. Neuhaus, *Ontohub: A semantic repository engine for heterogeneous ontologies*, Applied Ontology **12** (2017) 275, URL: https://doi.org/10.3233/AO-170190 (cit. on pp. 41, 157).

[52] D. Dig, K. Manzoor, R. E. Johnson and T. N. Nguyen, "Refactoring-Aware Configuration Management for Object-Oriented Programs", *29th International Conference on Software Engineering, (ICSE), Minneapolis, MN, USA*, 2007 427, URL: https://doi.org/10.1109/ICSE.2007.71 (cit. on p. 43).

[53] T. Ekman and U. Asklund, *Refactoring-aware versioning in Eclipse*, Electronic Notes in Theoretical Computer Science **107** (2004) 57, URL: https://doi.org/10.1016/j.entcs.2004.02.048 (cit. on p. 43).

[54] H. V. Nguyen, M. H. Nguyen, S. C. Dang, C. Kästner and T. N. Nguyen, "Detecting semantic merge conflicts with variability-aware execution", *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, (ESEC/FSE), Bergamo, Italy*, 2015 926 (cit. on p. 43).

[55] D. Asenov, B. Guenat, P. Müller and M. Otth, "Precise Version Control of Trees with Line-Based Version Control Systems", *Fundamental Approaches to Software Engineering - 20th International Conference, (FASE), Uppsala, Sweden, Proceedings*, 2017 152 (cit. on p. 43).

[56] J. Protzenko, S. Burckhardt, M. Moskal and J. McClurg, "Implementing real-time collaboration in TouchDevelop using AST merges", *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle, (MobileDeLi), Pittsburgh, PA, USA*, 2015 25 (cit. on p. 43).

[57] Y. Brun, R. Holmes, M. D. Ernst and D. Notkin, "Proactive detection of collaboration conflicts", *19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) and 13th European Software Engineering Conference (ESEC)*, 2011 168 (cit. on p. 43).

[58] K. Altmanninger,
"Models in Conflict - A Semantically Enhanced Version Control System for Models",
Proceedings of the Doctoral Symposium at the 10th International Conference on Model-Driven Engineering Languages and Systems, Nashville, USA, 2007 (cit. on p. 43).

[59] K. Altmanninger, W. Schwinger and G. Kotsis, *Semantics for Accurate Conflict Detection in SMoVer: Specification, Detection and Presentation by Example*,
IJEIS **6** (2010) 68, URL: https://doi.org/10.4018/jeis.2010120206 (cit. on p. 43).

[60] P. Brosch, "Improving conflict resolution in model versioning systems",
31st International Conference on Software Engineering, (ICSE), Vancouver, Canada, Companion Volume, 2009 355,
URL: https://doi.org/10.1109/ICSE-COMPANION.2009.5071020 (cit. on p. 43).

[61] A. Cicchetti, D. D. Ruscio and A. Pierantonio,
"Managing Model Conflicts in Distributed Development", *11th International Conference on Model Driven Engineering Languages and Systems, (MoDELS)*, 2008 311
(cit. on p. 43).

[62] S. Krusche and B. Brügge, *Model-based Real-time Synchronization*,
Softwaretechnik-Trends **34** (2014) (cit. on p. 43).

[63] H. Chong, R. Zhang and Z. Qin,
"Composite-based conflict resolution in merging versions of UML models",
17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, (SNPD), Shanghai, China, 2016 127, URL: https://doi.org/10.1109/SNPD.2016.7515890 (cit. on p. 43).

[64] T. B. Lee and D. Connolly,
Delta: an ontology for the distribution of differences between RDF graphs, tech. rep.,
W3C, 2001 (cit. on p. 44).

[65] M. Völkel and T. Groza, "SemVersion: An RDF-based Ontology Versioning System",
IADIS International Conference on WWW/Internet, IADIS, 2006 195 (cit. on p. 44).

[66] S. Cassidy and J. Ballantine, "Version Control for RDF Triple Stores",
2nd International Conference on Software and Data Technologies (ICSOFT), 2007 5
(cit. on p. 44).

[67] W. K. Edwards,
"Flexible Conflict Detection and Management in Collaborative Applications",
Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST, Banff, Alberta, Canada, 1997 139 (cit. on p. 44).

[68] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. van Deemter and R. Stevens,
"Towards Competency Question-Driven Ontology Authoring",
11th Extended Semantic Web Conference, (ESWC), 2014 752 (cit. on pp. 45, 51, 56).

[69] C. M. Keet and A. Lawrynowicz, "Test-Driven Development of Ontologies",
13th Extended Semantic Web Conference, (ESWC), 2016 642,
URL: http://dx.doi.org/10.1007/978-3-319-34129-3_39 (cit. on p. 45).

[70] D. Vrandecic and A. Gangemi, "Unit Tests for Ontologies",
On the Move to Meaningful Internet Systems, OTM Workshops and Posters, 2006 1012,
URL: http://dx.doi.org/10.1007/11915072_2 (cit. on p. 45).

[71] H. Knublauch and D. Kontokostas, *Shapes Constraint Language (SHACL)*, Accessed: 2017-05-08, 2017, URL: https://www.w3.org/TR/shacl/ (visited on 08/05/2017) (cit. on p. 45).

[72] I. Boneva, J. E. L. Gayo and E. G. Prud'hommeaux, "Semantics and Validation of Shapes Schemas for RDF", *16th International Semantic Web Conference, (ISWC), Vienna, Austria, Proceedings, Part I*, 2017 104, URL: https://doi.org/10.1007/978-3-319-68288-4_7 (cit. on p. 45).

[73] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen and A. Zaveri, "Test-driven evaluation of linked data quality", *23rd International World Wide Web Conference, (WWW)*, 2014 747, URL: http://doi.acm.org/10.1145/2566486.2568002 (cit. on p. 45).

[74] S. García-Ramos, A. Otero and M. Fernández-López, "OntologyTest: A Tool to Evaluate Ontologies through Tests Defined by the User", *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, 10th International Work-Conference on Artificial Neural Networks, (IWANN) Workshops, Salamanca, Spain. Proceedings, Part II*, 2009 91, URL: https://doi.org/10.1007/978-3-642-02481-8%5C_13 (cit. on p. 45).

[75] S. Yoo and M. Harman, *Regression testing minimization, selection and prioritization: a survey*, Software Testing, Verification and Reliability **22** (2012) 67, URL: https://doi.org/10.1002/stv.430 (cit. on p. 45).

[76] G. Rothermel and M. J. Harrold, "A Safe, Efficient Algorithm for Regression Test Selection", *Proceedings of the Conference on Software Maintenance, (ICSM), Montréal, Quebec, Canada*, 1993 358, URL: https://doi.org/10.1109/ICSM.1993.366926 (cit. on p. 45).

[77] C. Giuliano and A. M. Gliozzo, "Instance-Based Ontology Population Exploiting Named-Entity Substitution", *Proceedings of the 22nd International Conference on Computational Linguistics, (COLING), Manchester, UK*, 2008 265 (cit. on p. 51).

[78] M. Schmachtenberg, C. Bizer and H. Paulheim, "Adoption of the Linked Data Best Practices in Different Topical Domains", *13th International Semantic Web Conference, (ISWC), Riva del Garda, Italy, Proceedings, Part I*, 2014 245 (cit. on p. 52).

[79] D. Schober, B. Smith, S. E. Lewis, W. Kusnierczyk, J. Lomax, C. Mungall, C. F. Taylor, P. Rocca-Serra and S. Sansone, *Survey-based naming conventions for use in OBO Foundry ontology development*, BMC Bioinformatics **10** (2009) (cit. on pp. 52, 68, 69).

[80] N. F. Noy and T. Tudorache, "Collaborative Ontology Development on the (Semantic) Web", *Symbiotic Relationships between Semantic Web and Knowledge Engineering, AAAI Spring Symposium, Technical Report SS-08-07, Stanford, California, USA*, 2008 63 (cit. on p. 55).

[81] J. Seidenberg and A. L. Rector, "The State of Multi-User Ontology Engineering", *Proceedings of the 2nd International Workshop on Modular Ontologies, (WoMO), Whistler, Canada*, 2007 (cit. on p. 55).

[82] M. C. Suárez-Figueroa, A. Gómez-Pérez and M. Fernández-López, "The NeOn Methodology for Ontology Engineering", *Ontology Engineering in a Networked World.* 2012 9, URL: https://doi.org/10.1007/978-3-642-24794-1%5C_2 (cit. on p. 55).

[83] M. d'Aquin, A. Schlicht, H. Stuckenschmidt and M. Sabou, "Ontology Modularization for Knowledge Selection: Experiments and Evaluations", *Database and Expert Systems Applications, 18th International Conference, (DEXA), Regensburg, Germany, Proceedings*, 2007 874 (cit. on pp. 55, 169).

[84] J. Gracia, E. Montiel-Ponsoda, P. Cimiano, A. Gómez-Pérez, P. Buitelaar and J. P. McCrae, *Challenges for the multilingual Web of Data*, Journal of Web Semantics **11** (2012) 63, URL: https://doi.org/10.1016/j.websem.2011.09.001 (cit. on pp. 56, 71).

[85] T. Redmond, M. Smith, N. Drummond and T. Tudorache, "Managing Change: An Ontology Version Control System", *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany*, 2008 (cit. on p. 57).

[86] S. Lohmann, S. Negru, F. Haag and T. Ertl, *Visualizing ontologies with VOWL*, Semantic Web **7** (2016) 399, URL: https://doi.org/10.3233/SW-150200 (cit. on p. 57).

[87] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers, 2011 1, URL: https://doi.org/10.2200/S00334ED1V01Y201102WBE001 (cit. on pp. 57, 67).

[88] S. Phillips, J. Sillito and R. J. Walker, "Branching and merging: an investigation into current version control practices", *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, (CHASE), Waikiki, Honolulu, HI, USA*, 2011 9, URL: http://doi.acm.org/10.1145/1984642.1984645 (cit. on p. 63).

[89] E. Shihab, C. Bird and T. Zimmermann, "The effect of branching strategies on software quality", *2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, (ESEM), Lund, Sweden*, 2012 301, URL: http://doi.acm.org/10.1145/2372251.2372305 (cit. on p. 63).

[90] A. Schlicht and H. Stuckenschmidt, "Towards Structural Criteria for Ontology Modularization", *Proceedings of the 1st International Workshop on Modular Ontologies, (WoMO), co-located with the International Semantic Web Conference,(ISWC), Athens, Georgia, USA*, 2006 (cit. on p. 66).

[91] S. B. Abbès, A. Scheuermann, T. Meilender and M. d'Aquin, "Characterizing Modular Ontologies", *Proceedings of the 6th International Workshop on Modular Ontologies, Graz, Austria*, 2012 (cit. on p. 66).

[92] P. Bedi and S. Marwaha, *Versioning OWL Ontologies using Temporal Tags*, International Journal of Computer, Electrical, Automation, Control and Information Engineering **1** (2007) 686 (cit. on p. 67).

[93] J. P. Diane I. Hillmann Gordon Dunsire, "Versioning Vocabularies in a Linked Data World", *World Library and Information Congress, (IFLA), Lion, France*, 2014 (cit. on p. 67).

[94] M. Poveda-Villalón, "A Reuse-Based Lightweight Method for Developing Linked Data Ontologies and Vocabularies", *9th Extended Semantic Web Conference, (ESWC), Heraklion, Crete, Greece. Proceedings*, 2012 833, URL: https://doi.org/10.1007/978-3-642-30284-8%5C_66 (cit. on p. 67).

[95] C. Pedrinaci, J. S. Cardoso and T. Leidig, "Linked USDL: A Vocabulary for Web-Scale Service Trading", *11th International Conference - Trends and Challenges, (ESWC), Anissaras, Crete, Greece. Proceedings*, 2014 68 (cit. on pp. 67, 68).

[96] B. Haslhofer, F. Martins and J. Magalhães, "Using SKOS vocabularies for improving web search", *22nd International World Wide Web Conference, (WWW), Rio de Janeiro, Brazil, Companion Volume*, 2013 1253, URL: http://doi.acm.org/10.1145/2487788.2488159 (cit. on p. 68).

[97] N. A. A. Manaf, S. Bechhofer and R. Stevens, "The Current State of SKOS Vocabularies on the Web", *9th Extended Semantic Web Conference, (ESWC), Research and Applications, , Heraklion, Crete, Greece. Proceedings*, 2012 270, URL: https://doi.org/10.1007/978-3-642-30284-8%5C_25 (cit. on p. 68).

[98] T. Baker, S. Bechhofer, A. Isaac, A. Miles, G. Schreiber and E. Summers, *Key choices in the design of Simple Knowledge Organization System (SKOS)*, Journal of Web Semantics **20** (2013) 35 (cit. on p. 68).

[99] S. Peroni, D. M. Shotton and F. Vitali, *Tools for the Automatic Generation of Ontology Documentation: A Task-Based Evaluation*, International Journal on Semantic Web and Information Systems (IJSWIS) **9** (2013) 21, URL: https://doi.org/10.4018/jswis.2013010102 (cit. on p. 69).

[100] D. Schober, I. Tudose, V. Svátek and M. Boeker, *OntoCheck: verifying ontology naming conventions and metadata completeness in Protégé 4*, Journal of Biomedical Semantics **3** (2012) S4 (cit. on p. 69).

[101] V. Svátek and O. Šváb-Zamazal, "Entity naming in semantic web ontologies: Design patterns and empirical observations", 2010 1 (cit. on p. 69).

[102] E. Montiel-Ponsoda, D. Vila-Suero, B. Villazón-Terrazas, G. Dunsire, E. E. Rodriguez and A. Gómez-Pérez, "Style Guidelines for Naming and Labeling Ontologies in the Multilingual Web", *Proceedings of the International Conference on Dublin Core and Metadata Applications, DC, The Hague, The Netherlands*, 2011 105 (cit. on p. 69).

[103] V. Svátek, O. Sváb-Zamazal and V. Presutti,
"Ontology Naming Pattern Sauce for (Human and Computer) Gourmets",
Proceedings of the Workshop on Ontology Patterns (WOP), collocated with the 8th International Semantic Web Conference (ISWC), Washington D.C., USA, 2009
(cit. on p. 69).

[104] M. Kezadri and M. Pantel, "First Steps Toward a Verification and Validation Ontology",
Proceedings of the International Conference on Knowledge Engineering and Ontology Development, (KEOD), Valencia, Spain, 2010 440 (cit. on p. 71).

[105] S. Abburu, *A survey on ontology reasoners and comparison*,
International Journal of Computer Applications **57** (2012) 33 (cit. on p. 72).

[106] H. N. Boone and D. A. Boone, *Analyzing likert data*, Journal of Extension **50** (2012) 1
(cit. on p. 74).

[107] A. Kaur and K. S. Mann, *Component based software engineering*,
International Journal of Computer Applications **2** (2010) 105 (cit. on p. 78).

[108] N. Petersen, G. Coskun and C. Lange, "TurtleEditor: An Ontology-Aware Web-Editor for Collaborative Ontology Development",
10th International Conference on Semantic Computing, IEEE, 2016 183,
URL: http://dx.doi.org/10.1109/ICSC.2016.26 (cit. on p. 82).

[109] K. M. Endris, M. Galkin, I. Lytra, M. N. Mami, M. Vidal and S. Auer,
"MULDER: Querying the Linked Data Web by Bridging RDF Molecule Templates",
Database and Expert Systems Applications - 28th International Conference, (DEXA), Lyon, France, Proceedings, Part I, 2017 3 (cit. on p. 85).

[110] S. Lohmann, V. Link, E. Marbach and S. Negru,
"WebVOWL: Web-based Visualization of Ontologies", *Knowledge Engineering and Knowledge Management, (EKAW), Satellite Events, Sweden. Revised Selected Papers.* 2014 154, URL: https://doi.org/10.1007/978-3-319-17966-7%5C_21 (cit. on p. 85).

[111] I. Zaikin and A. Tuzovsky, "Owl2vcs: Tools for Distributed Ontology Development",
Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED) co-located with 10th Extended Semantic Web Conference (ESWC), Montpellier, France. 2013 (cit. on p. 86).

[112] J. Russo, E. Johnson and D. Stephens, *The validity of verbal protocols*,
Memory & Cognition **17** (1989) 759 (cit. on p. 89).

[113] C. Gutiérrez, C. A. Hurtado, A. O. Mendelzon and J. Pérez,
Foundations of Semantic Web databases,
Journal of Computer and System Sciences **77** (2011) 520,
URL: https://doi.org/10.1016/j.jcss.2010.04.009 (cit. on p. 98).

[114] N. Mihindukulasooriya, M. Poveda-Villalón, R. García-Castro and A. Gómez-Pérez,
"Collaborative Ontology Evolution and Data Quality - An Empirical Analysis",
OWL: - Experiences and Directions - Reasoner Evaluation - 13th International Workshop, (OWLED), Bologna, Italy, Revised Selected Papers, 2016 95,
URL: https://doi.org/10.1007/978-3-319-54627-8_8 (cit. on p. 109).

[115] N. F. Noy and M. C. A. Klein, *Ontology Evolution: Not the Same as Schema Evolution*,
Journal Knowledge and Information Systems **6** (2004) 428 (cit. on p. 109).

[116] P. Plessers and O. D. Troyer, "Ontology Change Detection Using a Version Log", *4th International Semantic Web Conference, (ISWC)*, 2005 578, URL: https://doi.org/10.1007/11574620_42 (cit. on p. 109).

[117] M. Mehrotra, "Ontology analysis for the Semantic Web", *Ontologies and the Semantic Web: AAAI Workshop ,Technical Report WS-02-11*, AAAI Press, 2002 (cit. on p. 109).

[118] R. M. Karp, "Reducibility Among Combinatorial Problems", *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, 2010 219, URL: https://doi.org/10.1007/978-3-540-68279-0_8 (cit. on p. 113).

[119] R. Drath and A. Horch, *Industrie 4.0: Hit or Hype?[Industry Forum]*, Industrial Electronics Magazine, IEEE **8** (2014) 56, ISSN: 1932-4529 (cit. on p. 126).

[120] E. A. Lee, "Cyber Physical Systems: Design Challenges", *11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, (ISORC), Orlando, Florida, USA*, 2008 363, URL: https://doi.org/10.1109/ISORC.2008.25 (cit. on p. 126).

[121] M. Mikusz, *Towards an Understanding of Cyber-physical Systems as Industrial Software-Product-Service Systems*, Procedia CIRP **16** (2014) 385 (cit. on p. 126).

[122] E. Tovar and F. Vasques, *Real-time fieldbus communications using Profibus networks*, IEEE Transactions Industrial Electronics **46** (1999) 1241, URL: https://doi.org/10.1109/41.808018 (cit. on p. 127).

[123] W. Mahnke, S.-H. Leitner and M. Damm, *OPC unified architecture*, Springer Science & Business Media, 2009, ISBN: 978-3-540-68898-3 (cit. on pp. 127, 129).

[124] P. Adolphs, H. Bedenbender, D. Dirzus, M. Ehlich, U. Epple, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, B. Kärcher, H. Koziolek, R. Pichler, S. Pollmeier, F. Schewe, A. Walter, B. Waser and M. Wollschlaeger, *Reference Architecture Model Industrie 4.0 (RAMI4.0)*, Status Report, ZVEI and VDI, 2015 (cit. on pp. 127, 128, 130, 145, 156).

[125] P. Adolphs, S. Auer, H. Bedenbender, M. Billmann, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, M. Jochem, M. Kiele, G. Koschnick, H. Koziolek, L. Linke, R. Pichler, F. Schewe, K. Schneider and B. Waser, *Structure of the Administration Shell. Continuation of the Development of the Reference Model for the Industrie 4.0 Component*, Status Report, ZVEI and VDI, 2016, URL: http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html (cit. on p. 127).

[126] K. Främling, M. Harrison, J. Brusey and J. Petrow, *Requirements on unique identifiers for managing product lifecycle information: comparison of alternative approaches*, International Journal of Computer Integrated Manufacturing **20** (2007) 715, URL: https://doi.org/10.1080/09511920701567770 (cit. on p. 129).

[127] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer and P. Doody, *Internet of things strategic research roadmap*, Internet of Things-Global Technological and Societal Trends **1** (2011) 9 (cit. on p. 129).

[128] M. Abu-Elkheir, M. Hayajneh and N. A. Ali,
Data Management for the Internet of Things: Design Primitives and Solution,
Sensors **13** (2013) 15582, URL: https://doi.org/10.3390/s131115582 (cit. on p. 129).

[129] R. Drath, *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*, Springer-Verlag, 2010 (cit. on p. 129).

[130] IEC, *IEC 61918:2013 Industrial communication networks - Installation of communication networks in industrial premises*, 2013,
URL: https://webstore.iec.ch/publication/6100 (cit. on p. 129).

[131] U. Enste and W. Mahnke, *OPC Unified Architecture.*,
Automatisierungstechnik **59** (2011) 397 (cit. on p. 129).

[132] E. Abele, A. Wörn, J. Fleischer, J. Wieser, P. Martin and R. Klöpper,
Mechanical module interfaces for reconfigurable machine tools,
Production Engineering **1** (2007) 421,
URL: https://doi.org/10.1007/s11740-007-0057-1 (cit. on p. 129).

[133] M. Hillier, *The role of cultural context in multilingual website usability*,
Electronic Commerce Research and Applications **2** (2003) 2,
URL: https://doi.org/10.1016/S1567-4223(03)00005-X (cit. on p. 129).

[134] M. Thoma, T. Braun, C. Magerkurth and A. Antonescu,
"Managing things and services with semantics: A survey",
IEEE Network Operations and Management Symposium, (NOMS), Krakow, Poland,
2014 1, URL: https://doi.org/10.1109/NOMS.2014.6838366 (cit. on p. 129).

[135] W. Wahlster, "Semantic Technologies for Mass Customization",
Towards the Internet of Services: The THESEUS Research Program, 2014 3,
URL: https://doi.org/10.1007/978-3-319-06755-1_1 (cit. on p. 129).

[136] A. Langegger, W. Wöß and M. Blöchl,
"A Semantic Web Middleware for Virtual Data Integration on the Web",
5th European Semantic Web Conference, (ESWC), Tenerife, Spain, Proceedings, 2008
493, URL: https://doi.org/10.1007/978-3-540-68234-9%5C_37 (cit. on p. 130).

[137] C. Bizer, C. Becker, P. N. Mendes, R. Isele, A. Matteini and A. Schultz,
"LDIF–A Framework for Large-Scale Linked Data Integration", *21st International World Wide Web Conference, (WWW), Developers Track, Lyon, France*, 2012 (cit. on p. 130).

[138] M. Graube, J. Pfeffer, J. Ziegler and L. Urbas,
"Linked Data as Integrating Technology for Industrial Data", *The 14th International Conference on Network-Based Information Systems, (NBiS), Tirana, Albania*, 2011 162,
URL: https://doi.org/10.1109/NBiS.2011.33 (cit. on p. 130).

[139] D. Bandyopadhyay and J. Sen,
Internet of Things: Applications and Challenges in Technology and Standardization,
Wireless Personal Communications **58** (2011) 49,
URL: https://doi.org/10.1007/s11277-011-0288-5 (cit. on p. 130).

[140] C. Bizer, T. Heath and T. Berners-Lee, *Linked Data - The Story So Far*,
International Journal on Semantic Web and Information Systems **5** (2009) 1,
URL: http://dx.doi.org/10.4018/jswis.2009081901 (cit. on p. 130).

[141] B. Quilitz and U. Leser, "Querying Distributed RDF Data Sources with SPARQL", *5th European Semantic Web Conference, (ESWC), Research and Applications, Tenerife, Spain, Proceedings*, 2008 524 (cit. on p. 130).

[142] J. Haupert, M. Seißler, B. Kiesel, B. Schennerlein, S. Horn, D. Schreiber and R. Barthel, *Object Memory Modeling*, Incubator Group Report, World Wide Web Consortium (W3C), 2011, URL: https://www.w3.org/2005/Incubator/omm/XGR-omm-20111026/ (cit. on p. 131).

[143] T. Aruväli, W. Maass and T. Otto, *Digital Object Memory Based Monitoring Solutions in Manufacturing Processes*, 24th DAAAM International Symposium on Intelligent Manufacturing and Automation **69** (2014) 449 (cit. on p. 131).

[144] H. Rijgersberg, M. van Assem and J. L. Top, *Ontology of units of measure and related concepts*, Semantic Web **4** (2013) 3, URL: https://doi.org/10.3233/SW-2012-0069 (cit. on p. 132).

[145] eCl@ss, *Standardized Material and Service Classification*, 2016, URL: http://www.eclass.eu/ (visited on 26/01/2016) (cit. on p. 132).

[146] S. Weyer, M. Schmitt, M. Ohmer and D. Gorecky, *Towards Industry 4.0-Standardization as the crucial challenge for highly modular, multi-vendor production systems*, 15th IFAC Symposium on Information Control Problems in Manufacturing **48** (2015) 579 (cit. on p. 136).

[147] V. Vyatkin, *Software Engineering in Industrial Automation: State-of-the-Art Review*, IEEE Transactions Industrial Informatics **9** (2013) 1234, URL: https://doi.org/10.1109/TII.2013.2258165 (cit. on p. 136).

[148] C. Pang and V. Vyatkin, "IEC 61499 function block implementation of Intelligent Mechatronic Component", *8th IEEE International Conference on Industrial Informatics*, IEEE, 2010 1124 (cit. on pp. 136, 141, 142).

[149] Y. Lu, K. Morris and S. Frechette, *Current standards landscape for smart manufacturing systems*, National Institute of Standards and Technology, (NISTIR) **8107** (2016) (cit. on pp. 137, 138).

[150] M. of Industry, I. technology of China (MIIT) and S. A. of China (SAC), *National Intelligent Manufacturing Standard System Construction Guidelines*, tech. rep., Standardization Administration of China, (SAC), 2015 (cit. on p. 137).

[151] Q. Li, H. Jiang, Q. Tang, Y. Chen, J. Li and J. Zhou, "Smart Manufacturing Standardization: Reference Model and Standards Framework", *OTM Workshops - Confederated International Workshops: EI2N, FBM, ICSP, Meta4eS, and OTMA, Rhodes, Greece, Revised Selected Papers*, 2016 16 (cit. on p. 137).

[152] P. Adolphs, S. Auer, M. Billmann, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, M. Jochem, M. Kiele, G. Koschnick, H. Koziolek, L. Linke, R. Pichler, F. Schewe, K. Schneider and B. Waser, *Structure of the Administration Shell*, Status Report, ZVEI and VDI, 2016 (cit. on p. 137).

[153] R. Drath, A. Lüder, J. Peschke and L. Hundt,
"AutomationML - the glue for seamless automation engineering",
Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, (ETFA), Hamburg, Germany, 2008 616 (cit. on p. 137).

[154] M. Schleipen, M. Damm, R. Henßen, A. Lüder, N. Schmidt, O. Sauer and S. Hoppe,
"OPC UA and AutomationML - collaboration partners for one common goal: Industry 4.0", *3rd AutomationML User Conference, Blumberg*, 2014 (cit. on p. 138).

[155] R. Henßen and M. Schleipen, *Interoperability between OPC UA and AutomationML*,
Procedia CIRP **25** (2014) 297, 8th International Conference on Digital Enterprise Technology (DET) - 2014 Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution, ISSN: 2212-8271 (cit. on p. 138).

[156] F. Himmler, "Function Based Engineering with AutomationML - Towards better standardization and seamless process integration in plant engineering",
Smart Enterprise Engineering: 12. Internationale Tagung Wirtschaftsinformatik, (WI), Osnabrück, Germany, 2015 16 (cit. on p. 138).

[157] S. Kozar and P. Kadera, "Integration of IEC 61499 with OPC UA",
21st IEEE International Conference on Emerging Technologies and Factory Automation, (ETFA), Berlin, Germany, 2016 1,
URL: https://doi.org/10.1109/ETFA.2016.7733538 (cit. on p. 138).

[158] S. Cavalieri and A. Regalbuto, *Integration of IEC 61850 SCL and OPC UA to improve interoperability in Smart Grid environment*,
Computer Standards & Interfaces **47** (2016) 77,
URL: https://doi.org/10.1016/j.csi.2015.10.005 (cit. on p. 138).

[159] S. Lohmann, P. Díaz and I. Aedo, "MUTO: the modular unified tagging ontology",
Proceedings the 7th International Conference on Semantic Systems, (I-SEMANTICS), Graz, Austria, 2011 95, URL: http://doi.acm.org/10.1145/2063518.2063531
(cit. on p. 138).

[160] M. Brettel, N. Friederichsen, M. Keller and M. Rosenberg,
How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 perspective, International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering **8** (2014) 37 (cit. on p. 145).

[161] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin and I. Horrocks,
"Capturing Industrial Information Models with Ontologies and Constraints",
15th International Semantic Web Conference, (ISWC), Kobe, Japan, Proceedings, Part II, 2016 325 (cit. on p. 145).

[162] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios",
49th Hawaii International Conference on System Sciences, (HICSS), Koloa, HI, USA, 2016 3928, URL: https://doi.org/10.1109/HICSS.2016.488 (cit. on p. 145).

[163] *IEC 62264-1: Enterprise-Control System Integration Part 1: Models and Terminology*,
Standard, International Electrotechnical Commission, 2013 (cit. on pp. 145, 149, 156).

[164] N. Petersen, M. Galkin, C. Lange, S. Lohmann and S. Auer,
"Monitoring and Automating Factories Using Semantic Models",
Semantic Technology - 6th Joint International Conference, (JIST), Singapore, Singapore, Revised Selected Papers, 2016 315,
URL: https://doi.org/10.1007/978-3-319-50112-3%5C_24 (cit. on p. 147).

[165] M. Uschold and M. Gruninger, *Ontologies: principles, methods and applications*,
Knowledge Engineering Review **11** (1996) 93,
URL: http://dx.doi.org/10.1017/S0269888900007797 (cit. on p. 147).

[166] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann and S. Hübner,
"Ontology-Based Integration of Information - A Survey of Existing Approaches",
Proceedings of the IJCAI Workshop on Ontologies and Information Sharing Seattle, USA, 2001 (cit. on p. 149).

[167] L. Bellatreche and G. Pierra, "OntoAPI: An Ontology-based Data Integration Approach by an a Priori Articulation of Ontologies", *18th International Workshop on Database and Expert Systems Applications (DEXA), Regensburg, Germany*, 2007 799,
URL: https://doi.org/10.1109/DEXA.2007.152 (cit. on p. 150).

[168] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini and R. Rosati,
Linking Data to Ontologies, Journal on Data Semantics **10** (2008) 133,
URL: https://doi.org/10.1007/978-3-540-77688-8%5C_5 (cit. on p. 150).

[169] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao,
Ontop: Answering SPARQL queries over relational databases,
Semantic Web **8** (2017) 471 (cit. on p. 150).

[170] P. Vandenbussche, G. Atemezing, M. Poveda-Villalón and B. Vatant, *Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web*,
Semantic Web **8** (2017) 437, URL: https://doi.org/10.3233/SW-160213
(cit. on p. 157).

[171] P. L. Whetzel, N. F. Noy, N. H. Shah, P. R. Alexander, C. Nyulas, T. Tudorache and M. A. Musen, *BioPortal: enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications*,
Nucleic Acids Research **39** (2011) 541 (cit. on p. 157).

[172] F. Baader, "Description Logics", *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School, Brixen-Bressanone, Italy, Tutorial Lectures*, 2009 1, URL: https://doi.org/10.1007/978-3-642-03754-2_1 (cit. on p. 163).

[173] D. Lukovnikov, A. Fischer, J. Lehmann and S. Auer, "Neural Network-based Question Answering over Knowledge Graphs on Word and Character Level", *Proceedings of the 26th International Conference on World Wide Web, (WWW), Perth, Australia*, 2017 1211, URL: http://doi.acm.org/10.1145/3038912.3052675 (cit. on p. 168).

APPENDIX A

List of Publications

- *Journal Articles and Book Chapters*:
  1. **Lavdim Halilaj**, Steffen Lohmann, Maria-Esther Vidal, Sören Auer. *EcoVoc: An Ecosystem for Distributed and Version-Controlled Ontology Development.* To be submitted in a relevant journal;
  2. **Lavdim Halilaj**, Irlan Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *SerVCS: Serialization Agnostic Ontology Development in Distributed Settings.* In Communications in Computer and Information Science (CCIS) 914 - Revised Selected Papers from 8th International Joint Conference, IC3K 2016, Porto, Portugal, 213-232, Springer;
  3. **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Steffen Lohmann, Sören Auer. *Git4Voc: Collaborative Vocabulary Development Based on Git.* In International Journal of Semantic Computing (IJSC), 1-24, World Scientific.
- *Conference Papers*:
  4. Irlán Grangel-González, **Lavdim Halilaj**, Maria-Esther Vidal, Omar Rana, Steffen Lohmann, Sören Auer, Andreas W. Müller. *Knowledge Graphs for Semantically Integrating Cyber-Physical Systems.* In 9th International Conference on Database and Expert Systems Applications (DEXA) 2018 Proceedings;
  5. Fathoni A Musyaffa, **Lavdim Halilaj**, Yakun Li, Fabrizio Orlandi, Hajira Jabeen, Sören Auer. *OpenBudgets.eu: A Platform for Semantically Representing and Analyzing Open Fiscal Data.* In 18th International Conference on Web Engineering (ICWE) 2018 Proceedings, Springer;
  6. **Lavdim Halilaj**, Irlán Grangel-González, Steffen Lohmann, Maria-Esther Vidal, Sören Auer. *EffTE: A Dependency-aware Approach for Test-Driven Ontology Development.* In 33rd ACM/SIGAPP Symposium On Applied Computing (ACM SAC) 2018 Proceedings, ACM;
  7. Niklas Petersen, **Lavdim Halilaj**, Irlán Grangel-González, Steffen Lohmann, Christoph Lange, Sören Auer. *Realizing an RDF-based Information Model for a Manufacturing Company – A Case Study.* (One of the two nominees for the Best In-Use Paper Award) In 16th International Semantic Web Conference (ISWC) 2017 Proceedings, 350-366, Springer;

8. Irlán Grangel-González, Paul Baptista, **Lavdim Halilaj**, Steffen Lohmann, Maria-Esther Vidal, Christian Mader, Sören Auer. *The Industry 4.0 Standards Landscape from a Semantic Integration Perspective.* In 22nd IEEE International Conference on Emerging Technologies And Factory Automation (ETFA) 2017 Proceedings;
9. **Lavdim Halilaj**, Irlan Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *Proactive Prevention of False-Positive Conflicts in Distributed Ontology Development.* (Best Paper Award) In 8th International Conference on Knowledge Engineering and Ontology Development Proceedings (KEOD), 43-51, SciTePress;
10. **Lavdim Halilaj**, Niklas Petersen, Irlán Grangel-González, Christoph Lange, Sören Auer, Gökhan Coskun, Steffen Lohmann. *VoCol: An Integrated Environment to Support Version-Controlled Vocabulary Development.* (Paper of the Month - Fraunhofer IAIS) In 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW) 2016 Proceedings, 303-319, Springer;
11. Irlán Grangel-González, Diego Collarana, **Lavdim Halilaj**, Steffen Lohmann, Christoph Lange, María-Esther Vidal, Sören Auer. *Alligator: A Deductive Approach for the Integration of Industry 4.0 Standards.* In 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW) 2016 Proceedings, 272-287, Springer;
12. Irlán Grangel-González, **Lavdim Halilaj**, Sören Auer, Steffen Lohmann, Christoph Lange, Diego Collarana. *An RDF-based Approach for Implementing Industry 4.0 Components with Administration Shells.* In IEEE Emerging Technologies and Factory Automation (ETFA) 2016 Proceedings, 1-8, IEEE;
13. Fathoni A Musyaffa, **Lavdim Halilaj**, Ronald Siebes, Fabrizio Orlandi, Sören Auer. *Minimally Invasive Semantification of Lightweight Service Descriptions.* In IEEE International Conference on Web Services (ICWS) 2016 Proceedings, 672–677, IEEE;
14. Irlán Grangel-González, **Lavdim Halilaj**, Gökhan Coskun, Sören Auer, Diego Collarana, Michael Hofmeister. *Towards a Semantic Administrative Shell for Industry 4.0 Components.* (Paper of the Month - Fraunhofer IAIS) In IEEE Tenth International Conference on Semantic Computing (ICSC) 2016 Proceedings, 230 - 237, IEEE;
15. **Lavdim Halilaj**, Irlán Grangel-González, Gökhan Coskun, Sören Auer. *Git4Voc: Git-based Versioning for Collaborative Vocabulary Development.* In IEEE Tenth International Conference on Semantic Computing (ICSC) 2016 Proceedings, 285 - 292, IEEE;
16. Andreas Poxrucker, Christian Mader, Peter Hevesi, **Lavdim Halilaj**, Steffen Lohmann, Paul Lukowicz, Sören Auer. *Towards a Smart Data Repository for the SDIL.* In 1st Smart Data Innovation Conference (SDIC) 2016 Proceedings, 23-29;
17. Irlan Grángel-González, **Lavdim Halilaj**, Gökhan Coskun, Sören Auer. *Towards Vocabulary Development by Convention.* In 7th Knowledge Engineering and Ontology Development (KEOD) 2015 Proceedings, 334-343, SciTePress.

- *Workshops and Demos*:

18. Irlán Grangel-González, **Lavdim Halilaj**, Maria-Esther Vidal, Steffen Lohmann, Sören Auer, Andreas W. Müller. *Seamless Integration of Cyber-Physical Systems*

in Knowledge Graphs. In 33rd ACM/SIGAPP Symposium On Applied Computing (ACM SAC) 2018 Proceedings, ACM;

19. **Lavdim Halilaj**, Irlán Grangel-González, Maria-Esther Vidal, Steffen Lohmann, Sören Auer. *DemoEffTE: A Demonstrator of Dependency-aware Evaluation of Test Cases over Ontology.* In 13th International Conference on Semantic Systems (Semantics) - Posters and Demo Track, 2017;
20. Fathoni A. Musyaffa, Fabrizio Orlandi, Tiansi Dong, **Lavdim Halilaj**. *OpenBudgets.eu: A Distributed Open-Platform for Managing Heterogeneous Budget Data.* In 13th International Conference on Semantic Systems (Semantics) - Posters and Demo Track, 2017;
21. **Lavdim Halilaj**, Steffen Lohmann, Christian Mader, Sören Auer. *Distributed Vocabulary Development with Version Control Systems.* In W3C Smart Descriptions and Smarter Vocabularies (SDSVoc), 2016;
22. **Lavdim Halilaj**, Niklas Petersen, Irlán Grangel-González, Christoph Lange, Steffen Lohmann, Christian Mader, Sören Auer. *Industrial Data Space: Semantic integration of Enterprise Data with VoCol.* In 12th International Conference on Semantic Systems (Semantics) - European Linked Data Contest, 2016.

List of Figures

List of Tables