Supporting Domain Modeling with Automated Knowledge Acquisition and Modeling Recommendations

vorgelegt von
Diplom-Informatiker (FH)
Henning Agt-Rickauer, geb. Agt

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Odej Kao
Gutachter:	Prof. Dr. Volker Markl
Gutachter:	Prof. Dr. Harald Sack
Gutachter:	Prof. Dr. Kurt Sandkuhl

Tag der wissenschaftlichen Aussprache: 4. November 2019

Berlin 2020

Abstract

Domain modeling is an important model-driven engineering activity, which is typically used in the early stages of software projects. Domain models capture concepts and relationships of respective application fields using a modeling language and domain-specific terms. They are a key factor in achieving shared understanding of the problem area among stakeholders, improving communication in software development, and generating code and software. Domain models are a prerequisite for domain-specific language development and are often implemented in software and data integration and software modernization projects, which constitute the larger part of industrial IT investment. Several studies from recent years have shown that model-driven methods are much more widespread in the industry than previously thought, yet their application is challenging.

Creating domain models requires that software engineers have both experience in model-driven engineering and detailed domain knowledge. While the former is one of the general modeling skills, the required domain knowledge varies from project to project. Domain knowledge acquisition is a time-consuming manual process because it requires multidisciplinary collaboration and gathering of information from different groups of people, documents, and other sources of knowledge, and is rarely supported in current modeling environments. Consistent access to large amounts of structured domain knowledge is not possible due to the heterogeneity of formats, access methods, schemas, and semantic representations. Besides, existing suitable knowledge bases were mostly manually created and are therefore not extensive enough. The automated construction of knowledge resources utilizes mainly information extraction approaches that focus on factual knowledge at the instance level and therefore cannot be used for conceptual-level domain modeling.

This thesis develops novel methods and tools that provide domain information directly during modeling to reduce the initial effort of using domain modeling and to help software developers create domain models. It works on the connection of the areas of software modeling, knowledge bases, information extraction and recommender systems to automatically acquire conceptual knowledge from structured knowledge sources and unstructured natural language datasets, to transform the aggregated knowledge into appropriate recommendations, and to develop suitable assistance services for modeling environments.

With this thesis, the paradigm of *Semantic Modeling Support* is proposed, the methodological foundation for providing automated modeling assistance. It includes an iterative procedure of model refinement, knowledge acquisition, and element recommendation that allows to query and provide the necessary domain knowledge for a range of support scenarios at each stage of domain model development, keeping the human in the loop. To address the lack of conceptual knowledge resources, new methods are developed to extract conceptual terms and relationships directly from large N-gram text data using syntactic patterns, co-occurrences, and statistical features of text corpora. A large *Semantic Network of Related Terms* is automatically constructed with nearly 6 million unique one-word terms and multi-word expressions connected with over 355 million weighted binary and ternary relationships. It allows to directly answer top-N queries. This thesis introduces an extensible query component with a set of fully connected knowledge bases to uniformly access structured knowledge with well-defined relationships. The developed *Mediator-Based Querying Architecture with Generic Templates* is responsible for retrieving lexical information from heterogeneous knowledge bases and mapping to modeling language-specific concepts. Furthermore, it is demonstrated how to implement the semantic modeling support strategies by extending the widely used Eclipse Modeling Project. The *Domain Modeling Recommender System* generates context-sensitive modeling suggestions based on the connected knowledge bases, the semantic network of terms, and an integrated ranking strategy. Finally, this thesis reports on practical experience with the application of the developed methods and tools in three research projects.

Zusammenfassung

Domänenmodellierung ist eine wichtige Methode der modellgetriebenen Softwareentwicklung, die oftmals in den frühen Phasen von Softwareprojekten verwendet wird. Domänenmodelle erfassen Konzepte und Beziehungen des jeweiligen Anwendungsgebietes unter Verwendung einer Modellierungssprache und domänenspezifischer Begriffe. Sie sind ein Schlüsselfaktor für ein gemeinsames Verständnis des Problembereichs unter den Projektbeteiligten, für die Verbesserung der Kommunikation während der Softwareentwicklung und für die Generierung von Code und Software. Domänenmodelle sind eine Voraussetzung für die Entwicklung von domänenspezifischen Sprachen und werden häufig bei Softwaremodernisierung sowie bei Software- und Datenintegrationsprojekten eingesetzt, die den größten Teil der industriellen IT-Investitionen ausmachen. Mehrere Studien aus den letzten Jahren haben gezeigt, dass modellgetriebene Methoden in der Branche verbreiteter sind, als bisher angenommen, ihre Anwendung jedoch eine Herausforderung darstellt.

Das Erstellen von Domänenmodellen setzt voraus, dass Softwareentwickler sowohl über Erfahrung in der modellgetriebenen Softwareentwicklung als auch über detaillierte Domänenkenntnisse verfügen. Ersteres gehört zu allgemeinen Modellierungskompetenzen, jedoch variiert das erforderliche Domänenwissen von Projekt zu Projekt. Der Erwerb von Domänenwissen ist ein zeitaufwendiger manueller Prozess, da er interdisziplinäre Zusammenarbeit und Informationsbeschaffung von verschiedenen Personengruppen, Dokumenten und anderen Wissensquellen erfordert und in aktuellen Modellierungsumgebungen kaum unterstützt wird. Ein einheitlicher Zugriff auf große Mengen von strukturiertem Domänenwissen ist aufgrund der Heterogenität von Formaten, Zugriffsmethoden, Schemata und semantischen Repräsentationen nicht möglich. Außerdem wurden vorhandene geeignete Wissensbasen meist manuell erstellt und sind daher nicht umfangreich genug. Bei der automatisierten Erstellung von Wissensressourcen werden hauptsächlich Ansätze der Informationsextraktion verwendet, die Faktenwissen auf Instanzebene erschließen und daher nicht für Domänenmodellierung auf Konzeptebene geeignet sind.

In dieser Arbeit werden neuartige Methoden und Tools entwickelt, die Domäneninformationen direkt während der Modellierung bereitstellen, um den anfänglichen Aufwand der Domänenmodellierung zu verringern und Softwareentwicklern bei der Erstellung von Domänenmodellen zu helfen. Sie befasst sich mit der Verknüpfung der Bereiche Softwaremodellierung, Wissensdatenbanken, Informationsextraktion und Empfehlungssysteme, um automatisch konzeptionelles Wissen aus strukturierten Datenquellen und unstrukturierten natürlichsprachlichen Texten zu erschließen, um das aggregierte Wissen in geeignete Empfehlungen zu transformieren, und um geignete Unterstützungsdienste für Modellierungswerkzeuge zu entwickeln.

Diese Dissertation führt die *Semantische Modellierungsunterstützung*, die methodische Grundlage für eine automatisierte Modellierungshilfe, ein. Sie umfasst ein iteratives Verfahren der Modellverfeinerung, Wissensbeschaffung und Elementempfehlung, welches das erforderliche Domänenwissen für eine Reihe von Unterstützungsszenarien in jeder Phase der Domänenmodellentwicklung abfragt und bereitstellt. Das Fehlen von konzeptionellen Wissensressourcen wird durch die Entwicklung neuer Methoden adressiert, die konzeptionelle Begriffe und Beziehungen direkt aus großen N-Gram Textdaten mit Hilfe von syntaktischen Mustern, Kookkurrenzen und statistischen Merkmalen großer Textkorpora extrahieren. Es wird ein großes *Semantisches Netz verwandter Begriffe* mit fast 6 Millionen Ein- und Mehrwortbegriffen automatisch konstruiert, die mit über 355 Millionen gewichteten binären und ternären Beziehungen verbunden sind, wodurch Top-N-Anfragen direkt beantwortet werden können. Die Dissertation führt eine erweiterbare Abfragekomponente mit einer Reihe fertig verbundener Wissensbasen ein, um einheitlich auf strukturiertes Wissen mit spezifischen Relationen zuzugreifen. Die entwickelte *Mediator-basierte Abfragearchitektur mit generischen Templates* fragt automatisch lexikalische Informationen aus heterogenen Wissensbasen ab und ordnet sie Modellierungssprachkonzepten zu. Außerdem wird gezeigt, wie die Strategien der semantischen Modellierungsunterstützung implementiert werden, indem das weit verbreitete Eclipse Modeling Project erweitert wird. Das *Domänenmodellierungsempfehlungssystem* generiert kontextsensitive Modellierungsvorschläge, basierend auf den verbundenen Wissensbasen, dem semantischen Begriffsnetz und einer integrierten Ranking-Strategie. Schließlich berichtet diese Arbeit über praktische Erfahrungen bei der Anwendung der entwickelten Methoden und Werkzeuge in drei Forschungsprojekten.

Declaration of Authorship

I, Henning Agt-Rickauer, declare that this dissertation titled "Supporting Domain Modeling with Automated Knowledge Acquisition and Modeling Recommendations" and the work presented in it are my own. I confirm that all sources and tools used are listed and duly cited.

Berlin, February 04, 2020 . . . . . .

Acknowledgements

I would like to express my gratitude to the people who made this dissertation possible.

First and foremost, I am particularly grateful to Dr. Ralf-Detlef Kutsche for bringing me to the TU Berlin, giving me the opportunity to carry out my research within two exciting research projects, and encouraging me to conduct this dissertation. This work would not have been possible without his persistent support and fruitful discussions during my long journey. I would like to thank Prof. Volker Markl for the great research environment he created at the DIMA group. His dedication to scalable data analytics has inspired me to focus on linking modeling and information extraction.

I would like to express my deep gratitude to Prof. Harald Sack for providing me with the opportunity to continue my work at the Hasso-Plattner-Institute. His valuable advice greatly improved the quality of my research, and the project in, which I participated, brought fresh ideas to my work. I have been fortunate to work with my colleagues Jörg Waitelonis, Tabea Tietz, Christian Hentschel and Magnus Knuth, who have so warmly integrated me into the team at HPI. Especially, I thank Jörg for his valuable comments during the last phase of my thesis.

I would like to thank Prof. Kurt Sandkuhl for acting as third reviewer for this dissertation.

Last but not least, I thank my beloved wife Kathy from the bottom of my heart for her patience and encouragement. She experienced the ups and downs of my work directly and was always there for me and gave me the freedom and time required to conduct my work. It was also a challenge for her because she often took the load off from me when she was taking care of our three children, who were born during the course of my dissertation.

To Kathy, Maila, Lenja, and Janik

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Automation is the technique of running a procedure with minimal human intervention. Usually, computers or similar technical devices replace human labor. Computers must be programmed by humans to perform the required tasks. Very early, computers were instructed with machine code that was entered directly through switches or read from punched cards. Later, low-level assembler programming languages were developed that provided commands and functions closely coupled with the computer's instruction set architecture. They were more readable to humans, but offered little abstraction, since one line code more or less corresponded to one machine instruction.

This changed in the late 1950s with the development of high-level programming languages, which are more machine-independent, easier to write and maintain, as they provide functions for common tasks, use natural language elements, and program code is translated into machine code using compilers or interpreters. The higher degree of abstraction makes programs more understandable and allows developers to focus on functionality. However, the discrepancy between the development of programming languages, computer hardware, operating systems and the resulting complexity in the 1960s led to the so-called "software crisis" [1], in which a large part of software projects ran over-budget and over-time, did not meet the requirements and were of low quality. At that time, the term *software engineering* was coined [2], which resulted mainly from a NATO conference initiative to develop new software methods and tools that were to handle the complexity.

Since then, software development has evolved into a mature engineering discipline, in which many useful principles, methods, languages, and tools (e.g., separation of concerns, V-Model, object-oriented programming languages, integrated development environments – just to name a few important ones) were created. Nevertheless, the capabilities of computers and user requirements have grown steadily, leaving the complexity inherent. At the level of programming languages, only a certain degree of abstraction can be achieved [3]. This has led to the use of models in software development that have always been a means of abstraction. Respective modeling methods emerged in the late 1990s and early 2000s mainly from Computer Aided Software Engineering (CASE) and Object-Oriented Analysis and Design (OOAD). The main advantage of modeling is that the user of the software, i.e. the domain expert, can be better integrated into the development of the software. This reduces the risk that requirements will not be properly implemented because, on the one hand the user cannot express them accurately, or they may be misinterpreted by the developer. In addition, these methods include the (semi-)automatic translation of models into program code, whereby a higher level of automation in software engineering is achieved.

As this dissertation was conducted within the context of research projects carried out in close collaboration with industry partners, this chapter first elaborates how important modeling in industrial software development has become, and how important domain modeling is: a method to capture concepts and relationships of a particular application field in domain models. Afterwards, the challenges of applying domain modeling will be discussed, especially with regard to the required knowledge acquisition. Finally, the contributions of this dissertation are presented: novel methods and tools to support domain modeling by developing, combining and improving methods of software modeling, knowledge bases, information extraction and recommendation systems.

1.1 Motivation

1.1.1 Industrial Relevance of Model-Driven Engineering

One of the approaches to handle complexity of software and software development processes is the use of model-driven engineering (MDE). The main goal of MDE is to raise the level abstraction by using models as primary development artifacts and to increase the degree of automation by reducing the amount of recurring development tasks [4, 5]. MDE is a solution to bridge the *"gap between the high-level concepts used by domain experts to express their specific needs and the low-level abstractions provided by general-purpose programming languages"* [6].

Model-driven solutions and adoptions have been developed by research and industry cooperation for almost 20 years. The most prominent example is the Unified Modeling Language (UML) [7], which was accepted as standard in 1997 by the Object Management Group (OMG) and is still being developed further. UML is considered the de facto **standard language** for software specification and design [8]. While UML is designed as a general purpose modeling language, the development of domain-specific solutions has a similar history [9]. Nowadays, domain-specific language (DSL) development has reached a maturity level comparable to general purpose solutions and is often used complementarily [10].

During the last two decades, model-driven engineering has not reached that much success compared to the adoption of object-oriented programming languages. The main reasons for the insufficient acceptance of MDE are tool usability and adoption, inconsistencies between software artifacts and models, as well as missing synchronization, and the difficulty of teaching real world design principles in education [11, 12]. Nevertheless, over the years, a number of reports have been published on the successful adoption of model-driven engineering in industry [13, 14, 15, 16, 17]. Apart from these success stories, very few studies were conducted in the past on how MDE is widely used in industry [18].

Missing empirical evidence of MDE in practice has been approached by several larger and long-term studies in the last five years [8, 19, 20, 21, 22, 23, 24, 10, 25, 26, 6, 27, 28, 29]. These studies revealed a **wider dissemination** of model-driven practice than it was argued in the past (cf., a systematic literature review of the years 2004-2008 by Budgen et al. [30]). The main outcomes of the investigations are: A large number of companies use modeling for software projects on a regular basis. Two studies report that 40%-55% of the developers and companies often use modeling during software projects, and 11%-13% of the developers and companies use modeling most of the time [31, 24]. Nearly all of the studies confirm that UML is the most widely used modeling language in industry (e.g., [22, 8]). The most frequently used diagram kind is the UML class diagram, and still the majority of developers use modeling techniques for problem understanding, documentation and communication [26, 8]. In addition, **domain-specific languages** *"have achieved a significant degree of penetration"* [22] (e.g., 85% of the study's respondents used UML, and 60% used DSLs).

1.1.2 Importance of Domain Modeling in Software Projects

Typically, early phases of software projects start with domain analysis and requirements engineering [32, 33]. Domain analysis is a *"process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable when creating new systems."* [33]. Domain analysis includes the creation of domain models that capture knowledge of an application domain. These models can be used for a variety of tasks: Requirements elicitation, specification, code generation, reverse engineering, explanation, documentation, design decisions, and training [34].

Domain models are a key factor in **improving communication** and understanding in software development [35, 36, 37, 38]. In a process of domain modeling different stakeholders agree on a common set of terms, relationships and descriptions. The resulting domain models enable a ubiquitous language [39] among groups of developers and domain experts and help to reduce ambiguities and misunderstandings [40]. Domain modeling is often used synonymously for conceptual modeling [41]. *"Conceptual modeling conventionally results in specifications that capture*

relevant knowledge about the application domain. These specifications then guide development by supporting communication between developers and users, promoting domain understanding and guiding the design process" [42].

Domain modeling is not exclusively dedicated to systematically collecting information about the respective domain and domain expert knowledge [43]. It is also about using the models in most software development phases. **Domain-driven design** [39] is a well-known approach that consequently employs domain models in the first place and connects them to implementation models very early. This development framework has proven successful in several industry projects [44, 45, 46, 47] and has been integrated into well-known software design patterns [48].

As described in the previous section, the development and usage of domain-specific languages has significantly increased in industry [22] and research [49] in recent years. Nevertheless, recent studies show that there is still a **high demand of domain-specific solutions** for software development [50] as well as insufficient opportunities to configure existing modeling languages for specialized domain-specific use [51]. Consequently, with the increasing development of domain-specific languages [52], domain analysis and domain modeling are growing in importance. This is emphasized by the fact that UML in its recent versions is considered to be too complex [53, 54] (e.g., *"For 80% of all software only 20% of UML is needed. However, it is not easy to find the subset of UML which we would call the "Essential" UML."* [55]).

1.1.3 Use Cases of Domain Modeling

In the previous sections we outlined general benefits of domain modeling and model-driven engineering settings, documented the growing dissemination of modeling solutions in industry and research, and described advantages of domain modeling for software projects in general. In this section we describe three use cases of domain modeling. Two of them are particular types of software projects that constitute the **larger part of industrial IT investments** (integration and modernization projects). The third use case (domain-specific languages) refers to software automation using small high-level languages suited for a particular problem field (domain).

Enterprise Application Integration (EAI) [56] is the process of integrating heterogenous software applications and systems within or across companies to enable interoperability between them and to build added-value applications on top of them. EAI includes data integration as well as process integration.

Application integration has become the most important software development activity. Already in the year 2000, *"Forrester Research estimates that up to 35 percent of development time is devoted to creating interfaces and points of integration for applications and data sources"* [56]. This trend was confirmed in 2012 by Gartner: *"By 2016, midsize to large companies will spend 33% more on application integrations than in 2013. By 2018, more than 50% of the cost of implementing 90% of new large systems will be spent on integration."* [57]. These estimations still hold true:

> *Through 2020, integration work will account for 50% of the time and cost of building a digital platform.*[1]

Enterprise Application Integration best practices lead to a set of EAI patterns [58] for implementing integration solutions. Most important patterns to translate data between software applications are the *Message Bus* pattern[2], that includes the development of a common data model, and the *Canonical Data Model* pattern[3]. The development of **common/canonical data models** is very similar to domain modeling. Both types of models incorporate domain concepts and relations used in a certain application field. A typical approach to overcome semantic heterogeneity in different datasets is the use of domain models and data mappings/transformations [59]. Consequently, the support of developing these data models will result in productivity increases for integration solutions.

[1] `https://www.gartner.com/smarterwithgartner/use-a-hybrid-integration-approach-to-empower-digital-transformation/`
[2] `http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageBus.html`
[3] `http://www.enterpriseintegrationpatterns.com/patterns/messaging/CanonicalDataModel.html`

Software Modernization, also known as software migration, is the task of rewriting or porting existing legacy software systems to new platforms, architectures or programming languages [60]. Modernization includes improvement of applications (e.g., adding new features), change of programming languages and runtime environments, migrating existing data architectures, and making components reusable [61].

Legacy systems are highly important, because they often play a crucial role in the daily business of enterprises. While in the past the focus of software modernization was on programming language modernization (e.g., COBOL to modern object oriented languages [62]) and improving reusability of software (e.g., Services Oriented Architecture (SOA) [63]), in recent years it has become more and more important to migrate existing software applications to cloud architectures and mobile environments [64, 65]. This includes porting applications to web-based execution environments as well as changing data management to distributed data stores.

Software modernization includes (partial) re-engineering or reverse engineering of existing implementations. This process usually starts with **gaining insight** into the current state of the system by the examination of source code, configuration and documentation. The analysis results are captured in representations (e.g., summaries, models, visualizations) different from the original legacy artifacts.

Model-Driven Reverse Engineering (MDRE) [66] proposes the construction of domain models for legacy system description. The process of reverse engineering is supported by frameworks that discover domain models and application models (semi-)automatically [67]. Models for legacy system description are then used to automatically transform parts of the original system to new target representations.

One of the benefits of using domain modeling for software modernization is the improvement of system comprehension [67]. It facilitates the understanding of complex coherencies by software users (domain experts) and software developers at the same time.

Domain-Specific Languages: A DSL *"is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain"* [68]. Domain modeling and domain-specific languages have a very close relationship. While domain modeling is used to express information about concepts and relationships of a particular domain, a DSL consists of these domain elements, proper notations, and interpreters to create models for that domain. It is common practice to use domain analysis and domain modeling during the development of domain-specific languages [68].

DSL development includes phases for decision, analysis, design, and implementation of the language [69]. Decision phase refers to the definition of purpose and intended use of a DSL. The analysis phase comprises gathering of domain knowledge and identification of domain abstractions and their relations. It results in a description of **domain concepts** and domain terminology, usually a domain model. Design phase deals with possible DSL architectures (e.g., based on a general-purpose programming language) and definition methods (e.g., grammars, metamodels). Finally, the implementation phase creates interpreters, compilers, and generators for the language.

In most cases, the analysis phase of DSL development is still conducted informally although guidelines have suggested otherwise for several years. A recent report of Kosar et al. [49] observed that only 5.7% of analyzed DSL studies had used formal approaches. Additionally, tool support for this phase is very limited [69]. The use of feature diagrams for DSL development is a step into the right direction [70]. Still, the use of domain models in the early phases of DSL development is very limited, but they especially improve communication with domain experts (the intended users of a DSL) [71].

1.1.4 Recommender Systems for Modeling

Recommender systems provide context-sensitive suggestions to users in order to support decision-making processes [72]. These systems recommend sets of items in a given context based on users' preferences, background data, and algorithms [73]. Items can be anything that is of interest to the user. Recommender systems are most widespread in e-commerce applications. In this field of

application the main goal is recommendation of interesting products based on the properties of products and what users bought before or had looked at.

Recommender systems are nowadays integrated into many other applications and systems. For example, in social networks they recommend friends or interesting persons based on the underlying graph structures. **Autocomplete** is a recommendation feature that has been successfully integrated into search engines (query autocompletion) and mobile phones (keyboard predictions) for many years and has become indispensable. It tries to predict the next user input based on the previous input, statistical models, background knowledge, and a set of rules.

Integrated development environments (IDEs) have essential recommendation features as well. For source code editors, autocomplete is usually referred to as code completion or **content assist**. It supports developers in two ways: Completions are offered according to the programming language grammar, and through context-sensitive pop-up lists of variables, methods, or objects by using existing source code. These features have also been adopted by textual domain-specific languages (DSL) that offer content assistance according to the DSL grammar.

At present, research in the recommender system community concentrates on solving large-scale input data problems, user interaction, and evaluation of recommender systems [74]. There is an active research community for recommendation systems in the software engineering domain [75]. Many works deal with building recommender systems and algorithms that suggest source code artifacts and that integrate question answering websites (e.g., StackOverflow) to suggest solutions based on expert opinions.

This thesis is motivated by the fact that recommender systems are hardly used in the area of model-driven engineering. It seeks to introduce features similar to autocomplete and knowledge graph widgets of search engines into modeling environments to reduce entry barriers when using model-driven software project methods.

1.2 Challenges of Domain Modeling

Although MDE has been an active field of research for approximately 20 years, it has been recently recognized by industry and research that model-driven engineering requires more connections across disciplines. This PhD thesis follows this direction and works on the support of modeling with knowledge-based systems:

> *Our MDE community has focused much on software and systems modeling, without much interaction with modeling activities in areas such as artificial intelligence, databases, the semantic web, or human-computer interactions. [12]*

Domain modeling, as other knowledge intensive processes, was and still is a challenging task [76, 77]. Over the years this has been addressed by research several times, for example in the area of requirements engineering:

> *Domain knowledge is acknowledged to be important for the acquisition and validation of requirements specification. Unfortunately this is one of the most error prone and costly activities in system development which has led a number of researchers to investigate how the process of requirements capture and validation may be improved. [78]*

Domain modeling requires gathering a lot of pieces of information delivered by different types of persons, documents and other knowledge sources. The support of this process is still an open issue, e.g., as described in a recent article by Ulrich Frank related to domain-specific modeling languages (DSML):

> *First, the development of a DSML requires domain-specific knowledge that may not be available in Information Systems or Computer Science. Thus, there is need for collaboration with other disciplines, e.g., Business and Administration, and with representatives of the respective domains in practice. [79]*

Software engineering is not the only area in which domain expertise plays a crucial role. The importance of domain knowledge and domain models was recently also addressed in the Big Data area by both the Data Engineering community [80] and the Conceptual Modeling community [81].

This section describes the key challenges that are addressed in this dissertation. First, the challenges of acquiring domain knowledge in software project settings are described (cf., Section 1.2.1). Then, the difficulties involved in obtaining knowledge from data sources that provide access in a structured way and that are only available in an unstructured form are characterized (cf., Section 1.2.2). Finally, the challenges arising from the unavailability of large conceptual knowledge resources for domain modeling are discussed (cf., Section 1.2.3).

1.2.1 Cost of Domain Knowledge Acquisition

In a software project, it is essential that developers understand the application domain and learn the respective concepts, their relationships and interactions. In essence, these concepts are represented by pieces of source code in the final software product. Consequently, it is very important that domain information is interpreted correctly to meet the user's requirements. Domain modeling is one of the important methods to gather domain information and domain knowledge using formal models and notations. These models help to make the domain knowledge explicit and available to all participants of a project.

Domain modeling and acquisition of domain knowledge is a time-consuming process [82, 83, 84]. It requires intensive collaboration between engineers and domain experts, studying large sets of documentation, additional modeling effort, and *"can take several person-years before a useful application program is delivered"* [82]. Although this quotation goes back to the eighties, it still very well characterizes the situation of software projects nowadays and why there is still a lot of research carried out to automate this process [85, 86]. The main reasons for the time-consuming process of domain knowledge acquisition are:

(1) **Cold Start Problem:** In a software project that aims to address an unsolved problem, it is often not an option to build on existing domain models. Domain engineering [87] tries to overcome this difficulty by building reusable libraries with domain information. It is a well-known problem [88, 89] that existing knowledge bases (often created manually) do not cover enough information or do not exist at all for certain target domains. This is usually referred to as the cold start problem: Reusable domain knowledge will become only available if enough solutions have already been developed using that methodology while new projects already want to benefit from the domain knowledge. Furthermore, in cases where it is possible to rely on previous projects of the same domain, commonalities of these projects have to be manually discovered to construct domain models. Even if existing domain models can be reused, it is still subject to discussion and agreement which concepts and relationships are relevant to a new software project and for which parts new domain information has to be collected. In the end, domain models frequently have to be developed from scratch [90, 91, 92].

(2) **Different Roles and Expertises:** A software project typically involves a number of stakeholders: e.g., project managers, software engineers, modeling experts, domain experts, and end users[4]. Literature commonly divides these roles into two groups: technical participants (developers) and non-technical participants (domain experts) [93]. The first group refers to persons involved in software design, development, and modeling. The second group usually includes representatives of the intended users of the software and persons with special knowledge in a particular area of interest (the domain for which the software is implemented). This thesis concentrates on domain modeling, consequently we will differentiate between modeling experts and domain experts (see Figure 1.1). Ionita et al. [93] very well characterize the challenges of collaboration between these two groups:

[4]The list is not intended to be complete, for more detailed description of software project roles we would like to refer the reader to http://zimmer.csufresno.edu/ sasanr/Teaching-Material/SAD/.

Conceptual models represent social and technical aspects of the world relevant to a variety of technical and non-technical stakeholders. To build these models, knowledge might have to be collected from domain experts who are rarely modelling experts and don't usually have the time or desire to learn a modelling language. [...] A conceptual model consists of concepts and relations among the concepts, by which people understand a part of the world. They are usually represented in (software) modelling tools using abstract graph-like structures containing boxes, arrows, and other symbols. The problem with these abstract representations is that domain experts whose input or feedback is needed to construct an adequate model may be unfamiliar with the notation, and may not be willing or able to learn it.

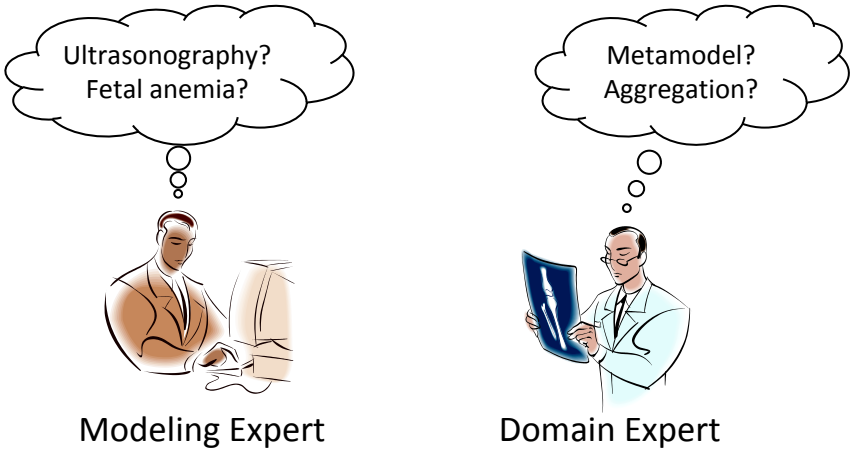


Figure 1.1: Examples of different mind-sets and roles participating in an MDE software project

The different mind-sets of both groups require a learning phase at the beginning of a project that is a time-consuming process. With respect to domain modeling, this process can be divided into two steps: First, the modeling expert has to learn a lot about the domain. The acquired knowledge is then transformed into appropriate models by him. Second, the models are presented to the domain expert by the modeling expert and discussed with him. At this point the domain expert has to deal with the modeling notation to suggest corrections. In the course of discussion, the modeling expert is already required to provide appropriate explanations of the models using the domain terminology of the domain expert. The two steps are usually repeated in several iterations, requiring additional meetings and telephone conferences. In the process of domain model creation, the modeling expert bears the greater burden. The modeling notation is a way of communicating the concepts and relationships of the domain that the domain expert already knows. It is easier for him to learn a small set of modeling constructs. The modeling expert is faced with using the correct domain terms and arranging the relationships correctly. This implies a deeper understanding of the domain. The importance of domain language is very well summarized by Evans [39]:

On a project without a common language, developers have to translate for domain experts. Domain experts translate between developers and still other domain experts. Translation is always inaccurate and hides disconnects in understanding between the domain experts and developers, between different developers and between different domain experts. [...] The overhead cost of all the translation, plus the risk of misunderstanding, is simply too high. A project needs a common language that is more than the lowest common denominator. With a conscious effort by the team, the domain model can provide the backbone for that common language, while connecting team communication to the software implementation.

To summarize the **first challenge**, the effort of acquiring domain knowledge and creating appropriate domain models is high. It is a costly activity including solitary work by the modeling experts (gathering information from different sources such as enterprise documents and specifications) as well as a lot of communication with domain experts. At the same time, it is a very important activity for the success of software projects [94].

1.2.2 Heterogeneity of Knowledge Bases

Generally, the field of knowledge acquisition differentiates between two types of knowledge: Tacit knowledge and explicit knowledge [95]. Tacit knowledge refers to experiences learned by individuals that are difficult to transfer or to verbalize [96] (know-how knowledge). Explicit knowledge is a type of knowledge that can be expressed by words, scientific formulae, or other notations (know-what knowledge) [97]. In this thesis, we focus on the acquisition and analysis of explicit knowledge that is "readily communicated and shared through print, electronic methods, and other formal means" [97] or that can easily be made explicit. Consequently, domain modeling is a way of extracting and transforming certain types of existing explicit knowledge (e.g., documentation, databases) to other representations of explicit knowledge (domain models).

One of the major difficulties in knowledge acquisition is that domain knowledge is contained in arbitrary sources. It is a time-consuming process to first find the relevant documents or artifacts containing the required information and then manually inspect the text or data with respect to the concepts and relationships of a domain. We differentiate between two types of knowledge sources: structured information and unstructured information. The *"intended meaning [of structured information] is unambiguous and explicitly represented in the structure or format of the data"* [98]. Unstructured information is not organized in a pre-defined manner. Its *"intended meaning is only loosely implied by its form"* [98].

Examples of structured information sources are databases, XML documents, knowledge bases, and models. Each of them provides access to information according to a certain schema that a user can rely on. Nonetheless, interfaces for search and retrieval may vary a lot. Consequently, additional effort is necessary for users to learn how to query the knowledge sources. From a user's point of view, uniform access to all sources is not available in most cases and may only be facilitated by building additional search engines on top of them. From a machine's perspective, standardized querying of all these different sources is not possible, because they all may rely on different data models and query languages. Not only syntactic and data model heterogeneity is a challenge, but also semantic heterogeneity [99] that concerns different meanings for the same identifiers and names in schemes (ambiguous terms).

Unfortunately, the number of structured information sources is negligible compared to unstructured information sources. It is estimated that 80% of existing data is unstructured[5]. Additionally, the majority of unstructured information is natural language text. Other examples of unstructured data are audio-visual content, log files, and source code. The amount of text data is growing rapidly[6]. Apart from acquisition from structured sources, this thesis focuses on knowledge acquisition from text corpora. Domain information is contained in a variety of documents, such as text books, manuals, dictionaries, encyclopedias, or requirements specifications. Relevant facts have to be located first and then interpreted. On the one hand, a major challenge is that humans may manually interpret text more precisely, but cannot process the amount of text in reasonable time. On the other hand, natural language understanding by machines is scalable, but it is a very difficult task, because it requires contextual knowledge and has to deal with ambiguities of words and formulations.

There are several research fields that work on sub-problems of the aforementioned challenges. In general, there is no ready-to-use solution to make available domain knowledge from heterogeneous data sources. In fact, several components, tools and partial solutions have to be tailored down to a domain knowledge extraction framework. Information retrieval [100] in general is one

[5] `https://www.forbes.com/sites/forbestechcouncil/2017/06/05/the-big-unstructured-data-problem/`
[6] `https://www.domo.com/learn/data-never-sleeps-5`

way of facilitating the search for relevant documents with domain information, but has limited capabilities in extracting fine-grained pieces of information from these documents. Information integration [101] is the merging of information from different data sources with different data structures into a common, uniform data structure. Information integration can be used in part for the acquisition of domain information from different sources, but often requires well defined schema information, a schema matching and a data fusion process.

Even if ready-made knowledge bases are available, as in the Linked Open Data Cloud[7], the combined access to them still presents a challenge: *"Moreover, these graphs are distributed over many different sources with very different characteristics. [...] Not only do we need methods to represent and analyse each kind of graph, we also require the means to combine them and to perform multi-criteria analyses on their combinations."* [102]

To summarize the **second challenge**, domain knowledge is contained in multiple sources, some in a structured form, but mostly unstructured. Consistent access to a large amount of structured domain knowledge is not available because of the heterogeneity of file formats, access methods, protocols, schemata, and the unavailability of explicit definitions of the meaning of the data.

1.2.3 Lack of Conceptual Knowledge Resources

Over the last decade, several projects have used automated processing techniques to create large knowledge resources from semi-structured and unstructured data. However, in the context of domain modeling, there is still a shortage of knowledge resources that contain conceptual knowledge.

The first type of these projects are projects that use Wikipedia as a source. The best known examples are DBpedia [103] and YAGO [104]. DBpedia is an extraction framework and knowledge base that converts semi-structured information of Wikipedia's info boxes into RDF triples. DBpedia uses a proprietary data model (ontology schema with roughly 770 classes[8]) and collaboratively developed mappings for extraction of class instances and relationship type instances. YAGO follows a similar approach, using Wikipedia info boxes, categories, as well as WordNet synsets. It was extended with temporal and spatial information in subsequent versions. WikiNet [105] also extracts facts from Wikipedia with a focus on multilingualism and the category system. These knowledge bases aim to collect as much factual knowledge as possible from semi-structured data at the instance level. Although they contain millions of facts, the proportion of conceptual knowledge is very low.

The second type of resource is collaboratively gathered knowledge. Freebase [106] was the first effort that collaboratively collected statements about real entities. Its development was discontinued and merged into Wikidata [107], a Wiki-based fact editor. Wikidata incorporates a more extensive data model and has already collected millions of facts for over 56 million objects[9] on a manual basis. As with the first type, the goal of these projects is to gather factual knowledge, and the collection of conceptual facts is not in the focus.

The third type of knowledge resources are linguistic databases that cover dictionary-related content and lexical-semantic knowledge. Most resources available are based on linguistic theories and have been created manually. On the one hand, they offer quality content, on the other hand, they do not provide as much coverage as automated approaches. The most popular resource is WordNet [108], a lexical database for the English language. It contains synsets (groups of synonyms) that are linked with a few lexical-semantic relations. FrameNet [109] is a lexical database that encodes the understanding of the English language in frame semantics, a way to organize knowledge to understand the meaning of words. VerbNet [110] focuses on the lexical entries of verbs with detailed syntactic-semantic descriptions and thematic roles. These knowledge bases can be better used as sources of domain knowledge, but are not extensive enough due to manual creation. BabelNet [111] was the first automated approach to extracting and aligning Wikipedia content to WordNet to create a larger encyclopedic dictionary. Later releases subsumed more and more resources such as VerbNet, FrameNet, and Wiktionary to create a single access

[7] `http://lod-cloud.net/`

[8] Based on `http://mappings.dbpedia.org/server/ontology/classes/` as of April 2019.

[9] `https://www.wikidata.org/wiki/Wikidata:Statistics` - as of May 2019

point for lexical information. In addition, it focuses on multilingual machine translation for entries that do not have appropriate lexicalizations.

Another resource of knowledge are common sense knowledge bases. They encode general world knowledge in machine-readable form. Cyc [112] was one of the first efforts to capture common sense concepts and knowledge. The Cyc Project invested a man-century during several years to manually collect facts and axioms. The Open Mind Common Sense (OMCS) project[10] was launched in 1999 as a crowdsourcing knowledge project to collect common sense knowledge in the form of English sentences. In 10 years, more than a million facts were collected. It became the main base for ConceptNet [113], a knowledge graph that encodes common concepts using words / phrases and labeled edges between them. ConceptNet has additionally been extended by converting portions of Wiktionary, WordNet, OpenCyc, and other manually created resources. Due to its integration with Wiktionary, ConceptNet also focuses on multilingualism.

In addition to knowledge resources that have been built manually or created by processing semi-structured content, there is a final category of methods that aims to create knowledge resources from unstructured data. Information extraction [114] utilizes various tasks from computational linguistics [115], artificial intelligence and information retrieval [116] to generate structured facts from natural language text. Most extraction systems aim to produce factual statements and differ in the degree of automation and initial effort required to run the extraction process (pattern-based extraction, (semi-)supervised learning, distant supervision, unsupervised learning). Well-known systems are: TextRunner [117], ReVerb [118], NeLL [119], which can be summarized by the term Relation Extraction (RE). The extraction of conceptual knowledge from text has received much less attention. There is the pioneering work of Hearst [120] for the discovery of *is-a* relationships and some work on taxonomy induction [121, 122], but in recent years research has focused on a simpler hypernymy relationship detection [123], which does not distinguish between class and instance levels.

To summarize the **third challenge**, there are a few usable lexical-semantic resources for domain modeling, but most have been created manually and are not extensive enough. Most other approaches to information extraction and auto-generated knowledge bases focus on factual knowledge at the instance level that cannot be used for conceptual-level domain modeling. In addition, much of the work is based on either WordNet or Wikipedia, which limits the structure, diversity, and domain coverage of knowledge bases.

1.3 Objectives, Contributions and Outline

This section summarizes the goals and main contributions of the thesis to the domain modeling challenges outlined before. The use of domain knowledge in domain models is an important factor in the implementation of software projects. As described in the previous section, domain modeling is hampered by time-consuming manual activities to capture domain information, the unavailability of uniform access to structured knowledge bases, and the lack of large conceptual knowledge sources.

This work aims at developing methods and tools to provide automated modeling recommendations for the creation of domain models, i.e., domain knowledge is made available directly during the modeling process. We propose to use automated knowledge acquisition from text corpora and integration with semantic databases to provide context-sensitive suggestions of model elements. The vision of semantic modeling support is as follows: The content of a domain model is analyzed during its development. Based on the terms used in the model, the developer receives information about related content and suggestions as to what he or she might include in the model. The suggestions will be adjusted with each model change.

[10] `https://github.com/commonsense/omcs`

Given the challenges of domain modeling and the vision of automated modeling support, the research question addressed by this thesis is *"How to improve the development of domain models through automated knowledge acquisition?"*. To answer this question, the following is considered.

(1) *Where does the required knowledge come from?* Semantic knowledge bases and ontologies are an important source of structured knowledge, and if insufficient, text datasets are analyzed to gain additional knowledge.

(2) *How can the necessary knowledge be acquired automatically?* We propose the automated construction of large semantic networks of related terms from text using natural language processing as well as automated querying of existing knowledge bases.

(3) *How can the acquired knowledge be used to improve modeling?* Acquired knowledge is transformed and used for context-sensitive recommendations, depending on the terms and relations in a domain model and what is currently being changed in the model.

(4) *How does model development affect the acquisition of knowledge?* At each step of a modeling process, guidance needs to be adapted to the changing content of a model.

To answer these questions, this work contributes to four areas. The first two contributions deal with the acquisition of domain knowledge from unstructured and structured sources. The third and fourth contributions focus on the supply of accumulated knowledge in the domain modeling process.

Contribution 1: Semantic Network of Terms The main contribution of this thesis addresses research challenge 3 (the lack of conceptual knowledge resources) with a semantic network of related terms (SemNet) focusing on the conceptual level. SemNet was constructed automatically using pattern-based extraction on a large N-gram text corpus. The approach relies on syntactic properties of sentences and statistical features of text corpora to perform a *domain-independent extraction*. The key features of the approach are: it only requires computational cheap shallow linguistic analysis and the extraction is directly performed on the N-grams that serve as proxy for the original text corpus. SemNet features almost 6 million unique one-word terms and multi-word expressions and 355 million weighted binary and ternary relationships between them.

Contribution 2: Ontology Connector The second contribution deals with research challenge 2 (heterogeneity of knowledge bases) and enables uniform access to lexical information contained in structured knowledge bases. The OntoConnector realizes a mediator-based querying approach that allows to integrate several types of knowledge bases on-the-fly with no effort. In case knowledge bases do not use standardized data models, a template-based strategy for SPARQL endpoints allows to integrate nearly every proprietary knowledge base.

Contribution 3: Semantic Modeling Support We approach the first research challenge (cost of domain knowledge acquisition) by analyzing the types of operations a user can perform to change a model and defining strategies for each modeling scenario to provide domain knowledge during the modeling process. In addition, we develop semantic mappings between domain modeling and knowledge representations to enable the automatic retrieval of domain knowledge and its transformation into modeling proposals. The semantic modeling support provides the theoretical basis for modeling recommendations.

Contribution 4: Domain Modeling Recommender System The fourth contribution addresses the first research challenge from a practical point of view, how domain knowledge and recommendations can be made available in a real modeling environment. The DoMoRe recommender system implements the developed support methods and integrates the semantic network of terms with connected lexical knowledge sources. This allows context-sensitive information to be provided during domain modeling in a Model Advisor and to propose semantically related names for model elements ordered by relevance in a Semantic Autocompletion feature.

Outline. The thesis is structured as follows. Chapter 2 introduces fundamentals of software modeling, knowledge bases, information extraction, and recommender systems relevant to the thesis. Chapter 3 presents our method for semantically supporting domain modeling by using knowledge contained in text datasets and structured knowledge bases. In Chapter 4 we describe the automated extraction of such knowledge from text and construct a large-scale semantic network of related terms (SemNet). Chapter 5 is devoted to the extraction of structured lexical and conceptual knowledge by utilizing knowledge base querying (OntoConnector). Chapter 6 presents the implemented domain modeling recommender system (DoMoRe) that generates context-sensitive modeling suggestions using SemNet and OntoConnector. Chapter 7 describes the research projects in which the developed methods and tools were applied and reports on practical experiences. Finally, Chapter 8 summarizes key research results and gives future work directions.

References. This work uses references to web addresses in addition to citations. The last access dates to the resources are not specified separately. All URLs referenced in this thesis were reviewed for availability during May 1-5, 2019.

Chapter 2

Foundations

2.1 Introduction

This chapter describes foundations in the relevant research fields for this work. This PhD thesis connects concepts, approaches and tools from mainly four areas of research, as shown in Figure 2.1.

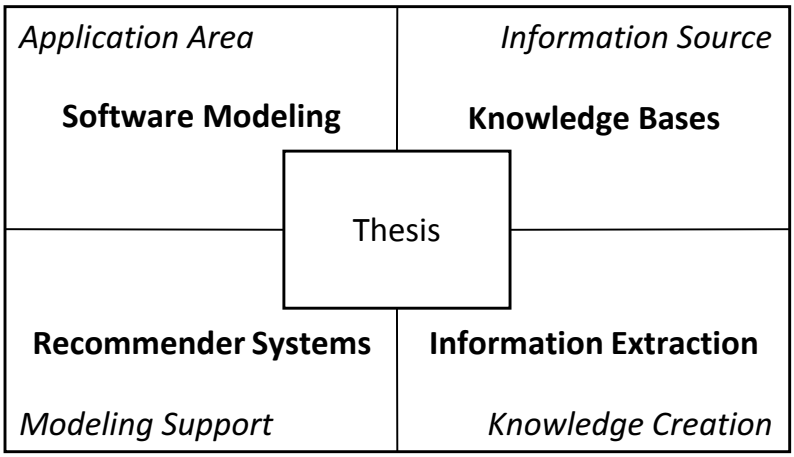


Figure 2.1: Main research areas related to this thesis

The application area of this thesis is *software modeling*, in particular *domain modeling*, an activity in software engineering to support analysis and development steps by using formal models. The field of *knowledge bases* deals with the explicit specification of factual knowledge about concepts and entities. We use knowledge bases as source of information to support domain modeling. *Information extraction* is the process of harvesting structured facts from unstructured data sources. This thesis utilizes methods from this field to create new knowledge sources. *Recommender systems* provide context-sensitive suggestions to users in order to support decision-making processes. The presented recommender system supports the development of domain models by using lexical information of knowledge bases and extracted information.

2.2 Foundations of Software Modeling

This section presents an introduction to the most important terms, concepts and methods of software modeling that are relevant to this thesis.

The Object Management Group (OMG)[1] describes software modeling or modeling in general as *"[...] the designing of software applications before coding"* [124]. The general goal of modeling is to raise the level of abstraction in complex software projects by using models (cf., Section 2.2.1). These models are usually created with the help of metamodels (cf., Section 2.2.2) and modeling languages (cf., Section 2.2.3), that have different purposes, such as the description of a system's static structure or the behavior of software components. This thesis concentrates on domain modeling (cf., Section 2.2.4) in which models are created that describe real-world concepts and relationships of the application areas in which software is used. Furthermore, this chapter describes important methods of software development that consistently use modeling (cf., Section 2.2.5 and Section 2.2.6).

2.2.1 Models

Models are used in a variety of engineering disciplines as representations of reality. They are usually created to study or analyze a certain aspect of a real thing before constructing it. In our case, the engineering discipline is software engineering that deals with the systematic construction of software systems. Software engineering is a complex task, consequently models help to reduce complexity in order to improve the way software is built.

A clear definition of the term *model* is difficult [125, 126]. A literature review of software modeling papers in the 2000s already revealed nine different definitions [127]. On the one hand, models have been used intuitively ever since, and on the other hand, the term is heavily overloaded. In this thesis, the common practice in literature [127] is followed that a description of models is approached by using certain properties. One of the most important notions on what a model constitutes was defined in Herbert Stachowiak's model theory. According to his theory a model must meet three criteria [128]:

1. Mapping criterion: Models are always models of something, namely illustrations, representations of natural or artificial originals, which themselves can be models again.
2. Reduction criterion: Models generally do not capture all the attributes of the original represented by them, but only those that seem relevant to the respective model creator or model user.
3. Pragmatic criterion: Models are not clearly assigned to their originals. They have a replacement function a) for certain model-utilizing subjects, b) within certain time intervals, and c) are subject to certain mental or actual operations.

These criteria mean that there must be an original, which is described by the model, that the model is an abstraction, and that it has a purpose, thus substituting the original. Abstraction is the key property of a model. It allows to focus on specific aspects without considering the full complexity. Among abstraction, Selic [16] defines five characteristics that a model must possess within a software engineering context:

1. Abstraction: A model is always a reduced rendering of the system that it represents.
2. Understandability: What remains by abstracting details must be presented in a comprehensible form.
3. Accuracy: A model must provide a true-to-life representation of the modeled system's features of interest.
4. Predictiveness: By using a model it should be possible to correctly predict interesting but nonobvious properties of the modeled system.
5. Inexpensiveness: It must be significantly cheaper to construct and analyze a model than the modeled system.

[1] `http://www.omg.org/`

Models may come in several forms. A useful categorization into graphical, mathematical, and textual models was proposed by Liddle [129]. The explanations of graphical and mathematical models are adopted in this thesis, but another definition of textual models by Jouault [130] is used, mainly for the reason that in software projects natural language text is considered too imprecise to be a model.

1. Graphical models are diagrams that depict concepts using lines, shapes, symbols, and usually some text.
2. Mathematical models describe aspects of systems as formulae.
3. Textual models describe parts of a system using specific syntaxes and grammars.

This thesis concentrates on graphical models used for software engineering tasks that consist of a visual notation (using a concrete syntax) and an internal data model (using an abstract syntax) to store the information that is expressed with the graphical model. Despite the difficulties of a clear definition of the term model, the definition of A. Brown [131] is most suitable for this work:

> *Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones.*

For a more detailed analysis of model definitions and relations the reader is referred to the work of Muller et al. [127] and Rodriguez-Priego et al. [132].

2.2.2 (Meta-)Metamodels

The creation of models requires to know what can be expressed and what assertions form a valid model. A metamodel describes how to formulate models. Gašević et al. [133] define a metamodel as follows:

> *A metamodel is an explicit model of the constructs and rules needed to build specific models within a domain of interest.*

This definition states that a metamodel is a model as well. Consequently, a metamodel adheres to the same properties and criteria described in the previous section. Following this recursive definition, the specification of how to construct a metamodel is provided on the next meta-level, in a meta-metamodel. Theoretically, this chain of meta-levels could be arbitrarily long. For practical reasons, it is common to limit the number of levels to four, which has been discussed by several authors (e.g., Bézivin[134], Atkinson & Kühne [135], Henderson-Sellers [136], Aßmann [137]) with its advantages and disadvantages. This four-level metamodeling architecture is common practice in model-driven engineering and was popularized by the Meta Object Facility (MOF) specification, which will be described in the following paragraph.

Meta Object Facility (MOF). MOF [138] is an Object Management Group (OMG) standard that provides a meta-modeling language to define metamodels. It also defines repository interfaces for model element storage and interchange across applications. MOF consists of the Essential MOF (EMOF), a subset of the Complete MOF (CMOF), which provides the core facilities to build metamodels. MOF is based on the principle that metamodels reside on one level and their models are located one level below. As illustrated in Figure 2.2, the practical implementation of this principle is a metamodeling architecture with four layers.

At the **M3 Layer**, the MOF meta-metamodel is located. It is the foundation for defining any modeling language and provides the key concepts for metamodel creation (e.g., meta-elements to specify classes, attributes or associations). MOF is self-defined, it conforms to its own language. This reflective mechanism allows the limitation to a practical number of modeling layers.

At the **M2 Layer**, metamodels are created using the MOF language, such as the UML metamodel, the CWM metamodel[2], or any other metamodel for a user-defined modeling language.

[2] `https://www.omg.org/spec/CWM/About-CWM/`

Metamodels at this level define the elements of which a model can be constructed, what properties these elements have and how these elements relate to each other.

The actual models that abstract from reality are located at **M1 layer**. For example, models on this level are class diagrams or use case diagrams, in case UML is used. In this metamodeling approach, the relationships between M1, M2, and M3 levels are called *conformsTo* (as described by Favre [139] and Bézivin [134]), meaning that a model was formulated according to the definitions and rules of its metamodel.

The lower **M0 Layer** differs from the other layers as it contains the things that were abstracted from (the originals according to Stachowiak). It separates reality from the modeling space. Real systems or other real objects on M0 layer are *representedBy* models on the M1 layer. Earlier versions of the MOF specification were based on *instanceOf* relationships between all levels. M0 layer was included in the modeling space, which led to a lot of confusion [140]. MOF is a strict modeling paradigm [141], in which a model element on one layer must have a correspondence to a model element on the layer above. An alternative approach is the ISO/IEC 24744 specification that allows relations across multiple levels and different types of relations between layers.

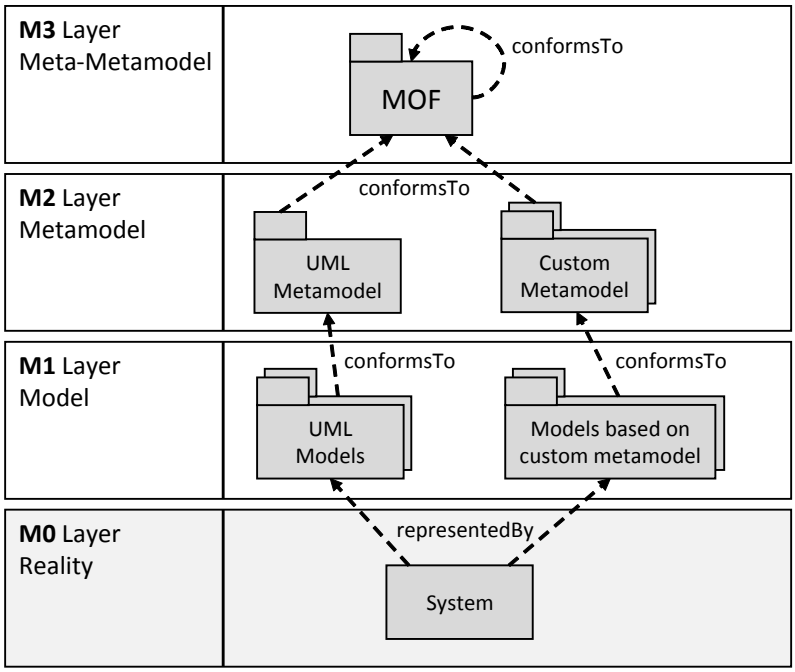


Figure 2.2: Four-layer metamodeling architecture, after [142] and [134]

Ecore. Ecore is a near-standard implementation of the Essential MOF specification [143]. Ecore is part of the Eclipse Modeling Framework (EMF)[3] and serves as a common meta-metamodel to build metamodels (abstract syntaxes) or domain models of domain-specific languages (DSLs). Ecore, EMF, and other development frameworks of the Eclipse Modeling Project[4] are the most widely used open source tools to build modeling languages, domain-specific languages and appropriate editors. The naming of Ecore in literature is often confusing, because sometimes it is referred to as the Ecore metamodel and sometimes it is called Ecore meta-metamodel. Nevertheless, its main purpose is to develop metamodels of DSLs. Thus, we follow the practice [142] that Ecore is located at M3 layer of the metamodeling architecture (cf., Figure 2.2) and call it Ecore meta-metamodel.

[3] `https://www.eclipse.org/modeling/emf/`
[4] `https://www.eclipse.org/modeling/`

2.2.3 Modeling Languages

In the previous sections, models, which are the artifacts that convey abstracted information about a software system, as well as metamodels that specify the concepts and rules to construct models have been introduced. But more parts are necessary, which make up a modeling language. A definition by Booch et al. compares modeling languages to natural languages [144]:

> *A language provides a vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system.*

Vocabulary and combination rules of a modeling language are defined in a metamodel. Thomas Kühne [145] relates languages to the metamodeling layers architecture as shown in Figure 2.3. The modeling language is a separate concept and the metamodel is an integral part of a modeling language. A model is expressed using a language and a language is defined by a metamodel. The same applies to the metalevel above. Instead of the *conformsTo* relationship Kühne uses *linguistic instanceOf* to emphasize that models are valid expressions of a language. It is an inter-level relationship in contrast to the intra-level relationship *ontological instanceOf*, a relationship on the same level between different logical units.

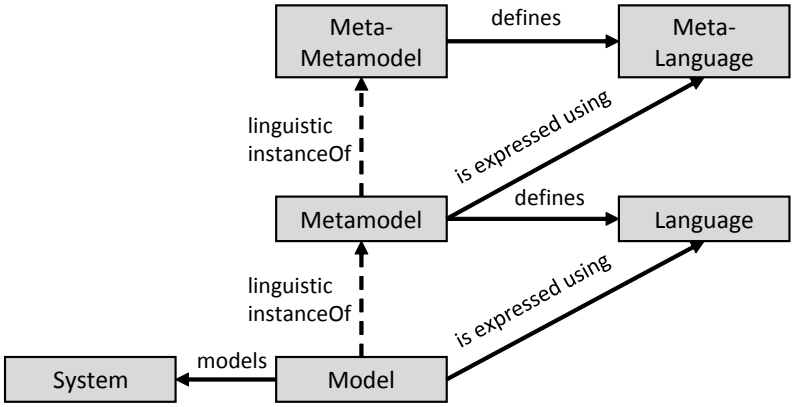


Figure 2.3: Language definition stack after Kühne [145]

So far only available elements and combination rules of a language have been discussed. These definitions only refer to the syntax of a modeling language. According to Karagiannis and Kühn, a modeling language is described by syntax, notation, and semantics [146]. These three components are also often referred to as abstract syntax, concrete syntax and semantics (cf., Selic [147] and Atkinson [148]).

Abstract syntax addresses the structure of the language separated from its notation. It defines the language concepts and how they can be combined. Additionally, it allows to verify the well-formedness of a model. Depending on the approach, well-formedness rules can also be defined separately from abstract syntax. The two most common techniques for abstract syntax definition of a modeling language are metamodels [148] and grammar definition languages (e.g., EBNF[5]). The terms abstract syntax and metamodel are often used interchangeably [149].

Concrete syntax (notation) refers to the visual representation of the modeling language concepts. Graphical modeling languages, often used in conjunction with metamodels, usually define shapes, labels and connections that are arranged in diagrams (e.g., boxes connected with arrows). Textual modeling languages, usually based on grammars, use keywords, special characters, and parameters for the language elements. Whereas a modeling language can have only one abstract syntax, it is possible to define multiple different concrete representations [147].

[5]Extended Backus-Naur form

Semantics define the meaning of the modeling language concepts. There are approaches that define semantics with the help of a semantic domain (e.g., with the help of ontologies or other formal languages) and a semantic mapping that links the language elements of the abstract syntax to elements in the semantic domain [146, 150]. It is also very common to describe semantics of language elements informally using natural language text (e.g., most of the UML is described like that). Finally, semantics can be described in terms of behavior of the model elements. These executable semantics are similar to programming languages that describe the behavior of the program when it is executed. A definition by da Silva [126] very well summarizes all the previously described concepts:

> *We define modeling language as a set of all possible models that are conformant with the modeling language's abstract syntax, represented by one or more concrete syntaxes and that satisfy a given semantics. Additionally, the pragmatics (of a modeling language) helps and guides how to use it in the most appropriate way.*

Unified Modeling Language. The UML is the most widespread modeling language for software construction [22, 8] (see also Section 1.1.1). It "is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system" [144]. UML is the outcome of an initiative to harmonize several object-oriented methods in the early nineties. It emerged from the Booch method by Grady Booch, the object-modeling technique (OMT) by Rumbaugh and the Object-Oriented Software Engineering (OOSE) by Jacobson ("the three amigos"). From the beginning, software organizations where integrated into the development of UML making it the de facto standard language for software specification and design in industry [8].

UML distinguishes two types of models for software construction: modeling the structure of a system and modeling the behavior of a system. Structural aspects refer to the static parts of models that represent conceptual or physical elements (e.g., classes, components). Behavioral aspects are the dynamic parts of models that describe the interactions between the structural elements and their internal states during lifetime. The objective of UML is not only the use of models for software construction. UML is a graphical language, thus it facilitates understanding of software systems and communication between different stakeholders involved in a software project.

UML provides several diagram types for structural description. The most important diagrams [22] are *class diagrams*, which describe sets of objects that share the same attributes, operations, relationships, and semantics [144], *component diagrams*, which show pieces of assembled software parts and their connections, and *deployment diagrams*, which describe an architectural view of run-time nodes and the distribution of components.

Most often used behavioral diagrams are *activity diagrams*, which show the flow of control among objects, *use case diagrams*, which describe a high-level view of the behavior of the system in relation to different types of users (actors), *sequence diagrams*, which describe a time-ordered interaction between objects using messages, and *statechart diagrams*, which model the event-ordered view of a class or interface using states and transitions.

We will not elaborate more on the large possibilities of the UML language(s), as this thesis concentrates on the support of modeling domain information that is predominantly knowledge about concepts and types of entities of domains. For further reading we refer the reader to the books of Booch et al. [144] and Oestereich [151], and to the UML specification 2.5 [152] that describe all the 14 diagram types of UML in detail. In this section, we will only provide an example of a UML class diagram that is the most widely used diagram type to capture domain information. A quote from the Unified Modeling Language User Guide very well describes the importance of classes [144]:

> *You use classes to capture the vocabulary of the system you are developing. These classes may include abstractions that are part of the problem domain, as well as classes that make up an implementation. You can use classes to represent software things, hardware things, and even things that are purely conceptual.*

Figure 2.4 depicts a set of classes from the school domain, their attributes and relations. The class diagram is based on a diagram of Jacobson et al. [144], to which we added additional

attributes for all classes. Such a diagram could emerge from the early development process of a school information system. It shows that a SCHOOL consists of one or more DEPARTMENTS (Composition relationship – departments cannot exist on their own), and one or more STUDENTS are members of a school (Aggregation relationship). COURSES are associated to departments (Association relationship), and any number of students can attend any number of courses. Every department has a set of INSTRUCTORS that teach the courses, and one of them may be the chair of a department. The attributes, such as `name`, `address`, and `phone_number`, are the properties that should be available in the school information system for the respective classes.

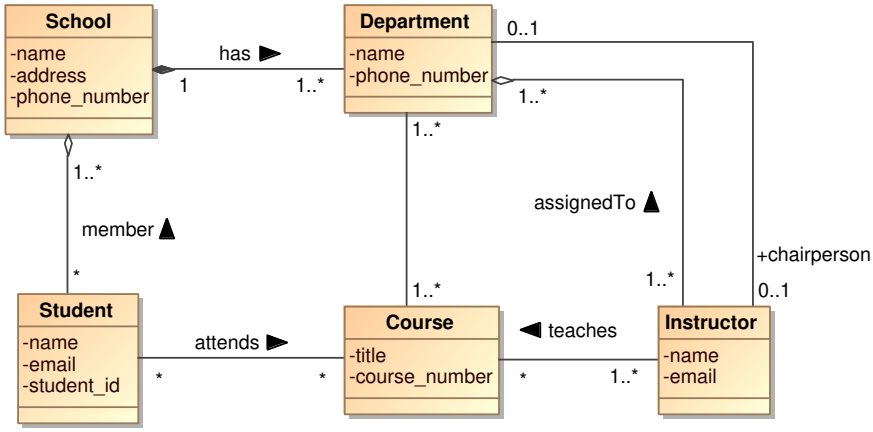


Figure 2.4: Example of a UML class diagram, after [144] with some additions

2.2.4 Domain Modeling

In software engineering, domain modeling is used to capture concepts and relationships of a certain application area in a domain model with respect to a specific software project. A definition by Iscoe et al. [34] describes:

> *Domain models are representations of an application domain that can be used for a variety of operational goals in support of specific software engineering tasks or processes.*

This definition focuses on the use of domain models for particular development tasks. Representing an application domain (also called the problem domain [32]) deals with organizing specific knowledge of that domain. A definition by Evans [39] pays attention to the captured knowledge:

> *The domain model is not a particular diagram; it is the idea that the diagram is intended to convey. It is not just the knowledge in a domain expert's head; it is a rigorously organized and selective abstraction of that knowledge.*

The previous sections described technical prerequisites of modeling and modeling languages. In contrast, domain modeling is concerned with the content of the models. Domain modeling is not necessarily bound to using a specific modeling language, different kinds of modeling techniques are possible. In most cases domain modeling focuses on the explicit formulation of technical terms and their relationships, as the following quotation of Atkinson indicates [35]:

> *One important purpose of domain models is to serve as a description of the problem that is understandable to the widest possible range of stakeholders.*

In the following, we shortly introduce the most important concepts related to domain modeling. **Domain Analysis** is the prerequisite for domain modeling. It comprises collecting information in the field of business or technology in which a piece of software is to be used. The goals of

domain analysis are the improvement of communication among stakeholders that will lead to faster development and the assembly of domain knowledge that will result in improved designs and more adaptable systems. In this thesis we adhere to the general definition of domain analysis by Prieto-Díaz: *"domain analysis can be seen as a process where information used in developing software systems is identified, captured, structured, and organized for further reuse"* [33]. Domain analysis results in the creation of several domain models with different purposes. Important for this thesis are domain dictionaries and notations [153] (thesauri, vocabularies, concept models, and concept representations).

Domain Engineering is a set of activities for creating software products based on reusable software assets. This software engineering method is two-fold and consists of two interlinked phases: One phase (domain engineering) is dedicated to the creation of reusable software artifacts (software product lines [154]), and the other phase (application engineering) concerns the development of applications adapting and customizing these software artifacts. Widely used domain engineering methods are based on feature models (e.g., Feature-Oriented Domain Analysis (FODA)) to determine commonalities and variabilities of software product lines and to derive software applications using configuration. These methods are not subject of the thesis. In contrast, we concentrate on the domain engineering phase, in particular on the domain analysis and domain design sub tasks [87].

Domain-Driven Design is an approach to complex software systems design that puts the domain and domain logic at the center of development. *"Every software program relates to some activity or interest of its user. That subject area to which the user applies the program is the 'domain' of the software."* [39]. The primary concepts of domain-driven design are the central domain model, the ubiquitous language, the consequent use of model-driven design and several design patterns and concepts to continuously extract relevant and refined information for software design. The domain model collects as much knowledge as possible about the domain as described in the previous paragraphs. The ubiquitous language is a common language tightly coupled with the domain model that both domain experts and modeling experts understand. The use of model-driven design ensures that the domain model and the source code are tightly coupled as well. This is mainly implemented using a layered architecture that separates domain logic from application logic, user interface, and infrastructure. Domain-driven design incorporates the following strategic design principles. (1) Bounded context: models have clear boundaries in which they should be kept consistent with terms and code. (2) Continuous integration: merging of implementation artifacts should be done with an automated process including automated tests. (3) Context maps: they provide the domain structure in terms of bounded contexts and their connections.

Conceptual Modeling has its origin in the database community and emerged from entity-relationship modeling. *"Conceptual modeling is about describing the semantics of software applications at a high level of abstraction"* [155]. There are three types [156] of conceptual modeling: (1) A software solution is modeled. That means a system is described in terms of structure models, behavior or functional models, and interaction and user interface models. (2) The domain in which a piece of software operates is modeled. This type of modeling is often used synonymously with domain modeling and has the same goals. A conceptual model of a domain is the abstract description of real-world concepts and their relationships. (3) The impact of the software solution on the domain is modeled. This type of conceptual modeling is less common and deals with the description of how a software solution interacts with domain entities and actors to satisfy stakeholder goals [156].

2.2.5 Domain-Specific Languages

Domain-specific languages (DSLs) are computer languages tailored to a specific application domain [69] in contrast to general purpose languages (GPLs). DSLs usually provide abstractions and notations for one particular field of interest that enable domain experts to participate in the development and configuration of software systems. At the same time they enable increase in productivity by providing the corresponding tooling to create models of the respective language and the infrastructure to process the models, generate code from them, or directly execute them.

Several criteria have to be considered for the development of domain-specific languages. DSLs can be internal/embedded, that means that they use a host language, add domain-specific language elements and restrict other language constructs. Embedded DSLs can reuse the host language infrastructure, such as compilers and interpreters, but are limited in the flexibility of how syntax is defined in the host language. In contrast, external DSLs are designed from scratch, and they are completely free in choice of constructs, syntax and notation, but the infrastructure must be developed from scratch as well. With respect to notation, textual DSLs are based on grammar definitions and accompanying parsers, whereas graphical DSLs have visual notations. Graphical DSLs are often called domain-specific modeling languages (DSMLs) [157]. Each of the categories allow hybrid approaches. More design decisions and guidelines are documented in [158] and [71]. Popular DSL examples are: Graphviz, SQL, HTML, and LATEX. Typical language workbenches for textual language development are xText[6], Jetbrains MPS[7], and MontiCore[8], and for graphical DSL development Eclipse Modeling Project (EMF+GMF)[9], Sirius[10], and MetaEdit+[11].

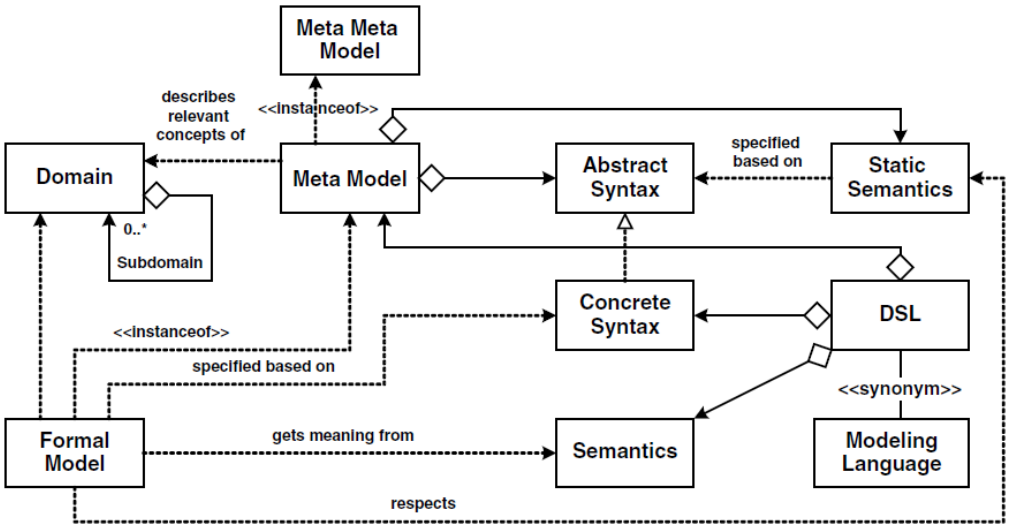


Figure 2.5: The concepts and components of domain-specific languages. Taken from [47]

Figure 2.5 shows the typical components of domain-specific languages. A DSL consists of a metamodel, a concrete syntax and semantics. A DSL is a modeling language to formulate formal models of a domain. Völter [47] sees both *abstract syntax* and *static semantics* (well-formedness rules) as part of the meta model. Other authors use metamodel and abstract syntax as synonyms [159]. Since the metamodel describes the relevant concepts of the domain, metamodel and domain model are also often used synonymously. The meta model defines relevant domain concepts and their relationships for the DSL. It is the internal data structure of a DSL that any other processing step will use. The abstract syntax can be defined using a grammar or a metamodel. DSLs can have multiple concrete syntaxes. The *concrete syntax*, either textual, graphical or hybrid, is the interface with which the DSL user interacts. *Semantics* describe the meaning of the language elements [150]. There are several approaches to semantic descriptions as described in Section 2.2.3.

Domain-specific modeling (DSM) is an approach of designing software systems with the systematic use of domain-specific languages. It includes the development of DSLs as well as the usage of DSLs. The main goal of DSM is to generate code from domain-specific models [13].

[6] https://www.eclipse.org/Xtext/
[7] https://www.jetbrains.com/mps/
[8] http://www.monticore.org/
[9] https://www.eclipse.org/modeling/
[10] http://www.eclipse.org/sirius/
[11] http://www.metacase.com/mwb/

2.2.6 Model-Driven Methods

Model-Driven Engineering (MDE). Model-driven engineering is an umbrella term for various approaches to software and systems engineering in which models play a first-order role. In general, MDE is also applied in other engineering disciplines. The emphasis is on **driven**, meaning that throughout all engineering tasks the work is centered around models: programming tasks are treated as secondary, and code shall be generated where suitable. **Model-Based Engineering** (MBE) and Model-Based Software Engineering (MBSE) are often used synonymously for MDE [4], although they have a slightly different meaning. "Model-based" is used in the sense that models still play an important role for development, but are not always used as first-class citizens[12]. Finally, there is also **Model-Driven Development** (MDD), which focuses on development tasks (e.g., requirements elicitation, analysis and design). For example, Model-Driven Architecture (MDA), which will be described in the following paragraph, can be seen as one concrete implementation of MDE [126] with a strong focus on MDD.

Model-Driven Architecture (MDA). A very important initiative for the use of metamodels, models and modeling in software engineering is the Model-Driven Architecture (MDA) of the OMG. MDA is comprised of a set of standards for the specification of models and their transformations into other models and source code. The essential characteristic of MDA is separation of concerns. It enables the specification of business concerns independent of system constraints and allows the specification of systems independent of the platforms they are supposed to run on. *"A platform is the set of resources on which a system is realized."* [160]. The most important standards of MDA are [161]:

- Meta Object Facility (MOF): A standard that enables modeling language definition.
- XML Metadata Interchange (XMI): A standard that facilitates interchange of models via XML documents.
- Unified Modeling Language (UML): A language for specifying the structure and behavior of systems.
- Common Warehouse Metamodel (CWM): A specification for data repository integration.
- Query View Transform (QVT): A mapping language to define transformations of one model to another.

MDA promotes the consequent use of these specifications for software systems development. It defines three architectural layers [160] (cf., Figure 2.6) that correspond to the level of abstraction on which the development takes place. On the most abstract level **business and domain models** are defined to capture information about real objects that are later managed by the system. In earlier versions of the MDA these models were called Computation Independent Models (CIMs). Technology-independent models of the software system reside on the **logical system models** layer (Platform Independent Models, PIMs). They describe interactions between business entities and software components independent of implementation-specific details. **Implementation models** (also called Platform Specific Models, PSMs) describe how a particular system or component is implemented using a specific technology (e.g., J2EE, .NET, Web Services).

MDA aims at automating transformations of models either from one representation to another on the same level or across levels of abstraction. Transformation specifications define a set of patterns that are applied to model elements of a source model. A transformation engine that executes the specification creates or modifies model elements in a target model. MDA transformations are specified using the QVT language [162]. Figure 2.6 illustrates the general principle of model transformation. The models in this example are: A platform independent model, which was created using a platform independent metamodel, a platform specific metamodel, and the

[12] `https://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/`

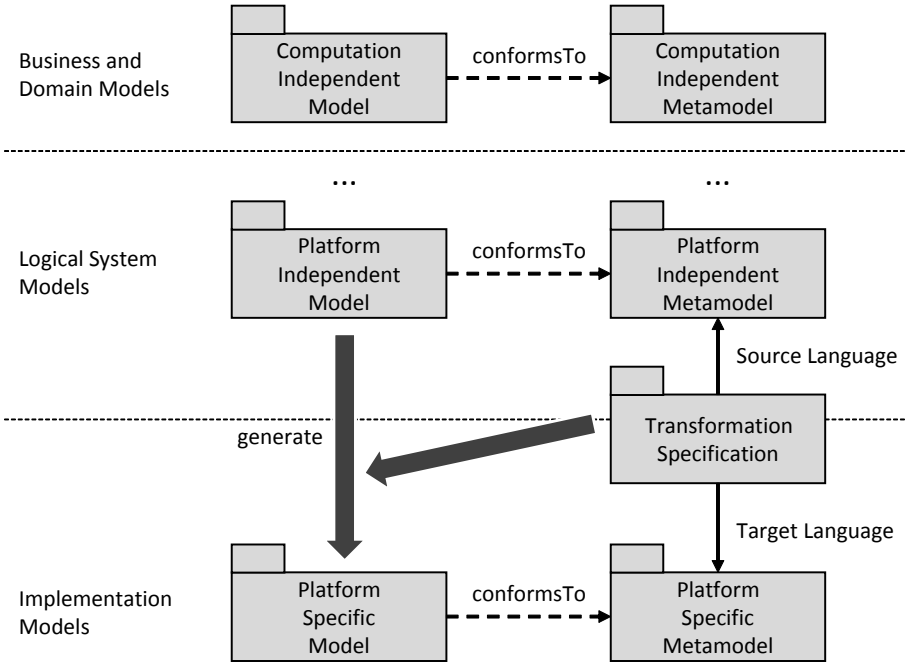


Figure 2.6: The principle of model transformation in MDA, after [161]

transformation specification, which defines a mapping on metamodel level between platform independent metamodel and platform specific metamodel. All four artifacts are used to generate the target platform specific model.

2.3 Foundations of Knowledge Bases

The field of knowledge bases deals with explicit specification and collection of factual and schematic knowledge about real-world concepts and entities and its storage in semantic databases to provide querying and reasoning services. The term knowledge base originates from the field of expert systems in AI research in the 1970s: *"One can define an expert system as a knowledge system that is able to execute a task that, if carried out by humans, requires expertise."* [163]. At that time expert systems were rule-based systems, usually consisting of three main components: a knowledge base that contains domain facts and domain rules, a reasoning or inference engine, and a user interface. Over time expert systems evolved to **Knowledge-Based Systems** (KBS) that incorporate more complex knowledge structures and more sophisticated automated reasoning services. Knowledge bases have not only been developed in the context of these systems, but have become complex systems themselves, as the following definition shows:

> *A knowledge base is a database that is used to manage knowledge. The information in the knowledge base can be accessed using logical operators to determine appropriate retrieval items. Often a knowledge base uses an ontology or data model to define its classification scheme. As applied here, the knowledge base is a highly contextualized set of classifications and relationships related to a focused core of content knowledge.*[13]

The following sections present what kinds of knowledge representations exist, what kinds of standards exist to encode knowledge, how knowledge can be stored and retrieved, and how knowledge is interlinked on the Web.

[13]From IGI Global Dictionary `https://www.igi-global.com/dictionary/knowledge-base/16266`

2.3.1 Knowledge Representation

Knowledge bases are part of the **Semantic Web** idea: describing the meaning of web documents in machine-readable form (first promoted by Tim Berners-Lee) [164] with the ultimate goal to express information in a machine-interpretable and machine-understandable form. Generally, the main intention of web documents and their markup languages is the presentation of information to humans. Semantic web documents extend normal web documents with additional information enabling machines to process the information. The Semantic Web provides a set of standards to implement semantic descriptions of documents with the goal to build models that describe the world in abstract terms, to draw meaningful conclusions from encoded knowledge, and to distribute, interlink, and reconcile knowledge on a global scale [165].

One of the prerequisites for marking up documents with additional meaning is the ability to point to already encoded knowledge. The fields of **knowledge engineering** [166] and **ontology engineering** [167] deal with acquiring and organizing knowledge in a machine-processable form to build information systems that manage the conserved knowledge and offer reasoning service on top of them. *"Knowledge engineering is not some kind of mining from the expert's head, but consists of constructing different aspect models of human knowledge."* [163].

There are several types of knowledge representation [168]: A **controlled vocabulary** is a way to consistently organize terms and expressions whose meanings are explicitly defined (to avoid homonyms). Optionally, each of the terms is assigned a preferred descriptor term (or index term) to additionally avoid synonyms. Controlled vocabularies usually do not contain any relationships between terms and are often used to index existing documents for search.

A **taxonomy** is a hierarchical classification system. It consists of controlled vocabulary terms that have relationships to each other so that a tree structure is formed. Terms can have exactly one relationship to a term located on the level above (superordinate terms) and one or many relationships to terms located on a level below (subordinated terms). Relationships on the same level are not allowed. There are variants of taxonomies that allow multiple parent terms (polyhierarchy) realized through duplication of terms. Terms in taxonomies usually represent classes (entity groups with common characteristics) [142]. Typical relationships are subclass or is-a relationships.

Semantic networks are knowledge representations using directed graph structures. They usually consist of nodes that represent concepts, and arcs that represent a conceptual relationship between two nodes [169]. Nodes have labels and can have types, usually defined by means of a taxonomy (concept hierarchy). A semantic network is not limited to a specific relationship type (e.g., linguistic, spatial, causal, or role relationships can be represented). Labels of relationships refer to the type of the relationship. Semantic networks are the most general form of knowledge representation, hence, in general, all other types of knowledge representation can be encoded as a semantic network.

The term **ontology** is used with different meanings in different contexts. A typical distinction in literature [170] is Ontology (with uppercase initial) in a philosophical sense that refers to the subject of existence and the nature and structure of things, and ontology (countable noun, *an* ontology or multiple ontologies) in a computational sense that refers to an artifact of consensual knowledge. The most common definition of an ontology was coined by Thomas Gruber: *"An ontology is an explicit specification of a conceptualization."* [171]. A conceptualization is an abstract view of reality and *"includes the objects presumed or hypothesized to exist in the world and their interrelationships"* [172]. The definition by Gruber was extended by several authors that the specification should be *formal* and that the conceptualization should be *shared*. A formal specification is a declarative and machine-readable representation. A shared conceptualization expects an agreement between a number of individuals or agents. An ontology usually provides a vocabulary to name the concepts, logical statements that describe the relationships between the concepts, and rules that describe how new knowledge can be deduced.

2.3.2 Representation Languages

The formal specification of knowledge requires appropriate languages. In this section, we introduce the most important open standards that enable encoding, exchange and interoperability of data

and knowledge. The World Wide Web Consortium[14] (W3C) is the main driver for realizing the Semantic Web idea. It has defined a number of specifications that are arranged in a layered architecture (see Figure 2.7 for a current version), called the **Semantic Web stack**.

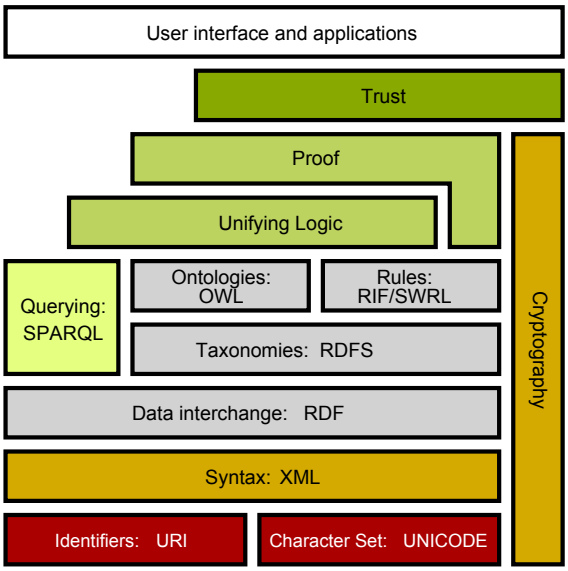


Figure 2.7: The Semantic Web stack from Wikipedia Commons[15]

The Semantic Web stack is based on existing web technologies, namely Unicode, URIs, and XML languages. **Unicode** is a standard for encoding and representing characters and text of different writing systems. **Universal Resource Identifier (URI)**[16] allow to specify globally unique names for resources. URI is the more general definition of the historically separated URL (the address where a resource is located) and URN (the unique name of a resource). Most of the Semantic Web languages rely on the **Extensible Markup Language (XML)**[17] to syntactically encode (and serialize) data in a structured way. The main concept of XML is the markup of content with tags that can have attributes. XML allows to use arbitrary tags, there is no fixed vocabulary. Consequently, XML can be used to define other markup languages. A fixed vocabulary for XML documents can be specified using an **XML Schema**. It defines allowed tag names, attribute names and how they can be nested and used. Using appropriate parsers, the well-formedness of an XML document can be validated against its schema.

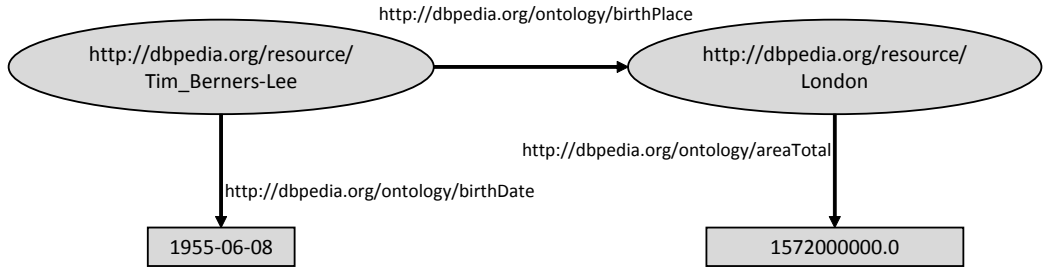
The most important standard of the Semantic Web stack is the **Resource Description Framework (RDF)**[18] that extends existing web languages. RDF has its origins in the metadata community, hence the main intention was the description of additional data for documents and other data:

> *"RDF as a W3C recommendation provides a data model for annotations in the Semantic Web. [...] RDF annotates Web resources in terms of named properties." [167]*

Annotations are realized as RDF statements in the form of SUBJECT PREDICATE OBJECT (RDF triples). The subject is a web resource about which a statement is made. The predicate (often called property) describes the kind of relationship. The object is either another web resource or a data value (usually called literal). Subjects, predicates and objects have to be named with URIs except for literal values and blank nodes (anonymous resources, only valid within a document).

[14] `https://www.w3.org/`
[15] Picture obtained from `https://commons.wikimedia.org/wiki/File:Semantic_web_stack.svg`
[16] `https://www.w3.org/Addressing/URL/URI_Overview.html`
[17] `https://www.w3.org/XML/`
[18] `https://www.w3.org/standards/techs/rdf`

RDF is a graph data model, hence, a set of triples forms a directed labeled graph. Figure 2.8 shows such a graph and the corresponding RDF triples. The example is taken from the DBpedia knowledge base[19]. It encodes three facts about the web resources of Tim Berners-Lee and London and demonstrates the linking of web resources and the annotation of web resources with named properties and their values.



SUBJECT	PREDICATE	OBJECT
http://dbpedia.org/resource/Tim_Berners-Lee	http://dbpedia.org/ontology/birthPlace	http://dbpedia.org/resource/London
http://dbpedia.org/resource/Tim_Berners-Lee	http://dbpedia.org/ontology/birthDate	1955-06-08
http://dbpedia.org/resource/London	http://dbpedia.org/ontology/areaTotal	1572000000.0

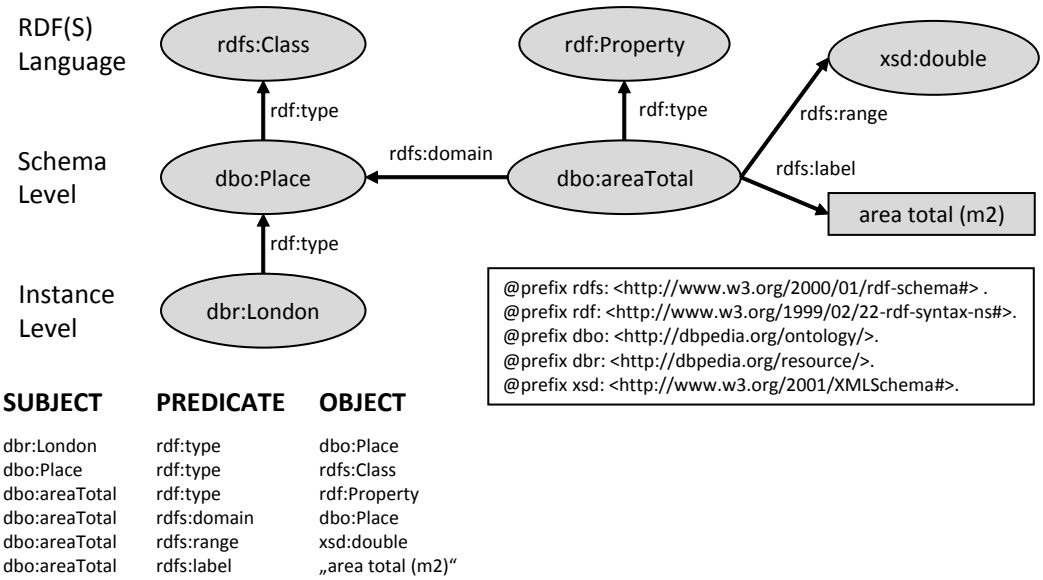
Figure 2.8: Examples of RDF statements in triple syntax and the corresponding RDF graph

The RDF data model itself does not yet provide meaning for web resources, but the annotations point to resources under which pre-agreed knowledge is encoded in terms of ontologies or other knowledge representations (see previous section). Nevertheless, URIs can be introduced on demand, in case vocabularies for certain fields are not yet available. In order to build new vocabularies the Semantic Web stack offers the **RDF Schema (RDFS)**[20] language. RDFS allows the assignment of types (the kinds of things) to resources. These types are defined with the help of RDFS classes. The distinction between classes (a set of resources that share the same characteristics) and individuals (often called instances) allows the specification of terminological schema knowledge.

> *"RDFS [...] does not introduce a topic-specific vocabulary for particular application domains [...]. Rather, the intention of RDFS is to provide generic language constructs by means of which a user-defined vocabulary can be semantically characterized." [165]*

This semantic characterization is achieved by describing properties of the vocabulary terms that apply in a domain of interest. Figure 2.9 shows an example of a class definition and associated properties as graph representation, including the corresponding triples using a prefix notation (name spaces). It defines a *Place* class and declares that the individual *London* belongs to that class using the type relationship. The example also defines the property *areaTotal*, that was used in the previous example. It is valid for all instances of *Place* (domain) and can hold *Double* values (range). The definition includes a textual name *area total (m2)* for the property. It seems to indicate that the measurement unit is square meters, but this textual description cannot be evaluated by machines without any further datatype definitions. Other important relationships of RDFS are the **subClassOf** and **subPropertyOf** properties to model subsumption hierarchies between classes and properties, respectively. As it can be seen in the example, RDFS statements are encoded as simple RDF triples. Thus, they constitute valid RDF documents. Vocabularies defined with RDFS are usually called **lightweight** ontologies [165]. On the one hand, they usually constitute taxonomies with simple binary property definitions. On the other hand, RDFS has limited expressiveness, e.g., it is not possible to model equivalence, negation, property restrictions, and cardinalities.

[19] `http://dbpedia.org/` – Triples were retrieved on May 9, 2019
[20] `https://www.w3.org/TR/rdf-schema/`



SUBJECT	PREDICATE	OBJECT
dbr:London	rdf:type	dbo:Place
dbo:Place	rdf:type	rdfs:Class
dbo:areaTotal	rdf:type	rdf:Property
dbo:areaTotal	rdfs:domain	dbo:Place
dbo:areaTotal	rdfs:range	xsd:double
dbo:areaTotal	rdfs:label	„area total (m2)“

Figure 2.9: Examples of class and property definitions using RDF Schema

The **Web Ontology Language (OWL)**[21] builds upon RDF and RDFS and provides more expressive power to model complex knowledge and relationships. OWL has a long history in the area of artificial intelligence. It emerged from the DAML+OIL languages [173] and became a W3C recommendation for the modeling of ontologies in 2004. OWL is based on formal logic (the language is mapped to a description logic [167]) to enable the derivation of implicit knowledge, called reasoning. The development of OWL always dealt with the trade-off between expressive power and efficient reasoning support (the more complex a language is, the more computationally inefficient the reasoning becomes). For that reason OWL version 1 comprises three different dialects: OWL Lite, OWL DL, and OWL Full.

OWL Full provides maximum expressiveness with no syntactic restrictions and allows combination of language primitives of OWL, RDF, and RDFS. It is interpreted with RDF-based semantics, hence, OWL Full is an undecidable language. That means, there are no guarantees that reasoning computation will end in finite time.

OWL DL is the description logic equivalent and still offers the complete language features. In order to guarantee decidability with NEXPTIME complexity it restricts the usage of the constructs from OWL and RDF (e.g., no class membership in other classes, and restrictions of functional and transitive properties) and uses the direct semantics of OWL.

OWL Lite is a reduced subset of OWL DL, thus, also a decidable language, which reduces worst-case computational complexity to EXPTIME. For example, it does not provide disjoint or complement constructs, and restricts cardinality to be either 0 or 1.

The most important elements of OWL are classes, individuals, datatype properties, object properties, property restrictions, and special properties (e.g., functional, transitive). Every OWL ontology comes with a header that contains metadata about the ontology, such as namespaces, version and author information. Figure 2.10 shows an example ontology definition using OWL together with its RDF/XML representation in Listing 2.1. The ontology contains four class definitions, namely *Person*, *Student*, *Professor*, and *FacultyMember*. Professor is a subclasss of FacultyMember and Student is defined as disjoint with Professor. There is one object property *hasAffiliation* valid for FacultyMember. The ontology contains two individuals, an instance of Professor *(ProfHaraldSack)* and an instance of Person *(HaraldSack)*. Both are connected with the sameAs-relationship.

[21] `https://www.w3.org/OWL/`

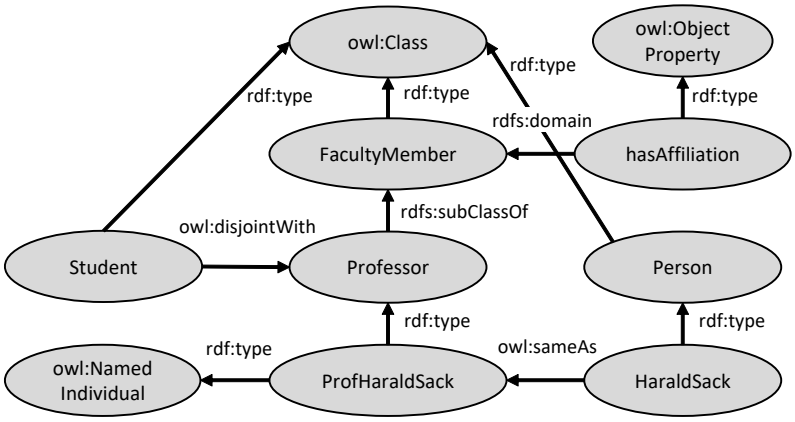


Figure 2.10: Examples of typical ontology definition statements with OWL. Examples partially taken from [165]

```
<rdf:RDF xmlns="http://www.example.org/"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
   <owl:Ontology rdf:about="">
      <rdfs:label>Example Ontology</rdfs:label>
      <dc:creator>Henning Agt-Rickauer</dc:creator>
   </owl:Ontology>
   <owl:Class rdf:about="FacultyMember" />
   <owl:Class rdf:about="Professor">
      <rdfs:subClassOf rdf:resource="FacultyMember" />
   </owl:Class>
   <owl:Class rdf:about="Student">
      <owl:disjointWith rdf:resource="Professor" />
   </owl:Class>
   <owl:Class rdf:about="Person" />
   <owl:ObjectProperty rdf:about="hasAffiliation">
      <rdfs:domain rdf:resource="FacultyMember" />
   </owl:ObjectProperty>
   <owl:NamedIndividual rdf:about="ProfHaraldSack">
      <rdf:type rdf:resource="Professor"/>
   </owl:NamedIndividual>
   <owl:NamedIndividual rdf:about="HaraldSack">
      <rdf:type rdf:resource="Person"/>
      <owl:sameAs rdf:resource="ProfHaraldSack" />
   </owl:NamedIndividual>
</rdf:RDF>
```

Listing 2.1: Corresponding RDF/XML representation of the OWL example. Examples partially taken from [165]

OWL 2 has become a W3C standard[22] in 2009. It is backward-compatible to OWL 1, that means OWL 1 ontologies expressed with RDF syntax are valid OWL 2 ontologies. OWL 2 comes with two types of dialects: OWL 2 Full and OWL 2 DL, which correspond to the OWL 1 dialects. Instead of OWL Lite OWL 2 profiles are available. There are three profiles[23]: OWL 2 EL, OWL 2 QL, and OWL 2 RL. OWL 2 EL is designed for very large ontologies with complex structures and guarantees polynomial time (PTIME) for reasoning on and querying of schemata and data with respect to the size of the ontology. OWL 2 QL was developed to achieve interoperability with relational database management systems with the goal of querying large volumes of instance data in LOGSPACE time. The expressive power allows to model main features of ER and UML class models. Finally, OWL 2 RL is designed for applications that require scalable reasoning using rule-based technologies. Implementations of this profile operate with RDF triples and can process any kind of OWL 2 ontology. It guarantees polynomial time reasoning, but only for correct answers (sound reasoning) that may not be complete.

[22] https://www.w3.org/TR/owl2-overview/
[23] https://www.w3.org/TR/owl2-profiles/

Other new features of OWL 2 are: A functional syntax and an updated Manchester syntax, syntactic improvements to reduce awkward expressions of certain constructs in OWL 1 (e.g., disjoint union, negative assertions), new property features (e.g., cardinality restrictions, (ir)reflexive and asymmetric object properties, keys), extended datatype definitions and restrictions, metamodeling via punning, and the support of IRIs.

In the remainder of this section we will only provide a very brief overview about the rest of the Semantic Web stack (cf., Figure 2.7), except for SPARQL, which is introduced in more detail in the next section. A family of languages in the stack refers to rules. *"A rule could be any statement which says that a certain conclusion must be valid whenever a certain premise is satisfied"* [165]. Rules are a supplement to OWL and RDFS to express knowledge and conditions that cannot be formulated with these languages. Rules are mainly based on first order logic (FOL). The two most widely known rule formalisms for Semantic Web are the Rule Interchange Format (RIF)[24] and the Semantic Web Rule Language (SWRL)[25]. RIF is a specification of a format for interchange of rules between different rule-based systems (e.g., FLORA-2, KAON2, Prolog) on the Semantic Web. SWRL is a combination of OWL DL and RuleML (Rule Markup Language, an initiative to standardize inference rules). From a semantics point of view, it combines description logic with Datalog rules. SWRL allows to express rules in terms of OWL classes, properties, and individuals. That means, symbols in rules can be OWL identifiers.

The "Unifying Logic" layer in the Semantic Web stack refers to the goal of combining the lower layers in a common language to execute queries and evaluate rules. The objectives of the Proof and Trust layers are to provide a mechanism for validating the correctness of information and the source of information. Finally, the Cryptography layer refers to technologies, such as secure protocols, identity verification, and access control, similar to those available on the common Web.

2.3.3 Knowledge Management

In this section we introduce concepts, languages and tools for storage, retrieval and management of RDF/OWL data and ontologies. The storage of RDF data deals with the preservation of RDF documents, triples and schema information. The retrieval of RDF data is responsible for efficient access to the stored RDF data and the possibility to query it according to desired aspects. Management includes services built on top of the storage and retrieval system, such as content preparation and reasoning.

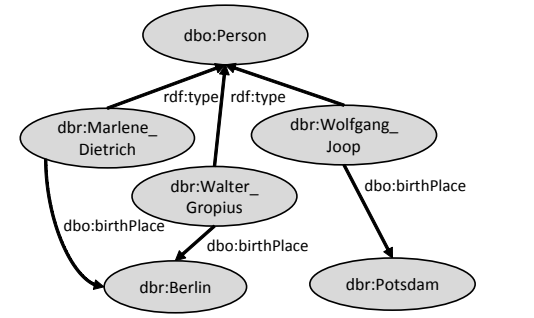
*"An **RDF data store** is a special database system built for the storage and retrieval of RDF statements."* [174]. RDF data store, RDF store and triple store are often used synonymously. A triple store usually follows a layered architecture and consists of a **repository** that physically stores the RDF data in files, databases, or main memory. Access to the repository is encapsulated with well-defined generic interfaces, so that it is possible to replace the repository engine. The data model of RDF and RDF schema requires either to transform the RDF graph structure into appropriate representations (e.g., database tables) or to directly store the data as a graph (e.g., in a node-edge or in an object-relational representation). Apart from the storage functionality, an RDF store offers APIs to add and remove RDF statements, export and import data, and interfaces for administration. State of the art RDF stores are: 4store[26], Blazegraph[27], Fuseki[28], and Virtuoso[29].

The second most important functionality of a triple store is a **query** component that allows to retrieve and filter RDF statements. In general, a triple store can implement multiple query interfaces to offer access to the storage layer with multiple query languages (e.g., RQL, RDQL, SPARQL). As **SPARQL** became a W3C recommendation in 2008, this query language is the standard access method to stored RDF data for most of the triple stores. SPARQL allows to extract values and (partial) statements from RDF graphs. It is also capable to extract RDF

[24] `https://www.w3.org/TR/rif-overview/`
[25] `https://www.w3.org/Submission/SWRL/`
[26] `https://github.com/4store/4store`
[27] `https://www.blazegraph.com/`
[28] `https://jena.apache.org/documentation/fuseki2/`
[29] `https://virtuoso.openlinksw.com/`

subgraphs, to construct new RDF graphs [142], to perform joins of separate RDF graphs, and to transform instances based on one vocabulary to another.

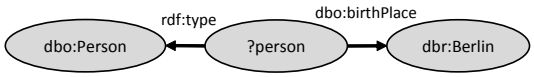
A **SPARQL query** is composed of the following parts: An optional prefix declaration for abbreviating URIs (PREFIX), a mandatory query result clause (SELECT), an optional dataset clause (FROM), a mandatory query pattern (WHERE), and one or more optional query modifiers (e.g., ORDER BY, GROUP BY). Figure 2.11 shows a query example using the mandatory SELECT and WHERE clauses. The dataset is an excerpt of the English DBpedia containing three persons and their birth places. The query asks for nodes in the graph that have a *dbo:Person* type and a link to *dbr:Berlin* via the *dbo:birthPlace* property. The query contains one variable and two triple patterns, one for the type relationship and one for the birth place relationship. They match all RDF statements that have the same structure. The result set in turtle notation contains the two matching persons that were born in Berlin.



(a) RDF dataset

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?person
WHERE {
  ?person rdf:type dbo:Person .
  ?person dbo:birthPlace dbr:Berlin .
}
```

(b) SPARQL query



(c) Triple patterns

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix res: <http://www.w3.org/2005/sparql-results#> .
_:ResultSet2053 rdf:type res:ResultSet .
_:ResultSet2053 res:resultVariable "person" .
_:ResultSet2053 res:solution _:ResultSet2053r0 .
_:ResultSet2053r0 res:binding _:ResultSet2053r0c0 .
_:ResultSet2053r0c0 res:variable "person" .
_:ResultSet2053r0c0 res:value dbr:Marlene_Dietrich .
_:ResultSet2053 res:solution _:ResultSet2053r1 .
_:ResultSet2053r1 res:binding _:ResultSet2053r1c0 .
_:ResultSet2053r1c0 res:variable "person" .
_:ResultSet2053r1c0 res:value dbr:Walter_Gropius .
```

(d) SPARQL result set

Figure 2.11: Examples of an RDF dataset, a SPARQL query, the corresponding triple pattern, and the result set

A **SPARQL endpoint** is an interface to the RDF store that accepts queries and returns result sets, either for requests from humans or from applications [174]. It implements the SPARQL protocol[30] to process SPARQL queries submitted using HTTP, to parse input formats, and to return results in the appropriate formats (e.g., tables in XML or HTML format). A SPARQL endpoint usually offers a web interface for users to execute queries and browse results. Since version 1.1 the SPARQL query language not only supports the retrieval of RDF data but also allows to add/delete triples to/from the RDF store. The query syntax was extended by optional INSERT and DELETE clauses that contain the respective RDF statements.

An RDF store usually incorporates an **inference engine**. The task of an inference engine is the deduction of additional facts that are not explicitly defined in the existing knowledge base. A typical use case for RDF data is the application of RDF(S) entailment rules[31] to the stored data. The most widely used entailment rules are the inference of the transitive closure for subClassOf and subPropertyOf relationships and the inference of class memberships using properties and their domain/range restrictions [167]. The two most popular methods to perform the deduction are: Inference in advance using forward-chaining (starting from known facts and applying rules until a goal is reached) and inference at query runtime using backward-chaining (starting from a list of goals and checking whether there are facts available that support the goals) [175]. Besides inference engines for RDF data there are several well developed **reasoners** (e.g., FaCT++[32],

[30] `https://www.w3.org/TR/sparql11-protocol/`
[31] `https://www.w3.org/TR/rdf11-mt/`
[32] `http://owl.cs.manchester.ac.uk/tools/fact/`

Racer[33], Pellet[34]) for decidable description logic in order to perform deduction on OWL ontologies. Most RDF stores and ontology tools integrate one or more of these reasoners.

There exist several **ontology engineering tools** that support the development of ontologies. They offer graphical user interfaces for editing, browsing, documenting, and visualizing ontologies, as well as export/import features for common languages and formats. Protégé[35] is the most popular ontology editor. It was developed by the Stanford Center for Biomedical Informatics and is a free and open-source tool. Protégé has been used for knowledge acquisition and domain ontology building for many years [167]. With its plug-in architecture and customizable user interface it can be easily tailored to specific needs of various ontology development tasks. Other popular ontology engineering tools are: NeOn toolkit[36], OWLGrEd[37], TopBraid Composer[38] (commercial tool), and OntoStudio[39] (commercial tool).

2.3.4 Linked Data

In the early years of the Semantic Web idea, publication of semantic data often resulted in large isolated RDF dumps that were put on the Web for download [176] (e.g., the OpenCyc common sense ontology [112], the GALEN medical terminology, the WordNet lexical database [108]). Although these initiatives were very valuable and still belong to the most frequently used resources in the Semantic Web and computational linguistics community, they could only be processed offline, they were not interlinked, and it was difficult to retrieve single elements or excerpts of the RDF documents.

Friend of a Friend (FOAF)[40] was the first project that applied interconnection of RDF documents by linking FOAF profiles to other friends' profiles using a *knows* relationship. The project was the first step towards a *Web of Linked Data*. Publication of more and more isolated datasets continued, so that Tim Berners-Lee proposed the **Linked Data principles** [177], a guideline of how to publish Linked Data *"in such a way that it is machine readable, its meaning is explicitly defined, it is linked to other external datasets, and it can in turn be linked to from external datasets as well."* [174]. Over time the principles evolved to more detailed specifications (e.g., how to publish vocabularies [178]). The four Linked Data principles, which enable discovering and consuming semantic data on the web by both humans and machines, are [177]:

1. *Use URIs as names for things.*
2. *Use HTTP URIs so that people can look up those names.*
3. *When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).*
4. *Include links to other URIs, so that they can discover more things.*

The first principle refers to RDF's basic rule of using **Unique Identifiers** for everything published as Linked Data. The second principle is the so-called **Dereferencing**, the ability to access the URI over HTTP using globally accessible domains. The third principle, when accessing a URI with a client, recommends providing structured RDF data to return additional information about an object identified by a URI. This includes **Content Negotiation**, the process of providing different representations of the same RDF data depending on the request. For the same URI, for example, a user using a conventional Web browser receives an HTML page rendered from the RDF data, and a machine using an RDF client receives an RDF document in exactly the requested format. The latter principle advocates the use of **Interlinking** and allows us to examine more information in other datasets and to avoid duplicate information (for example, reusing a type of a dataset in another dataset or stating that two things in different datasets are the same).

[33] https://www.ifis.uni-luebeck.de/~moeller/racer/
[34] https://github.com/stardog-union/pellet/
[35] https://protege.stanford.edu/
[36] http://www.neon-toolkit.org/
[37] http://owlgred.lumii.lv/
[38] https://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/
[39] http://www.semafora-systems.com/en/products/ontostudio/
[40] http://www.foaf-project.org/

Publishing Linked Data following these principles has created a web of data that has grown very rapidly within the last 10 years. This has been reinforced by several efforts to automatically create large knowledge bases using semi-structured data and textual documents on the Web. One of the first successful projects is **DBpedia** [103]. *"DBpedia is a crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects. This structured information resembles an open knowledge graph (OKG) which is available for everyone on the Web."*[41]. DBpedia extracts existing structured information from Wikipedia pages. Extracted information is converted into RDF documents that form a large RDF record. The dataset is provided as Linked Data using an instance of Virtuoso Server[42] and can be queried using a public SPARQL endpoint[43].

Because of its popularity, DBpedia has become the focal point of the **Linked Open Data Cloud**[44]. The LOD Cloud is a community project that collects metadata of Linked Data datasets and provides analysis of the content, its connections, and interactive entry points into the datasets. A visualization of the datasets is shown in Figure 2.12. It contains more than 1200 records[45], which are divided into different domains.

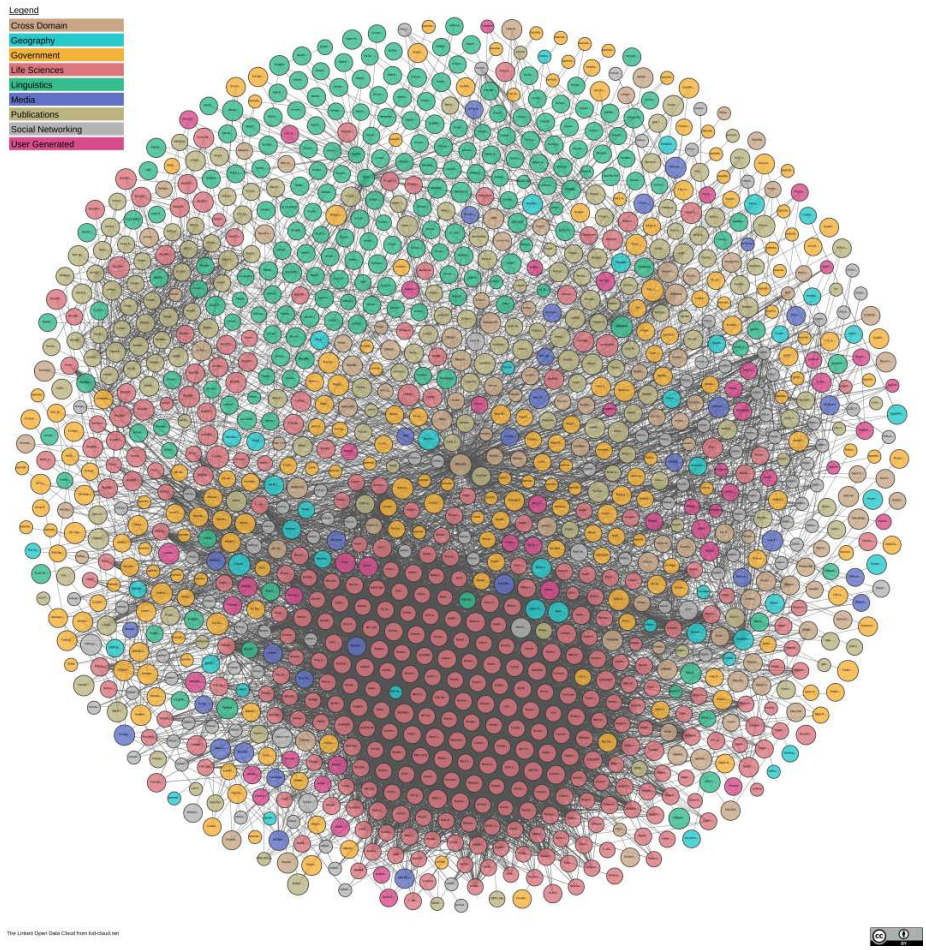


Figure 2.12: Linked Open Data Cloud diagram as of March 2019, by John P. McCrae, from http://lod-cloud.net/

[41] `https://wiki.dbpedia.org/about/`
[42] `https://virtuoso.openlinksw.com/`
[43] `https://dbpedia.org/sparql/`
[44] `https://lod-cloud.net/`
[45] As of March 29, 2019

2.4 Foundations of Information Extraction

Information extraction is the process of creating structured facts from unstructured data sources [114]. This thesis utilizes methods from this field to create new sources of knowledge. Information extraction itself is associated with several other research areas, such as: computational linguistics [115], artificial intelligence and information retrieval [116]. This section introduces the most important terms, concepts, and methods of information extraction that are relevant to this thesis. Marie-Francine Moens [116] defines information extraction as follows:

> *Information extraction is the identification, and consequent or concurrent classification and structuring into semantic classes, of specific information found in unstructured data sources, such as natural language text, making the information more suitable for information processing tasks.*

Unstructured data includes all types of text documents and media files (audio, video, image) that need to be processed to make the contained information interpretable. Structuring into semantic classes means that the raw information contained in the linguistic structure of sentences and their formation is organized into several linguistic models at different levels of abstraction, or the visual and audible content is classified accordingly. These models allow further analysis at syntactic, semantic, and pragmatic levels to interpret the intended meaning.

2.4.1 Basic Computational Linguistics Methods

Information extraction, in most cases, involves natural language processing, as the majority of unstructured data is text [179]. Several computational linguistics methods have to be applied to access the information that is expressed by natural language text. In the following we describe the most important linguistic preprocessing techniques.

The first step of processing a natural language document or corpus is checking the data for corrupt files, checking the character encoding and if necessary, the removal of markup such as XML or HTML tags (**boilerplate detection**) in order to obtain the raw text for further processing. The division of text into sections or paragraphs is called **text segmentation** and performed in case the data or the extraction task requires it. The content of a corpus may originate from arbitrary sources, such as the Web, hence a **language identification** may be required.

Tokenization splits the text into words, numbers, punctuation, so-called tokens. In most European languages tokens are usually separated by white space (space-delimited languages). Unsegmented languages (e.g., Chinese, Japanese) require additional dictionaries or morphological analysis. One of the main tasks of tokenization is to correctly identify whether punctuation or hyphenation should be considered as part of the word or not (e.g., periods for abbreviations, periods as decimal point, and periods to mark the end of the sentence). Closely connected to tokenization is the preprocessing step **sentence segmentation**, *"the process of determining the longer processing units consisting of one or more words"* [180]. Figure 2.13 shows an example piece of text that is tokenized and segmented[46].

Lexical analysis operates on word level and deals with decomposing words into their parts, morphological processing, and eliminating word variants (for example, $\{organizes, organizing, organized\} \rightarrow organize$). Morphological variants are subsumed by their **lemma**, the invariant root form of a word. While lemmatization is the more complex reduction of inflectional forms, **stemming** is a pure heuristic process that cuts off the end of words [181]. These procedures are often used in the context of **text normalization**, which in general has the goal to turn tokens and text into a canonical form (e.g., lowercasing, expanding abbreviations, removing stopwords).

A very important step in natural language processing is **part of speech tagging** (POS tagging). It operates on sentence level and determines the corresponding lexical category (e.g., noun, adjective) for each token (words or tokens are tagged, respectively). The sets of possible tags vary a lot between different POS tagging systems and languages. In general there are three universal

[46]The sentence was processed with `http://text-processing.com/demo/`

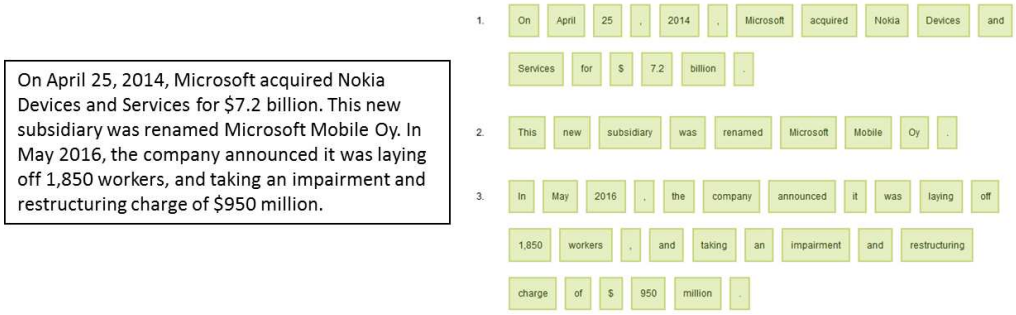


Figure 2.13: Example text from Wikipedia's page on Microsoft and its segmented and tokenized form, processed with the Treebank Word Tokenizer of the Python NLTK

primary parts of speech: noun, verb, and adjective [180]. Further categories are subcategories of the primary word classes that reflect tense or number and categories of secondary importance (e.g., determiner, conjunction). The most important **tagsets** are: Penn Treebank and Wall Street Journal (WSJ) tagset (48 tags) [182] and the Brown Corpus tags (87 tags)[47] for the English language as well as the STTS tagset and modifications (54 tags)[48] for German. In 2012, Petrov et al. [183] proposed a universal tagset consisting of only 12 categories covering the most frequent part of speech for 22 languages. Figure 2.14 shows the previous example text tagged with the Stanford Tagger [184] using the Penn Treebank tagset compared to tags from the Google Cloud Natural Language Services[49]. The main challenges in POS tagging are ambiguous words that may have different parts of speech depending on the context or their usage (e.g., *patient* as an adjective and *patient* as a noun), and how to process unknown words that cannot be handled by handcrafted rules or were not present in the training data for machine learning based models. Nevertheless, POS tagging is a very well-studied field of research. State-of-the-art taggers achieve an accuracy of about 97%.

(a) Original Text

On April 25, 2014, Microsoft acquired Nokia Devices and Services for $7.2 billion. This new subsidiary was renamed MicrosoftMobile Oy. In May 2016, the company announced it was laying off 1,850 workers, and taking an impairment and restructuring charge of $950 million.

(b) Tagged with Stanford Tagger using Penn Treebank Tagset

On/**IN** April/**NNP** 25/**CD** ,/**,** 2014/**CD** ,/**,** Microsoft/**NNP** acquired/**VBD** Nokia/**NNP** Devices/**NNPS** and/**CC** Services/**NNPS** for/**IN** $/**$** 7.2/**CD** billion/**CD** ./**.** This/**DT** new/**JJ** subsidiary/**NN** was/**VBD** renamed/**VBN** Microsoft/**NNP** Mobile/**NNP** Oy/**NNP** ./**.** In/**IN** May/**NNP** 2016/**CD** ,/**,** the/**DT** company/**NN** announced/**VBD** it/**PRP** was/**VBD** laying/**VBG** off/**RP** 1,850/**CD** workers/**NNS** ,/**,** and/**CC** taking/**VBG** an/**DT** impairment/**NN** and/**CC** restructuring/**NN** charge/**NN** of/**IN** $/**$** 950/**CD** million/**CD** ./**.**

(c) Tagged with Google Cloud Natural Language Services using the Universal Tagset

On/**ADP** April/**NOUN** 25/**NUM** ,/**PUNCT** 2014/**NUM** ,/**PUNCT** Microsoft/**NOUN** acquired/**VERB** Nokia/**NOUN** Devices/**NOUN** and/**CONJ** Services/**NOUN** for/**ADP** $7.2/**NUM** billion/**NUM** ./**PUNCT** This/**DET** new/**ADJ** subsidiary/**NOUN** was/**VERB** renamed/**VERB** Microsoft/**NOUN** Mobile/**NOUN** Oy/**NOUN** ./**PUNCT** In/**ADP** May/**NOUN** 2016/**NUM** ,/**PUNCT** the/**DET** company/**NOUN** announced/**VERB** it/**PRON** was/**VERB** laying/**VERB** off/**PRT** 1,850/**NUM** workers/**NOUN** ,/**PUNCT** and/**CONJ** taking/**VERB** an/**DET** impairment/**NOUN** and/**CONJ** restructuring/**NOUN** charge/**NOUN** of/**ADP** $950/**NUM** million/**NUM** ./**PUNCT**

Figure 2.14: Comparison of Stanford/Penn Treebank and Google part-of-speech tags for a sample text

In a linguistic processing pipeline, the POS tags are usually the starting point for **syntactic analysis**. Syntactic analysis deals with the grammatical structure of a sentence by determining relations between words. In general, there are two main approaches to syntactic natural language analysis: grammar-driven parsing and statistical **parsing** [180]. The former is mainly based on context-free grammars and their extensions (e.g., Head-driven Phrase-Structure Grammars (HPSG)), and tries to match a grammar against a given sentence. Statistical approaches also use

[47] `http://www.helsinki.fi/varieng/CoRD/corpora/BROWN/tags.html`
[48] `http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/GermanTagsets.en.html`
[49] `https://cloud.google.com/natural-language/?hl=de`

grammars but try to induce grammars based on probabilistic models (e.g., Probabilistic Context-Free Grammars (PCFG) are often used). Output of parsing is either a hierarchical structure of the input sentence, called a syntax tree or **constituent structure**, which is recursive decomposition of a sentence (e.g., consisting of noun phrases, verb phrases, etc.), or a **dependency structure**, which establishes binary relationships between words, and labels words according to their functional role (e.g., subject, object). Figure 2.15 shows such a dependency structure for a sample sentence (orange: functional roles, green: dependencies).

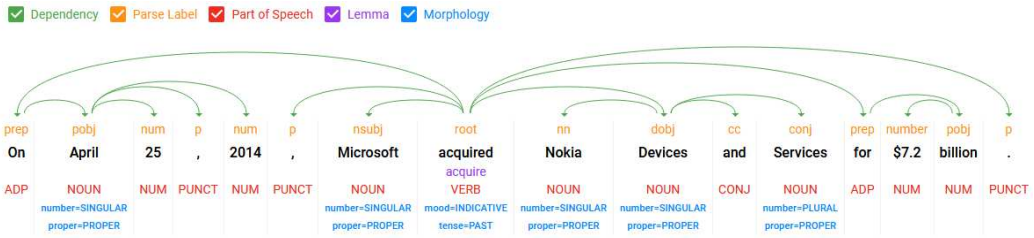


Figure 2.15: Dependency structure for a sample sentence obtained from the Google Cloud Natural Language Services

The methods described above are among **deep parsing** systems that are able to capture the full grammatical structure of sentences including long-distance relationships. As the application of these grammars is computationally intensive, they may be insufficient for large amounts of data. Instead, **shallow parsing** is often used, a rudimentary syntactic analysis, also called chunking. It tries to identify boundaries between basic elements by separating the sentence into chunks (usually phrases without any roles).

Finally, when it comes to analyzing larger text collections, **probabilistic language models** are often used to examine sequences of words and their context. These language models are mainly used to predict a possible next word for a given sequence of words. This is a fundamental task in speech and handwriting recognition, and it is also used for spelling correction. These models are mainly based on **N-gram** analysis. In the context of corpus analysis, an N-gram is a sequence of n consecutive words. The frequency of an N-gram is determined by counting all its occurrences in a text collection. If for example a trigram model is used, all occurrences of three consecutive words are counted in a corpus and for a 2-word input sequence the probability of the third word can be estimated using, for example, a maximum likelihood estimation [185]. In practice, it is common to use trigram models, sometimes 4-gram or 5-gram models, but the longer the context the more sparse the data becomes (e.g., simply counting 6-grams will result in fewer counts and too many distinct N-grams).

2.4.2 Term Extraction

In this section we introduce methods for extracting interesting terms from natural language text. The notion of **term** is used quite differently in different areas of research. In the context of this thesis we refer to a definition by the Merriam Webster dictionary[50], which says *"a word or expression that has a precise meaning in some uses or is peculiar to a science, art, profession, or subject"*. Particularly, we want to highlight that "term" refers to both words and phrases. By "interesting terms" is meant that these terms carry a meaning that stands out from other terms. There are a number of concepts that closely relate to the notion of term, such as key terms, collocations and multiword expressions (MWE), which will be described in more detail below.

*"**Key terms** [...] are sets of significant terms in a text document that give high-level description of its content for readers."* [186]. **Key term extraction** or keyword extraction has its roots in information retrieval to select keywords in document collections that are used to retrieve the

[50] `https://www.merriam-webster.com/dictionary/term`

most relevant documents according to a keyword query. The baseline model in this area is **TF-IDF** [187], a composite weight of the *term frequency*, the occurrences of a term in a document, and the *inverse document frequency*, the logarithm of the total number of documents in a collection divided by the document frequency (in how many documents of the collection the term occurred). As this model is not very robust, several extensions of it exist to improve finding and ranking candidate terms. They combine other weights, such as distance/position of a term in a document or keyphrase-frequency. Other approaches use external sources, such as Wikipedia, to determine links between candidate terms and their importance. Several methods exist that build a graph model of the document, and use graph-based ranking algorithms [188] or community detection [186]. If these methods are applied in a domain-specific context, it is often called the extraction of **technical terminology**.

*"A **collocation** is a word combination whose semantic and/or syntactic properties cannot be fully predicted from those of its components, and which therefore has to be listed in a lexicon."* [189]. This refers to the limited compositionality of collocations (some meaning is added). They are often idiomatic (but they need not be) or have a specialized meaning, thus these fixed expressions are used more frequently than other combinations. Collocations include a broader range of combinations (e.g, compounds and phrasal verbs), for example *disk drive* may also be a key term, but the collocation *make up* will most probably not be regarded as a key term (which are nouns and noun phrases, normally). A common method to find collocations is counting sequences of two or more adjacent words (bigrams or N-grams in general) and applying part-of-speech filters [190].

Multiword expression (MWE) is an umbrella term for several syntactic categories, to some extent overlapping with key terms and collocations. A linguistic definition of MWE is: *"Multiword expressions (MWEs) are lexical items that: (a) can be decomposed into multiple lexemes; and (b) display lexical, syntactic, semantic, pragmatic and/or statistical idiomaticity"* [180]. Multiple lexemes refer to whitespace-separated expressions, hence it is common that fused words (e.g., kickboxing) are not considered as MWEs. While collocations do not necessarily imply idiomaticity, it is required for multiword expressions. Idiomaticity, sometimes also called idiosyncrasy, means that these expressions are peculiar in their use and deviate from the basic properties of their components (e.g., *bull market*). They are treated as a single unit and may be a continuous or discontinuous sequence of words. This is also a difference in comparison to collocations (usually treated as consecutive words), but there exist research works that both analyze collocations in a discontinuous manner and multiword expressions only in a continuous manner. The different types of idiomaticity refer to how they are lexically or syntactically constructed (e.g., *ad hoc*, parts are not in a lexicon), that there is a mismatch between the semantics of the parts and the whole (e.g., *kick the bucket*), in which situations they are used (e.g., *good morning*), and how often certain expressions are used (e.g., dialect differences *mail man* vs. *post man*).

2.4.3 Fact Extraction

This section presents information extraction methods that typically build on the basic tasks of natural language processing and the extraction of terms to derive further information contained in the text.

A very common information extraction task is **Named Entity Recognition** (NER), the process of finding *"each mention of a named entity in the text and label its type"* [185]. A **named entity** is a real object or an abstract object that can be referenced with a proper name (e.g., Albert Einstein, Berlin). NER can be considered to be a special case of term extraction (locating the named entity), but also includes a classification task: mapping entity types to localized mentions. Entity types are the classes that subsume named entities: people, locations, organizations, etc. Typically, numeric expressions are also treated as named entities (for example, dates, prices). The set of entity types depends heavily on the extraction goal and the domain. General types are *person*, *location*, *date*, domain-specific classes are for example *gene* or *protein* names. Fine-grained NER aims to resolve more specific types (e.g., *politician*, *scientist*, or *film star* for the type *person*). State-of-the-art NER tagging systems use feature-based models (conditional random fields) [191], neural networks (LSTM) [192], and rule-based approaches [193].

Closely related to NER is the task of **coreference resolution**, which resolves which mentions of entities refer to the same entity. Normally, pronouns (e.g., *he*, *it*) must be associated with the correct entity, and variations of named entities, such as descriptive noun phrases, must be joined together (e.g., *United States of America* and *The States*). Coreference resolution attempts to find all the referencing expressions of an antecedent and arranges them in chains. In contrast, **anaphora resolution** works in the opposite direction and identifies the referring expression for a single pronoun. Current state-of-the-art coreference resolution systems are based on clustering and learning-to-search algorithms [194]. Figure 2.16 shows an example of NER tagging and coreference resolution[51].

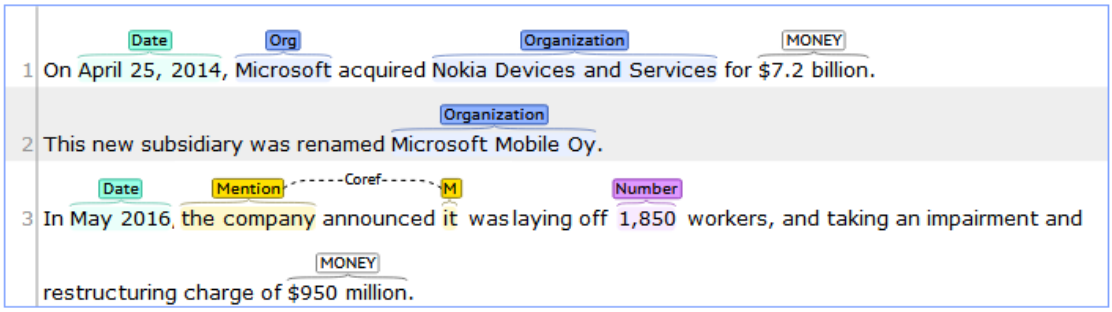


Figure 2.16: Named Entity Recognition and Coreference Resolution applied to an example text using the Stanford CoreNLP tools

Semantic role labeling, often also called thematic role labeling, *"[...] regards the assignment of semantic roles to the (syntactic) constituents of a sentence."* [116]. If a sentence is segmented into noun phrases and verb phrases, its task is to determine the arguments of verb phrases (predicates) and to assign a proper role to them. Typical generalized thematic roles are: AGENT, RESULT, SOURCE, or GOAL. The sets of roles used for annotation are dependent on what external source is used (e.g., PropBank [195], VerbNet [110], FrameNet [109]) and what kind of linguistic theory is implemented (e.g., categorial roles vs. prototype roles). Figure 2.17 shows an example of semantic role labeling[52].

Figure 2.17: Example text enriched with semantic role labels using the SRL demo of the Cognitive Computation Group at the University of Pennsylvania

While the previous methods enrich parts of a given sentence (e.g., words, phrases) with additional semantic information, the task of **relation extraction** (RE) is to determine relations between already identified and tagged parts. This is essentially establishing and classifying binary relationships (or in general n-ary relationships) between named entities. An extension of relation detection is **event extraction** that also allocates a temporal and spatial dimension to the extracted relationships. The supported relation types of an extraction system depend on the extraction tasks and the domain of the target text corpus. Typically, relationships with respect to locations (e.g., cities *locatedIn*), to persons (e.g., *marriedTo*, *bornIn*) and to organizations (e.g,. *subsidiaryOf*, *ownerOf*) are detected. A common set of relations is provided by the Linguistic Data Consortium in the Automatic Content Extraction (ACE) program[53]. Methods of relation extrac-

[51] The sentence was processed with `http://nlp.stanford.edu:8080/corenlp/`
[52] The sentence was processed with `http://cogcomp.org/page/demo_view/SRL`
[53] `https://www.ldc.upenn.edu/collaborations/past-projects/ace/annotation-tasks-and-specifications`

tion can be roughly classified into rule-based approaches as well as supervised, semi-supervised and unsupervised machine learning approaches, that will be described in the following.

Rule-based relation extraction is based on manually developed extraction rules that mainly rely on lexico-syntactic features of sentences. The decision as to whether a particular structure represents a particular relationship type is made by explicit pattern recognition. The oldest and most popular approach is the use of **Hearst Patterns** [120] to extract *is-a* relationships (hypernyms) from natural language text. These patterns contain surface-level phrases (e.g., *such as*, *or*), punctuation marks, placeholders for named entities, and elements of regular expressions. For example, the pattern *"NP_0, such as NP_1, NP_2, ..., (and |or) NP_n"* extracts hypernymic relationships (*programming language*→{*Java,Python*}) from an example sentence fragment *"If you are used to a programming language, such as Java or Python"*. The main advantage of rule development is that, because of their declarative nature [193], these patterns are easy for humans to understand, and the effects of change are directly visible (compared to a machine learning model, which requires a training phase and an extraction phase). This is also confirmed by recent studies showing that rule-based systems are **still prevalent in industry** [196]. In contrast, rule-based extractions have difficulty with scalability, namely the high costs of rule development and the manageability of large sets of rules. This is mainly addressed by pattern learning approaches. Nevertheless, it has recently been shown that especially for hypernym detection rule-based approaches still outperform distributional methods [197].

Supervised relation extraction *"requires labelled data where each pair of entity mentions is labelled with one of the pre-defined relation types."* [198]. Supervised RE is based on lexical, syntactic and semantic features that are extracted from the labeled training data. These features are used to train machine learning algorithms for classification that assign the most probable relationship type to respective entity mentions. **Features** in this case are observed phenomena in the original sentences. These include for example word-based features (word before and after the entities), grammatical features (phrase heads), and several semantic features (e.g., types of the entities, part-of-speech of words dependent on the entities). For manually labeled training data and/or manually engineered features in general, any standard **classification** algorithm can be used (support-vector machines, logistic regression, etc.). The advantage of feature-based relation extraction is that cross-validation can easily be applied to evaluate the features. The disadvantage is that the creation of labeled training data is very costly. An overview of these kinds of systems can be found in [199].

Semi-supervised relation extraction aims to reduce the high effort of creating labeled training data. The first approach is to use **bootstrapping** algorithms [185]. In this procedure, only a small amount of example relation instances, so-called **seeds**, are supplied at the beginning (e.g., if seeking for book authors, {*J. K. Rowling, Harry Potter and the Philosopher's Stone*}, {*Stephen King, Misery*}). Occurrences of these examples are searched in a large unlabeled corpus, and patterns are learned from the occurrences. Newly discovered patterns are used to extract new relation instances, and then the process is repeated. SNOWBALL [200] was one of the first systems implementing this principle based on **Dual Iterative Pattern Relation Expansion** (DIPRE). The steps of the algorithm are shown in Figure 2.18. The main challenge of this iterative process is the assessment of discovered patterns, because no gold standard for validation exists: *"**Semantic drift** occurs when an erroneous pattern leads to the introduction of erroneous tuples, which can then, turn, lead to the creation of problematic patterns."* [185]. To address this problem, confidence values are estimated for new patterns and new tuples based on how many tuples a pattern finds in the set of already extracted tuples and in the whole document collection. Also limited closed-world knowledge (e.g., implemented in DARE [201]) is used to improve the assessment.

The second approach of semi-supervised relation extraction is called **distant supervision** (DS). *"The distant supervision assumption is that if two entities participate in a relation, any sentence that contain those two entities might express that relation."* [202]. In contrast to bootstrapping a large number of seeds is provided to the extraction system by obtaining tuples of already existing knowledge bases. These tuples are then used to find sentences in which they occur, and respective patterns are extracted. The advantage over bootstrapping is that it allows many more features to be gained. The disadvantage is that a lot of noise is introduced by finding

sentences just by named entity occurrences. DS has some difficulties with overlapping relations. That means, multiple relations are valid for the same tuples (e.g., many persons were born in, lived in, and died in the same city), and patterns learned from these examples contribute to all relations. This is addressed by introducing negative examples [203].

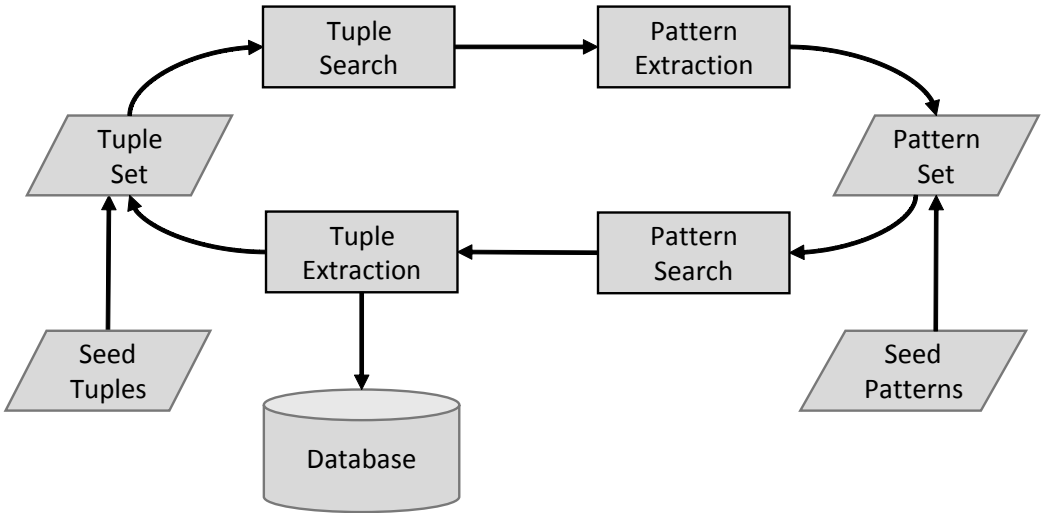


Figure 2.18: Bootstrapping principle for semi-supervised relation extraction, after Jurafsky and Martin [185]

Unsupervised relation extraction (URE) is a line of research that aims to extract relations and relation instances from natural language text without any human intervention. The biggest difference to the previous methods is that URE is not restricted to a fixed set of relations. Hence, URE can still be applied, if a knowledge base of a certain domain is not available. Most of the approaches (e.g., [204],[205]) rely on **clustering** to determine sets of entity pairs that belong to a relation type. In general, the procedure is as follows [198]. (1) Named entity recognition is performed on the complete text corpus. (2) Pairs of co-occurring named entities, their order and their context (intermediate and surrounding words) are recorded. (3) Context similarities are determined, for example, by word frequencies, lexical information, and dependency structures. (4) Hierarchical clustering is applied to the entity pairs using the similarity values. Each of the resulting clusters represents a relation that is then labeled.

The most influential work in the area of unsupervised relation extraction is **Open Information Extraction** (OpenIE), first proposed by Banko et al. in the TEXTRUNNER system [117]. Three phases are required to obtain a set of extractions: (1) Self-supervised learning: Because deep parsing is not practical for large corpora, only a small sample of the corpus is fully processed first and a set of heuristics is applied to obtain positive and negative training data fully automatically. The training examples are used to train a Naive Bayes classifier. (2) Single-pass extraction: The complete corpus is analyzed with noun phrase detection (chunking) and their connecting words. Relation candidates are accepted or discarded by the aforementioned classifier. Extractions are in the form of *(Berkeley, located in, Bay Area)*. (3) Redundancy-based assessment: All relation tokens are stemmed into their base form and tokens are omitted that may lead to overspecification (e.g., *"Young scientists from many universities are studying exciting new technologies"* is reduced to *"scientists are studying technologies"*). Identical tuples are then merged and it is counted from how many distinct sentences the relation was extracted. This allows to estimate a probability whether a tuple is a correct instance of a relation. Starting with TEXTRUNNER a series of extraction systems and methods have been developed to improve unsupervised relation extraction. This includes: REVERB, OLLIE, RELNOUN, SRLIE, and OPENIE 5.0. Mausam [206] provides an overview of these systems.

2.4.4 Distributional Semantics

This section presents methods that automatically build semantic representations of words and phrases by observing their occurrence in natural language text. *"The terms distributional, context-theoretic, corpusbased or statistical can all be used (almost interchangeably) to qualify a rich family of approaches to semantics that share a "usage-based" perspective on meaning, and assume that the statistical distribution of words in context plays a key role in characterizing their semantic behavior."* [207].

Distributional models have a long history in linguistic, cognitive and computational research reaching back to the fifties. In his early work on discourse analysis, Zellig Harris first coined in 1954 the **Distributional Hypothesis**: *"the parts of a language do not occur arbitrarily relative to each other: each element occurs in certain positions relative to certain other elements"* [208]. This theory was first used in phonemic analysis and later proposed for all linguistic levels. The famous slogan of J.R. Firth (1957) *"You shall know a word by the company it keeps"* [209] shows that with respect to words (words with similar meanings occur in similar contexts). Hindle [210] restricted that to words having a grammatical dependency on the target words.

Vector Space Models (VSM) [211] are the most common representation of word contexts. Context in that regard are the words in the neighborhood of a target word. The general idea of VSM is to represent documents, phrases or words as points in space, and the distance between these points corresponds to their semantic similarity. This geometric measurement was generalized to capture various aspects of co-occurrence in natural language text usually with the use of vectors, matrices, and higher-order tensors. As VSM are often called vector **semantics**, semantics here refers to the meaning of a word or phrase. **Semantic similarity** in this context is used in a broader sense and refers to the **semantic relatedness**. Two words are similar if they are near-synonyms. Antonyms are considered highly related but not similar [185], and words having other functional dependencies (e.g., *car* and *gasoline*) are also considered to be related but not similar.

The construction and use of vector space models is mainly concerned with four aspects: (1) **Matrix type** – what kind of properties of the text are represented by columns and rows (e.g., a word-document, word-context, or pair-pattern matrices). The construction usually requires some linguistic preprocessing of the corpus under study. (2) **Weighting** – values in the matrices are usually filled with frequencies and then normalized according to a certain weighting model (e.g., probabilities, TF-IDF, Pointwide Mutual Information (PMI)) in order to make them comparable and to favor discriminative information content. (3) **Dimensionality reduction** – word matrices usually have a huge number of columns and rows and are sparse (most values are 0). *"The goal of dimensionality reduction is eliminate rows/columns that are highly correlated while bringing similar things together and pushing dissimilar things apart."* [212] (e.g., through Latent Semantic Analysis (LSA), Principal Components Analysis (PCA), or Latent Dirichlet Allocation (LDA)). (4) **Vector comparison** – there are several ways of measuring the similarity of VSM's vectors (e.g. Cosine, Manhattan or Euclidean distance, Kullback-Leibler divergence). This often involves sparse-matrix multiplication methods or distributed parallel computation because matrices may not fit in memory completely [185].

	bite	buy	drive	eat	get	live	park	ride	tell
bike	0	9	0	0	12	0	8	6	0
car	0	13	8	0	15	0	5	0	0
dog	0	0	0	9	10	7	0	0	1
lion	6	0	0	1	8	3	0	0	0

Figure 2.19: Example of a word-context matrix, based on [213]

Figure 2.19 shows an example of a word-context matrix of nouns and their neighbor verbs. The rows represent the target words (word vectors), and the columns represent the context (context

vectors). The embedding of words into a vector space is called **word embedding**. The values in the cells are the frequencies with which the words occurred together. From this simple representation, it can already be seen that *bike* and *car* share similar contexts as well as *dog* and *lion* (through the respective counts in the rows). No matter what type and mathematical principle is used, vector space models can be summarized as mutual-information weighted word co-occurrence matrices.

Vector space models are one technique of word embedding that are all based on count models. Another family of approaches that became very popular in recent years is based on **neural language models** and uses predictive models to create distributional semantic models (DSM): *"Instead of counting co-occurrences, prediction DSMs are neural network algorithms that directly create low-dimensional implicit distributional representations by learning to optimally predict the contexts of a target word"* [213]. That means the most probable set of context words is directly learned from the input data.

The most prominent approach that boosted research in this direction is **Word2Vec** [214, 215]. Although there have been other approaches before, which used neural networks for language models (e.g., feedforward NNLM [216]), Word2Vec was the first work that minimized the training to log-linear computational complexity. This made it possible to learn vector representations using much more data than it was possible before. Thus, the models outperformed previous work in various word similarity tasks. Additionally, the output of neural language models are dense vector representations that also allow for computationally efficient usage.

Learning low-dimensional word vectors (usually 50 to 600 dimensions) means that for each observation in a large corpus the vector of the target word and the vectors of the context words are updated to be closer in vector space and all others are updated to be less close to the vector of the target word. Word2Vec implements two models, the first is a Continuous Bag-of-Words (**CBOW**) model. This model predicts a target word based on a set of n history and n future context words (often $n = 2$ is used). The order of the context words is ignored. The second approach is named Continuous **Skip-gram** Model that aims at predicting the context words from a given target word. Skip-grams allow a context with gaps. As CBOW always predicts the most probable word, infrequent words are not well represented with this model. CBOW is more efficient in terms of memory requirements and training time because the skip-gram model will possibly preserve multiple relevant contexts for one target word (capturing multiple semantics, and thus also working better with infrequent words). Skip-gram model respects the order of the context words in the sense that closer context words will get higher weights. CBOW performs better on large corpora, where skip-gram should be used if only smaller training data is available[54]. The neural network either uses hierarchical softmax or **negative sampling** for the output layer, both being computationally more efficient than previous NNLMs. The specialty about negative sampling is that for a target word the observed context words are treated as positive examples, and negative examples are obtained by randomly choosing words from the lexicon not co-occurring with the target word.

Although neural language models have been treated as the holy grail for a wide range of word similarity and classification tasks in recent years, a systematic comparison of count-based and predictive models is still ongoing research [213, 217]. Moreover, it was recently shown, that count-based models can still compete with predictive models or even outperform them if less training data is available [218].

2.5 Foundations of Recommender Systems

Recommendation systems (RS) provide users with context-sensitive suggestions to support the decision-making process [72]. Typically, an RS collects information about users and items they interact with (such as movies, songs, products). RS help users to deal with a large amount of available information by providing recommendations for items that may be of interest to them (e.g., which movies to watch, which music to listen to, or which products to buy). The main

[54] `https://groups.google.com/forum/#!searchin/word2vec-toolkit/c-bow/word2vec-toolkit/NLvYXU99cAM/E51d8LcDxlAJ`

goal of an RS is to *predict* that an item is worth recommending. *"RS make use of different sources of information for providing users with predictions and recommendations of items. They try to balance factors like accuracy, novelty, dispersity and stability in the recommendations"* [219]. This section introduces the basic concepts and types of recommendation systems proposed in the literature. After that, we highlight two types of RS, semantics-aware recommendation systems and recommendation systems in software engineering (RSSE).

2.5.1 Types of Recommender Systems

Recommendation systems depend mainly on the available data used to generate recommendations, how the data is processed (filtering), and the goal to be achieved (e.g., top-N recommendations, scalability, precision). On the one hand, data refers to properties of the objects of interest (e.g., product features and customer attributes), and on the other hand to external sources such as ratings or social information. Not surprisingly, data mining methods are common to recommender systems [72], such as preprocessing, classification, and clustering.

The most common **classification** of RS is based on filtering algorithms [220]. They determine what information is used and how it is processed to generate recommendations. These approaches include collaborative filtering, demographic filtering, content-based filtering, hybrid filtering, as well as constraint-based, knowledge-based, and context-sensitive systems.

Collaborative filtering (CF) *"... methods produce user specific recommendations of items based on patterns of ratings or usage (e.g., purchases) without need for exogenous information about either items or users."* [72]. Two users are considered similar if they offer similar ratings for similar items or buy similar products. A preference for certain types of items can be obtained automatically by analyzing the activities of the users (e.g., history of the products browsed in a store, number of times songs are heard). The similarity between users is determined by comparing the scores of all the items they evaluate, and the similarity between items is determined by comparing the scores of all the users who evaluated the items. A common recommendation algorithm for CF is k Nearest Neighbors (kNN) using standard similarity measures such as Pearson Correlation (CORR) or Mean Squared Differences (MSD).

Content-based filtering (CBF) identifies similarities between items based on their properties (such as genre, ratings, opinions, tags) and recommends similar objects to those that a user has interacted with in the past. CBF matches the preferences of a user profile against the features of the items. A user profile is either manually specified as a set of interests or built from the features of items that the user previously rated, bought, or viewed. Item characteristics are usually retrieved as a set of keywords from product descriptions or product catalog attributes. A typical architecture of content-based recommender systems is as follows [221]. A *content analyzer* extracts information from item descriptions using standard methods of information retrieval and creates structured element representations. A *profile learner* collects user feedback on items, either through explicit evaluation of items or implicitly through user interaction with the system. Along with the article representations, a profile is learned that represents the interests of the user. A *filtering component* compares the profile with element representations and estimates the relevance of new elements. The result is either a binary relevance judgment (relevant or not) or a continuous representation (top-N recommendations).

Demographic filtering (CF) is based only on features of users such as age, gender, country, and assumes that users with similar personal attributes have a similar interest in items.

Knowledge-based recommendation systems use additional knowledge of users, elements, and recommendation criteria to improve the recommendations. Knowledge is usually coded by rules on how user interests are satisfied by article features. Recommendations are generated by applying reasoning on user, item, and rule data. In the context of RS, the term knowledge-based is usually associated with the encoding of explicit knowledge in the sense of **logical implications**, but not with the use of external knowledge bases that contain factual knowledge, as we know from Linked Data. In general, there are two types of knowledge-based RS: **constraint-based** and **case-based**, both using this type of encoded knowledge. *"Case-based recommenders determine recommendations on the basis of similarity metrics whereas constraint-based recommenders pre-*

dominantly exploit predefined recommender knowledge bases that contain explicit rules about how to relate customer requirements with item features." [72].

Hybrid filtering combines the above approaches to take advantage of each strength and to overcome the disadvantages. The cold start issue for new items in CF and for new users in CBF is usually addressed by combining CF or CBF with demographic filtering and knowledge-based approaches. **Context-aware** recommendation systems additionally use time, location, and other sensor-based information to improve the recommendations.

2.5.2 Semantics-Aware Recommender Systems

An extension of recommender systems that was recently introduced to classical RS uses external semantic databases containing knowledge encoded according to the Linked Data principles [222]. These systems are sometimes referred to as a knowledge-based RS [223] leading to confusion with traditional knowledge-based approaches, as described in the previous section. In general, this new type of RS falls into the category of content-based RS because it mainly uses additional information from Linked Open Data (LOD) to learn features of items. We adopt the term semantics-aware recommender systems, often also called **Linked Data-based RS**, which describe a type of RS that relies *"... on the integration of external knowledge, such as machine readable dictionaries, taxonomies (or IS-A hierarchies), thesauri or ontologies (with or without value restrictions and logical constraints), for annotating items and representing user profiles in order to capture the semantics of the target user information needs."* [224].

The knowledge sources used by semantics-aware recommender systems reach from common sense, linguistic and domain-specific ontologies to encyclopedic knowledge bases, such as DBpedia and Wikidata, or in general, any dataset from the Linked Open Data Cloud[55]. According to a study of Figueroa et al. [223] DBpedia is the most widely used dataset because it is one of the largest sources available and suitable for a variety of domains. LOD datasets are mainly used because of the available information for heterogenous items and the lack of semantic information for them. For example, named entity linking to DBpedia for item descriptions helps to overcome polysemy and synonymy problems of keyword based approaches. Additionally, further information on linked entities can be retrieved from knowledge bases to discover new similarities, for example, movies with different directors that in turn have many commonalities (e.g., birthplace, awards won) [225] will be preferred for recommendations. Finally, schema information with taxonomic knowledge is used to categorize items according to their classes and to discover items that have common super classes [226].

Linked Data is used for RS in the following four ways as reported by Figueroa et al. [223]: *"Linked Data driven RS rely mainly on Linked Data to perform their tasks, hybrid RS use Linked Data and also other techniques, representation only RS do not provide Linked Data-based recommendations but use Linked Data for representing data based on RDF, and finally exploratory search systems that are not RS but may help users to find concepts or topics and have some similar features to RS especially in the use of Linked Data."*

2.5.3 Recommendation Systems in Software Engineering

Research and development for recommendation systems have strong roots in e-commerce applications, but they become more and more important in the field of software engineering (SE). *"Key factors giving rise to practical RSSEs include large stores of publicly available source code for analyzing recommendations, mature software-repository mining techniques, and mainstream adoption of common software development interfaces, including Web interfaces such as Bugzilla and tool-integration platforms such as Eclipse."* [227].

The main difference between traditional RS and recommendation systems in software engineering (RSSE) is that the context of recommendations is difficult to grasp. This is mainly because of the **variety of tasks** associated with building software systems and because the application domain is not known beforehand. Additionally, programming tasks are very often accompanied with

[55] `http://lod-cloud.net/`

information seeking (e.g., searching on Q&A websites such as Stack Overflow[56]). Consequently, data mining and external knowledge sources play an important role.

RSSE may analyze many different sources of information [75]: Source code (e.g., of a project, of a framework, external source code repositories), project history (e.g., changes in a version control system), communication archives (e.g., forums, issue trackers), APIs and their documentation, the integrated development environment (IDE) itself, interaction traces (e.g., logs of user actions), execution traces (e.g., of a running software system), and the Web (e.g., tutorials, Q&A websites).

As a holistic recommendation system is not feasible for SE, RSSE are often task-specific and focus on specific aspects of software development (e.g., source code specific recommendation, such as refactoring and code reuse, or recommendations related to bug reports and feature requests). The most well-known recommendation feature implemented in most IDEs (e.g., Eclipse) is **code completion**, which helps developers in finding variables, methods and parameters efficiently with context-sensitive pop-ups. It is based on the type system of the programming language as well as language features of the core language and available frameworks. In order to improve recommendation ranking, learning from examples is used [228]. Important RSSE implementations are: Strathcona [229] for code example recommendations for third-party libraries, Hipikat [230] for the recommendation of project information for maintenance tasks, and CodeBroker [231] for recommending methods to fulfill an implementation task.

Like the above approaches, most available recommendation systems deal with tasks that are related to programming, because they are closely connected to integrated development environments. Recommendation systems for **software modeling** have so far not received much attention, although modeling is considered a very important task in software engineering by both research and industry [36]. One reason is that modeling is still a separate activity often performed in separate tools. A few approaches touch some aspects of modeling, such as model completion for state machines using constraint logic programs [232], guided DSL development using constraints and presenting possible editing operations [233], and automatization of software product line engineering [85]. Recommendation for modeling is still predominantly based on guidelines and on examples using model repositories, but their adoption is still challenging [234].

2.6 Summary

In this chapter, the fundamentals of four research areas were presented, whose interaction is addressed in this dissertation. We introduced the foundations of software modeling, general concepts of models and modeling languages as a software engineering tool at a higher level of abstraction. **Domain Modeling**, the application field of this dissertation, and related methods were presented to demonstrate the importance of achieving a common understanding among stakeholders in software projects by explicitly creating domain models using domain-specific terms and relationships. The field of knowledge bases was discussed, including various types of knowledge representation and knowledge management using Semantic Web technologies. **Linked Open Data** plays the most important role as it provides the standards for a variety of datasets that include common sense and domain-specific knowledge. We have presented the field of information extraction and basic methods of natural language processing that allow the (semi-)automatic generation of structured facts from unstructured data sources. Most important is the **extraction of terms** from natural language text and the application of distributional methods to derive semantic relationships between them. Finally, we reviewed the basics of recommender systems, highlighted trends in the use of Linked Data in **Semantics-Aware Recommender Systems**, and determined the lack of implementations of recommendation systems for modeling tasks.

[56] `https://stackoverflow.com/`

Chapter 3

Semantic Modeling Support

Domain modeling is a challenging task because it requires multidisciplinary collaboration and information gathering from different groups of people, documents and other sources of knowledge. Collecting appropriate information for a software project and implementing it in domain models is often time-consuming and decoupled from modeling tools. In this chapter, a new methodology is developed to provide domain information directly during modeling, addressing the first challenge of domain modeling: the high cost of acquiring domain knowledge.

3.1 Introduction

In this chapter we introduce the concept of semantic modeling support and detail the approach of the thesis. *Modeling*: The activity of creating and refining models. In our case these models are domain models that focus on capturing concepts and relationships of particular application areas. *Support*: Modeling activities are assisted with context-sensitive pieces of information. Support is completely automated in contrast to guidelines or methodologies. *Semantic*: Modeling support focuses on the domain-specific terms and their relationships in domain models in contrast to syntactic modeling language assistance.

The chapter is organized as follows. First, related modeling methods are reviewed in Section 3.2. The overall approach of modeling support with automated knowledge acquisition is introduced in Section 3.3. Then, the model refinement operations are described that are extended by the semantic modeling support (cf., Section 3.4). This includes the description of respective domain modeling constructs and a number of scenarios that include the definition of the proposals the user will receive. Section 3.5 analyzes how conceptual relationships between terms in domain models are represented across different linguistic and knowledge modeling paradigms. Based on the analysis, a mapping between these relationships is developed. After that, we focus on what the retrieval of related terms should look like according to the analyzed semantic relationships (cf., Section 3.6). We then describe the conceptual framework and prerequisites of knowledge acquisition from unstructured text datasets (cf., Section 3.7) and from structured knowledge bases (cf., Section 3.8), and we finally conclude the chapter in Section 3.9.

3.2 Related Modeling Methods

Semantic modeling support has been predominantly investigated in the area of connecting ontology development with model-driven development [136]. A recent book by Nalepa and Baumeister [235] describes synergies between knowledge engineering and software engineering by applying software and modeling methods in ontology development and vice versa. At the **conceptual level** there is still ongoing work to unify ontological and software modeling paradigms [236], approaches to adopt modeling concepts from each other [237], and a series of works by Giancarlo Guizzardi to extend MDE languages with ontological foundations [238, 239, 240].

Tairas et al. [241] describe how the domain analysis phase of DSL development benefits from the use of **ontologies**. Their approach is based on manual ontology construction during early stages of domain-specific language development. Thonggoom et al. [242] support conceptual modeling using data model instance repositories. The repositories are created from SQL schema libraries with several hundred relations, and thus contain patterns from prior database designs to allow **modeling knowledge reuse**. The REBUILDER UML system [243] aims at a similar goal for UML diagram reuse. The design assistant uses case-based reasoning. The OntoDSL framework [244] uses ontology technologies at the meta-model level (such as reasoning) to help DSL users identify model-level inconsistencies. The CoCoViLa tool [245] generates metamodels of domain-specific languages from OWL descriptions.

3.3 General Support Procedure

The procedure of our intended modeling support is shown in Figure 3.1. The approach incorporates an iterative process with three steps. Each of the steps is associated with tools that are used in the activity, and with artifacts that are produced or consumed, respectively. Starting with model refinement, one iteration reads as follows: A manual change in a domain model is made. Based on the current state of the model, domain knowledge is acquired automatically. The acquired knowledge is transformed automatically into appropriate suggestions that influence the next refinement step.

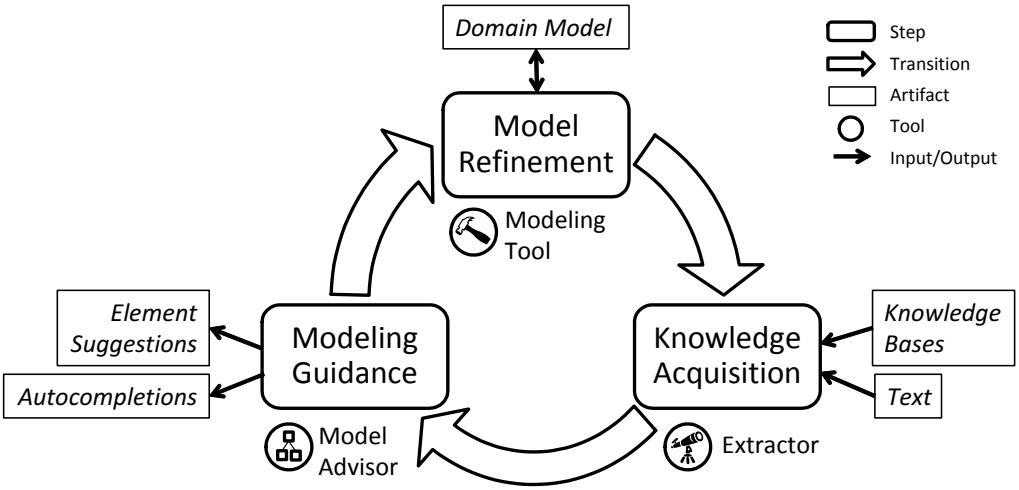


Figure 3.1: Iterative approach of supported modeling

Model Refinement is a manual activity where a developer creates a new model or changes an existing one. It is the developer's task to represent real-world concepts and relationships in a domain model using a modeling language (e.g., UML class diagrams, ER diagrams). Usually, an existing *modeling tool* is used to perform this task. The tool implements the modeling language and offers the possibility to create models conforming to the language. The objectives of the thesis are the identification of model change operators that can be semantically supported. This requires the mapping of modeling relationships to semantic relationships of knowledge representations. In addition, the proper extraction of the content of domain models as well as the connection of suitable modeling assistance services to the modeling environment will be developed.

Knowledge Acquisition is the step in which domain knowledge for a specific model is gathered. The acquisition is based on the terms that are used to name the elements (e.g., class names or association names). We pursue two strategies: First, we exploit existing structured knowledge sources to acquire the required domain knowledge. Knowledge bases and ontologies (e.g., WordNet or Cyc) are automatically queried for terms of a model to retrieve related terms. The objectives

of the thesis are the connection of an exemplary set of knowledge bases and the implementation of an easy-to-use mechanism to plug in new knowledge sources. Secondly, it is a well-known problem [88, 89] that existing knowledge bases (often created manually) do not contain enough information or do not exist at all for respective target domains. We address this issue by the automated creation of own semantic knowledge sources from natural language datasets that cover a variety of domains. The objective of the thesis is to build domain-independent methods and tools to extract terms and their related terms from text corpora. The extracted information is used as a primary source for modeling suggestions. *Extractor* components implement the extraction from text datasets and knowledge bases.

Modeling Guidance is the activity in which domain knowledge obtained from different sources is aggregated and transformed into appropriate recommendations. Suitable excerpts of the data are generated and element suggestions are displayed to the user. The recommendations are highly context-sensitive depending on what the developer currently selects, creates or modifies in the domain model. It is the goal to present semantically related model elements that support the developer's decisions on what to include in the model and how to connect the elements. The objectives of the thesis are the development of data aggregation and transformation methods and tools to translate between the knowledge representation world and the modeling world. It is a challenge for a recommender system to suggest items that are of importance for the current situation. This thesis will develop appropriate ranking algorithms for the obtained knowledge.

3.4 Domain Modeling Support Scenarios

In this section we describe in detail the modeling support to be performed by our method. During domain model development a user has several options to change the model. These options usually depend on the used modeling tool and the respective implementation of the modeling language. In order to provide modeling support for a manageable set of operations we exactly define here for which modeling activities what kind of support shall be accomplished. We first examine the modeling notation and modeling constructs commonly used for domain modeling (cf., Section 3.4.1). Based on possible model refinement operations, we then specify a set of scenarios and expected suggestions by using examples. We distinguish between two different kinds of support. First, context information will be provided if an element of a domain model is selected by the developer. The context information includes possible related model elements with all kinds of relationships (cf., Section 3.4.2). Second, if a new element is created, automated suggestions will be provided on how to name the element. The support is dependent on the type of connection between the new element and existing elements of the model (cf., Section 3.4.3)

3.4.1 Domain Modeling Languages

There are several ways to create and manage domain models. The methods and tools developed in this thesis apply to several modeling languages and modeling means. Prominent representatives include UML class diagrams and Entity-Relationship (ER) diagrams. In software engineering, UML class diagrams are often used to express concepts and relationships of a business domain [246]. Typically, they are used to derive design models for object-oriented programming languages during development. ER models usually play a role in data-driven applications to express types and relationships [8]. They are transformed into relational models for database design. Other examples for domain modeling languages include Object-Role Modeling (ORM) [247], Conceptual Modeling Language (CML) [248], and multi-level modeling [249, 136].

Although there is a continuing fundamental discussion in the conceptual modeling community [237, 249] about the correct representation of real-world concepts with modeling languages, all approaches have in common that the respective modeling language expresses conceptual structures of a domain with specific terms to improve the understanding of the problem area. To provide concrete examples and implement appropriate support tools, we will exemplarily use UML-like class diagrams as an example to illustrate our work. In fact, several recent empirical studies have

shown that the UML class diagram is the most widely used modeling paradigm in the industry [19, 25, 250, 8]. As semantic modeling support focuses on the terms contained in the models (how things are named), it can easily be transferred to other modeling languages.

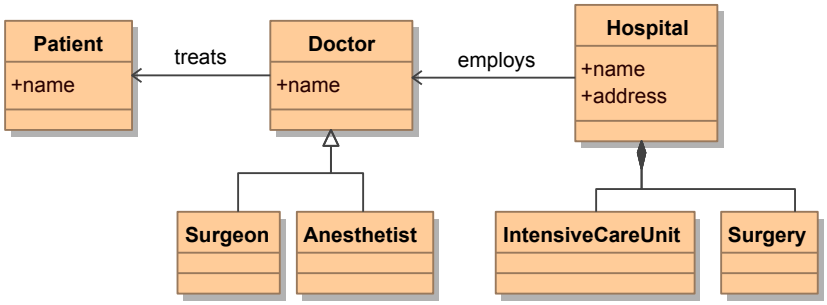


Figure 3.2: Simple domain model example in the healthcare domain

Figure 3.2 depicts a simple domain model in the health care domain using the UML class diagram notation[1]. The diagram shows several facts with domain-specific terms that can be interpreted as follows: Hospitals employ different types of doctors, and these doctors treat patients. In addition, hospitals are made up of different parts. The graphical symbols used and the underlying modeling means determine the interpretation of the domain information. Domain models are problem-specific and therefore follow the closed-world assumption [251]. This means that every concept and relationship used in development must be explicitly stated. In this particular case, the domain model defines that there are only two types of doctors and two different parts of a hospital (although it is known that there are more entities in the real world).

The domain model was created using parts of the UML metamodel and class diagram graphical notation. The following modeling constructs are relevant to our work. **Classes** group sets of instances that have common features. They are shown as rectangular boxes in the diagram. For example, the class *Patient* represents the concept of persons under medical care. Classes can have attributes that represent certain properties of the concept (terms inside the class box). In the example, hospitals can have *names* and *addresses*. At a certain level of refinement attributes are usually equipped with cardinality restrictions and with datatypes (not shown in the figure).

The illustration shows three different types of **relationships**. (1) Generalization / specialization relationships connect more abstract classes to more specific classes or vice versa (arrow with closed arrowhead)[2]. The domain model example defines that *Surgeon* and *Anesthetist* are specific doctors. (2) The second relationship shown is the composition (connection with a filled diamond). It expresses that a class is part of another class (e.g., an *IntensiveCareUnit* is part of a *Hospital*). The composition is a special kind of aggregation, i.e., the parts cannot exist without the whole. Using a normal aggregation connection, part and whole may exist independently of each other (e.g., tires of a car). Both aggregation and composition are referred to as containment relationships. It is the developer's decision which is appropriate. (3) The association is the third type of relationship (arrow with open arrowhead). It is a weaker kind of relationship and expresses that one class affects the other (e.g., *Patient* and *Doctor*). Normally, the type of influence is described by the association name (*treats*), but not necessarily. The association can have a direction (as shown in the picture) and may also have cardinalities.

The following is important for semantic modeling support: Concepts of the respective domain are represented with classes using **normal nouns**. Two or more classes can either be linked with part-whole (*"is part of"*) or taxonomic (*"is a"*) relations, or with an association using arbitrary **verbs**. Consequently, if the model is transformed to a linguistic representation, the domain-specific

[1]Other domain model examples in the hospital domain and other domains are available from: `http://www.uml-diagrams.org/examples/hospital-domain-diagram.html?context=cls-examples`

[2]Note that only one arrow is visible, but for each of the sub-classes one relationship exists in the model.

information can be largely expressed using verbs that link nouns [252]. This corresponds to the design steps for entity relationship diagram creation, in which textual descriptions are analyzed to identify entity types via noun phrases and relationship types via verb phrases [253].

3.4.2 Providing Contextual Information

The goal of providing contextual information is the recommendation of possible connected model elements together with their relationship types for a selected domain model element. The information displayed corresponds to the previously described modeling means of UML-like class diagrams. The following two scenarios are supported:

Scenario 1 – Selecting a class. If the developer selects a class, the user will see various types of information. A class can be linked with a generalization, a specialization, an aggregation (part or whole), and named / unnamed association relationships. Figure 3.3 shows all six types of modeling suggestions and respective examples. The first two types of suggestions refer to the generalization/ specialization link. (1) Subclasses indicate possible specializations, in this case special types of hospitals. (2) Superclasses show more general terms for generalizations of the hospital class. (3) Aggregated classes refer to possible parts (e.g., parts of a hospital). (4) Container classes show what a hospital can be part of. The last two types of suggestions refer to the association link to other classes. (5) Related classes show possible linked classes for which the relationship is not specified further (no verb). (6) Associated classes show possible linked classes together with their association names.

Interaction	**Modeling Support**
Select Class	*Context Information for the Class*

- **Sub classes**: psychiatric hospital, sanatorium, private hospital, ...
- **Super classes**: healthcare facility, medical care institution, …
- **Aggregated classes**: ward, hospital room, ...
- **Container classes**: hospital association, ...
- **Related classes**: clinic, admission, physician, nursing home, ...
- **Associated classes**: patient *is admitted to*, patient *is discharged from*, *includes* school, *trains* nurse, *employs* nurse, ...

Figure 3.3: Scenario 1 – Contextual information for a selected class

Scenario 2 – Selecting an association When an association is selected, model elements are proposed in relation to the association and its already connected classes. There are three different types of suggestions, as shown in Figure 3.4: (1) Alternative associations: Other association names are displayed, to which both classes can be connected. The example shows that a doctor could examine or visit a patient, and a patient could consult a doctor. (2+3) Associated classes: For each association end, alternative connected classes are also provided. Modeling suggestions in the example show that a doctor can treat a disease, an animal or a wound. For *patient* and *treat* different types of treatments are proposed.

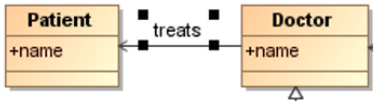
Interaction	**Modeling Support**
Select Association	*Context Information for the Association*
	- **Alternative associations**: examine, consult, visit, ... - **Associated classes for doctor + *treat***: disease, animal, wound, … - **Associated classes for patient + *treat***: surgery, placebo, chemotherapy, antibiotic, ...

Figure 3.4: Scenario 2 – Contextual information for a selected association

In summary, when selecting model elements, the upcoming modeling step is not yet known. Consequently, the provision of contextual information aims to propose a variety of information for all possible modeling actions to **assist in completing the model**.

3.4.3 Providing Suggestions for Element Names

In domain modeling, new elements are added to diagrams and named with domain-specific terms. The goal of providing suggestions for element names is to develop an autocomplete function for creating new model elements similar to search engines. Autocomplete is a feature in several applications that helps users to type in input fields. It is used successfully in mobile phones, web browsers, search engine interfaces and integrated development environments. Assistance is either provided immediately during typing or triggered with a keyboard shortcut. Autocomplete tries to predict the user input based on already entered characters, background knowledge, rules and heuristics.

Autocomplete in integrated development environments provides context-sensitive pop-up lists of possible completions for a particular programming language, depending on the grammar of a language and existing source code (also known as content assist). Search engine autocompletion is indispensable in today's search engines. It suggests keyword terms while typing in search fields. Search query suggestions are usually based on natural language statistics, past user searches, and the popularity of terms.

Autocomplete for domain modeling suggests a set of domain-specific terms while typing the name of a model element. Since the proposals should match the existing model elements, the suggested terms must be in semantic relationship to the existing terms. The suggestions depend on (1) the type of the new element (class or association), (2) the current state of the model (existing names of elements and relationships) and (3) the type of relationship that is drawn from the new element to another model element (e.g., a specialization). The main features of autocomplete are the adaptation of the suggestions during the typing and the ranking of the terms by relevance. Below we describe scenarios in which new elements are created and what the support should look like.

Scenario 3 – Creation of a class (no connection). Modeling tools usually offer the creation of new classes in a model without any connection. Typically, this happens, when classes are added to the diagram and the respective connections are drawn afterwards. In this case, class name suggestions are dependent on all existing class names in the model. Particularly, in the list of suggestions class names should appear that are related to all the existing classes ordered by relevance (cf., Figure 3.5).

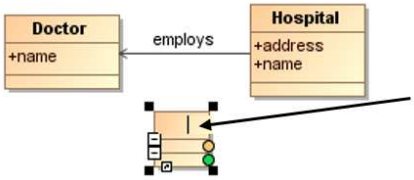
Interaction *Add Class Without Connection*	**Modeling Support** *Suggestions for the name of the related class*
	Nurse, Office, Clinic, Care, Admission, Discharge, School, Dentist, Treatment, Condition, Disease, ...

Figure 3.5: Scenario 3 – Suggestions for related class names when adding a disconnected class

Scenario 4 – Creating a subclass. A subclass is created when the developer uses the specialization link from an existing class to empty space in the diagram (cf., Figure 3.6). In this case, the class name suggestions depend on the associated superclass. The example shows different types of doctors (different types of medical specialists). Existing subclass names must not appear in the list of suggestions, so the appropriate context of the modeling action must be determined (all existing subclasses of the associated superclass must be considered).

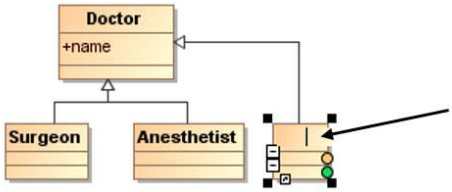
Interaction *Add Subclass*	**Modeling Support** *Suggestions for the name of the subclass*
	House physician, Allergist, Ear Doctor, Neurologist, Oncologist, Dermatologist, Medical Toxicologist, ...

Figure 3.6: Scenario 4 – Suggestions of subclass names when adding a specialization

Scenario 5 – Creating a superclass. Similar to creating a subclass, the generalization link creates a superclass. The example shows the recommendation of more general terms for the doctor class (see Figure 3.7). Existing superclass names may not appear in the list of suggestions.

Interaction	**Modeling Support**
Add Superclass	*Suggestions for the name of the superclass*

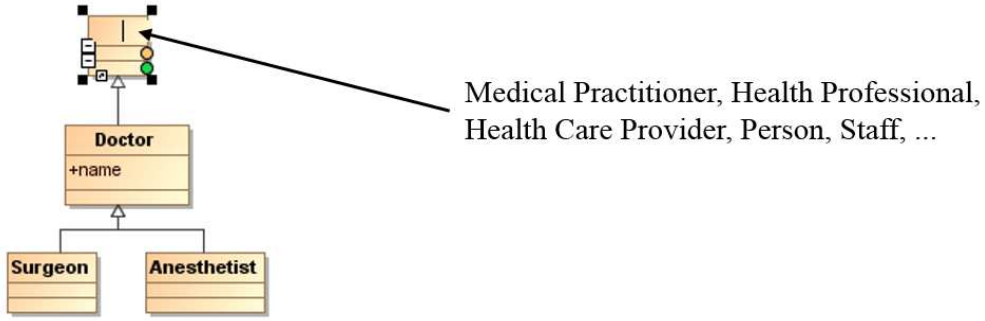


Figure 3.7: Scenario 5 – Suggestions of superclass names when adding a generalization

Scenario 6 – Creating an aggregated class. If the developer uses a composition or aggregation link that starts from an existing class, an aggregated class is created in the diagram. The example in Figure 3.8 shows possible parts of a hospital. The suggestions depend on the associated container class, existing parts, and the type of aggregation (either compositional or non-compositional).

Interaction	**Modeling Support**
Add Aggregated Class	*Suggestions for the name of the aggregated class*

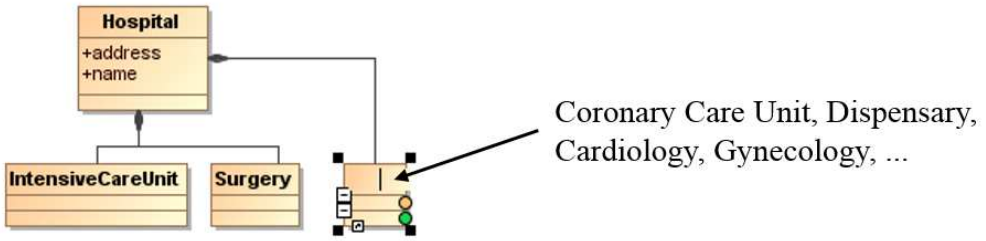


Figure 3.8: Scenario 6 – Suggestions of aggregated class names when adding an aggregated class

Scenario 7 – Creating a container class. If the opposite direction of a composition or aggregation relationship is used, a container class is created. In the example used in Figure 3.9, suggestions are made of which a hospital can be a part. Similar to the aggregated class scenario, already linked classes must be taken into account by the recommender.

Interaction	**Modeling Support**
Add Container Class	*Suggestions for the name of the container class*

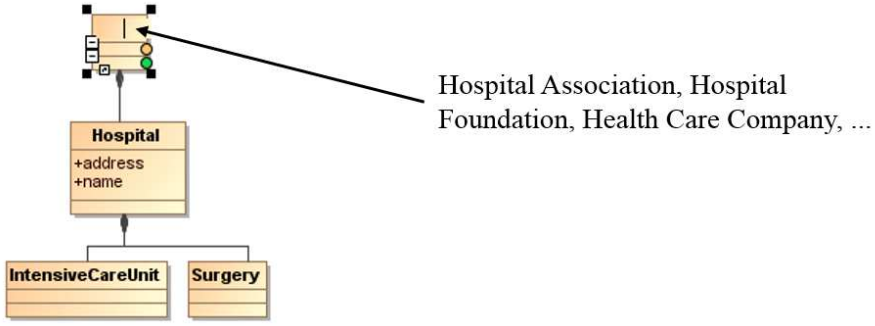


Figure 3.9: Scenario 7 – Suggestions of container class names when adding a container class

Scenario 8 – Creating an associated class. An associated class is created when the developer draws an association link from a class to an empty area in the diagram. The association is created without a name and a new class. Names for the new class will be recommended that are related to *hospital*. This scenario is very similar to the third scenario, with the exception that related classes are proposed only for the linked class and not for all classes in the diagram.

Interaction	**Modeling Support**
Add Associated Class	*Suggestions for the name of the associated class*

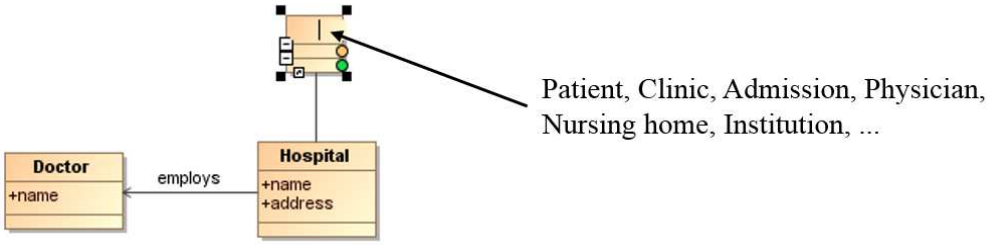


Figure 3.10: Scenario 8 – Suggestions of associated class names when adding an associated class

Scenario 9 – Creating an association When the developer creates an association link between two classes, recommendations are provided for the association name (verbs). The suggestions depend on both class names. If the association has no direction, verbs are suggested that apply to both directions (e.g., Nurse *observe* Patient, and Patient *ask* Nurse).

Interaction	**Modeling Support**
Add Association	*Suggestions for the name of the association*

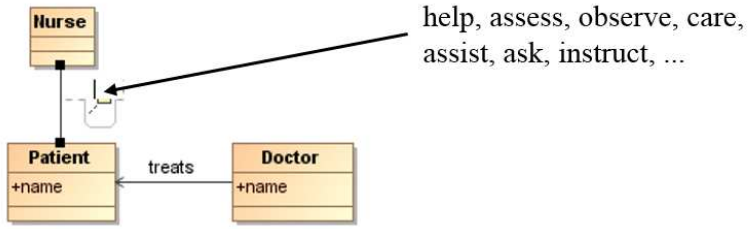


Figure 3.11: Scenario 9 – Suggestions of association names for a newly created association link

In summary, when creating new elements in a diagram, the most **relevant names** for the newly created classes or associations **are proposed**, depending on the type of the relationship. Normally, all scenarios also apply when a model element is renamed.

3.5 Mappings of Domain Model Relationships

The conceptual relationships used in domain models can be found in other research fields with different representations. Our goal is to combine different types of knowledge sources into a single recommendation system. In this section we analyze the inherent semantic relationships of domain models from a lexical perspective and their representation in other sources of knowledge. We focus on classes that represent the concepts and three types of relationships: generalization / specialization, aggregation, and association. We review literature from database research [254, 155], linguistics [255, 256, 257], information systems [258, 238, 136, 259], and semantic web research [260, 261, 262, 263], and relate the different types of relationships to each other. Table 3.1 provides an overview, details are given as follows.

Specialization and Generalization are hierarchical abstraction mechanisms to refine abstract classes to more specific ones and to group specific classes into more abstract classes. Specialized classes inherit the properties of the abstract class and introduce additional attributes or operations. Classes are structured into a taxonomy using specialization and generalization. In lexical semantics these conceptual relationships are referred to as hyponymy and hypernymy between words or phrases. Hypernyms are the more general terms and hyponyms the more specific ones. In linguistics, this relationship is in the family of class inclusion [256] (different types of subordinate relationships). In thesauri specification (e.g., based on ISO 25964) both relationships are expressed with the narrower term and broader term relation. The relationship can be mapped to the subClassOf-relationship [264] in ontology definition with RDF/OWL as well as to the broader- and narrower relationship of the SKOS specification [265]. Across disciplines it is often called *is-a* relationship. Unfortunately, this term is used ambiguously, on the one hand referring to subordinate relationships between classes and on the other hand referring to class-member relationships (instance relationships).

Aggregation is used to specify a part-of relationship between two classes. One class acts as the whole (the composite) and one class acts as the part (the component). We summarize both aggregation (parts can exist independently of each other) and composition (parts cannot exist independently of each other) under the term aggregation. In linguistics, part-whole relationships are called meronymic relationships (meronyms are the parts and holonyms are the wholes). Six to seven types of meronymy (e.g., component-integral object, portion-mass) are discussed in related works [266, 254]. Part-whole relationships are not supported directly in the thesaurus definition nor in the RDF / OWL ontology specification. There exist specialized vocabularies and approaches

to model these relationships in ontologies. There is a W3C Best Practice specification "Simple Part Whole" (SPW) that includes *hasPart* and *partOf* relationships. However, there are knowledge bases that contain part-whole relationships but use a non-standard vocabulary (such as WordNet).

Modeling Language Relationship	**Lexical-Semantic Relationship**	**Knowledge Source Relationship**
Specialization	Hyponymy	Subclass, Narrower Term
Generalization	Hypernymy	Subclass (inv.), Broader Term
Aggregation (Part)	Meronymy	HasPart (SPW), *Meronym (WN)*
Aggregation (Whole)	Holonymy	PartOf (SPW), *Holonym (WN)*
Association (named)	Agent-Action	Object Property
Association (unnamed) or group of classes	Semantic Relatedness	Related Term

Table 3.1: Corresponding semantic relationship types of different modeling paradigms

Association is the third kind of conceptual relationship we have analyzed in terms of other representations. We distinguish between two types: unnamed associations, to express a simple dependency between two domain model classes, and named associations that further specify the type of association (usually with a verb). In linguistics, named dependencies fall into the category of case relationships [256], more specifically in our case in *agent-action* relationships (e.g., *dog – bark*). To a certain extent, RDF/OWL *object properties* with domain/ range restrictions can be compared with named associations because they affect two classes of an ontology, as long as those restrictions are specified (e.g., *spokesman(PoliticalParty, Person)).* While thesauri do not contain named associations, the related term (RT) relationship is used to define a relationship between two terms in a non-hierarchical way [267], and thus can be mapped to the unnamed association. In lexical semantics, the unnamed association is referred to as *semantic relatedness*, an associative relationship that describes any functional relationship between two words. From a lexical point of view, the unnamed association is similar to a group of classes (the diagram is the container).

In summary, taxonomic relationships in domain models can be well mapped to other structured knowledge sources such as thesauri and ontologies. Other domain model relationships are not fully represented in these resources. As a result, they are a good source for acquiring knowledge for our modeling support, but they are not enough. All domain model relationships and their inherent conceptual relationships are rooted in various linguistic theories. Thus, the combination of knowledge base queries and natural language analysis allows to retrieve related domain model elements for all our modeling support scenarios.

3.6 Retrieval of Lexical Information

The modeling support scenarios and the semantic relationship mappings have essentially shown that the semantic modeling support is focused on retrieving appropriate domain-specific terms that have specific relationships to the other terms in a domain model. In this section, a series of technology-independent queries for terms is derived with a focus on lexical relationships that abstract from the details of the underlying knowledge bases to provide the content for the model element suggestions.

Each query represents a function that accepts domain-specific terms as input and returns a set of domain-specific nouns or verbs. Noun terms may be either single nouns (e.g., doctor or hospital) or multi-word expressions (e.g., health care provider, x-ray diffraction, or rate of DNA synthesis). The functionality of the queries is described, different types of input and expected outputs are defined, and they are illustrated using examples.

The first four queries provide straightforward taxonomic and partonomic knowledge. The query "**Get Broader Nouns**" retrieves more general noun terms for a single input noun. Generalization relationships are organized in hierarchies, so the query should allow you to specify an optional

maximum distance. For example, a distance of two means that for any broader noun all the broader nouns are also retrieved. The result of the query is a set of zero or more tuples containing the broader nouns and their distance from the input term.

```
GetBroaderNouns(hospital,2)
--> (medical care institution,1)
    --> (organization with individual clients,2)
    --> (health care organization,2)
--> (healthcare facility,1)
    --> (building,2)
```

Listing 3.1: Example of the broader nouns query using a distance of two

The query "**Get Narrower Nouns**" determines more specific noun terms for a single input noun. The functionality and structure of input and output is similar to querying broader nouns. The distance should be specified carefully. Compared to the broader terms query, a distance of two may already lead to a very large set of narrower terms. Listing 3.2 shows an example of retrieving narrower terms up to a distance of two.

```
GetNarrowerNouns(doctor,2)
--> (allergist,1)
--> (medical specialist,1)
    --> (anesthesiologist,2)
    --> (baby doctor,2)
    --> (cardiologist,2)
    --> ...
--> (surgeon,1)
    --> (neurosurgeon,2)
    --> (cosmetic surgeon,2)
    --> ...
--> ...
```

Listing 3.2: Example of the narrower nouns query using a distance of two

The query "**Get Part Nouns**" returns noun terms that are in a part-of relationship to the input noun. The distance is also applicable, therefore parts of parts can be retrieved. Listing 3.3 shows getting part terms for the term *pregnancy*.

```
GetPartNouns(pregnancy,2)
--> (segmentation,1)
--> (parturiency,1)
    --> (uterine contraction,2)
    --> ...
--> (morning sickness,1)
--> ...
```

Listing 3.3: Example of the part nouns query using a distance of two

The query "**Get Whole Nouns**" works in the opposite direction in the part-whole hierarchy and retrieves a set of nouns to which the input term belongs, as shown in Listing 3.4.

```
GetWholeNouns(cardiogram,1)
--> (medical checkup,1)
--> (medical examination,1)
--> ...
```

Listing 3.4: Example of the whole nouns query using a distance of one

The query "**Get Related Nouns**" delivers nouns that have a functional dependency to the input terms in a non-hierarchical way. The query allows three types of input: (a) A single noun term is queried. The result of the query is a set of noun terms that depend on this single input term. (b) As input a set of nouns is used. The query determines the noun terms that relate to all the terms entered together. This is especially important to provide suggestions for new classes in domain models if multiple classes already exist and the new class is not linked. (c) The third type of input is a noun term together with a verb term. In this case, the query identifies relevant noun terms that are associated with the verb to the input noun.

A distance as it exists in a hierarchy does not apply to a related nouns query. The relationship between related terms is based on the semantic relationship [268] (see Section 3.5). Consequently, the result of the query should be ranked based on this measurement. An example is shown in Listing 3.5.

```
GetRelatedNouns(pregnancy)   GetRelatedNouns(hypertension,induce)
--> #1 (childbirth)          --> #1 (diabetes)
--> #2 (woman)               --> #2 (oral contraceptive pill)
--> #3 (mother)              --> #3 (preeclampsia)
--> #4 (month)               --> #4 (pregnancy)
--> #5 (parturition)         --> #5 (toxemia)
--> #6 (termination)         --> ...
--> ...
```

Listing 3.5: Examples of the related nouns query using a single noun term and a noun-verb combination

The query "**Get Related Verbs**" retrieves verbs that have a functional dependency on noun terms (e.g., agent-action, see Section 3.5). There are three different types of input allowed for the query: (a) For a single noun, the query returns a set of verbs that depend on the noun. (b) There is a variant of this query (Get Related Verbs With Nouns) that retrieves verbs together with linked nouns that are related to the input term. (c) If two noun terms are provided as input, the query will return verb terms that connect both nouns together. In all cases, the ranking of verbs should be similar to the ranking of related nouns described in the previous paragraph.

```
GetRelatedVerbs(pregnancy)   GetRelatedVerbs(patient,doctor)
--> #1 (occur)               --> #1 (examine)
--> #2 (terminate)           --> #2 (tell)
--> #3 (associate)           --> #3 (ask)
--> #4 (contraindicate)      --> #4 (treat)
--> #5 (avoid)               --> #5 (consult)
--> #6 (prevent)             --> #6 (visit)
--> ...                      --> ...
```

Listing 3.6: Examples of the related verbs query using a single noun term and two noun terms

Table 3.2 summarizes all queries for lexical information retrieval.

Query Name	Input	Expected Output
Get Broader Nouns	Noun term	Set of more general noun terms
Get Narrower Nouns	Noun term	Set of more specific noun terms
Get Part Nouns	Noun term	Set of component noun terms
Get Whole Nouns	Noun term	Set of composite noun terms
Get Related Nouns	(a) Noun term	Set of related noun terms
	(b) Set of noun terms	Set of related noun terms
	(c) Noun + verb term	Set of related noun terms
Get Related Verbs	(a) Noun term	Set of related verb terms
	(b) Noun term	Set of related verb + noun terms
	(c) Two noun terms	Set of related verb terms

Table 3.2: Summary of the technology-independent term queries

3.7 Knowledge Acquisition from Text Datasets

In this section, we detail the concept of how natural language text is analyzed to extract terms and their relationships on a conceptual level. The methods of information extraction used have been chosen so that they can be applied universally to all possible domains. Figure 3.12 shows a general high-level extraction process. The process begins by collecting text documents from various sources, either domain-specific (e.g., news articles, medical documents) or domain-independent (e.g., web crawls, digital libraries). The next step is pre-processing (e.g., boilerplate removal) the text document collection [269] to create a clean text corpus that can be accessed in a consistent way. The main task is the implementation of the information extraction pipeline. It is called a pipeline, since natural language processing is usually done in several steps, and the output from a previous step is used as input to the next step and enriched with further annotations [270] (e.g., part-of-speech tags, parse trees, relation instances, patterns). Evaluations are performed to assess the quality of the extracted results. If necessary, the extracted facts are compared with manually annotated examples, generated test sets, gold standards or existing knowledge sources. In many cases, classical information retrieval measurements are used [116]. It is common practice that evaluation of the extraction process results in refinement of algorithms, patterns, and corresponding implementations (applicable to machine learning and rule-based approaches).

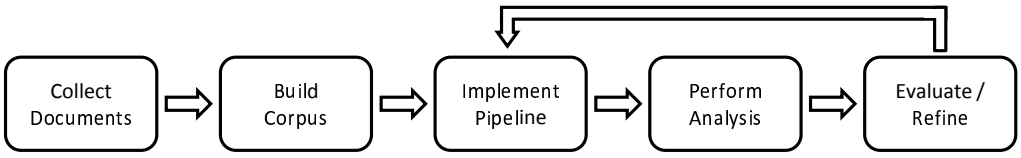


Figure 3.12: General procedure of information extraction

Redundancy and Paraphrases. Popescu [271] and Banko [272] analyzed large corpora (especially web document collections) and discovered that they include a lot of redundancy. Additionally, not only the same facts are mentioned in different documents, but natural language formulations of the same statements vary within documents and from document to document (paraphrasing [273]). This is not surprising, but a very important prerequisite for our text analysis procedure. Most information extraction systems focus on the identification of named entities (class instances) and their relationships by learning natural language patterns (e.g., *"X is located in*

Y" as a pattern for the relation CITYLOCATEDINCOUNTRY, example instance: (Berlin,Germany)). In contrast, as outlined before, our main interest is identification of relationships on a conceptual level. Consequently, we analyze the **mentions of concepts** in natural language text that occur redundantly and paraphrased as well, as illustrated in Figure 3.13.

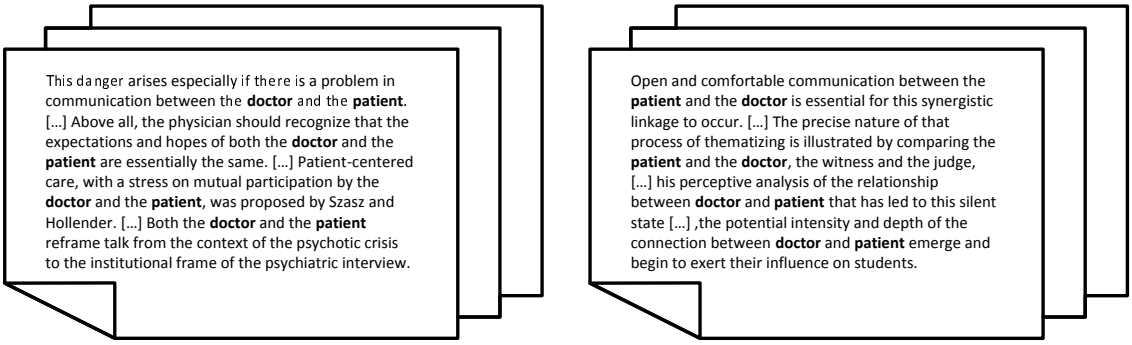


Figure 3.13: Examples of redundancy and paraphrasing in text documents for mentions of concepts; sentences were obtained from [274] using the Google Books search with the respective keywords

Word Co-occurrences and Distributions. For several decades, linguists have been arguing that *"certain lexical items tend to co-occur more frequently in natural language use than syntax and semantics alone would dictate"* [275]. This phenomenon, also known as collocation, was first popularized by Firth [209] in the Fifties. Closely related is the *distributional hypothesis* [208] that implies that words with similar meanings occur in similar contexts. These foundational works led to a number of studies in computational linguistics on how meanings can be derived from statistics of language use (e.g., statistical semantics [211]). Usually, these approaches depend on the analysis of large text corpora [189, 276]. Distributional semantics are still subject to research, for example in the field of measuring word similarity [277] or detecting multi-word expressions [278, 279]. It is important for our work that we can derive the **latent semantic relationship** between domain-specific terms from frequently co-occurring words and phrases. Additionally, it is possible to identify technical terms with the help of collocations [190].

N-Grams. Analyzing words and word sequences is a low-level method of statistical natural language processing [275] and is typically used to create probabilistic language models. These models predict the most likely next word for a sequence of input words and are used in OCR tasks, spelling correction, or machine translation [190]. An n-gram is a sequence of n consecutive words. An N-gram analysis determines how often a particular word or phrase appears in a text corpus. The result is an n-gram corpus containing these counts, which is usually filtered by very rare N-grams (frequency below a certain threshold) and contains sequences of up to 5 or 6 words. From these statistics, such as trigram counts, it is possible to determine the most likely subsequent word for two input words based only on the raw frequencies. The most interesting feature of N-grams is that they act as a **proxy for the original corpus** [280]. As a result, information extraction tasks can be performed on much smaller data. For the thesis, it is important that N-gram counts can be used to derive semantic relationships between co-occurring words and terms.

Part-Of-Speech and Technical Terms. One step in the syntactic analysis of natural language text is part-of-speech (POS) tagging. It is the process of determining the lexical category of each word in a sentence and a prerequisite for further analysis of the grammar. Available part-of-speech taggers [184] work with high accuracy. For the English language the widely adopted Penn Treebank tagset [182] is used. Recent advances in POS tagging suggest using simplified universal tagsets [183] to enable cross-language tagging. For example, such a tagset was used in the Google

N-Gram corpus with syntactic annotations [281]. The tags provided have the disadvantage that it is not possible to distinguish between normal nouns and proper nouns. In this dissertation, however, it is required to identify mentions of concepts and technical terms that normally are comprised of normal nouns. One approach to terminology extraction in text documents is the direct use of part-of-speech tags for keyword recognition. Justeson & Katz [282] analyzed that more than 99% of the multi-word noun phrases for technical terms consist only of nouns, adjectives and the preposition "of" and that these POS sequences can be used for filtering collocations. Feldman et al. [283] used lemmatized word forms for mining term taxonomies. Both approaches and recent work on part-of-speech tag analysis in the field of ontologies [284] confirm that **lexico-syntactic patterns** is a sophisticated method of extracting terms. Nevertheless, recent information extraction systems (e.g., PATTY [285], ReVerb [118]), which use POS patterns in a similar fashion, focus on learning relationships for later association of named entities. In this thesis, we adapt these methods for domain-independent concept extraction.

Extraction Approach of the Thesis. The following steps are taken to build an extensive domain-independent semantic network of terms (cf., Figure 3.14). The dissertation focuses on the extraction of semantically related terms directly from N-gram data. That is, in a first step, n-grams and their corpus frequencies are determined. The n-gram dataset serves as a proxy to the original text corpus. Therefore, it is not necessary to perform information extraction tasks on the original sentences, which would require orders of magnitude greater processing overhead. The identification of concept terms is done using a pattern-based approach that relies on shallow semantic features of sentence fragments. Relation extraction is based on distributional semantics of co-occurring terms. The semantic network aims at encoding identified terms as nodes in a graph and connects them with weighted edges if a relationship has been identified.

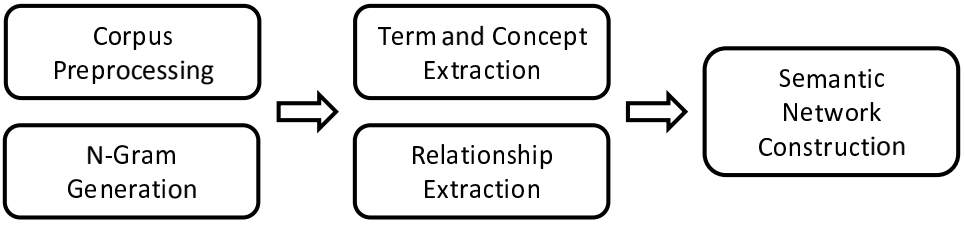


Figure 3.14: Text extraction approach of the thesis

3.8 Knowledge Acquisition from Knowledge Bases

This section details how knowledge bases are used to acquire additional structured information about domain-specific terms and their relationships. The creation of knowledge bases has a long tradition in research in artificial intelligence and is based on methods of ontology and knowledge engineering [166, 167]. Figure 3.15 shows the general concept of knowledge base creation for manual and automated methods from a practical point of view. Usually, an ontology schema is defined (see Figure 3.15, upper level in the knowledge bases). The schema is a vocabulary that defines concepts, properties, descriptions, and rules of ontology. It is often referred to as TBox and contains the terminological knowledge of ontology [286]. In most cases, schema definition is a manual step for both the manual and automated creation of knowledge bases. RDF Schema Language (RDFS) [261] and Web Ontology Language (OWL) [287] are the standards for the schema definition.

A knowledge base is populated by adding factual knowledge at the instance level (see Figure 3.15, lower level in the knowledge bases) using the vocabulary at the schema level. The instance level is called ABox and contains the assertional knowledge. In the manual case, both levels are created, managed and updated manually by ontology engineers. The automated knowl-

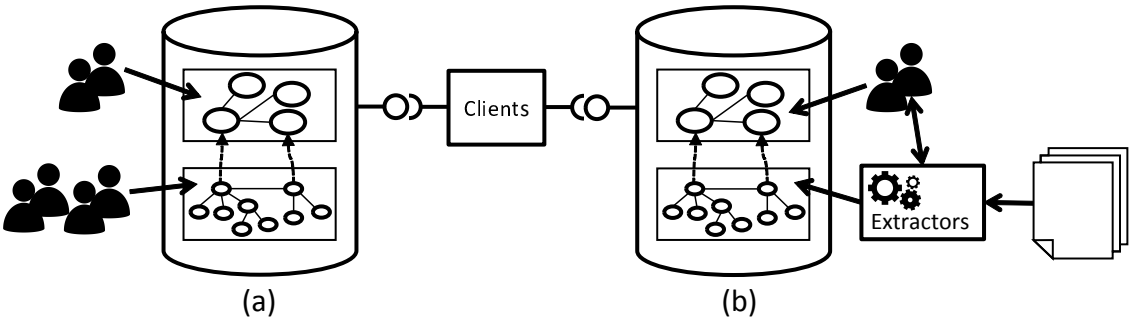


Figure 3.15: General concept of knowledge base creation and access: (a) Manual knowledge base creation. (b) Knowledge base creation with the help of automated extractors

edge base population will use extraction tools to acquire instance knowledge from external sources and automatically add facts to the knowledge base. The RDF language [288] is the standard representation for knowledge base facts in the form of subject-predicate-object triples (RDF statements).

RDF databases (triplestores) are responsible for storing and providing access to knowledge base data and related schema information. While small knowledge bases can be locally managed on a document-driven basis, extensive knowledge bases (e.g., DBpedia [103], Geonames [289]) require sophisticated management, indexing, searching, and access to the contained semantic data. A triplestore provides a SPARQL endpoint, which is a web-based interface for querying the knowledge base. Knowledge-based applications are built on top of knowledge bases using the SPARQL language [290] and the processing of the stored facts.

Existence of Appropriate Knowledge Bases. The development and use of ontologies in information systems has its roots in the biomedical domain [291, 292]. In the last two decades a shift from isolated, small, and hand-crafted ontologies to large, interlinked, and automatically generated knowledge bases can be observed [293]. Making ontologies and semantic datasets publicly available on the web was mainly driven by several Semantic Web and Linked Open Data initiatives [294, 103]. DBpedia, a knowledge base created from Wikipedia containing encyclopedic knowledge, became the central hub of the linked data cloud. Nevertheless, manually created knowledge bases are still required and are very reliable foundational works. The most prominent example is the lexical database WordNet [108, 257] on which many projects are based. Although the amount of linked and semantic data is growing[3], there is still a lack of large domain ontologies [167, 295].

Lexical Information in Knowledge Bases. Although the availability of knowledge bases has improved in recent years, the publication of linked data focuses almost exclusively on the instance level. It is common for knowledge bases to consist of a relatively small ontology schema describing the model of the data and a large amount of factual knowledge. On the one hand, to provide modeling suggestions at the conceptual level, lexical information and domain-specific terms hidden in ontology schemata must be used. On the other hand, specialized knowledge bases that focus on lexical knowledge have to be used. As mentioned earlier, WordNet [108] is the most widely used lexical database for the English language. Foundational ontologies that describe top-level concepts and common sense knowledge bases are also a source of domain modeling knowledge (e.g., Cyc [112], and Suggested Upper Merged Ontology (SUMO) [296]). The Lexicon Model for Ontologies (lemon) [297] is a promising vocabulary for the exchange of lexical information. So far, however, only a few ontologies use this model. The creation of linked data resources for linguistic applications is still a subject of research [298, 299, 300].

[3]E.g., see the Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. `http://lod-cloud.net/`

Of course, the creation of lexical resources for linguistic research and natural language applications does not exclusively refer to the ontology and linked data community. Thesauri and controlled vocabularies have been created for many years [267] (e.g., Roget's Thesaurus [301]). Many of them still exist in isolation, use proprietary data formats (e.g., OpenThesaurus[4]) or are not freely available (e.g., WordWeb[5]). Recently, several efforts have been made to convert publicly available vocabularies into RDF representations, such as: the Library of Congress Subject Headings (LCSH) linked dataset [302] (2008), EuroVoc and related European vocabularies [303] (2013), and Lexvo [304] (2015). Many RDF representations use the Simple Knowledge Organization System (SKOS) [265] because of the broader and narrower relationships that are defined in the SKOS vocabulary [305]. Nevertheless, there are many thesauri and vocabularies that use their own data models.

Heterogeneous Data Models. As described in the introduction, the variety of lexical resources poses the challenge of making unified access to lexical information more difficult. Heterogeneity exists on two levels: First, the schemata for lexical information organization differ from knowledge base to knowledge base. Although there are initiatives (e.g., Open Linguistics Working Group [306]) to harmonize lexical resources, this problem remains unresolved. An example of how the word *dentist* and its concept is modeled in WordNet, OpenCyc, and EuroVoc is shown in Figure 3.16. In WordNet, the literal "dentist" is a written representation that is linked to a canonical form of a particular sense (not shown in the illustration), that is in turn linked to lexical entry that belongs to a synset. In OpenCyc, "dentist" is a specific label type of an OWL class and an instance of a medical specialist type. In EuroVoc, "dentist" is a literal form of a preferred label of a thesaurus concept.

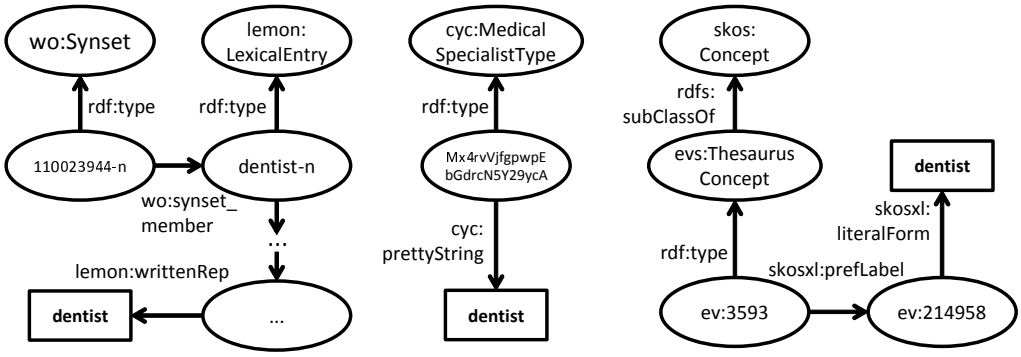


Figure 3.16: Knowledge base data model heterogeneity: Representation of the concept "dentist" in WordNet, OpenCyc and EuroVoc

Second, there is also a semantic heterogeneity. Depending on the scope of the knowledge base and the requirements of the creators, the actual lexical and conceptual information differs between the knowledge bases. In WordNet, for example, the more general concept (hypernym) of the dentist is "medical practitioner". In OpenCyc, a dentist is a kind of "prescriber" and an instance of "medical specialty". In EuroVoc, the dentist is associated with the broader term "health care profession".

Extraction Approach of the Thesis. It can be summarized that a lot of lexical information is already available as queryable RDF data. Vocabularies like SKOS and lemon aim at standardization but are not yet sufficiently implemented. Lexical information of terms and their relationships

[4] `https://www.openthesaurus.de/about/download`
[5] `http://wordweb.info/`

exist on schema level, intermediate proprietary data models and on instance level of knowledge bases. Our term queries can be partially answered using the available data. On the one hand, it is required to deploy transparent access to the diversity of lexical information. Ontology matching [307] would be one option to create a large unified lexical knowledge base. However, it has the disadvantages that a unified data model has to be developed, full access to all of the knowledge bases has to be provided and a computationally intensive alignment process has to be employed. The strategy of this thesis is to leave the knowledge bases as they are and to provide an easy plug-in mechanism that makes it possible to query and extract required information on demand (cf., Figure 3.17). On the other hand, it is required to integrate and match the obtained query results. The thesis aims at collecting as much information as possible in favor of recall and focuses on ranking strategies in the recommender systems.

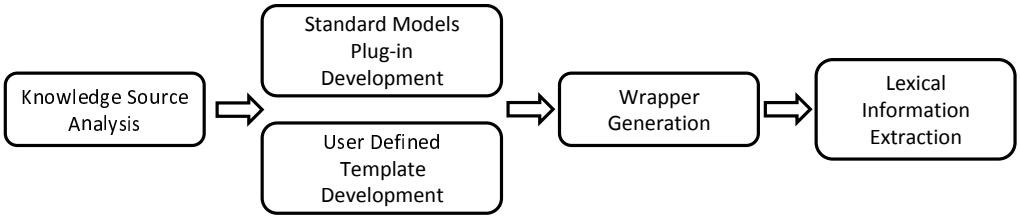


Figure 3.17: Knowledge base extraction approach of the thesis

3.9 Summary

In this chapter, we presented the detailed approach of domain modeling support with semantic model element suggestions. The modeling assistance follows an iterative procedure in which after each refinement step domain knowledge is acquired according to the domain-specific terms in the model and recommendations are derived and presented to the user to be used in the next modeling operation. We exemplarily chose domain modeling based on UML class diagrams, the most widely accepted standard for model-based software development in industry, to illustrate the modeling support. Nine modeling scenarios have been identified in which contextual information and search-based term autocompletion is provided. The assistance focuses on recommendations of class names, represented as nouns and noun phrases, as well as suggestions of association names, represented as verbs. The semantic relationships used in domain models have been mapped to lexical and knowledge modeling relationships as a basis for information extraction from text datasets and structured knowledge sources. We defined ten technology-independent queries that deliver the necessary domain-specific terms for the modeling recommendations. The implementation of these queries is twofold: First, pattern-based and n-gram-based information extraction from text corpora is developed to create a large body of terms and relationships that quantify the degree of semantic relatedness using statistical semantics. Second, extensible knowledge base querying is developed to retrieve lexical information and structured semantic relationships from heterogenous knowledge bases. Finally, the integrated information from extracted terms and connected knowledge bases is used in a recommendation system.

Chapter 4

SemNet: Extraction of Semantically Related Terms

One of the biggest difficulties of knowledge acquisition is that domain knowledge is largely contained in unstructured information sources. The proportion of structured information sources is very low compared to natural language information such as textbooks, dictionaries, encyclopaedias, requirement specifications or technical documentation. Over the past decade, several projects have proposed automated processing techniques to create large-scale factual knowledge resources from semi-structured and unstructured data, mostly focused on the instance level. As domain modeling takes place at the conceptual level, many of these sources cannot be used. This chapter develops novel methods and tools for extracting conceptual terms and relationships, addressing the third challenge of this thesis: the lack of conceptual knowledge resources.

4.1 Introduction

In this chapter we describe the automated construction of SemNet, a large-scale semantic network of terms and their relationships. SemNet is created using a huge natural language dataset that covers virtually every possible domain. We apply natural language processing to extract concept terms and multi-word expressions and their latent semantic relationships. SemNet comprises a large body of knowledge with almost 6 million noun and verb terms and over 355 million quantified binary and ternary relationships.

The chapter is structured as follows: In section 4.2 we compare our work to state-of-the-art systems. Section 4.3 describes in detail the complete text analysis process and respective implementations. We describe the Google Books N-gram dataset, dataset processing steps, linguistic analysis, concept extraction, relation extraction, extension of the analysis context and the construction of SemNet. Section 4.4 provides insights into analysis results of each step and discusses our findings. Finally, in Section 4.5 the semantic network is evaluated by comparing term and relationship coverage against existing semantic databases.

4.2 Related Extraction Methods

In this section we summarize the state of the art in related term extraction. We start with recent methods that cover certain aspects of extraction with respect to term detection, analysis models and data sources. After that we analyze latest approaches of neural word embeddings and describe how they relate to our work.

4.2.1 Keyword and Relationship Extraction

State-of-the-art systems for **keyword extraction** use supervised learning techniques or graph-based methods and external knowledge sources (e.g., [188, 186]). These approaches usually concentrate on grouping relevant keywords to determine the main topics of documents. Hasan and Ng [308] and Beliga et al. [309] provide recent surveys on keyphrase extraction and graph-based methods. In general, these approaches are not concerned with the relationships between terms that we require. Nevertheless, they use similar methods (e.g., syntactic information, N-grams) to select candidate terms from text documents and to group terms according to semantic relatedness measurements as we do.

There are three works on information **extraction from N-gram data** but with different goals. Tandon et al. [310] focus on the discovery of syntactical patterns in Web N-Gram data to populate isA, partOf, and hasProperty relationships based on seeds from ConceptNet. While the main goal is pattern extraction and evaluation, the approach does only consider single word terms as arguments for the relations. Also, the length of most of the patterns does not allow to extract longer terms from a fivegram context. Nulty et al. [311] also analyze Web N-Gram data, but in the context of collocations (noun-noun compounds). They use lexical patterns to determine semantic relations of the modifying word in a compound (e.g., *temporal* relationship in "summer travel"). Their work is similar to ours in the way compound nouns are identified. Lin et al. [312] summarize a workshop on N-gram analysis and provide a few tools for working with N-gram data and three use cases for information extraction from N-grams (gender and animacy knowledge discovery, noun phrase parsing, supervised classifiers).

There are several ways to determine the degree of **semantic relatedness** between identified terms in natural language texts. The relationship is often derived from statistics of word collocations [211], as introduced in Section 2.4.2. Other approaches use external knowledge sources to compute semantic relatedness. A very popular approach [313] is the use of Wikipedia-based explicit semantic analysis, and other works combine WordNet concept hierarchies and collaboratively constructed knowledge sources [314]. Zhang et al. [315] provide a survey on methods of lexical semantic relatedness. Comparison was very difficult because of different datasets, different background knowledge sources and different evaluation methods. They point out that general purpose knowledge sources (e.g., Wikipedia, WordNet, Wiktionary) have limited coverage of specialized vocabularies, and that there is a gap between the research of lexical semantic relatedness in general, in the biomedical domain and the application of general methods in other domains. Word context-based methods are a good starting point for measuring semantic relatedness as they do not require any additional background information. As described in Section 1.2.3 and confirmed by the aforementioned study, general purpose background sources are not sufficient to be applied to domain modeling scenarios in virtually every domain.

4.2.2 Word Embeddings

Traditional vector space models were predominantly used for word similarity tasks, but they faced challenges with sparse vector representations, dimensionality reduction, and scalability in using large corpora. In late 2013, work on neural network word embeddings [214, 215] (see also Section 2.4.4) has advanced a series of researches that surpassed the state-of-the-art in several natural language processing tasks, such as word similarity, part-of-speech tagging, word analogy, and text classification. The main purpose of these models and follow-up work is the syntactic and semantic analysis of the lexical meaning [213] to serve as background knowledge for NLP tasks. These types of word embeddings are optimized for fast training and low-cost vector operations (e.g., to calculate cosine similarity between word vectors), resulting in dense low-dimensional vectors with a user-defined fixed number of dimensions and vocabulary size. At first glance, neural word embeddings seem to be a good source for related terms for domain models, but there are some points in which they are very different from what we need. In the following, we will discuss these differences by analyzing four recent major approaches in this field:

- Word2Vec (2013) [215] - Distributed representations of words: A major breakthrough in neural word embeddings based on large input data.
- GloVe (2014) [316] - Global vectors for word representation: A count-based model using word-word co-occurrences.
- DEPS (2014) [317] - Dependency-Based Word Embeddings: A model that uses non-linear dependencies as context.
- Fasttext (2018) [318] - Word embeddings in several languages: Efficient text classification and representation learning.

Term Identification. All approaches are mainly concerned with **word** vector representations based on whitespace delimited strings. That is, all models learn vector representations for single tokens (unigrams) only. These tokens can be anything that occurs in a corpus: nouns, adjectives, verbs and other word classes, punctuation, URLs, numbers, etc. (e.g., they contain vectors for words like *"###-####"*, *"(309)"*, or *"ImageJPG"*). None of the works directly incorporates multiword expressions[1]. Word2Vec uses a trick while the corpus is preprocessed to integrate multiword terms. Phrases are identified based on a ratio between bigram and unigram frequencies. The spaces in these phrases are replaced by an underscore so that they are treated as individual tokens during the training phase. This trick requires several passes through a corpus and is in fact a simplified approach to mutual information. Although Fasttext is based on Word2Vec, none of the published models contain multiword terminology. In contrast, SemNet considers multiword expressions to be the number one priority, as they are very important for domain-specific vocabularies.

Vocabulary Size. Learned word representations normally use a user-defined vocabulary size by taking the n most frequent words of the corpus. The published pre-trained models of the respective approaches provide arbitrary vocabulary sizes: Word2Vec reports on 692K words [215] and publishes a model with 3M terms (containing multiword strings produced with the trick described before)[2]. The GloVe vectors have a vocabulary size of 400K or 2.2M depending on the corpus[3] used. Fasttext uses 1M (Wiki-News) and 2M (Common Crawl) word vocabularies[4]. The authors of DEPS did not publish any pretrained models. As shown before, these vocabularies contain any type of tokens. This also includes all plural and tense variations. Considering unigrams, a threshold of 1M or 2M words is absolutely feasible. Including more words would only introduce noise vectors. We analyzed the vocabularies of all available models by POS tagging each vocabulary entry and sorting out lines that contain numbers and special characters. We keep singular **normal nouns** (excluding proper nouns) and adjective noun combinations (in the case of Word2Vec) as we do in SemNet. The proportion of nouns in the respective word embeddings is on average 26%. SemNet contains 5.9M terms (5.8M singular nouns and singular adjective noun combinations and 0.1M verbs). Table 4.1 provides an overview.

Dimensionality. As mentioned at the beginning of this section, word embeddings are optimized for computational inexpensive vector operations to effectively compare words in NLP tasks. Consequently, all approaches aim to produce low-dimensional word vectors, either by directly learning dense representations or by dimensionality reduction in count-based models. Literature describes practical settings with sizes between 50 and 1000 dimensions [214]. The use of these models in a retrieval context (top-N recommendations) is rather ineffective, since for every query the entire vocabulary must be iterated, all similarities must be calculated, and then the closest Nth words are returned (see, for example, distance[5] implementation of Word2Vec). However, Sem-

[1] However, the models contain hyphenated words because they are not split during tokenization and therefore treated as a single token. Space-separated MWEs are not included. Mikolov et al. [318] state *"Directly incorporating the N-grams in the models is quite challenging as it clutters the models with uninformative content due to huge increase of the number of the parameters."*

[2] `https://code.google.com/archive/p/word2vec/`

[3] `https://nlp.stanford.edu/projects/glove/`

[4] `https://fasttext.cc/docs/en/english-vectors.html`

[5] `https://github.com/tmikolov/word2vec/blob/master/distance.c`

Net stores terms and their relationships as a graph so that top-N related terms can be retrieved directly.

Syntactic Information. Continuous vector representations created by Word2Vec, Fasttext, and GloVe are based on the linear context of words, such as those used by the CBOW and SkipGram models. Although the syntactic context is captured to some extent by these models, the actual syntactic information of the words is not taken into account. Dependency-Based Word Embeddings (DEPS) [317] use another context model that considers the dependency structure of sentences in a nonlinear way for vector representation learning. The resulting models capture the similarity between words (functional similarity) better than the relatedness between words (topical / domain similarity). This is also confirmed by a recent study on various dependence-based approaches [319]. Since our goal is to extract related terms in favor of domain similarity, these approaches do not help. Additionally, the resulting models are a purely mathematical representation of words as vectors that no longer contain syntactic information. The resulting SemNet model distinguishes between noun phrases and verbs and also provides subject-predicate-object structures for retrieval.

	Word2Vec GNews [215]	GloVe Wiki/Giga [316]	GloVe CCrawl [316]	Fasttext Wiki/News [318]	Fasttext CCrawl [318]	SemNet N-Gram [320]
Terms	Words+ Phrases[6]	Words	Words	Words	Words	Words+ MWEs
Voc. Size	3M	400K	2.2M	1M	2M	5.9M
Noun Terms[7]	462K (˜15%)	234K (˜58%)	599K (˜30%)	277K (˜27%)	605K (˜30%)	5.8M (˜99%)
Dimension	300	50/100/300	300	300	300	n/a
Syntactic Info	n/a	n/a	n/a	n/a	n/a	Nouns,Verbs SPO

Table 4.1: Comparison of Recent Word Embedding Approaches to SemNet

To summarize, none of the current word embedding models provides enough information that is directly usable in a domain modeling context and none of the approaches is able to generate models preserving syntactic information of words or phrases.

4.3 Extraction Process

4.3.1 Overview

To achieve our intended semantic modeling support with automated model element suggestions, we consider the following: We require a dictionary of terms that is big enough to cover a large portion of domains with all possible terms that are used in those domains. The terms should be interconnected if they are semantically related, thus constituting a semantic network. The degree of relatedness should be quantified to enable ranking of related terms. It is required to identify multi-word noun terms [321] and verbs as well as binary and ternary relations between them. The network should allow for retrieving related terms for a single term and for multiple terms at once. Figure 4.1 gives an overview of our approach.

[6] Phrases were obtained by a preprocessing trick using merged single words.
[7] Proper Nouns / Named Entities not included

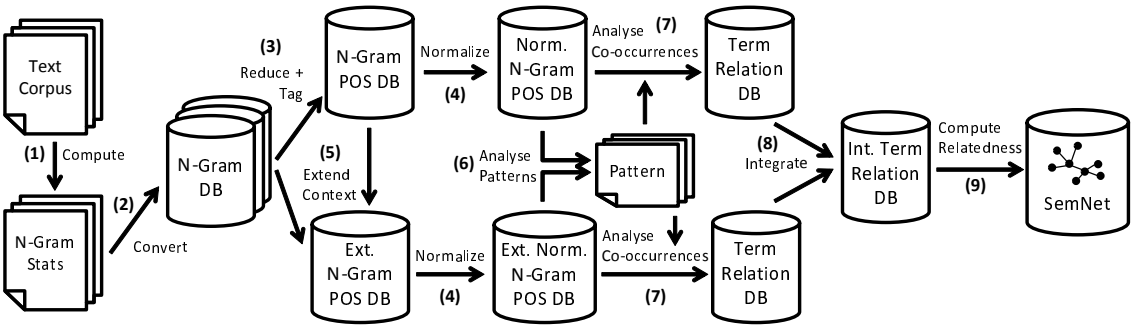


Figure 4.1: Procedure of creating a semantic term network based on natural language analysis

(1) The approach relies on automated analysis of natural language text to extract information about domain terms and their relationships. Input to the analysis process is a large text corpus of which frequencies for words and word sequences (N-grams) are determined. We do not build the text corpus ourselves but rely on an existing one. Nevertheless, all methods described in this chapter are applicable to other text datasets.

(2) We make use of the existing Google Books N-gram dataset, described in Section 4.3.2. First step of the preprocessing is the transformation of N-gram statistics delivered as plain text files into queryable databases (Section 4.3.3).

(3) We apply filtering to exclude useless N-grams (Section 4.3.4) and perform part-of-speech tagging on the reduced dataset (Section 4.3.5).

(4) Last preprocessing step is the normalization of word variants (Section 4.3.6).

(5) In parallel, we apply heuristic methods to extend the limited context of five words in the N-gram dataset to be able to extract longer domain-specific terms (Section 4.3.10).

(6) On both the normal and extended dataset we conduct pattern analysis and derive a set of syntactic patterns to extract terms (Section 4.3.7).

(7) The patterns are input to a co-occurrence analysis to extract term-term relationships (binary and ternary) for nouns and verbs (Section 4.3.8).

(8) Both resulting datasets of term relationships are aggregated separately (Section 4.3.8.2) and integrated (Section 4.3.10.3).

(9) Finally, based on term frequencies and statistical semantics, degrees of relatedness for the complete semantic network are computed (Section 4.3.9) and a queryable database is constructed (Section 4.3.11). SemNet is a large-scale graph of terms with weighted edges denoting the degree of relatedness, and it provides several interfaces for querying.

4.3.2 Google Books N-Gram Dataset

The Google Books project aims at providing a searchable digital library of a huge amount of books. Since 2004, Google Inc. has digitized over 15 million books [322] for full text book search on the web using optical character recognition. Most of the books are provided by university libraries or publishers who participate in the partner programs.

Google selected a subset of approximately 5 million books from the years 1500 to 2008 and built a text corpus of roughly 500 billion words in several languages for quantitative text analysis. Most of the corpus is in English (361 billion words), other languages are: French (45 billion), Spanish (45 billion), German (37 billion), Chinese (13 billion), Russian (35 billion), and Hebrew (2 billion) [322].

The complete natural language text was processed with an N-gram analysis that counts how often a certain word or word sequence occurs within the corpus. The resulting dataset includes word frequencies for all 1,2,3,4 and 5-grams that occurred at least 40 times. The N-gram frequencies are separated by years of publication allowing an analysis of word evolutions through time[8]. The dataset is split into languages, and can be downloaded[9] as tab-separated plain text files.

Different versions of the N-gram dataset exist. The first release of the dataset was published in 2009 [322]. It contains the 1,2,3,4 and 5-grams of the original text corpus, the publication year in which the N-grams occurred, the absolute frequencies of the N-grams, and in how many distinct books each N-gram occurred.

The second version of the N-gram dataset was published in the year 2012 [281] with several enhancements. Quality of the dataset was improved using revised OCR technology and more precise tokenization. Additionally, the new release provides syntactic annotations for each N-gram. These annotations include part-of-speech tags, sentence boundaries, and word dependency relations. The POS tags used in the dataset conform to a language independent tag set with 12 different lexical categories (nouns, verbs, adjectives, adverbs, pronouns, determiners and articles, prepositions and postpositions, numerals, conjunctions, particles, punctuation marks, and tags for all remaining word categories) [183].

In 2013 another dataset was released that is based on the English Google Books corpus [323]. It focuses on dependency tree fragments that were obtained by parsing the original sentences. The work includes more details than the syntactic annotations. In contrast to the other datasets the structure of the data is completely different because of the labeled dependencies between words.

When we started with our work on terminology extraction, we worked with the 2009 dataset and then later switched to the 2012 dataset for quality reasons. The descriptions in this chapter are based on the *English all dataset, version 20120701*. It was derived from the English Google Books corpus (approximately 361 billion words in total). The dataset is 2.5 terabytes in size and contains over 61 billion lines of text.

Table 4.2 shows the structure of the tab-separated files. The dataset contains N-grams with syntactic annotations as well as without annotations. As an example we show fivegrams. Most of our work is based on the fivegrams, because they offer the largest context for analysis. The first part of the table shows fivegrams without annotations. Each line represents the occurrences (*match count*) of the respective fivegram in the respective publication year and in how many different books it was found (*volume count*). The Second part of the table shows example fivegrams with part-of-speech annotations. The tags are concatenated to each of the words. Each N-gram in the dataset is contained at least twice (once without annotations and at least one variant with annotations). N-grams without annotations represent the occurrences regardless of the lexical categories. The same N-grams can have different syntactic annotations in different contexts. The last two segments of the table show two different versions of the fivegram *"the doctor and the patient"*. In a few contexts the word *patient* has an adjective role.

We decided to use the Google Books N-Gram dataset because it covers an extremely large variety of literature and terminology in almost every domain. It allows an analysis of terminology in great breadth.

4.3.3 Dataset Conversion

In this section, we describe the conversion process of the N-gram dataset plain text files into a relational database. We decided to conduct the transformation for the following reasons: (1) Size

[8] Evolution of words over time can be explored under `http://books.google.com/ngrams`

[9] The dataset can be downloaded at `http://books.google.com/ngrams/datasets`

N-Gram	Year	Match Count	Vol. Count
a doctor or a nurse	1998	116	113
a doctor or a nurse	1999	86	80
a doctor or a nurse	2000	110	104
...			
the doctor and the patient	2006	323	253
the doctor and the patient	2007	375	270
the doctor and the patient	2008	216	183
a_DET doctor_NOUN or_CONJ nurse_NOUN practitioner_NOUN	2002	13	13
a_DET doctor_NOUN or_CONJ nurse_NOUN practitioner_NOUN	2003	18	18
a_DET doctor_NOUN or_CONJ nurse_NOUN practitioner_NOUN	2004	33	13
...			
the_DET doctor_NOUN and_CONJ the_DET patient_NOUN	2006	313	244
the_DET doctor_NOUN and_CONJ the_DET patient_NOUN	2007	366	263
the_DET doctor_NOUN and_CONJ the_DET patient_NOUN	2008	211	179
...			
the_DET doctor_NOUN and_CONJ the_DET patient_ADJ	2006	10	10
the_DET doctor_NOUN and_CONJ the_DET patient_ADJ	2007	9	9
the_DET doctor_NOUN and_CONJ the_DET patient_ADJ	2008	5	5

Table 4.2: Structure of the Google Books N-gram dataset files. Examples of fivegrams without and with syntactic annotations

reduction: The huge size of the input files makes it difficult to analyze and process the data. Just extracting and copying the data files already takes days[10]. (2) Year aggregation: Our analysis does not require the separation into publication years. In order to make statements about the semantic relatedness between terms it is necessary to analyze total frequencies. (3) Querying: The conversion allows easier querying of the N-gram data to conduct manual pre-analysis steps, such as finding patterns and creating samples.

The conversion process is split into two steps. We first create a database containing an index of all possible text tokens (the words) using the unigrams of the Google Books Ngram dataset. Secondly, the other N-grams are converted into databases using the unigram index. As a result it was possible to reduce the size of the data by two orders of magnitude. The size reduction also allows us to perform most of the analysis tasks in the main memory.

Most of our subsequent analysis steps are based on the fivegrams of the dataset. In the following sections we mainly describe the steps and algorithms using examples of fivegrams.

4.3.3.1 Database Backend

We use an SQLite[11] database backend to store all N-gram information. Although the database system is not optimized for processing huge datasets it offers several important features. A complete database with all its metadata is stored in a single file. It is not necessary to install a database server. The database client directly accesses the database files. Database APIs are available for a variety of programming languages. Furthermore, SQLite offers a very easy mechanism to create, load and process in-memory SQL databases, which is important for us to reduce processing time.

[10] For example, the conversion process of the 2-gram data files (127 gigabytes in compressed format, roughly 843 gigabytes raw text files) took 16 hours.

[11] `http://www.sqlite.org/`

4.3.3.2 Unigrams

Figure 4.2 depicts the conversion of the unigram data file. The text file is parsed, years and volume counts are omitted, and match counts are aggregated. Each word together with its total corpus frequency is inserted into the unigram database table. The resulting table contains 10,254,948 distinct words/tokens. 385 megabytes disk space are required to store the database including an index on the words. Listing 4.1 shows the SQL statements used to create the unigram database table.

Figure 4.2: Process of the unigram data aggregation and database table creation (10 million unique tokens)

```
1 CREATE TABLE words (id INTEGER PRIMARY KEY, text TEXT, frequency INTEGER);
2 CREATE INDEX wordtext on words (text ASC);
```

Listing 4.1: Create table statements for the unigram database table

The conversion process turned the unigram text file with 1.4 billion lines of text (28 gigabytes) into a database with roughly 10 million distinct tokens (385 megabytes). Now we can determine the total number of tokens in the original text corpus by summing up the individual token frequencies: 935 billion tokens existed before the N-gram analysis. This value includes the number of spaces (468 billion), thus resulting in approximately 467 billion tokens (including punctuation).

4.3.3.3 Fivegrams

In this section we describe the conversion of the fivegram dataset text files. The dataset contains a mixture of two different fivegram variants: with and without part-of-speech tags. In the resulting fivegram database we include all fivegrams together with their part-of-speech tags. Table 4.3 lists all possible tags of the general Google part-of-speech tagset. The tagset itself is also stored in the database.

ID	Tag	Explanation	ID	Tag	Explanation
1	NOUN	Nouns, proper nouns	7	ADP	pre- and postpositions
2	VERB	Verbs	8	NUM	Numerals
3	ADJ	Adjectives	9	CONJ	Conjunctions
4	ADV	Adverbs	10	PRT	Particles
5	PRON	Pronouns	11	.	Punctuation marks
6	DET	Determiners, articles	12	X	Other

Table 4.3: Universal language independent part-of-speech tagset used by the Google Ngram dataset [281]

Listing 4.2 shows the SQL statements used to create the fivegram and the tagset database tables. Each word of a fivegram is stored as a separate foreign key reference (w1-w5) to the unigram table and each tag with a foreign key reference (p1-p5) to the tagset table. Please note that we do not use indices on the fivegrams so far, because at this stage of the analysis process we just require sequential processing of the fivegrams.

```
1 CREATE TABLE fivegramswithpos
2   (id INTEGER PRIMARY KEY, w1 INTEGER, w2 INTEGER,
3           w3 INTEGER, w4 INTEGER, w5 INTEGER,
4           p1 INTEGER, p2 INTEGER, p3 INTEGER, p4 INTEGER,
5           p5 INTEGER, f INTEGER);
6 CREATE TABLE tags (id INTEGER PRIMARY KEY, tag TEXT);
7 CREATE INDEX tagindex on tags (tag ASC);
```

Listing 4.2: Create table commands for the tagset database table and the fivegram database table

The conversion process of the fivegrams as shown in Figure 4.3 is similar to the unigram conversion. Occurrences of the same fivegrams in different years are aggregated. The part-of-speech tags are cut off from the words and stored separately. The resulting table only contains identifiers of the respective words and tags. 59.8 billion lines of text (2.4 terabytes) have been processed. The created database contains 704,355,409 fivegrams and requires 23 gigabytes disk space.

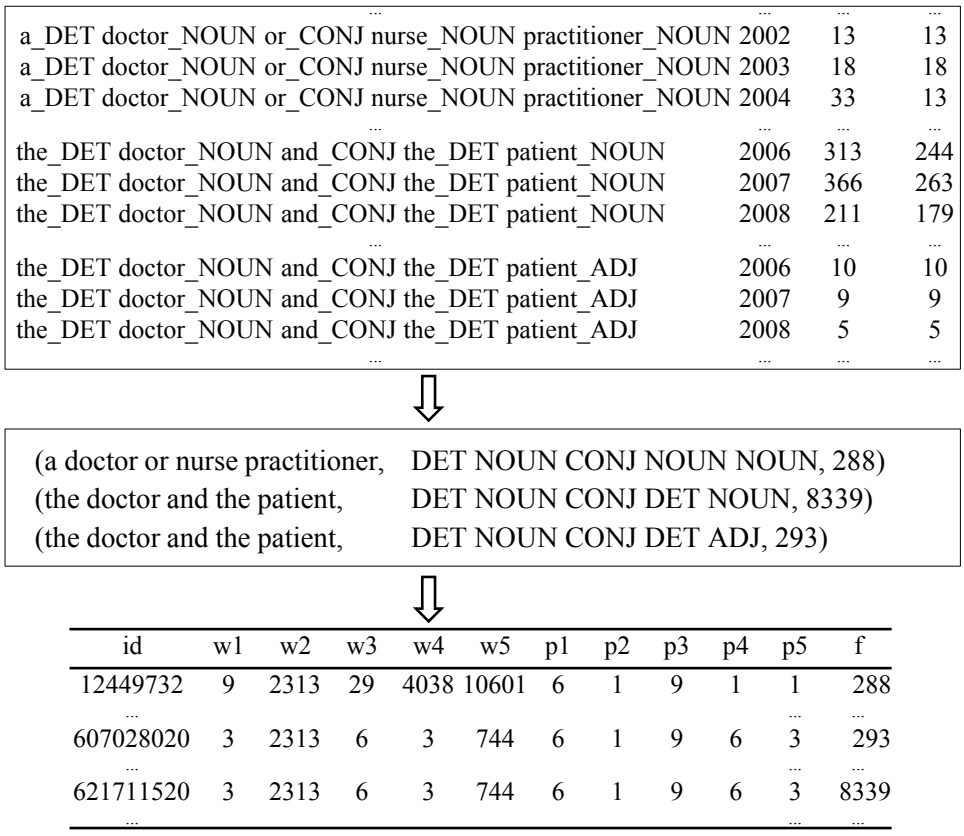


Figure 4.3: Process of the fivegram data aggregation and database table creation (700 million unique fivegrams)

4.3.3.4 Fivegram Queries

At this stage of the analysis process, it is already possible to query the N-gram database for specific phrase fragments. Listing 4.3 shows how to determine groups of fivegrams that contain the word *doctor* and the word *nurse* at certain positions. First, we retrieve the identifiers of both words. After that, we match all the fivegrams with the words on second and fourth positions. The results are sorted by decreasing frequency and limited to the first ten rows.

```
1  SELECT * FROM words WHERE text = 'doctor' or text = 'nurse';
2  --> 2313|doctor|19086489
3  --> 4038|nurse |10379228
4
5  SELECT words1.text, words2.text, words3.text, words4.text, words5.text, f FROM
       fivegramswithpos
6  LEFT JOIN words AS words1 ON words1.id = w1
7  LEFT JOIN words AS words2 ON words2.id = w2
8  LEFT JOIN words AS words3 ON words3.id = w3
9  LEFT JOIN words AS words4 ON words4.id = w4
10 LEFT JOIN words AS words5 ON words5.id = w5
11 WHERE w2 = 2313 and w4 = 4038
12 ORDER BY f DESC LIMIT 10;
```

Listing 4.3: Example query of the fivegram database to find fivegrams of specific words at specific positions. Please note that for reasons of space the select and join statements for the tags have been excluded

w1	w2	w3	w4	w5	p1	p2	p3	p4	p5	f
a	doctor	,	nurse	,	DET	NOUN	.	NOUN	.	2520
,	doctor	,	nurse	,	.	NOUN	.	NOUN	.	1924
your	doctor	or	nurse	if	PRON	NOUN	CONJ	NOUN	ADP	1734
your	doctor	,	nurse	,	PRON	NOUN	.	NOUN	.	1697
the	doctor	,	nurse	,	DET	NOUN	.	NOUN	.	1693
a	doctor	or	nurse	.	DET	NOUN	CONJ	NOUN	.	1340
the	doctor	or	nurse	.	DET	NOUN	CONJ	NOUN	.	1202
your	doctor	or	nurse	.	PRON	NOUN	CONJ	NOUN	.	1111
Your	doctor	or	nurse	will	PRON	NOUN	CONJ	NOUN	VERB	1083
a	doctor	or	nurse	,	DET	NOUN	CONJ	NOUN	.	1041

Table 4.4: Result of the fivegram query, showing the ten most frequent fivegrams that contain doctor and nurse at the second and fourth position, respectively

4.3.4 Dataset Reduction

For further analysis steps we built a stopword list that contains the most frequent non-noun and non-verb words. It is required to identify fivegrams that contain no information for our purposes and to identify words in multi-word expressions that were incorrectly tagged. The list was built by using the 1000 most frequent unigrams and by manually removing all nouns and verbs from it. The resulting list consists of about 500 distinct tokens, mainly comprising punctuation, conjunctions, pronouns, determiners and particles. Additionally to the list we implemented a stopword function that eliminates words that contain special characters (e.g., numbers, non-alphabetic characters).

The N-gram dataset was created using purely statistical means and therefore contains much information that is not relevant for extracting the domain terminology. Our analysis process aims to find terms that are expressed with adjectives, nouns and verbs. We process the complete fivegram database by iterating every fivegram and apply two rules: (1) If the fivegram contains four or five stopwords, it is discarded. (2) If it does not contain a noun, it is discarded. With these heuristics, we were able to reduce the dataset to 60 percent relevant fivegrams.

4.3.5 Part-Of-Speech Tagging

Although the Google N-gram dataset already contains part-of-speech tags, we performed part-of-speech tagging of the fivegram sentence fragments. There are two reasons why the general Google tagset is not sufficient for our purposes. First, it does not differentiate between proper nouns and normal nouns. This is an important issue, because we address modeling support on conceptual level and concepts are usually expressed as nouns or noun phrases using normal nouns [253]. Secondly, the Google tagset also does not differentiate between different word forms (e.g., plural nouns, verbs in the third person form or gerunds). In order to aggregate word variations it is required to reduce inflected nouns and verbs to their root form using stemming. This process requires the correct lexical category of the words.

ID	Tag	Explanation	ID	Tag	Explanation
2	NN	Noun, singular or mass	30	VBP	Verb, non-3rd person singular present
11	NNS	Noun, plural	38	VBZ	Verb, 3rd person singular present
9	NNP	Proper noun, singular	1	JJ	Adjective
33	NNPS	Proper noun, plural	13	JJR	Adjective, comparative
36	VB	Verb, base form	12	JJS	Adjective, superlative
17	VBD	Verb, past tense	44	:	Colon, hyphen, etc.[12]
19	VBG	Verb, gerund or present participle	32	SYM	Symbol
25	VBN	Verb, past participle	10	FW	Foreign word

Table 4.5: Excerpt of the Penn Treebank Tagset used by the Stanford Part-Of-Speech Tagger

The part-of-speech tagging of the fivegrams was performed using the Stanford Log-linear Part-Of-Speech Tagger[13] of the Stanford NLP toolkit [184]. The Stanford POS tagger uses the University of Pennsylvania (Penn) Treebank Tagset [182]. The tagset includes 46 different POS tags. Table 4.5 shows an excerpt of the tags that are relevant for our work.

The Stanford POS tagger includes several language-dependent tagger models. We use the English model that incorporates left-to-right dependencies and distributional similarity features (english-left3words-distsim.tagger) to tag the English fivegrams. The tagging process is as follows: We iterate through the reduced fivegram database table and rebuild the textual sentence fragments using the unigrams. Using the word and sentence structure of the tagger the fivegram text is processed by MaxentTagger component delivering the more detailed Penn Treebank tags. Afterwards, the tags are replaced in the database, as shown in Figure 4.4. The precise tags now allow the distinction between the different word forms that we use during the normalization procedure described in the next section.

4.3.6 Normalization

Words and multi-word expressions have different spellings in terms of their functions, positions in the sentence, and dependencies on other words. Normalization is the process of reducing the word variants to reduce the number of terms in the analysis result that mean the same but are spelled differently. We mainly perform normalization to eliminate capitalization and to determine the base forms of nouns, verbs, and adjectives.

We iterate through the fivegram database table and apply the following rules to each token (word) of each fivegram: (1) Unchanged words: Proper nouns are not touched, as they are usually

[12] Please note that the original definition of this tag only includes colons and semi-colons, but in practice it captures also hyphens and slashes. We use this tag to identify hyphenated words.

[13] The POS tagger can be downloaded at `http://nlp.stanford.edu/software/tagger.shtml`. We used version 3.5.1 of the tagger and its accompanied tagging models.

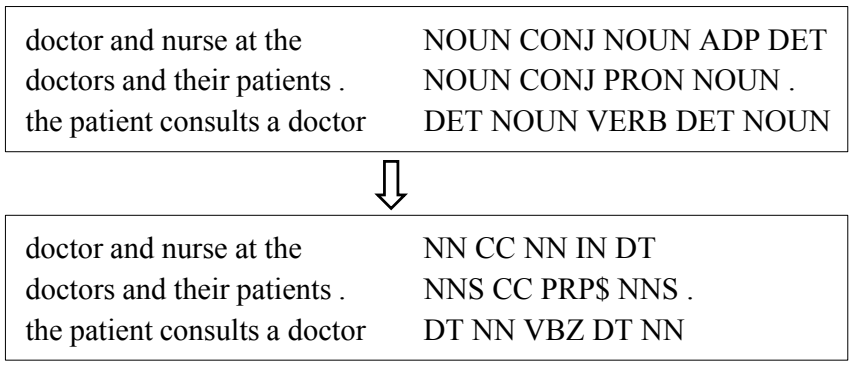


Figure 4.4: Part-of-speech tagging of the fivegrams, replacing the general Google tags with the more detailed Penn Treebank tags

written in uppercase. Words with more than one uppercase letter are not changed, since they usually refer to acronyms (except for plural variants, see the fourth rule). Stopwords are not changed. (2) Lowercasing: Adjectives (POS tags beginning with JJ), verbs (POS tags beginning with VB), and normal nouns (POS tag is NN or NNS) are lowercased. (3) Word forms: Plural nouns are reduced to their singular form. Verbs that use different tense, person or number are reduced to their base form. Comparative and superlative adjectives are reduced to their normal form. We use the stemming feature of the morphology component of the Stanford NLP kit for word form reduction. (4) Special Rules: We normalize the two different versions of a hyphen (- and –) to a single hyphen character. If an acronym uses a plural-s (all characters are uppercase except the s-suffix), it will be truncated. Figure 4.5 shows examples of the respective normalization rules.

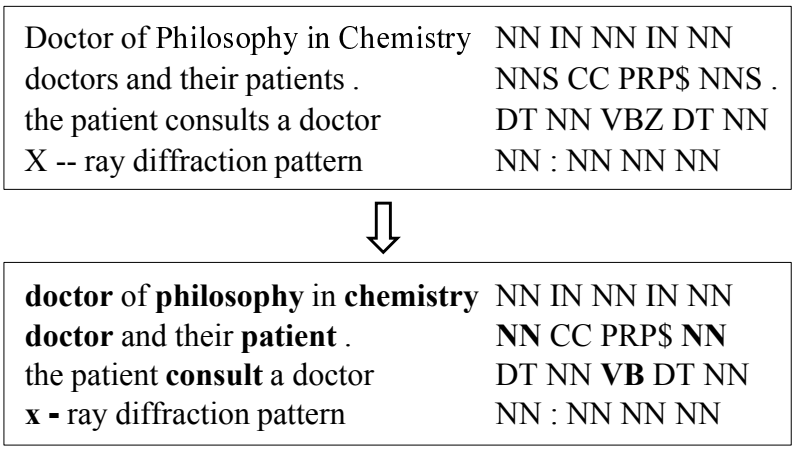


Figure 4.5: Examples of the applied normalization rules on the fivegrams

4.3.7 Syntactic Patterns

Terminology is expressed using single words or multi-word expressions. In order to find terms in N-gram natural language fragments we analyze sequences of part-of-speech tags [282]. The analysis is performed in two steps. (1) We first determine the most frequent part-of-speech patterns of terminology used in several existing dictionaries and ontologies. (2) We then complement the patterns by patterns specific to the N-gram dataset and formulate a set of extraction rules.

4.3.7.1 Automated Noun Pattern Analysis

First, we examine several existing English dictionaries, lexical databases, vocabularies and ontologies on how concept names, class names and noun terminology are expressed. The most important resource is the lexical database WordNet [108], because it allows to retrieve English terminology that is already classified as nouns, verbs and adjectives. We extract all nouns excluding those that refer to instance synsets (e.g., city names). The complete list of noun terms contains 104,182 entries. We also analyze these databases: DBpedia ontology (3,475 classes and properties), OpenCyc (108,937 classes), Schema.org (628 classes), ConceptNet (2.3 million concepts), and Linked Open Vocabulary (48,005 classes). The part-of-speech tags are determined by tagging each term of the respective resource using the Stanford POS tagger. Most of the terms consist of only one or two words. The results of the tagger have to be treated with care, because the missing context (a sentence) leads to imprecise tagging results. We cross-validated the tagger results by looking up each term in the respective N-gram dataset and retrieved their most frequent part-of-speech tags.

Table 4.6 summarizes the results of the automated part-of-speech pattern analysis. For each database we provide the number of analyzed terms and the ten most frequent syntactic patterns. The following observations with respect to the obtained patterns have been made.

4.3.7.2 Dominant patterns

The most obvious result of automated tagging is that the most frequent syntactic patterns are a single noun, two subsequent nouns and an adjective followed by a noun. On average 61.3% of the terms conform to these three patterns. The percentage of these patterns in OpenCyc and ConceptNet is comparatively low because these two databases tend to have more concepts with detailed, long labels (e.g., OpenCyc includes a lot of labels such as "*formation of symbiont germ tube hook structure on or near host*"). Additionally, ConceptNet includes a large number of concepts that are described with short natural language phrases, not only referring to noun terminology. (e.g., "*more important than others to remember*"). Hence, the distribution of the patterns is more diverse.

4.3.7.3 Important patterns

In the top ten list of patterns, additionally important patterns appear that have up to four parts. These include noun combinations up to four nouns and adjective-noun combinations up to four parts. Another noteworthy pattern is a noun followed by a preposition followed by a noun. This pattern refers to terms such as *date of birth* or *master of science*. Variations are two nouns or an adjective and a noun after the preposition. The preposition pattern unfortunately has some drawbacks. It also extracts multi-word terms that are idioms used in almost every context (e.g., *number of doctors*, or *part of life*). Nevertheless, these combinations can be sorted out using statistical measures applied later on.

4.3.7.4 Impractical patterns

A fairly common tag is a single adjective. This is obviously a wrong POS pattern. The first reason for this result is the missing context for the POS tagger, resulting in incorrect tags for homonymous words, such as adjectives and nouns having the same spelling. Surprisingly, the single adjective is the fourth most frequent pattern in the set of WordNet nouns. The reason is

	WordNet Noun synset members (104,182)		**DBpedia** Class & property labels (3,475)	
1	[Noun]	44,3%	[Noun]	31,6%
2	[Noun] [Noun]	27,1%	[Noun] [Noun]	23,5%
3	[Adjective] [Noun]	12,9%	[Adjective] [Noun]	11,9%
4	[Adjective]	2,2%	[Noun] [Noun] [Noun]	4,5%
5	[Verb gerund]	1,3%	[Noun] [Prepo.] [Noun]	4,5%
6	[Verb gerund] [Noun]	1,1%	[Adjective] [Noun] [Noun]	3,2%
7	[Noun] [Noun] [Noun]	1,0%	[Verb gerund] [Noun]	1,6%
8	[Noun] [Prepo.] [Noun]	0,9%	[Adjective]	0,8%
9	[Adjective] [Noun] [Noun]	0,9%	[Noun] [Adjective] [Noun]	0,8%
10	[Foreign Word]	0,8%	[Noun] [Prepo.] [Adj.] [Noun]	0.8%
	Schema.org Class names (628)		**OpenCyc** Class labels and aliases (108,937)	
1	[Noun] [Noun]	37.7%	[Noun] [Noun]	15.1%
2	[Noun]	20.2%	[Noun]	13.0%
3	[Adjective] [Noun]	10.8%	[Adjective] [Noun]	8.3%
4	[Verb] [Noun]	9.7%	[Noun] [Noun] [Noun]	7.1%
5	[Noun] [Noun] [Noun]	5.9%	[Adjective] [Noun] [Noun]	4.1%
6	[Adjective] [Noun] [Noun]	4.1%	[Noun] [Noun] [Noun] [Noun]	3.0%
7	[Verb gerund] [Noun]	1.3%	[Noun] [Adjective] [Noun]	1.6%
8	[Noun] [Conj.] [Noun] [Noun]	1.0%	[Adj.] [Noun] [Noun] [Noun]	1.4%
9	[Noun] [Noun] [Noun] [Noun]	0.8%	[Verb gerund]	1.3%
10	[Noun] [Conj.] [Noun]	0.8%	[Adjective] [Adjective] [Noun]	1.2%
	ConceptNet Concept names (2,315,095)		**Linked Open Vocabulary** Class labels (48,005)	
1	[Noun] [Noun]	28.3%	[Noun]	34.5%
2	[Noun]	20.7%	[Noun] [Noun]	16.3%
3	[Adjective] [Noun]	6.5%	[Noun] [Noun] [Noun]	6.1%
4	[Noun] [Noun] [Noun]	6.1%	[Adjective] [Noun]	5.1%
5	[Adjective]	2.6%	[Adjective] [Noun] [Noun]	1.7%
6	[Adjective] [Noun] [Noun]	1.8%	[Adjective]	1.6%
7	[Noun] [Prepo.] [Noun]	1.6%	[Noun] [Noun] [Noun] [Noun]	1.6%
8	[Noun] [Adjective] [Noun]	1.3%	[Noun] [Prepo.] [Noun]	1.5%
9	[Noun] [Noun] [Noun] [Noun]	1.3%	[Verb] [Noun]	1.4%
10	[Verb] [Noun]	1.1%	[Verb] [Noun] [Noun]	0.7%

Table 4.6: Results of the automated syntactic analysis of terms in lexical and semantic databases

that WordNet contains a large number of synsets that refer to nationalities (e.g., synset {*French*}) and to languages (e.g., synset {*German language, German*}), that cannot be distinguished from the respective adjective by the tagger. Furthermore, a lot of synsets contain synonyms that are an adjective short form of the actual noun (e.g., *Abyssinian cat* and *Abyssinian*).

The verb in gerund form and the gerund followed by a noun are also part of the top ten list. In general, these patterns are not wrong, but difficult to apply for extraction. For example *answering machine* would be a correct term with respect to our extraction goals, but in *he enjoys playing football* the extraction would be incorrect. In case there is enough context available in front of the term the extraction of the latter example could be avoided with exception rules.

4.3.7.5 Special Noun Patterns

The Google Ngram dataset was created applying a tokenization that separates hyphenated words into several single tokens (e.g., *first-aid kit* will be represented by a four-gram {*first, -, aid, kit*}). Patterns for these cases could not be detected by our automated pattern analysis, because hyphenated words are usually treated as single nouns. The extraction of these terms from the fivegrams requires a few extra patterns. The part-of-speech tagging of the separated hyphenated words results in the detection of single lexical categories separated by punctuation. For "*first - aid kit*" the categories are [Adjective] [Punctuation] [Noun] [Noun]. A few variants are possible with respect to how the fivegrams are classified (noun or adjective-noun combinations after the hyphen). If the part before the hyphen is just one or two characters long (e.g., {*x, -, ray*}), it will be tagged as a symbol. Additional patterns are required for these cases.

4.3.7.6 Verb Patterns

The goal of this thesis is not only to identify noun terminology, but also to identify relationships between nouns and verbs to suggest relationship names in a domain model. Therefore, part-of-speech patterns for verbs are required as well. The extraction is straightforward: patterns of part-of-speech tags for single verbs and sequences of verbs are matched with the N-grams. Individual verbs are extracted directly. In sequences of verbs, the last verb is always retrieved. This simple and effective heuristic extracts in almost all cases the verb that carries the semantic content of the phrase and skips modal and other auxiliary verbs as well as auxiliary verbs required for participles or passive constructs (e.g., in the fivegram *the doctor has been consulted* the normalized noun-verb relationship {doctor,consult} is extracted.

4.3.7.7 Implemented Extraction Patterns

In this section we present the complete set of part-of-speech patterns that was built based on the previously described pattern analysis. As described in Section 4.3.5, the reduced fivegram dataset was tagged using the Penn Treebank tagset. Table 4.7 summarizes all patterns that are used during the extraction process. For each pattern we list the respective sequence of part-of-speech tags and provide examples of terms that will be extracted by the pattern. Patterns 1-11 extract terms consisting of different adjective-noun combinations. Patterns 12-17 are responsible for extracting hyphenated terminology. Patterns 18-20 extract terms that make use of a preposition. Patterns 21-24 extract verbs and verb combinations from the fivegrams.

Note that some of the patterns identify terminology consisting of four parts. A four-word term in a fivegram leaves space for a single word before or after the term to establish a relationship with it. It is not possible to derive a noun-noun relationship in this case. Maximally, we can identify a noun-verb relationship. All terms with four words are stored anyway and can later be connected to other terms in a context extension step (see Section 4.3.10).

For each of the 24 patterns we derive a regular expression that determines the position of a term in a fivegram. The list of regular expressions is extensible and will be loaded dynamically by the extraction component.

ID	Pattern	Example
1	NN	the **doctor** and the **nurse**
2	NN NN	**family doctor** for a checkup
3	NN NN NN	advice of **health care provider**
4	NN NN NN NN	a **health care team member**
5	NN JJ NN	**DNA double helix** , chromatin
6	NN JJ NN NN	the **mouse mammary tumor virus**
7	JJ NN	**medical care** and treatment .
8	JJ JJ NN	nose and **upper respiratory tract**
9	JJ JJ NN NN	**nuclear magnetic resonance spectroscopy** ,
10	JJ NN NN	platelet or **white blood cell**
11	JJ NN NN NN	**coronary artery bypass surgery** was
12	JJ : NN	**sick - bed** to attend
13	JJ : NN NN	the **first - aid kit**
14	NN : NN	**birth - rate** has been
15	NN : NN NN	the **doctor - patient relationship**
16	SYM : NN	**x - ray** of chest
17	SYM : NN NN	a **semi - government authority**
18	NN IN NN	received a **doctor of philosophy**
19	NN IN NN NN	**type of health insurance** that
20	NN IN JJ NN	a **doctor of dental surgery**
21	VB	**consult**
22	VB VB	**was consulted**
23	VB VB VB	**has been consulted**
24	VB VB VB VB	**may have been consulted**

Table 4.7: Implemented part-of-speech pattern to identify noun key terminology and associated verbs, based on the Penn Treebank tagset. Note that the verb pattern examples are not normalized for better readability

4.3.8 Co-occurrence Analysis

In this section, we describe how to apply the syntactic patterns to extract co-occurring terms from fivegrams, and how to aggregate the extraction results.

4.3.8.1 Extraction of Related Terms

The identification of semantically related terms is grounded in the *Distributional Hypothesis* first discovered by Harris [324] in the fifties. It describes that words with similar meanings occur in similar contexts. In our case the context is a five word window given by a fivegram. The absolute frequencies of the fivegrams provide information on how often a specific context occurred. Consequently, terms that co-occur more often have a stronger relationship.

In corpus linguistics the extraction of words and their surrounding words is usually referred to as collocation. The application of the previously described syntactical patterns extracts certain collocations from the fivegrams. The goal of the extraction process is to identify at least two noun terms or at least one noun term and one verb in a fivegram. Noun-noun relations require the collocations to be separated by at least one word. Therefore, the respective POS tag of the separation word must not be part of the syntactical patterns (e.g., a coordinating conjunction). In the extraction of noun-verb relations the patterns can be consecutive.

Pattern matching is applied in a hierarchical and non-overlapping way. Figure 4.6 shows possible matching scenarios and what kind of relations are extracted from the fivegrams. Each regular expression of the respective POS pattern is applied to the POS tag sequence. If applicable,

the matching determines a set of positions for each pattern. After that all overlapping matches on the same level and all consecutive matches are discarded. A longer match on a higher level discards all matches on the respective lower levels.

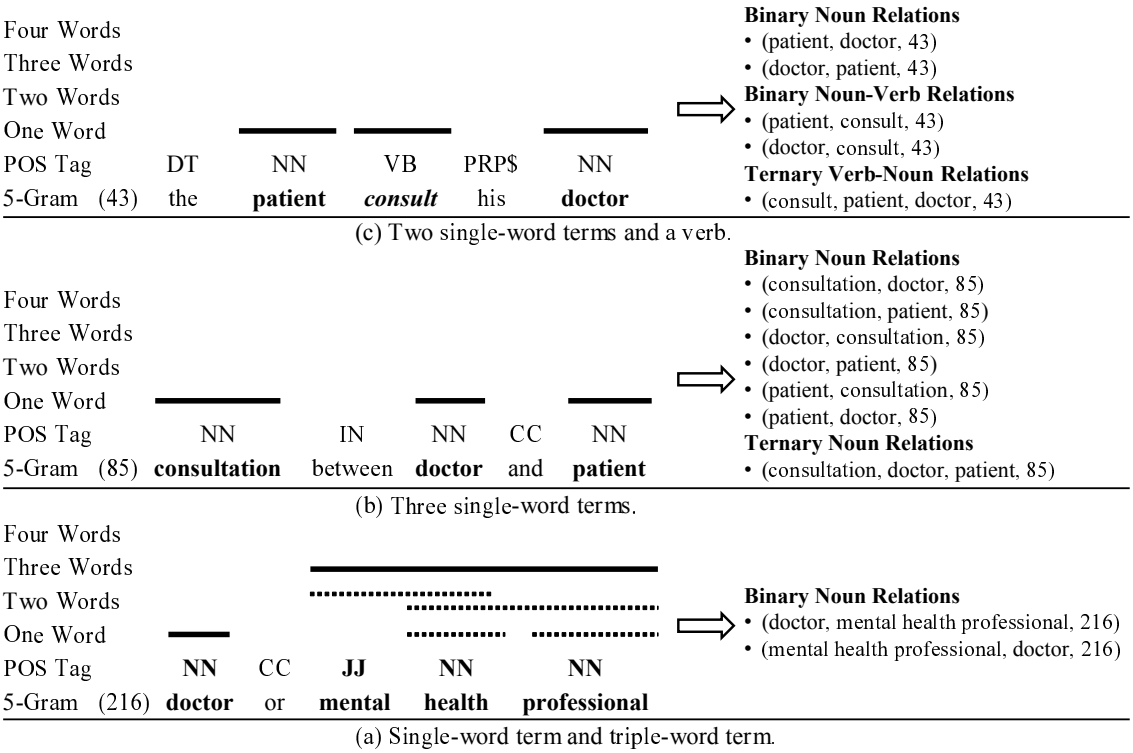


Figure 4.6: Hierarchical POS pattern matching to determine positions and relations of terms in the fivegrams

For example, in Figure 4.6a the single noun pattern (NN) occurs three times on the lowest level. Two double-word matches and one triple-word match are determined. The JJ NN NN pattern match on the highest level remains and the single noun in the first position is kept because they do not overlap with another term and are separated by conjunction from each other. From that fivegram two binary noun-noun relations are extracted: *(doctor, mental health professional)* occurred 216 times and vice versa. It is necessary to store both directions, because each term will occur in different contexts, thus its collocation will be considered separately.

In Figure 4.6b the pattern matching is simple: three single word terms will be determined. The extraction records all term combinations together with their absolute frequencies (six relations). The example illustrates that additionally a ternary relation will be stored.

Figure 4.6c depicts the extraction of noun-verb relationships. These relationships are extracted if at least one verb and one noun occur in the fivegram. In the example, the fivegram contains two single-noun terms and a verb. Three different types of relations will be extracted: Noun relations between *doctor* and *patient*, noun-verb relations between *consult* and *doctor* and *patient*, respectively, and a ternary verb-noun relation.

The results of the co-occurrence analysis are large occurrence and co-occurrence tables (roughly 1 billion rows in total). For each type of relationship a separate table is created. During the analysis the extracted terms are inserted into the tables together with absolute frequencies of the fivegram in which they occurred. Listing 4.4 shows the respective SQL statements to create these tables. A noun term can consist of up to four parts (*termw1, termw2, termw3, termw4*). Table *nouncooccurrences* stores binary noun term relationships (a term, a related term and the frequency f). Table *nounverbcooccurrences* stores co-occurrences of noun terms and verbs. Ternary relations

are stored in the respective tables shown in Listing 4.4. Finally, we also store in a separate table how often each extracted noun term occurred and by which pattern it was detected (table *singlenounoccurrences*). The information is required for later relatedness computation. The tables for single occurrences also include terms for which no relation could be detected (e.g., all four-word terms).

```
1  CREATE TABLE nouncooccurrences (termw1 INTEGER, termw2 INTEGER, termw3 INTEGER,
       termw4 INTEGER, relw1 INTEGER, relw2 INTEGER, relw3 INTEGER, relw4 INTEGER, f
       INTEGER);
2
3  CREATE TABLE nounverbcooccurrences (termw1 INTEGER, termw2 INTEGER, termw3 INTEGER,
        termw4 INTEGER, verb INTEGER, f INTEGER);
4
5  CREATE TABLE nounternarycooccurrences (term1w1 INTEGER, term1w2 INTEGER, term2w1
       INTEGER, term2w2 INTEGER, term3w1 INTEGER, term3w2 INTEGER, f INTEGER);
6
7  CREATE TABLE nounverbternarycooccurrences (term1w1 INTEGER, term1w2 INTEGER,
       term1w3 INTEGER, term2w1 INTEGER, term2w2 INTEGER, term2w3 INTEGER, verb
       INTEGER, f INTEGER);
8
9  CREATE TABLE singlenounoccurrences (termw1 INTEGER, termw2 INTEGER, termw3 INTEGER,
        termw4 INTEGER, pattern INTEGER, f INTEGER);
10
11 CREATE TABLE singleverboccurrences (verb INTEGER, f INTEGER);
```

Listing 4.4: Create table commands for the extracted relationships tables

4.3.8.2 Duplicate Aggregation

The result of the co-occurrence analysis contains a huge number of duplicates, because the same terms (co-)occur in different contexts. Consequently, we aggregate frequencies of duplicate entries in the respective single occurrence and co-occurrence tables. The necessary database operation is a simple aggregation of frequencies using the sum and group by operators. Due to the large size of the database tables we implemented an in-memory hashmap aggregation, which does not require separate sorting of the input data. Hence, we were able to optimize processing time by only sequentially reading the tables, dynamically building the hashmaps, and then writing the result back to disk. Figure 4.7 shows examples of (a) noun-noun relations and (b) single noun occurrences. For reasons of readability the tables contain the textual representation of the internal word identifiers.

There are two special rules for single occurrences and ternary co-occurrences: The table for aggregated single occurrences also includes the pattern by which the term was detected. As the part-of-speech tagging is non-deterministic for different contexts, the same term could be detected by multiple patterns. In those cases, the final result includes the most frequent pattern. In contrast to binary relations, the order of ternary relations is not important. For aggregation we sort the noun terms by ascending identifiers. Hence, we can avoid duplicate entries, e.g., for noun-noun-verb relations, such as *(doctor, patient, consult)* and *(patient, doctor, consult)*.

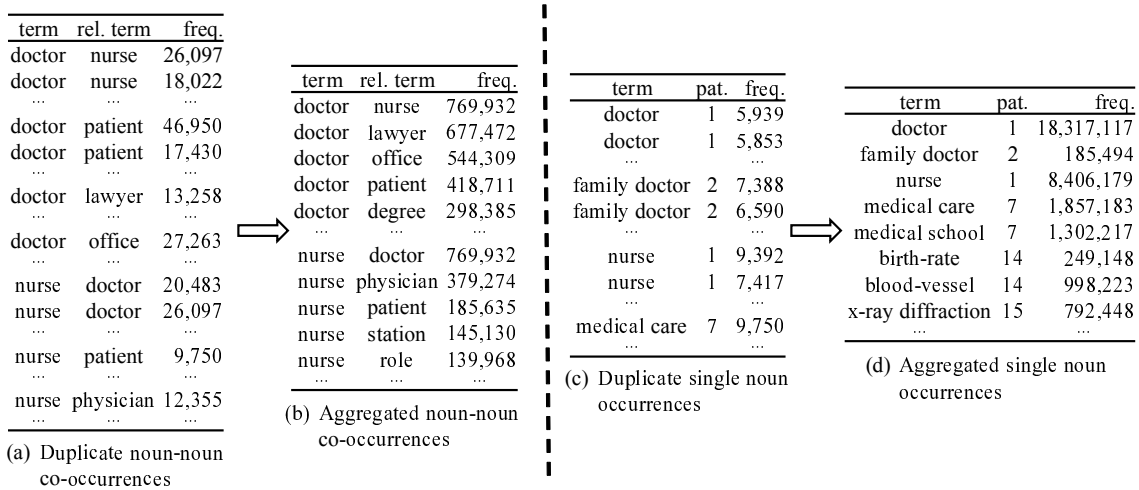
At this stage of the analysis process, we can already query a term for related terms ordered by strength of the relationship using the total absolute frequencies.

4.3.9 Relatedness Computation

Having obtained the total frequencies of co-occurring terms and occcurrences of individual terms, we can now determine the degree of relatedness between extracted terms. We calculate two relatedness measures: Relative frequency and (Normalized) Pointwise Mutal Information.

4.3.9.1 Relative Frequency

The result of the binary relationship extraction contains absolute frequencies for term pairs. For each extracted term, a set of related terms is known along with each frequency. To compare



(a) Duplicate noun-noun co-occurrences

(b) Aggregated noun-noun co-occurrences

(c) Duplicate single noun occurrences

(d) Aggregated single noun occurrences

Figure 4.7: Aggregation of duplicate extracted co-occurrences (a)+(b) and occurrences (c)+(d) (textual representation of the database content)

the relatedness between terms, we calculate the relative frequencies for each set of related terms. Let $T = \{t_1, \ldots, t_n\}$ be the set of terms extracted from the fivegrams (n denotes the number of distinct terms in the binary relationship extraction result). For every $t_i \in T$ there exists a set of tuples with related terms r and their absolute occurrences o: $R_i = \{(r_1, o_1), \ldots, (r_m, o_m)\}$ (m denotes the number of related terms). For each r_j we calculate the relative frequency f_j using Equation 4.1.

$$f_j = \frac{o_j}{\sum_k o_k} \tag{4.1}$$

Relative frequencies can be easily calculated for noun-noun relationships. As mentioned in Section 4.3.8.1, terms extracted during co-occurrence analysis are stored in the co-occurrence table in the following way: If a fivegram contains *term1* and *term2*, two entries are inserted: *(term1,term2)* and *(term2,term1)*. We give an example of why it is important to treat the direction of the relationship separately.

The relationship between *doctor* and *patient* is stored using two entries: *(doctor,patient,418711)* and *(patient,doctor,418711)*. Of course, the absolute frequency is the same for both entries. After calculating the relative frequencies, the entries are as follows: *(doctor,patient,0.03893)* and *(patient,doctor,0.00765)*. The reason for the difference is that, from the doctor's point of view, the relationship with the patient is stronger, but from the patient's point of view there are other more important relationships and there are also many more relationships (the corpus frequency of *patient* is about three times higher than of the term *doctor*)).

Relative frequency can be used to perform top-k queries on relationships to retrieve rankings of related terms. One of the disadvantages of relative frequency is that very general terms tend to be ranked more highly because they occur very often and in almost every context (e.g., the term *time* is in the top 25 related terms of *doctor*, *nurse*, *patient*, and *physician*).

4.3.9.2 Pointwise Mutal Information

Pointwise mutal information (PMI) measures the dependency between the probability of coinciding events and the probability of individual events [325]. The general definition is as follows: X and Y are discrete random variables and x and y are particular outcomes of X and Y, respectively. PMI is the ratio of their actual joint probability to the joint probability assuming independence [189] as defined in equation 4.2.

$$pmi(x, y) = \log \left[\frac{p(x, y)}{p(x)p(y)} \right] \tag{4.2}$$

The application of PMI to the extracted terms obtained from the fivegrams means that the observed events x and y terms[14] are extracted from the fivegrams. PMI of the terms relates the probability of their coincidence $p(x, y)$ with the probabilities of observing both terms independently $p(x)p(y)$. PMI is an associativity score of two terms considering their individual corpus frequency. The individual probabilities are estimated by counting the occurrences $f(x)$ and $f(y)$ within the complete text corpus. These frequencies can be either obtained from the 1/2/3/4-gram datasets (depending on the number of words that make up a term) or from a table of individual noun occurrences created during the extraction process. The probabilities $p(x)$ and $p(y)$ are then computed by dividing $f(x)$ and $f(y)$ by the corpus size N. The probability of co-occurrence $p(x)p(y)$ is calculated by dividing $f(x, y)$ (obtained from the noun co-occurrence table) by the corpus size N.

In order to compare associativity scores with each other, there exists a normalized variant of PMI. Normalized pointwise Mutal Information (NPMI) is calculated according to the Equation 4.3. It normalizes PMI values between [-1,+1].

$$npmi(x, y) = \frac{pmi(x, y)}{-\log\left[p(x, y)\right]} \tag{4.3}$$

We show an example illustrating the performance of NPMI. The extraction process has detected that the term *doctor* co-occurred with *order* and *dentist* at a similar absolute frequency (about 170,000 times). Considering the absolute or relative frequency, both related terms receive a similar degree of relatedness. NPMI of both terms takes into account that *order* is a very general term that occurs in many contexts. Therefore, there is no interesting relationship between *doctor* and *order*. Consequently, *npmi(doctor,order)* is much lower than *npmi(doctor,dentist)*.

However, PMI has the disadvantage that rare events (low frequencies) are rated with high associativity. This happens especially when a term very rarely occurs together with another term. For example, the term *fiction audience* has only occurred with *doctor* and only a few times, but *npmi(doctor,midwife)* is almost the same *npmi(doctor,fiction audience)*. The latter, of course, is not an interesting relationship in a medical context.

All three associative measures (relative frequency, pmi and npmi) are calculated for each binary relationship and stored in the resulting database.

4.3.9.3 PMI Computation for Ternary Relations

In order to calculate PMI scores for ternary noun relationships and ternary noun-noun-verb relationships we extend Equations 4.2 and 4.3 to three variables (cf., Equation 4.4). That means, that x, y and z are terms and PMI relates the probability of their coincidence $p(x, y, z)$ with the probabilities of observing the three terms independently $p(x)p(y)p(z)$.

$$pmi(x, y, z) = \log\left[\frac{p(x, y, z)}{p(x)p(y)p(z)}\right] \quad npmi(x, y, z) = \frac{pmi(x, y, z)}{-\log\left[p(x, y, z)\right]} \tag{4.4}$$

4.3.10 Context Extension

So far, terminology extraction was based on fivegrams, thus only five consecutive tokens are available to find related terms. It limits the extraction of related terms to the following relationships:

- One-word term ↔ one-word term
- One-word term ↔ two-word term
- One-word term ↔ three-word term
- Two-word term ↔ two-word term
- One-word term ↔ one-word term ↔ one-word term

[14] Note that the terms can be multi-word expressions

Terms consisting of four words cannot be related to other terms at all. In order to relax these limitations, we describe in this section the generation of sixgrams from the fivegram data.

4.3.10.1 Sixgram Generation

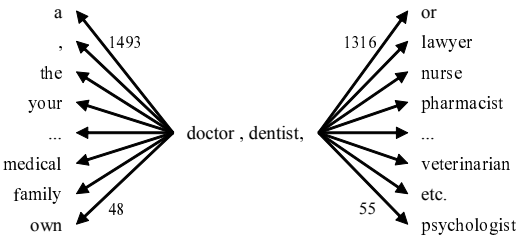
It is a typical use case for N-gram data to predict the most likely subsequent word for a sequence of input words. For example, a sequence of four words can be used to retrieve a set of fivegrams that start with the particular sequence. Figure 4.8a and b show examples of the sequence *[doctor , dentist ,]*. The dataset contains 21 fivegrams which start with the sequence and 21 fivegrams in which the sequence occurs at the end. The most likely subsequent word of the sequence is *"or"* and the most likely subsequent noun is *"lawyer"*.

a	doctor , dentist ,	1493
,	doctor , dentist ,	1221
the	doctor , dentist ,	720
your	doctor , dentist ,	389
(	doctor , dentist ,	249
any	doctor , dentist ,	141
medical	doctor , dentist ,	113
family	doctor , dentist ,	89
...	... ...	...
their	doctor , dentist ,	64
The	doctor , dentist ,	60
A	doctor , dentist ,	52
Your	doctor , dentist ,	51
own	doctor , dentist ,	48
new	doctor , dentist ,	46
with	doctor , dentist ,	41

(a) Fivegrams with "doctor, dentist," at the end.

doctor , dentist , or	1316
doctor , dentist , and	903
doctor , dentist , lawyer	592
doctor , dentist , nurse	325
doctor , dentist , pharmacist	162
doctor , dentist , hospital	152
doctor , dentist , etc.	125
doctor , dentist , accountant	92
... ...	...
doctor , dentist , optician	65
doctor , dentist , barber	62
doctor , dentist , psychologist	55
doctor , dentist , engineer	52
doctor , dentist , optometrist	48
doctor , dentist , architect	46
doctor , dentist , chiropractor	41

(b) Fivegrams with "doctor, dentist," at the beginning.



(c) Graph representation of the combination.

medical	doctor	, dentist ,	lawyer	352
medical	doctor	, dentist ,	pharmacist	137
...	...	...	...	
family	doctor	, dentist ,	psychologist	72

(d) Generated six-grams (excerpt)

Figure 4.8: Process of the six-gram generation

We exploit the inherent structure of the fivegrams and how they have been obtained to materialize N-grams of length six. Since the fivegrams have been extracted by traversing the original text with a sliding window, it is possible to derive their most likely predecessor and successor words. In general, this is done by connecting the left neighbor and the right neighbor of each fourgram within the fivegram data. An example of the graph representation is shown in Figure 4.8c. In this particular example, a permutation would already result in 441 possible sixgrams. Processing the huge amount of sixgrams would be impractical. Therefore, we apply a number of heuristics to generate only useful sixgrams. In our context "useful" means that more term relations can be extracted from the generated sixgrams (e.g., the sixgram *[a doctor , dentist , or]* does not contain additional information, but *[family doctor , dentist , pharmacist]* does).

The process of the sixgram generation is as follows: (1) We determine a set of unique fourgrams within the fivegrams. (2) All fourgrams starting with a sentence start tag (_START_), ending with a sentence end tag (_END_), and ending with a period are sorted out, because an extension beyond sentence boundaries makes no sense. In case the fourgram does not contain a noun, it will be eliminated as well. (3) For each remaining fourgram we query left and right neighbors in the fivegram dataset. (4) We iterate through all possible combinations and keep a sixgram in case the following conditions are met: the sixgram must start with a noun or an adjective-noun combination and must end with a noun. That means added words must contribute to the intended extraction. We also accept the sixgram if it conforms to the former condition on one side and a verb was added on the other side. All rules ensure that additional relations can be detected which have not been seen in the fivegram dataset. (5) Additionally, we implement a heuristic that discards a fourgram in case too many neighbours have been found. This usually happens if the phrase is too general (e.g., the original fivegram was *[doctor during the course of]* and the derived fourgram*[during the course of]* would have almost 1000 right neighbors). (6) The frequency of the resulting sixgram is determined by taking the lower frequency from the two fivegram that were used at the beginning and end of the sixgram.

In the example shown in Figure 4.8 there are only two beginnings that meet the previously described conditions and there are 16 possibilities that end with a noun. Consequently, 32 sixgrams are generated (an excerpt is shown in Figure 4.8d). Part-of-speech tags (not shown in the Figure) are stored together with the sixgrams.

4.3.10.2 Terminology Extraction

After having generated the sixgrams, we process the sixgram dataset in the same manner as described in Sections 4.3.6 to 4.3.8. The sixgrams are normalized, and the terminology extraction uses exactly the same POS patterns. As a result, we are now able to additionally extract the following types of relationships (new types are marked in bold):

- One-word term ↔ one-word term
- One-word term ↔ two-word term
- One-word term ↔ three-word term
- **One-word term ↔ four-word term**
- Two-word term ↔ two-word term
- **Two-word term ↔ three-word term**
- One-word term ↔ one-word term ↔ one-word term
- **One-word term ↔ one-word term ↔ two-word term**

Furthermore, ternary noun-noun-verb relationships include terms consisting of up to four words.

4.3.10.3 Integration of Fivegram and Sixgram Extraction Results

The integration of the extraction results from the fivegram and the sixgram dataset is straightforward. For both sets, the extraction process generates binary noun-noun relationships, binary noun-verb relationships, ternary noun-noun-noun relationships, and ternary noun-verb-noun relationships. Prior to relatedness computation, all four types are iterated and sixgram relationships that are not included in the fivegram extraction results are added. Terms, not yet included, are added to the vocabulary and term frequencies are added together. The degree of relatedness between terms in the integrated dataset is determined using relative frequency and normalized PMI as described in Section 4.3.9.

4.3.11 SemNet Construction

The final step of the extraction process is the construction of the SemNet graph. The results of the extraction process and co-occurrence analysis are so far large-scale tables of word sequences, related word sequences and the respective relatedness measures. Multiword expressions (terms consisting of more than one word) are still stored as separated words using a unigram index because of the context extension algorithm.

The first step in SemNet construction is to merge word sequences and create an index of terms (the vocabulary). All relationship tables containing the binary and ternary relationships are converted to use the new index afterwards. Second step is the creation of a semantic network, which is in essence a large-scale graph in which each term is a node (with a type, either noun or verb) and each directed edge denotes a weighted relationship between the terms. SemNet includes 5.9 million unique one-word terms and multiword expressions connected with 355 million relationships. Each relationship is quantified by the absolute frequency of co-occurrence, a calculated relative frequency, the pointwise mutal information (PMI) measurement and its normalized form (NPMI).

Figure 4.9 shows the four aspects of SemNet for an example term. (a) On top the most related noun terms for *hospital* are shown. The ranking is based on the lexicographer's mutual information (LMI) as it is implemented in the recommender system (cf., Section 6.6). Terms that have a relationship are connected with two directed edges in order to capture the different relative frequencies. (b) On the right-hand side examples of the most related verb terms are shown. There are also edges for the opposite direction (not shown in the figure for space reasons). (c) On the left of Figure 4.9 ternary relationships are shown for three noun terms that have been observed together with their weights. (d) At the bottom ternary relationships are shown for verbs that connect two nouns.

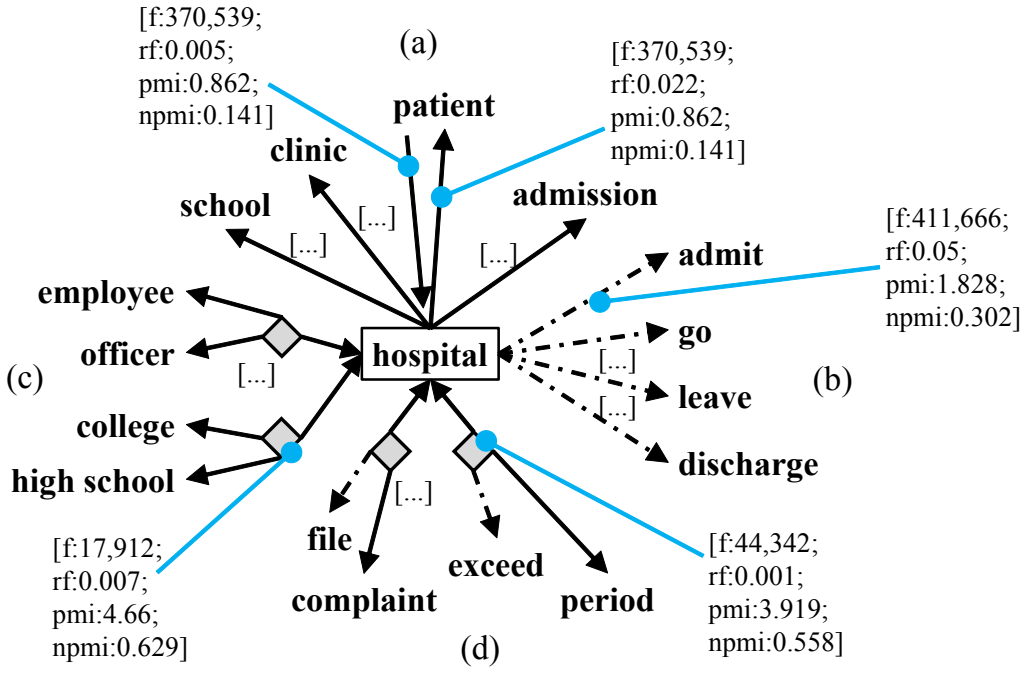


Figure 4.9: Excerpt of the SemNet graph for the term "hospital". (a) binary noun-noun relationships, (b) binary noun-verb relationships, (c) ternary noun-noun-noun relationships, (d) ternary noun-verb-noun relationships (*f* – absolute frequency, *rf* – relative frequency, *(n)pmi* – (normalized) point wise mutual information)

4.4 Extraction Results

This section summarizes the outcomes of the different analysis and extraction steps. In particular, we provide statistics of our analysis methods on the Google Books N-Gram dataset. Table 4.8 shows the structure of this section in relation to the extraction process described in Section 4.3.

Analysis Step	**Results**
Section 4.3.2 Google Books N-Gram Dataset Section 4.3.3 Dataset Conversion Section 4.3.4 Dataset Reduction	Section 4.4.1 Conversion Results
Section 4.3.5 Part-Of-Speech Tagging Section 4.3.6 Normalization	Section 4.4.2 Normalization Results
Section 4.3.7 Syntactic Patterns	Section 4.4.3 Pattern Match Results
Section 4.3.8.1 Extraction of Related Terms	Section 4.4.4 Co-occurrence Results
Section 4.3.8.2 Duplicate Aggregation	Section 4.4.5 Aggregation Results
Section 4.3.10 Context Extension Section 4.3.11 SemNet Construction	Section 4.4.6 Context Extension and Integration Results

Table 4.8: Overview of the analysis sections and their corresponding result sections

4.4.1 Conversion Results

In Section 4.3.3 we described in detail the conversion of the Google Books N-Gram dataset text files into database representations of reasonable size. Apart from the 1-gram and 5-gram dataset, it was also required to convert the 2-grams, 3-grams, and 4-grams for later analysis steps to be able to retrieve frequencies of multi-word expressions. Table 4.9 summarizes the results of the conversion process and the properties of the datasets.

	1-grams	2-grams	3-grams	4-grams	5-grams
Compressed size	5.3 GB	144 GB	1508 GB	294 GB	236 GB
Uncompressed size	27 GB	872 GB	10193 GB	2674 GB	2411 GB
Lines of text	1.4 B	35.6 B	341 B	76.2 B	59.8 B
Conversion time	0.5 h	16 h	175 h	45 h	25 h
Database size	0.38 GB	4.0 GB	28.5 GB	22.8 GB	22.3 GB
Number of N-grams	10 M	181 M	1160 M	820 M	704 M

Table 4.9: Summary of the N-gram dataset conversion process

It is surprising that the trigram dataset is four times the size of the four-gram dataset, but the resulting database does not contain four times as many trigrams. The reason is that the input files of the datasets contain many variations of trigrams with and without part-of-speech tags for all years (e.g., one word with POS tags and two words without tags as well as two words with POS tags and one word without tag for the same trigram). Google has excluded all N-grams with less than 40 occurrences from the dataset. In general, sequences of three words occur more frequently than sequences of four words. Therefore, there are more trigram variations in the dataset that are aggregated by our conversion process.

For speeding up the later analysis steps we performed a **reduction** of the fivegram database as described in Section 4.3.4. We excluded all fivegrams that only contained stopwords or punctuation and fivegrams that do not contain at least one noun word (according to the POS tags provided by Google). The reduction examined 704,355,409 fivegrams and sorted out 40% useless fivegrams resulting in **427,030,743** fivegrams for subsequent analysis steps.

4.4.2 Normalization Results

Before normalization, the fivegram dataset was tagged with the Stanford POS tagger and the Google POS tags were replaced by the more precise Stanford tags (cf., Section 4.3.5). The normalization process examined 427 million fivegrams. 244 million fivegrams (57%) have been changed. Thus, in half of the fivegrams at least one word was lowercased, stemmed, or a special rule was applied. The distribution of the applied rules is shown in Table 4.10. In total 2.14 billion words have been examined and 326 million words (15.3%) have been normalized according to the rules.

Rule	**Occur.**	**Perc.**	**Example**
Verb stemming	178.61 M	53.51%	examined → examine
Noun to singular	120.17 M	36.00%	doctors → doctor
Lowercase noun	19.67 M	5.89%	Doctor → doctor
Lowercase adjective	9.25 M	2.77%	Medical → medical
Lowercase verb	3.30 M	0.99%	Prescribe → prescribe
Change hyphen	2.68 M	0.80%	– → -
Lowercase symbol	0.05 M	0.02%	X - ray → x - ray
Acronym to singular	0.07 M	0.02%	ICUs → ICU

Table 4.10: Summary of the fivegram normalization results

4.4.3 Pattern Match Results

The co-occurrence analysis was conducted on the reduced fivegram dataset consisting of 427 million fivegrams. 386 million fivegrams contained a term that conformed to one of the 20 noun patterns. We expected that in all fivegrams at least one pattern will match, because the reduction rules defined that at least one noun had to occur in a fivegram (see Section 4.3.4). The reason is as follows: The reduction was applied using the original Google part-of-speech tags. After the reduction we tagged the reduced fivegram set with the more precise Stanford tagger and the tags were replaced. Different lexical categories were determined in these cases, hence 41 million fivegrams originally contained a noun according to Google POS tagging that were tagged otherwise with the Stanford tagger.

Table 4.11 summarizes how often the POS patterns (see Section 4.3.7.7) occurred in the dataset (*after* discarding overlapping matches). In total, 477 million occurrences of terms were detected. The single noun pattern was responsible for finding 351 million single-word terms. 90 million double-word terms were found (patterns JJ NN and NN NN). Roughly 30 million triple-word terms were detected and 5.4 million terms consisting of four parts. In total, 220 million verb expressions have been found. The examples presented in the table are the most frequent terms that were extracted by the respective pattern.

The distribution of the pattern occurrences corresponds to the vocabulary analysis as described in Section 4.3.7.1. The **three dominant patterns** NN, JJ NN, and NN NN constitute 91.67 percent of the terms found.

No.	Pattern	Occurrences	%	No.	Pattern	Occurrences	%
1	NN	351,813,830	73.665	14	NN : JJ NN	355,179	0.074
2	JJ NN	69,679,552	14.590	15	NN JJ NN	326,669	0.068
3	NN NN	20,520,297	4.297	16	JJ : JJ NN	227,849	0.048
4	NN IN NN	20,054,684	4.199	17	NN : VBN NN	174,430	0.037
5	NN IN JJ NN	3,004,718	0.629	18	SYM : NN	121,501	0.025
6	JJ NN NN	2,798,153	0.586	19	JJ NN NN NN	101,877	0.021
7	JJ JJ NN	2,654,912	0.556	20	JJ JJ NN NN	99,577	0.021
8	NN : NN	2,013,229	0.422	21	NN NN NN NN	52,445	0.011
9	NN NN NN	1,148,578	0.240	22	SYM : JJ NN	32,366	0.007
10	NN IN NN NN	993,481	0.208	23	SYM : NN NN	31,325	0.007
11	JJ : NN NN	469,888	0.098	24	NN JJ NN NN	23,124	0.005
12	NN : NN NN	464,997	0.097	25	JJ NN JJ NN	19,597	0.004
13	JJ : NN	401,854	0.084				
Total Noun Term Pattern Matches 477,584,112 (100%)							
1	VB	195,376,321	88,671	3	VB VB VB	1,235,048	0,561
2	VB VB	23,719,974	10,765	4	VB VB VB VB	7,617	0,003
Total Verb Term Pattern Matches 220,338,960 (100%)							

Table 4.11: Number of noun and verb pattern matches. The figures denote how often a certain pattern matched in the set of 427 million fivegrams (after discarding overlapping matches)

4.4.4 Co-occurrence Results

This section summarizes the results of the extraction process for detected relationships in the fivegram dataset. We present numbers on how many fivegrams contained co-occurring terms and what lengths they had. Table 4.12 presents recognized binary relationships for noun-noun and noun-verb term pairs. Percentages refer to the total of 427 million fivegrams in the dataset.

The first part of the table lists numbers with respect to noun terms. A noun-noun relationship was extracted from a fivegram when two terms in the fivegram were detected according to the noun POS patterns. The terms had to be separated by at least one token (e.g., "the doctor *and* nurse"). This does not allow to extract certain combinations (e.g., a 1-word term that occurs along with a 4-word expression) because a context larger than 5 tokens is required. 89 million fivegrams contained a binary noun-noun relationship.

The second part of the table shows the results of nouns that occur along with verbs. A noun-verb relationship was extracted from a fivegram when a noun and a verb were found in the fivegrams. A separation token is not necessary. For example, "the doctor *prescribed* the medication" is a valid phrase that results in an extraction. Therefore, almost twice as many relationships could be extracted. 159 million fivegrams contained a binary noun-verb relationship.

Where possible, the analysis extracted ternary relationships as well. The extraction of a ternary relationship with three noun terms was only possible for 1-word terms. Other extracted relationships are combinations of a verb term and two noun terms. Table 4.13 presents statistics of the detected ternary relationships. Note that ternary relationship extraction will always result in two binary noun-verb relationships or three binary noun-noun relationships, respectively.

In summary, 265 million co-occurrences have been extracted from the fivegram dataset that resulted in 248 million binary relationships and 17 million ternary relationships (still including duplicates). 164 million fivegrams could not be used for relation extraction because they contained only one term.

Relationship Type	Number of Fivegrams	Percentage
1-word noun term + 1-word noun term	70,819,765	16.58%
1-word noun term + 2-word noun term	15,422,408	3.61%
1-word noun term + 3-word noun term	1,942,865	0.45%
2-word noun term + 2-word noun term	611,610	0.14%
Fivegrams with binary noun-noun relationships	88,796,648	20.79%
1-word noun term + verb term	131,893,156	30.89%
2-word noun term + verb term	22,610,633	5.29%
3-word noun term + verb term	4,358,409	1.02%
4-word noun term + verb term	458,629	0.11%
Fivegrams with binary noun-verb relationships	159,320,827	37.31%

Table 4.12: Statistics on the distribution of the binary relationships contained in the fivegram data

Relationship Type	Number of Fivegrams	Percentage
2x 1-word noun term + verb	14,221,482	3.330%
3x 1-word noun term	1,309,719	0.307%
1x 1-word + 1x 2-word noun term + verb	1,286,032	0.301%
1x 1-word + 1x 3-word noun term + verb	38,975	0.009%
2x 2-word noun term + verb	14,682	0.003%
Fivegrams with ternary noun-noun and noun-verb relationships	16,870,890	3.951%

Table 4.13: Statistics on the distribution of the ternary relationships contained in the fivegram dataset

4.4.5 Aggregation Results

So far, the analysis process has recorded pattern matches of nouns and verbs and their co-occurrences in different contexts within the five-gram dataset. The results still contain duplicate entries, e.g., the term *doctor* and *nurse* may appear together in different five-grams with different frequencies, hence, multiple relationships and pattern matches with the same content have been created. The purpose of duplicate aggregation is to merge duplicate matches of individual terms and of duplicate extracted relationships by adding up their individual frequencies.

Table 4.14 provides an overview of how many distinct terms and relationships have been extracted throughout the process. Roughly, 700 million pattern matches are reduced to 8.7 million uniquely identified keyword terms. 265 million extracted relationships contain about 51 million unique connections between the identified terms.

Element	Before	After
Noun term pattern matches	477,584,112	8,682,195
Verb term pattern matches	220,338,960	44,877
Binary noun-noun relationships	88,796,648	30,391,650
Binary noun-verb relationships	159,320,827	11,830,236
Ternary noun-noun-noun relationships	1,248,088	719,762
Ternary noun-noun-verb relationships	15,369,594	8,004,178

Table 4.14: Number of extracted terms and relationships before and after duplicate aggregation

No.	Pattern	Unique terms	Most frequent term
2	JJ NN	2.300.095	first time
4	NN IN NN	1.432.029	point of view
5	NN IN JJ NN	1.131.213	museum of natural history
3	NN NN	1.123.914	interest rate
7	JJ JJ NN	562.307	central nervous system
6	JJ NN NN	546.892	fair market value
10	NN IN NN NN	372.916	weapon of mass destruction
1	NN	259.720	time
9	NN NN NN	252.823	health care provider
8	NN : NN	105.427	x - ray
15	NN JJ NN	101.743	junior high school
12	NN : NN NN	88.837	right - hand side
14	NN : JJ NN	80.696	cross - sectional area
11	JJ : NN NN	77.467	short - term memory
16	JJ : JJ NN	49.971	mid - nineteenth century
17	NN : VBN NN	40.113	state - owned enterprise
20	JJ JJ NN NN	37.644	average annual growth rate
19	JJ NN NN NN	36.642	high school graduation rate
21	NN NN NN NN	22.915	attention deficit hyperactivity disorder
13	JJ : NN	17.282	anti - semitism
24	NN JJ NN NN	10.189	community mental health center
18	SYM : NN	9.832	co - operation
25	JJ NN JJ NN	9.272	high performance liquid chromatography
23	SYM : NN NN	6.858	cod - liver oil
22	SYM : JJ NN	5.398	co - operative society

Table 4.15: Noun POS pattern ranked by number of distinct extracted terms

The most interesting result of the aggregation is which of the POS patterns contributed most to the set of unique terms. In Section 4.4.3 we saw that "NN", "JJ NN" and "NN NN" were the three most common patterns (73.7%, 14.6%, resp. 4.3% of the matches), but in fact the patterns "JJ NN" (26.5%), "NN IN NN" (16.5%), "NN IN JJ NN" (13%), and "NN NN" (13%) were responsible for extracting the most unique terms. Table 4.15 displays the patterns by uniquely extracted terms and shows the most frequent term extracted by the respective pattern.

4.4.6 Context Extension and Integration Results

In this section, we summarize the achievements of extending the extraction context from five-grams to six-grams. The starting point of the analysis is the non-normalized and reduced set of 427 million five-grams. In a first step, the set of distinct four-grams included in the five-gram dataset was determined by taking the first four and the last four tokens of each five-gram, respectively. 299 million different four-grams were extracted. Four-grams, which either started with a sentence-start token or ended with a sentence-end token or no longer contained a noun, cannot be extended and have been discarded. The remaining 218 million four-grams were used as input to determine left and right neighbor combinations for the six gram generation. Table 4.16 provides the exact numbers.

About half of the four-grams (109 million) had both left and right neighbor tokens for the six-gram generation. Theoretically, all possible combinations would have made about 2 billion six-grams. The heuristics described in Section 4.3.10 produced a manageable set of 439 million useful six-grams. Useful means that the additional token contains either a noun, an adjective, or a verb that contributes to one of the POS patterns.

Processing Step	**Number of N-grams**
Five-grams analyzed	427,030,743
Four-grams extracted	298,723,990
Four-grams analyzed	217,894,500
Four-grams extended	108,833,227
Six-grams generated	439,059,211

Table 4.16: Number of N-grams processed for context extension

After having generated the six-grams, the co-occurrence analysis was re-run on the extended dataset. The main objective of the context extension was the full support of noun terms consisting of four words that could only be associated with verbs so far. Furthermore, it was the goal to maximize the number of extracted relationships. Table 4.17 shows the extracted distinct terms and relationships of SemNet before and after the integration with the six-gram analysis. The semantic network contains five times more connected 4-word terms, and the number of ternary relationships was increased by more than an order of magnitude.

Distinct Terms	**Before**	**After**	**Increase**
1-word terms	236,474	414,014	75 %
2-word terms	2,118,938	2,944,694	39 %
3-word terms	1,054,045	1,512,475	43 %
4-word terms	193,215	1,040,323	**438** %
All terms	3,602,672	5,911,506	64 %

Distinct Relationships	**Before**	**After**	**Increase**
Binary noun-noun relationships	30,391,650	174,709,510	475%
Binary noun-verb relationships	11,830,236	41,542,072	251%
Ternary noun-noun-noun relationships	719,762	38,329,325	**5225**%
Ternary noun-noun-verb relationships	8,004,178	100,749,574	**1159**%
All relationships	59,672,898	355,330,481	495 %

Table 4.17: Number of distinct terms and relationships contained in SemNet before and after the context extension

4.5 Evaluation

This section presents the evaluation of SemNet. The evaluation of a newly developed information extraction method and source of knowledge is challenging, as there are generally no gold standards available that exactly match the type of results obtained with the method. Consequently, we assess the content of the semantic network by comparing it with existing knowledge bases with similar content. Section 4.5.1 describes the datasets we use for comparison. The evaluation procedure is described in Section 4.5.2. The results of the evaluation are shown in Section 4.5.3.

4.5.1 Datasets

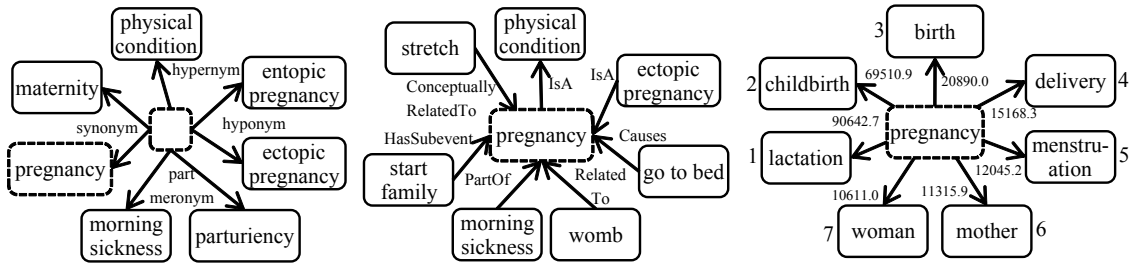
SemNet is compared to two existing manually created semantic databases: WordNet V3.1 [108] and ConceptNet V5.1 [113] (the latter also contains automatically generated content). They were selected for the following reasons. On the one hand, they contain information about terminology and its semantic relationships, similar to SemNet. On the other hand, both projects focus on conceptual knowledge that can be used in domain-specific modeling. Automatically created knowledge bases such as YAGO[15] and DBpedia[16] have limited benefits for domain modeling as they concentrate on factual knowledge (at the instance level). In the following, the term *pregnancy* is used as an example to illustrate what kind of information is contained in the respective networks and how it is modeled.

WordNet. WordNet is a lexical database for the English language [108]. It models synsets that group words that have the same meaning. It contains word senses for nouns, verbs, adjectives and adverbs. Most of the information relates to nouns (a total of 117,659 synsets, 82,115 nouns synsets, and 102,249 noun relationships). WordNet mainly covers synonymous, taxonomic and part-whole relationships. Figure 4.10a shows 7 of the 32 relationships that exist in WordNet for the word sense *pregnancy*. The words *pregnancy* and *maternity* are both synonyms associated with the word sense object in the middle. There are several relationships to other word senses, including the fact that pregnancy is a physical condition, that there are two specific types of pregnancies, and that morning sickness and parturiency are part of the pregnancy. For reasons of space, we omit the word sense objects for the other terms. Each of the above relationships is actually connected to a word sense that has a synonym link to the actual word.

ConceptNet. ConceptNet is a "large semantic graph that describes general human knowledge" [113]. It models concepts which are expressed with natural language phrases. It was created manually based on the Open Mind Common Sense project[17] and partially automatically from Wiktionary[18] and the ReVerb[206] project. Lexical types are indistinguishable, ConceptNet includes all kinds of concepts such as named entities (*barack obama*), noun phrases (*database software*), adjectives (*beautiful*), and activities (*build aircraft*) (1.7 million English concepts in the core version of ConceptNet and 5.9 million relationships between them). ConceptNet also features taxonomic, synonym and part-whole relationships. Additionally, it contains several other relationship types (e.g., *AtLocation*, *HasProperty*). Figure 4.10b shows examples (7 of 37 relationships) for the concept *pregnancy*.

SemNet. SemNet is a large-scale graph of related terms with almost 6 million single-word terms and multi-word expressions classified into nouns and verbs. Relationships between terms are modeled as weighted edges between the terms using several corpus-based and information-theoretic measurements (355 million relationships). Figure 4.10c shows the term *pregnancy* together with its 7 most related terms (14,143 relations in total, for space reasons we omit back references).

[15] `http://www.yago-knowledge.org`
[16] `http://dbpedia.org`
[17] `http://csc.media.mit.edu/`
[18] `https://www.wiktionary.org/`



(a) The word sense *pregnancy* in WordNet (7 out of 32 relationships).

(b) The concept *pregnancy* in ConceptNet (7 out of 37 relationships).

(c) The term *pregnancy* in SemNet (First 7 out of 14,143 relationships).

Figure 4.10: Examples of how terminology information for *pregnancy* is represented in WordNet, ConceptNet and SemNet

4.5.2 Quantitative Evaluation Procedure.

The evaluation of the information contained in SemNet from WordNet and ConceptNet takes place in two steps. We first determine how many synsets/words from WordNet and how many concepts from ConceptNet are included in SemNet. Second, we take the synsets and concepts found and determine how many of their relationships are contained in SemNet. Therefore, we can examine how well the specific knowledge base relationships are represented in SemNet.

To generate an evaluation dataset from WordNet, we iterate through the synsets of WordNet, which are classified as nouns (82,192 synsets). We exclude 18,114 instance synsets (e.g., city names, countries, animal orders) because these types of words and expressions were explicitly filtered during the creation of SemNet. We also filter synsets with words that contain numbers and other special characters (789 synsets), and synsets that have no relationship destinations due to the filtering rules (6,230 synsets). As a result, 57,059 noun synsets are evaluated. These synsets comprise 77,554 unique terms and are connected with 154,035 relationships.

Comparing ConceptNet with SemNet is more challenging because concept names in ConceptNet can contain all sorts of lexical expressions and are not categorized. Therefore, not all nouns can be selected. We have used POS tagging for concept names to identify nouns, but it is too inaccurate for individual words without context. Therefore, we have determined all concepts in ConceptNet that have a corresponding noun in WordNet. As a result, 53,612 concepts in conjunction with 634,128 relationships are evaluated.

The evaluation of the term coverage is carried out as follows. We iterate through the WordNet and ConceptNet terms and count whether the respective term is in the SemNet vocabulary. A WordNet synset is considered to be found if at least one of the synonyms is included in SemNet.

The relationships in WordNet and ConceptNet are evaluated as follows. All previously found synsets/concepts are iterated and for each term the corresponding related terms are retrieved from SemNet. We then determine how many WordNet/ConceptNet relationship targets are included in SemNet's list of related terms.

4.5.3 Quantitative Evaluation Results

Term Coverage. Table 4.18 presents the results of the term coverage evaluation for WordNet and ConceptNet. On average, SemNet covers 83.9% of the evaluated noun concepts of WordNet and ConceptNet. 87.3% of WordNet synsets, 78.6% of WordNet noun terms, and 87.9% of ConceptNet concepts were found. WordNet's noun term coverage is lower compared to synset coverage, because often one synonym is found in SemNet, but other rare synonyms cannot be found due to the threshold of at least 40 occurrences in the original Google N-gram dataset from which SemNet was constructed. On the one hand, term coverage can be improved by creating custom N-gram datasets based on other text corpora with lower thresholds. On the other hand,

this leads to a multitude of rare terms that rarely occur together with other terms, resulting in poor PMI values for relationships. It must therefore be considered which goal is more important.

Dataset	Quantity	Contained in SemNet	Percentage
WordNet Noun Synsets	57,059	49,827	87.3%
WordNet Noun Terms	77,554	60,956	78.6%
ConceptNet Noun Concepts	53,612	47,124	87.9%
All	188,225	157,907	83.9%

Table 4.18: Term coverage results for WordNet and ConceptNet

Relation Coverage. The results of the WordNet relationship evaluation are shown in Table 4.19. In WordNet, noun synsets are connected with 11 different kinds of relationships. These relationships are essentially four main types: is-a relationships (hypernym / hyponym), part-whole relationships (meronym / holonym), antonyms and categorization relationships (category / member). Relationships are modeled as unidirectional connections, so hypernym and hyponym have the same number. This also applies to meronym / holonym and category relationships. There are slight differences in the reverse direction, as some targets were filtered during dataset preparation (see Section 4.5.2). The average coverage is relatively low at 56.3 percent. The only exception is the antonym relationship (84.9% of the relationships could also be found in SemNet). The main reason for this difference is the small context of five words in the Google N-gram dataset. Antonyms usually appear closer together in natural language expressions, and hypernym / hyponym relationships for normal nouns / concepts often require sentence-level analysis [120, 310], which is not possible using fivegrams.

Relationship type	WordNet	SemNet	Perc.	Example
Hypernym	55,360	31,016	56.0%	herring → saltwater fish
Hyponym	55,360	31,016	56.0%	meal → afternoon tea
Meronym	5,512	3,139	56.9%	
- *Part Meronym*	*4,395*	*2,562*	*58.3%*	bowling → frame
- *Member Meronym*	*452*	*239*	*52.9%*	family → child
- *Substance Meronym*	*665*	*338*	*50.8%*	coffee → caffeine
Holonym	5,519	3,145	57.0%	
- *Part Holonym*	*4,404*	*2,570*	*58.4%*	antenna → transmitter
- *Member Holonym*	*450*	*237*	*52.7%*	letter → alphabet
- *Substance Holonym*	*665*	*338*	*50.8%*	clay → roofing tile
Antonym	1,976	1,678	84.9%	dissonance → harmony
Category	3,003	1,549	51.6%	split → tenpin bowling
Category Member	3,997	2,066	51.7%	grammar → clause
All	130,727	73,609	56.3%	

Table 4.19: Relationship coverage results for WordNet

The results of the ConceptNet relationship evaluation are shown in Table 4.20. There are a few more than 22 relationship types, but we filtered all types that had fewer than 500 instances. On average, 68.8 percent of ConceptNet's relationships are contained in SemNet. In general, SemNet performs better in the ConceptNet evaluation because both networks follow similar aims, namely to build a network of conceptual terms and their relationships.

About half of ConceptNet's relationships are classified as *IsA*-relationships, of which 61.2% were also extracted by SemNet. The very good results for the relationships *RelatedTo* (83.2%) and *ConceptuallyRelatedTo* (76.6%) support that our methods accomplish the identification of semantically related terms. SemNet performs best at the relationships *LocatedNear* and *HasA* (over 90 percent). *HasA* is primarily an attribute or ownership relationship and is often expressed in terms that resemble for example "the national anthem of the country". Consequently, the probability of simultaneous occurrence of these words in a five or six word window is higher. Similar observations were made by Nulty et al. [311]. The *Antonym* relationship also scored very good results for the same reasons as in the WordNet evaluation. Although it is questionable whether this relationship should be modeled at all, the results of the *NotIsA* relationship are also very good, in contrast to the *IsA* relationship. Looking closer at these relationships, one can quickly see that the majority of instances of the relationship are similar to the *RelatedTo* relationship. The connected words are correct for the *NotIsA* relationship but are mostly from the same context (e.g., "apple" *NotIsA* "banana, cherry, citrus orange, vegetable").

Relationship type	**ConceptNet**	**SemNet**	**Perc.**	**Example**
IsA	287,626	175,907	61.2%	acid → liquid
AtLocation	59,033	42,735	72.4%	alarm clock → bedroom
HasProperty	56,962	42,317	74.3%	address → street name
ConceptuallyRelatedTo	35,274	27,023	76.6%	art → gallery
RelatedTo	29,866	24,846	83.2%	coat → cold weather
UsedFor	26,282	21,340	81.2%	basement → storage
HasA	18,716	16,915	90.4%	country → national anthem
LocatedNear	8,158	7,394	90.6%	paper → printer
PartOf	10,017	7,161	71.5%	keyboard → computer
Synonym	8,958	5,027	56.1%	silence → quietness
CapableOf	4,963	3,813	76.8%	scientist → research
NotIsA	4,548	3,679	80.9%	daughter → mother
MadeOf	4,286	3,520	82.1%	chocolate → cocoa bean
SimilarSize	2,845	2,436	85.6%	mattress → bed
Causes	2,760	1,832	66.4%	affair → divorce
Desires	2,083	1,818	87.3%	person → acknowledgment
HasContext	4,308	1,723	40.0%	crown → dentistry
ReceivesAction	1,957	1,496	76.4%	king → overthrow
HasPrerequisite	1,571	1,165	74.2%	fire → oxygen
Antonym	1,264	1,008	79.7%	decrease → increase
HasSubevent	1,372	925	67.4%	dream → eye movement
CreatedBy	720	636	88.3%	electricity → generator
All	573,569	394,716	68.8%	

Table 4.20: Relationship coverage results for ConceptNet

Relationship coverage can be improved by enlarging the analysis context. This is either possible by creating custom N-gram datasets with a window size of seven (or even larger) or by directly analyzing the original sentences of the corpus. Datasets with longer N-grams have the disadvantage that the number of N-grams grows exponentially with the window size if not pruned at a certain threshold. That is why there are hardly any datasets that go beyond a length of five [326]. In addition, the threshold must be set so low that a large part of the N-grams occurs only very few times. Thus, frequencies of the terms and all relatedness computations hardly have any significance, and ranking of terms will be difficult. However, if all original sentences of the corpus are processed (POS tagging, normalization, co-occurrence analysis), this means a much higher computational effort than when using the N-gram dataset as a proxy.

In summary, the automated identification of semantically related terms shows very good results,

although only a context of six words is available for the extraction. Compared to manually created knowledge bases with a few hundred thousand terms and relationships, SemNet comprises a variety many times greater.

4.6 Working with SemNet

There are several ways of how to use SemNet. This section describes the interfaces that have been developed to interact with the semantic network.

4.6.1 Data Serializations

We provide two different serializations of the SemNet graph. The first serialization is a relational database in the SQLite format. The complete semantic network is contained in a single file. Each edge of the graph is one row in a database table. It is not required to install any database server or additional driver to use the database. A standalone client (also just a single precompiled file) for several operating systems can be downloaded from SQLite website[19]. All major programming languages (e.g., Java, PHP, Python) have available modules to access SQLite databases. Using this format it is easy to embed SemNet into other applications that require ranked lists of related terms or top-N recommendations. The database is publicly available for download[20]. It can be queried using standard SQL statements.

As SemNet naturally is a graph, we also generated a Neo4j[21] serialization of SemNet. The Neo4j version requires an installation of the Neo4j community edition that runs a web interface to browse the graph. Each term in SemNet is a node in the Neo4j graph. They are connected with directed edges that have additional properties for frequencies, relative frequencies, PMI and normalized PMI. The network can be queried using the Cypher graph query language.

4.6.2 Application Programming Interfaces

We have developed Java and PHP application programming interfaces to programmatically access SemNet. The APIs provide methods for accessing SemNet's vocabulary, retrieving related terms from SemNet based on noun and verb inputs, and querying the semantic network for top-N recommendations. The provided functions are in detail:

- getIdForTerm – retrieves the vocabulary identifier based on an input string
- getTermForId – retrieves the vocabulary term based on the input identifier
- contains – returns true if the input string is contained in the SemNet vocabulary
- searchTerm – retrieves a set of terms in which a (partial) input string is contained
- getRelatedNounTerms – retrieves a set of related noun terms for an input noun or verb term
- getRelatedVerbTerms – retrieves a set of related verb terms for an input noun term
- getRelatedTernaryTerms – retrieves a set of single terms or tuples of terms for input noun or verb terms or tuples

The first three functions provide access to the vocabulary and the mappings of string terms to identifiers that can be used in other functions. The *searchTerm* function allows to find vocabulary terms by specifying substrings. The *getRelated*-functions determine the set of related terms for the respective input terms depending on the type (noun or verb) and the cardinality (binary or ternary). Each of the functions allows to rank the result sets based on the respective relatedness measurements (absolute frequency, relative frequency, PMI, and normalized PMI).

[19] `http://www.sqlite.org/`
[20] `http://semnet.henning-agt.de/`
[21] `https://neo4j.com/`

4.6.3 Web Interface

We also provide an online version of SemNet that allows to query related terms in a web interface. The interface provides query forms for the four types of relationships of the semantic network (cf., Figure 4.11).

- Noun queries: Enter a noun term and retrieve related noun and related verb terms.
- Verb queries: Enter a verb term and retrieve related noun terms.
- Ternary noun queries: Enter one or two noun terms and retrieve triples of related noun terms.
- Ternary noun-verb queries: Enter one noun term and a verb or enter two noun terms and retrieve triples of noun pairs and related verbs.

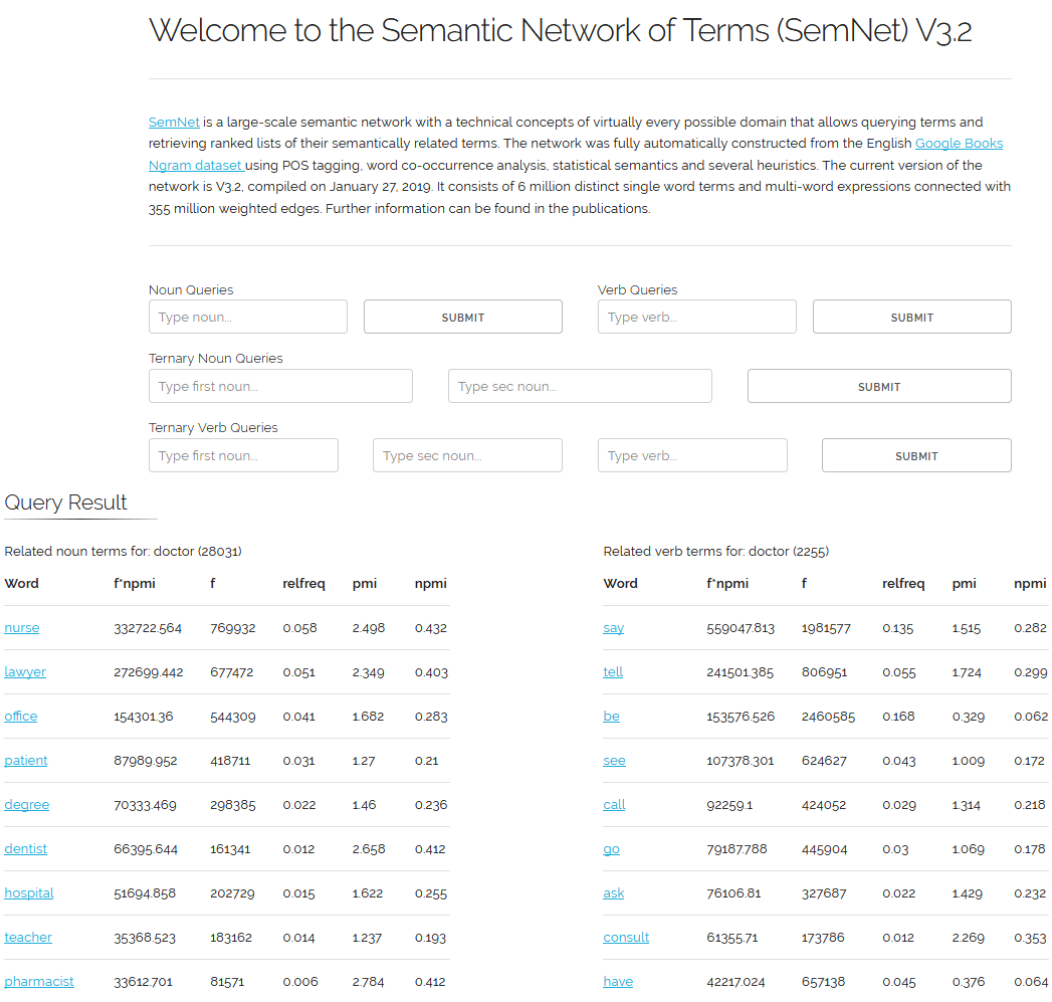


Figure 4.11: Screenshot of SemNet's web interface

By default, the query results are ordered in descending order according to the lexicographer's mutual information (LMI), that is, the absolute frequency multiplied by the normalized pointwise mutual information. The website allows you to browse the semantic network directly by clicking on the terms in the result lists. The term is then queried, and corresponding related terms are displayed. The website can be accessed at `http://semnet.henning-agt.de/`.

4.6.4 Top-N Examples

This section presents examples of SemNet by querying SemNet for terms with different degrees of specificity and showing top-N related terms. We selected terms that are very common with very high corpus frequencies (around 100 million occurrences), terms with high frequencies (around 10 million occurrences), terms with average frequencies (around 1 million occurrences), rare terms with low frequencies (around 100 thousand occurrences), and very rare terms with very low frequencies (around 10 thousand occurrences).

Table 4.21 shows the 10 most closely related noun terms for five noun terms with decreasing degrees of specificity. f indicates the absolute corpus frequency of the query term. The last row contains the total number of related noun terms connected to the query term.

	patient	**emotion**	**computer science**	**concentrated sulphuric acid**	**photographic paper**
f	115M	10M	1.2M	114K	10K
1	family	feeling	engineering	drop	film
2	condition	thought	mathematics	ml	image
3	physician	voice	degree	volume	means of ink
4	symptom	passion	chemistry	action	stamp
5	care	mind	electrical engineering	part	sensitive surface
6	treatment	expression	information	weight	means of pressure
7	disease	cognition	physics	heating	drawing
8	doctor	motivation	economics	cubic centimeter	real image
9	age	sensation	computer engineering	desiccator	means
10	hospital	intellect	lecture note	excess	impression
...	...	...	...	...	...
#	118,149	19,741	903	124	97

Table 4.21: Top 10 related noun terms for the respective noun query terms with different corpus frequencies in descending order

Table 4.22 presents the 10 most closely related verb terms for five noun terms with different degrees of specificity. f indicates the absolute corpus frequency of the query term. The last row contains the total number of related verb terms connected to the query term.

	patient	**emotion**	**computer science**	**concentrated sulphuric acid**	**photographic paper**
f	115M	10M	1.2M	114K	10K
1	have	express	major	add	record
2	treat	be	study	dissolve	expose
3	suffer	feel	concern	heat	print
4	complain	overcome	teach	treat	sensitise
5	die	control	include	contain	impregnate
6	place	arouse	deal	boil	moisten
7	ask	choke	train	dilute	coat
8	instruct	experience	attempt	moisten	saturate
9	see	conflict	imperil	warm	soak
10	occur	tremble	hope	pour	dip
...	...	...	...	...	...
#	3,329	1,888	240	62	68

Table 4.22: Top 10 related verb terms for the respective noun query terms with different corpus frequencies in descending order

Table 4.23 presents the 10 most closely related noun terms for five different verb terms with different degrees of specificity. f indicates the absolute corpus frequency of the query term. The last row contains the total number of related noun terms connected to the query term.

	increase	generate	fetch	decode	reformulate
f	110M	10.9M	1M	105K	11K
1	number	heat	water	message	term
2	risk	electricity	high price	bit	conventional gasoline
3	size	income	price	receiver	problem
4	amount	data	good price	unit	policy
5	rate	revenue	doctor	speech	way
6	demand	random number	instruction	cue	question
7	likelihood	process	deep sigh	rule	light
8	population	order	home	soviet espionage	issue
9	pressure	ability	glass	ability	fuel
10	efficiency	magnetic field	kitchen	instruction	theory
...	...	...	...	...	...
#	150,124	40,620	4,804	844	347

Table 4.23: Top 10 related noun terms for the respective verb query terms with different corpus frequencies in descending order

Table 4.24 presents the 10 most closely related triples with noun terms (ternary relationships) for the respective query noun terms. The numbers in brackets indicate the absolute corpus frequency of the query term. The last row contains the number of total triples that contained the query term.

	patient (115M)			pregnancy (4.3M)		
1	diabetes	patient	type	pregnancy	delivery	labor
2	alzheimer	disease	patient	pregnancy	breast	woman
3	crohn	disease	patient	incest	rape	pregnancy
4	hodgkin	disease	patient	lactation	pregnancy	use
5	symptom	sign	patient	uterus	contraction	pregnancy
6	morbidity	mortality	patient	cervix	carcinoma	pregnancy
7	cushing	syndrome	patient	infanticide	abortion	pregnancy
8	graves	disease	patient	pregnancy	drug	birth defect
9	neck	patient	head	nausea	vomiting	pregnancy
10	caution	patient	use	childbirth	pregnancy	effective care
...	...	...	...	...	...	...
#	206,270			16,850		
	soybean (1M)			tree house (104K)		
1	soybean	wheat	corn	arsenal	fort	tree house
2	sorghum	soybean	oats	quay	fort	tree house
3	soybean	vegetable	corn	settlement	tree house	church
4	soybean	corn	cotton	stone	tree house	church
5	soybean	alfalfa	oats	opera	tree house	church
6	soybean	oats	maize	apartment	tree house	church
7	sorghum	soybean	peanut	expensive	tree house	car
8	soybean	corn	plant	permanent	tree house	road
9	sorghum	soybean	alfalfa	dwelling	tree house	church
10	soybean	maize	potato	domed	tree house	church
...	...	...	...	...	...	...
#	4557			13		

Table 4.24: Top 10 related triples of noun terms for the respective noun query terms with different corpus frequencies

Table 4.25 presents the 10 most closely related triples with subject-predicate-object terms (ternary relationships) for the respective query noun terms. The numbers in brackets indicate the absolute corpus frequency of the query term. The last row contains the number of total triples that contained the query term.

	patient (115M)			pregnancy (4.3M)		
1	supine	lie	patient	pregnancy	delivery	labor
2	pain	complain	patient	pregnancy	breast	woman
3	patient	acquire	immunodeficiency syndrome	incest	rape	pregnancy
4	patient	have	right	lactation	pregnancy	use
5	patient	complain	pain	uterus	contraction	pregnancy
6	patient	be	supine	cervix	carcinoma	pregnancy
7	history	have	patient	infanticide	abortion	pregnancy
8	patient	acquire	immunodeficiency	pregnancy	drug	birth defect
9	patient	admit	hospital	nausea	vomiting	pregnancy
10	patient	represent	percent	childbirth	pregnancy	effective care
...	...	...	...	...	...	...
#	311,159			13,704		
	soybean (1M)			tree house (104K)		
1	soybean	be	glycine max	side	expose	tree house
2	sunflower seed	sesame	soybean	tree house	be	wood
3	soybean	dry	oils	carpenter	build	tree house
4	corn	be	soybean	tree house	climb	sailor
5	soybean	paste	miso	tree house	appear	wood
6	soybean	be	crop	ball	be	tree house
7	sugar beet	hop	soybean	distance	appear	tree house
8	crop	grow	soybean	tree house	discover	travel
9	soybean	be	field of corn	garden	be	tree house
10	soybean	develop	expert system	wood	be	tree house
...	...	...	...	...	...	...
#	487			27		

Table 4.25: Top 10 related triples with subject-predicate-object terms for the respective noun query terms with different corpus frequencies

4.7 Summary

In this chapter, we presented the detailed approach of how to extract semantically related terms from a large-scale N-gram dataset to create a semantic network of terms (SemNet) for virtually every possible domain. The N-grams serve as proxies for a large text corpus, so it is not necessary to process the complete original sentences. At the same time, they provide absolute frequencies of recurring phrases and co-occurring terms that can be used to derive the degree of relatedness between them. Terms and multiword expressions (MWEs) were extracted using a set of 25 syntactic part-of-speech patterns used on preprocessed fivegrams of the Google Books N-gram dataset. The syntactic category (noun/ verb) of the terms is retained throughout the extraction process and named entities are explicitly excluded. Therefore, SemNet contains almost exclusively conceptual terms and expressions. Relationships are identified by applying a hierarchical pattern matching to the fivegram elements and recording the absolute frequencies of co-occurring terms in different contexts. Since the five-word window limits the extraction of noun-noun relationships to a maximum of three-word terms and allows only a very limited extraction of noun-verb relationships with four words, we have extended the context by generating a sixgram dataset from the fivegrams. To prevent the combinatorial explosion of the search space, some effective heuristics have been developed to produce only useful sixgrams that contain conceptual terms. This not only increases the number of recognized terms by more than 60%, but also leads to additional relationships of more than an order of magnitude. SemNet includes nearly 6 million nouns, noun phrases and verb terms, as well as over 355 million binary and ternary relationships. Each relationship is

quantified using common relatedness measurements that allow to directly answer top-N queries. The evaluation shows that SemNet can detect over 80% of the terms, on average over 65% of the relationships, and on average around 80% of the related-to relationships contained in manually created knowledge bases. SemNet is made available online[22].

[22] http://semnet.henning-agt.de/

Chapter 5

OntoConnector: Integration of Lexical Knowledge Bases

The previous chapter developed automated methods for extracting semantically related terms that help to create a rich conceptual knowledge resource with millions of terms and hundreds of millions of relationships. It provides an unprecedented amount of lexical information classified with information theory measurements. However, with the techniques used for extraction, the types of the relationships cannot be determined more closely. In contrast, knowledge bases that use RDF typically encode domain knowledge using specific relationships with well-defined meaning and rarely include probabilistic weights. As this type of information is required to support domain modeling, this chapter examines structured knowledge bases with respect to the contained lexical knowledge. Consistent access to this information in a combined way is challenging, because knowledge bases exist in a distributed fashion, provide different access methods, syntactically encode information using different data models, and semantically describe the content in various ways. This chapter develops new methods and tools for transparent access to lexical information in knowledge bases, addressing the second challenge of this thesis: the heterogeneity of knowledge bases.

5.1 Introduction

In this chapter we describe how lexical and conceptual knowledge contained in existing knowledge sources is used to support domain modeling. We detail the development of OntoConnector, a component responsible for querying heterogenous knowledge bases, for extracting the lexical information, for integrating the results, and for transforming the results into appropriate representations to be used for domain modeling.

The chapter is structured as follows: In Section 5.2 we review related approaches to knowledge integration. Section 5.3 describes the general procedure of using knowledge bases for modeling suggestions. Section 5.4 shows how lexical and conceptual knowledge is organized in different knowledge bases using standard and proprietary data models. In Section 5.5 we detail our solution of a mediator-based architecture to provide uniform access to heterogenous knowledge bases. We exemplarily have selected three knowledge bases that represent widely used data models and develop their integration in Section 5.6. In Section 5.7 we describe how intermediate results of different knowledge bases are integrated. Finally, from the exemplary integration we derive a set of templates for the easy integration of other knowledge bases that use the same or similar data models (cf., Section 5.8).

5.2 Related Knowledge Integration Methods

OntoConnector aims to leverage lexical and conceptual knowledge from heterogeneous knowledge bases. The provision of unified access to these sources can be exploited through approaches from

four main categories: methods for deriving alignments between knowledge bases, approaches with a translator, methods for creating centralized data sources, and methods for jointly querying multiple knowledge bases.

5.2.1 Ontology Matching

Ontology matching, often referred to as ontology alignment, is the process of finding correspondences between entities, classes, and properties of the respective knowledge bases [327]. An alignment is a set of correspondences, often encoded as a set of RDF statements with binary relationships (e.g, using *owl:sameAs, owl:equivalentClass, rdfs:subClassOf, rdfs:subPropertyOf, owl:equivalentProperty, skos:exactMatch, skos:closeMatch*) or using dedicated languages (e.g, EDOAL, SEKT, SKOS, OWL, SWRL). There are some systems that can also create complex alignments, but this area is still subject of research (see the recently published report of the Ontology Alignment Evaluation Initiative (OAEI) 2018 [328]). The AgreementMakerLight (AML) [329] system delivered the best overall performance in most of the tasks in recent OAEI competitions. Recent comparisons of ontology matching systems have been done by Shvaiko & Euzenat [307] and Otero-Cerdeira et al. [330]. An alignment can either be used to translate a knowledge base into a target representation, to merge knowledge bases, or to implement the answering of queries. In general, ontology matching is applicable to overcome the heterogeneity of lexical and conceptual knowledge bases. However, this is not feasible in our environment. It would be necessary to match each knowledge base with all others, or to use a knowledge base as a reference, and all others are reconciled. In addition, the process must be repeated for each new knowledge base to be connected to the system. The effort would be too big, for example, matching large ontologies with millions of instances, such as YAGO and DBpedia, takes 5-11 hours, even in highly-optimized systems [331]. In addition, matching is only applicable to the content of a knowledge base that has counterparts in another knowledge base. The goal of OntoConnector is to integrate complementary knowledge distributed in heterogeneous knowledge bases.

5.2.2 Knowledge Translation

Knowledge translation is an approach in which a schema or intermediate knowledge base acts as a translator between different knowledge bases. It is mainly applied in scenarios where semantic heterogeneity cannot be resolved by ontology matching systems. This semantic heterogeneity usually occurs when n-ary facts are formulated [332] (e.g., events with date, place, and people involved). These facts can be modeled in many different ways with the RDF data model. FrameBase [333] provides a knowledge base schema that uses linguistic frames for a more concise representation of n-ary relationships. The integration of knowledge bases is achieved by transforming the respective instances into FrameBase instances using integration rules. These integration rules have to be developed manually and can partially be generated automatically [334]. Although the lexical information we need for domain modeling can be represented without n-ary relationships (or without the respective triple patterns), applying this approach would cost too much effort. Nevertheless, it is an interesting opportunity to represent relatedness values and ternary relationships of SemNet (cf., Section 4.3.11).

5.2.3 Data Centralization

Data centralization is an option to create a consolidated knowledge base from multiple sources to provide centralized access to the information. Although the Semantic Web is generally based on a linked decentralized architecture, it may be necessary to create such data warehouses and provide appropriate linked data interfaces, which is often the case in companies [335]. The methods involved are very similar to traditional database management systems and data warehouses, which implement an Extract, Transform, Load (ETL) process and also provide Online Analytical Processing (OLAP). The RDF Data Cube Vocabulary[1] is the linked data representation of the

[1] `https://www.w3.org/TR/vocab-data-cube/`

multi-dimensional data model in data warehouses. For our purposes, it is not feasible to copy and transform the complete data of existing knowledge bases because developing a domain model requires only a small portion of it. In addition, this would prevent the most recent versions of knowledge bases from being used without repeating the ETL process over and over again.

5.2.4 Query Federation

Query federation deals with integrated access to distributed RDF data sources using transparent SPARQL queries. Federated querying of RDF datasets is based on the concepts of federated databases and federated information systems [99], and mediator-based information systems [336]. Our OntoConnector implements a very similar architecture (see Section 5.5). A common workflow for processing distributed RDF queries in the literature [337] is: (1) Parsing: A SPARQL query is parsed and decomposed into triple patterns. (2) Query Planning: This step involves creating subqueries and selecting the sources (to which endpoints the sub-queries will be submitted). Sources may be determined at run time or set in advance, for example, using the SERVICE operator of the SPARQL 1.1 Federated Query extension[2]. (3) Optimization: A query execution plan is generated, including, for example, rewriting queries or determining join orders as in conventional database systems. (4) Query Execution: Subqueries are executed from the respective data sources, and the results are integrated. Federation systems research in recent years has focused mainly on the planning and optimization steps (see recent surveys by Saleem et al. [338] and Wylot et al. [339], and systems like HiBISCuS, FedX, SPLENDID). Compared to the query framework described above, our goal is not to formulate integrated SPARQL queries but rather rely on a *separated execution approach* [340]. This gives us more flexibility in adding and removing knowledge bases to the system. We only use the default federated query extension and the SPARQL-LD extension to integrate non-SPARQL endpoints (see Section 5.8.4).

5.3 General Querying Procedure

Figure 5.1 gives an overview of our approach. (1) Domain models under development contain domain-specific terms as well as relationships and are monitored for changes. For each relevant change (e.g., a class / relation has been created or renamed – see Section 3.4 for the full list of support scenarios), the domain terminology is extracted from the model into a lexicon representation. OntoConnector's task is to gather information about related domain model elements. (2) Based on the scenario and the extracted expressions, technology-independent queries are generated (for example: retrieve all the narrower terms of a class name). (3) Each registered knowledge base normally incorporates its own data model. Therefore, technology-independent queries are translated to knowledge base specific queries and executed. (4) The execution and retrieval of the result is done on a mediator basis. The original content of the knowledge base is wrapped and remains unchanged. For each of the queries and for each knowledge base, the results are retrieved separately. (5) The result integration is performed per query type (e.g., subclass query results of the knowledge base 1..n) and translated back to the lexical representation. (6) The results are processed and mapped to the respective modeling language constructs and are offered as recommendations in the frontend. (7) The developer can use the suggestions and refines the domain model according to his needs. Then the process starts again from the beginning. Except for the refinement step, the procedure is completely automated.

5.4 Sources of Modeling Knowledge

In this section, we introduce three knowledge bases that have been selected by way of example as representatives of specific data models. We chose WordNet, OpenCyc, and ConceptNet because they come closest to the conceptual and common sense knowledge that is required to support domain modeling.

[2] `https://www.w3.org/TR/sparql11-federated-query/`

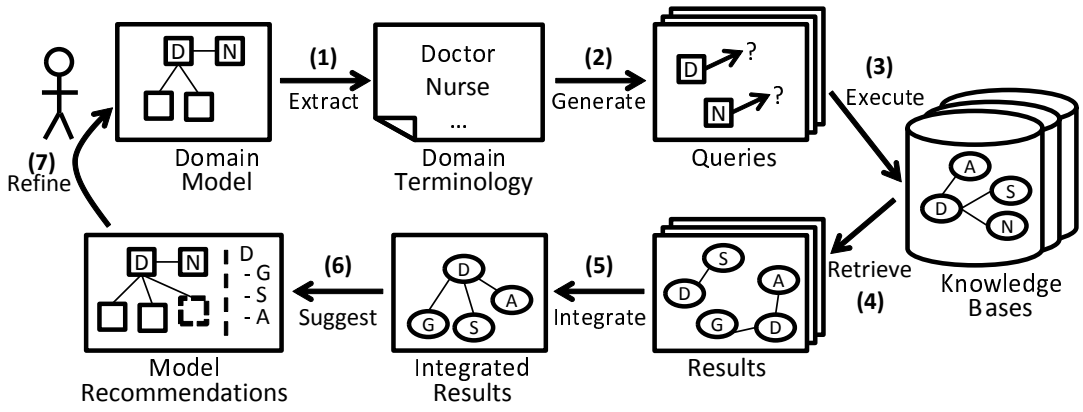


Figure 5.1: Automated procedure of querying semantic knowledge bases to provide model element suggestions

5.4.1 Lemon-Based Lexicons: WordNet

WordNet is a lexical database for the English language [108]. It models synsets that group words that have the same meaning. It contains word senses for nouns, verbs, adjectives and adverbs (over 117,000 synsets in total). Most of the information relates to nouns (82,115 noun synsets). WordNet mainly covers synonymous, taxonomic and part-whole relationships, and to some extent instance knowledge. It was created manually, thus featuring high quality structured information on lexical relationships. In the last decade, WordNet was refined and updated several times (latest version is V3.1). It can be downloaded and queried through a web interface[3].

The original WordNet, developed by Princeton University, is based on a proprietary data model that uses a series of ASCII files[4]. Several efforts have been made to represent WordNet directly as RDF[5]. A few years ago, WordNet was published as a lexical resource as part of the Linguistic Linked Open Data Cloud [306]. This version of WordNet uses *lemon* [341], a model for representing ontology dictionaries and lexicons. Lemon is an important resource for publishing lexical resources on the Web and is still being developed in the W3C OntoLex group[6].

Figure 5.2 shows the core model of the lemon and its relationship to the WordNet ontology. Words are represented as LexicalEntry from the lemon model. A lexical entry can have multiple meanings (LexicalSense). A sense refers to a Synset in the actual WordNet ontology. A synset groups all words that have the same sense, using the synset member relationship. Synsets can be related by taxonomic relationships (hyponyms, hypernyms) and several other relationships (e.g., instance and part-whole relationships, not shown in the picture).

In Figure 5.3 we give an example of how the words *doctor* and *dentist* are modeled in WordNet-lemon. The word *doctor* has several meanings. Two of the lexical senses (*1-n*, *4-n*) are shown in the figure. The first meaning is a person who holds a PhD (synset *617*). The second meaning is the medical doctor (synset *615*). The words *doctor* and *physician* are members of the same synset and thus being synonyms. The word *dentist* has only one meaning and belongs to synset *944*. Both the medical doctor synset and the dentist synset have a common hypernym (synset *469*). This means that the broader term for them is the word *medical doctor*.

[3] http://wordnetweb.princeton.edu/perl/webwn

[4] https://wordnet.princeton.edu/documentation/wnintro5wn

[5] https://semanticweb.cs.vu.nl/lod/wn30/, https://github.com/jrvosse/wordnet-3.0-rdf, and https://www.w3.org/TR/wordnet-rdf/

[6] https://www.w3.org/2016/05/ontolex/

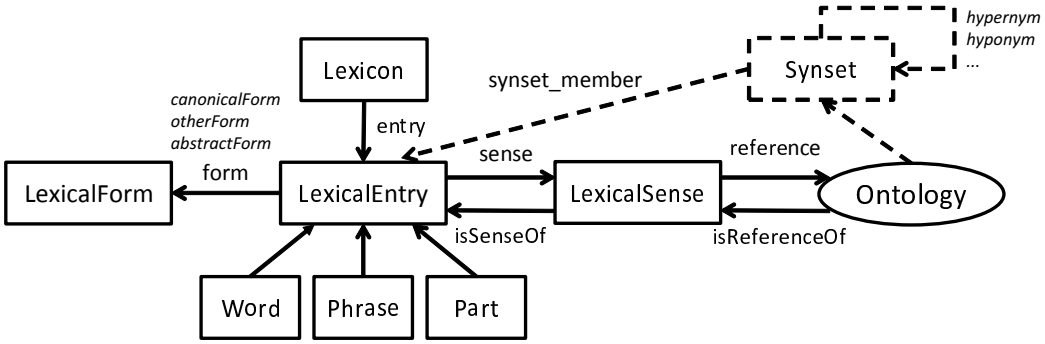


Figure 5.2: The core lemon model (after [342]) and relationships to the WordNet model

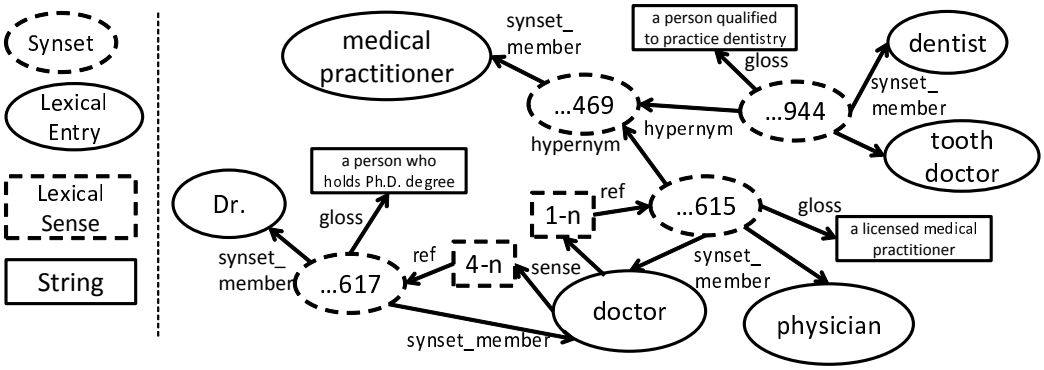


Figure 5.3: Excerpt of WordNet: Relationships between the word *doctor* and *dentist*. For reasons of readability we simplified some details of the model

5.4.2 OWL Schemata: OpenCyc

Cyc is a common sense ontology [112]. Its development began in 1984 to formalize a large amount of knowledge facts in machine-readable format. While WordNet mainly models concepts and their lexical relationships, Cyc also encodes some additional facts about concepts and appropriate rules to enable reasoning. OpenCyc is the publicly available version of Cyc, which contains only part of the rules. Cyc originally used a proprietary language (CycL) and a schema for knowledge modeling. Later it was converted into an OWL ontology. We chose OpenCyc to use it for domain modeling support because it is a popular representative of conceptual knowledge coding in an ontology schema.

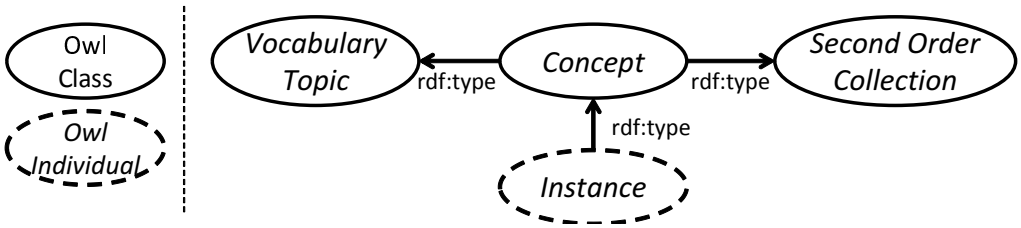


Figure 5.4: OpenCyc's data model based on OWL

Figure 5.4 shows in general how OpenCyc organizes its content. The OWL representation of OpenCyc models concepts as OWL classes. Concepts are first-order collections, so their instances are OWL individuals. Each concept has a label and zero or more synonym terms (not shown in the figure). In the case of a homonymous term, separate concepts are modeled. The predominant re-

lationship between concepts is the RDFS subclass relationship to express taxonomic relationships. OpenCyc also models second order collections (the type of concept). Because they group similar concepts, concepts are instances of these collections. In addition, OpenCyc contains information about topics of a concept (also expressed by an RDF type relationship). Second order collections and topics are also organized with taxonomic relationships.

In Figure 5.5 we review the WordNet example of *doctor* and *dentist* in OpenCyc. The doctor (in the medical sense) is represented as OWL class *Doctor_Medical*. OpenCyc does not have a concept for a person who holds a PhD, but a concept for the doctoral degree itself (not shown in the picture). Both the *Doctor_Medical* and the *Dentist* have the same super concept *Prescriber* (a person who can prescribe medication). The *Prescriber* is a sub-concept of *MedicalCareProfessional*. A synonym for it is *medical practitioner*. Compared to WordNet, the information on the doctor and the dentist is similar (both are medical practitioners). In addition, both the doctor and the dentist are instances of the second-order collection *MedicalSpecialistType*. This means that they belong to the group of *medical specialty* (e.g., dermatologists and x-ray specialists are other specialties). Furthermore, the concepts are also classified according to their topic. Both the doctor and the dentist are examples of *Medical_Topic*, which is a *Healthcare_Topic*. For example, the medical topic also includes the concepts *Nurse* and *Pharmaceutical Product*.

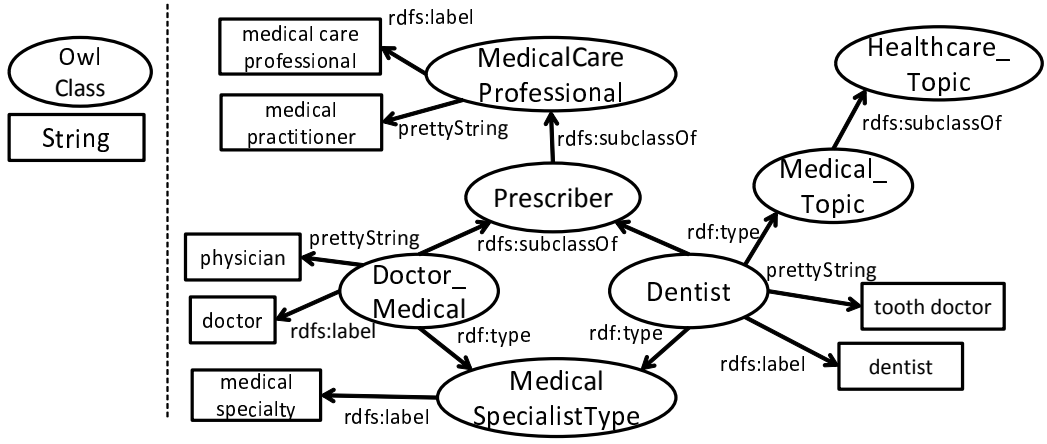


Figure 5.5: Excerpt of OpenCyc: Relations between the concept doctor and dentist

5.4.3 Proprietary Models: ConceptNet

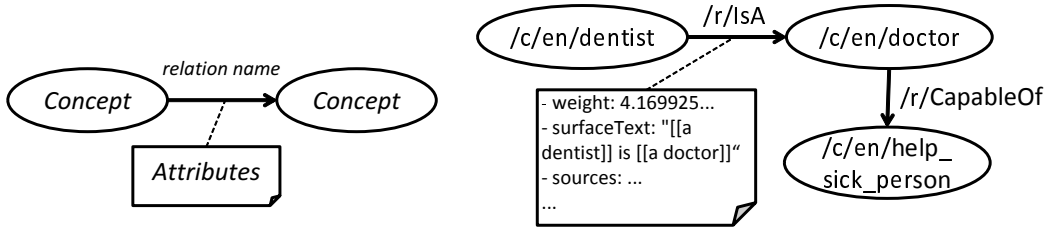
ConceptNet is a *"large semantic graph that describes general human knowledge"* [113]. It also aims to create a large semantic database of common sense knowledge. ConceptNet focuses on multilingualism and the integration of multiple sources of knowledge. It was created based on manually developed projects (e.g., Open Mind Common Sense[7] and Wiktionary[8]) and on automatically generated knowledge sources (e.g., ReVerb [118]).

The ConceptNet data model is very simple: it is a directed labeled graphic. The nodes of the graph are words or short phrases in natural language (the concepts), and the edges are labeled links (the assertions) that express the relationship between the concepts (e.g., *IsA*, *UsedFor*, *CapableOf*). The relationships have additional attributes, such as the source of the assertion or text fragments in which the concepts occur. Relationships can be reified. Concepts do not have type or lexical information. ConceptNet can be queried via a REST API or a web interface. The serialized version of it uses a CSV format. To some extent, ConceptNet supports links to semantic web resources (such as the RDF version of WordNet or DBpedia), but to the best of our

[7] `http://csc.media.mit.edu/`
[8] `https://www.wiktionary.org/`

knowledge, there is no Linked Data version of ConceptNet available online that could be queried using SPARQL[9].

Figure 5.6 shows the data model of ConceptNet and parts of the graph for the concept *doctor*. Both the concept *doctor* and *dentist* are nodes in the knowledge graph. The syntax of the node ID specifies a concept (*/c/*) and indicates English language (*/en/*). They are connected with a relationship called *IsA* (identifier begins with */r/*). Attributes of the relationship include the sources of assertion, the text in natural language in which both concepts appear, and some other fields, e.g. license and dataset information. The third concept, *help sick person*, shows what kind of information ConceptNet allows. This concept is a sentence in natural language and the relation is neither taxonomic nor a part-of relationship compared to the prevailing relationships and WordNet and OpenCyc. Although the name of the relationship is *IsA* and the surface text *[[a dentist]] is [[a doctor]]* implies that *doctor* is the broader concept of *dentist*, ConceptNet does not contain a precise definition of its relationships.



(a) ConceptNet's graph data model (b) Related concepts of „doctor"

Figure 5.6: ConceptNet's data model and an excerpt of the graph for the concept doctor

5.5 Mediator-Based Approach

In the previous section, we described knowledge bases that have similar goals: they encode conceptual, common sense, and lexical knowledge in machine-readable format. They all contain valuable information that can be used to support domain modeling. The biggest challenge in using these sources of knowledge is the lack of unified access to them. Heterogeneous data models prevent the knowledge databases from being consistently queried. In addition, the same information is represented differently. Information about terms and their relationships exists at the schema level, in proprietary intermediate data models, and at the instance level. To make the information available jointly for domain modeling, several steps must be taken.

One possibility would be to manually study the knowledge bases and use the information directly to refine the domain models under development. This is a time-consuming process: all intended knowledge bases must be searched for relevant terms and the results must also be integrated manually.

A second option is the integration of the respective knowledge base in order to create a common, more comprehensive knowledge base that can be used to support the modeling. The problem of different data representations has been well studied in the areas of data and information integration [346] and ontology alignment [330, 307]. In general, this solution is possible, but it is necessary to know all knowledge bases and their data models in advance for full integration. In addition, the alignment and integration of large knowledge databases is computationally expensive. For our purposes, these methods are impractical because we only need specific small parts of a set of knowledge bases, depending on the terms contained in a domain model.

We propose a mediator-wrapper solution to implement the knowledge base querying. A mediator allows the interaction of a user or system with heterogeneous data sources in a uniform way [347]. Knowledge bases remain as they are, a wrapper is responsible for content translation,

[9]There are some works [343, 344] about encoding ConceptNet in RDF, but they still face issues with a succinct representation [345].

and the mediator provides a single point of access to the information for the modeling recommendations. Figure 5.7 shows the architecture of our approach. We differentiate between three different layers.

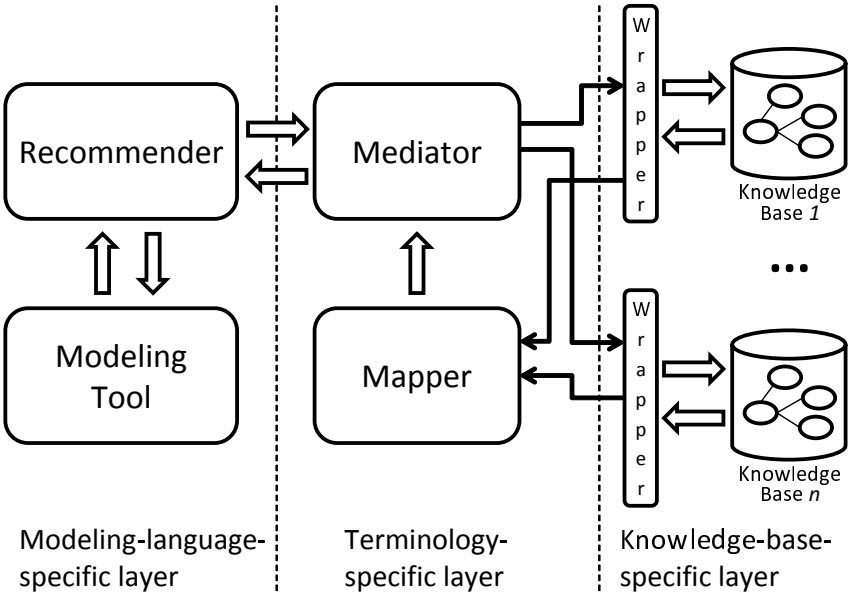


Figure 5.7: Three layer mediator-wrapper architecture

In the *modeling-language-specific level*, the developer uses the modeling tool and interacts with the recommender. This layer treats elements such as classes and associations, and the recommender proposes these types of elements based on the content of a model.

The mediator and the mapper are in the *terminology-specific level*. Domain-specific terms used in a model are relevant in this layer (e.g., nouns and their related terms). The mediator is responsible for translating terminology-specific content into the modeling layer and vice versa. It also manages a set of knowledge bases and their corresponding wrappers and sends queries to them as needed. The mapper collects and integrates results of the wrappers and provides the information to the mediator.

In the *knowledge-base-specific layer*, the wrappers communicate with the knowledge bases. Each wrapper must handle different query languages and formats (such as OWL, RDF, SPARQL, JSON) and different types of modeling (e.g., graphs, concepts, synsets).

The advantage of our architecture is that only a specific set of queries needs to be developed for a small portion of the information from each knowledge base and a mapping of the results. The OntoConnector component supports the **automated integration** of these types of data models without any development effort, as we exemplarily integrated WordNet and OpenCyc, based on these models:

- Ontology schemata: concepts and relationships modeled using OWL or RDFS classes and object properties
- SKOS-based vocabularies: terms modeled with concepts and broader, narrower, and related relationships [265]
- Lemon-based knowledge bases: a specific vocabulary for modeling lexicons of ontologies [297]

If none of these data models are present, we support **semi-automatic integration** of any knowledge base that offers a SPARQL endpoint. The effort to add a new knowledge base to the system is relatively small, it is only necessary to specify a small set of queries for taxonomic, part/whole, related and verbal relationships, as we show in the next section.

5.6 Knowledge Base Specific Queries

In this section, we describe in detail how the wrappers implement knowledge base specific queries for each knowledge base and how the results are mapped to the terminology specific layer. Figure 5.8 shows the kind of data that is exchanged between the components of our mediator-wrapper architecture. (1) Each of the wrappers is able to process knowledge base independent queries. The mediator submits the queries to the wrappers with the input terms. (2) The wrapper executes the respective knowledge base specific query using the provided query language (e.g., SPARQL or via HTTP REST interfaces). (3) The knowledge base returns a result set using the provided result language (e.g., XML or JSON). (4) The wrapper extracts the terminology information and returns it to the mapper. (5) The mapper integrates all intermediate result sets and returns the result back to the mediator.

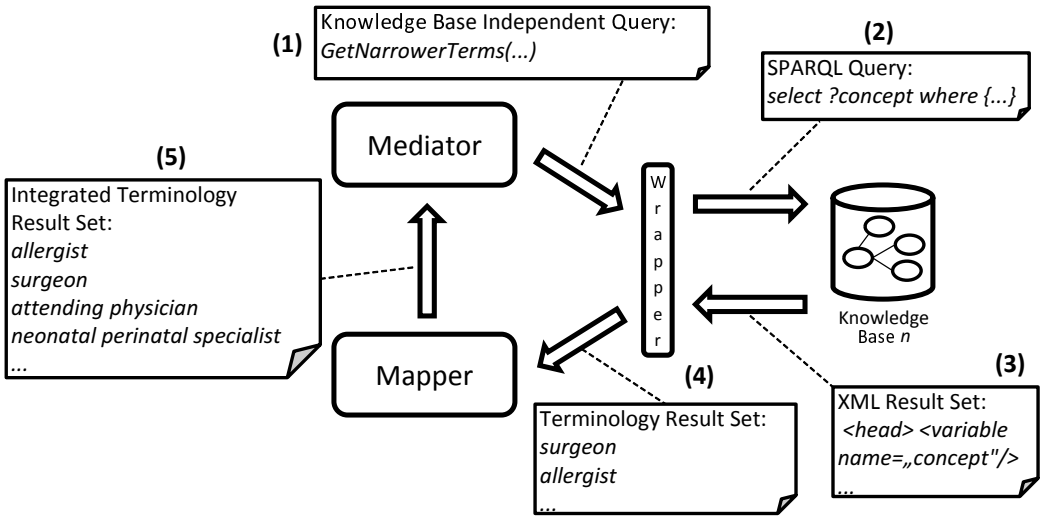


Figure 5.8: General approach of translating knowledge base independent queries to knowledge base specific queries

The following section is an overview of the steps that each wrapper implements, and after that the technical details regarding each knowledge base are presented.

5.6.1 Query Procedure

Input to a wrapper and thus input to a knowledge base specific query are one or more noun / verb terms of the terminology-specific layer. The terms, if present in the respective knowledge base, are represented by a particular object depending on the particular data model. The following steps are performed when a term is requested.

Keyword Search. To find related terms for an input term, the term is searched for presence in a knowledge base. Usually, terms are encoded as string label, name, or identifier of a knowledge base object (e.g. "doctor").

Target Object Retrieval. Once the object has been identified, it is retrieved along with its properties (e.g., the concept "Medical_Doctor").

Related Object Retrieval. The knowledge base is queried for related objects that match a specific relationship.

Term Retrieval. The terms used for the associated objects are determined.

Filtering. This optional step excludes related objects and / or terms from the results that meet certain criteria.

5.6.2 WordNet Specific Queries

In Section 5.4.1 we described the data model of WordNet and gave an example of how terminology is represented and linked in the knowledge base. The most important concept of WordNet is that words are modeled separately and grouped in synsets (the senses). Words can have multiple meanings (homonyms) and can belong to more than one synset. Semantic relationships (e.g., hypernym/hyponym) connect synsets and not words. Therefore, it is first necessary to search for the words and then to determine the respective synset in order to query related terms.

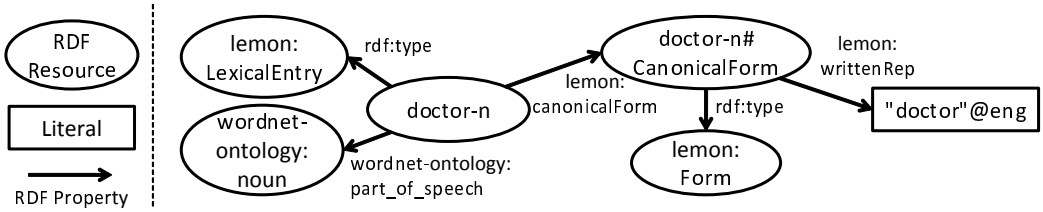


Figure 5.9: Lexical entries in WordNet RDF

Keyword Search. Figure 5.9 shows an example of how the noun "doctor" is modeled in WordNet RDF. The *doctor-n* object is classified as a *LexicalEntry* from the lemon model. The lexical entry has a noun *part_of_speech* relationship to the WordNet ontology. The noun has a canonical form, and its written representation is *"doctor"* in English. To find the appropriate nouns in WordNet RDF, we must search for canonical forms that contain a written representation with the keyword, and then retrieve the lexical entries that belong to the canonical forms. Please note that nouns, verbs and adjectives can have the same written representation. For this reason, the part of speech must be included in the query.

Listing 5.1 shows our SPARQL query for WordNet RDF[10] that returns a lexical entry for the corresponding keyword "doctor". Since the lexicon is separately modeled by WordNet, this query should always return exactly one noun for a given keyword[11]. Lines 1-3 define abbreviations for the respective knowledge base URIs. Line 5 returns a variable for the noun, line 6 filters all forms that are equal to "doctor", line 7 retrieves the lexical entries of the forms, and line 8 filters for nouns. The query result is shown in line 11 of the listing.

```
1  PREFIX lemon: <http://lemon-model.net/lemon#>
2  PREFIX wordnet: <http://wordnet-rdf.princeton.edu/>
3  PREFIX wordnet-ontology: <http://wordnet-rdf.princeton.edu/ontology#>
4
5  select ?noun where {
6   ?form lemon:writtenRep "doctor"@eng.
7   ?noun lemon:canonicalForm ?form.
8   ?noun wordnet-ontology:part_of_speech wordnet-ontology:noun.
9  }
10 ------------------------------------------------------------
11 http://wordnet-rdf.princeton.edu/wn31/doctor-n
```

Listing 5.1: WordNet SPARQL query that retrieves nouns for the keyword "doctor"

[10]The SPARQL endpoint of WordNet RDF is available at `http://wordnet-rdf.princeton.edu/sparql/`

[11]Note that written representation is case-sensitive. In WordNet, there is a second noun "Doctor" that is not retrieved by this query.

Target Object Retrieval. The next step is to determine the synset of the noun in WordNet. Many words have multiple meanings and therefore belong to multiple synsets. In this paragraph, we describe a simple version of the object retrieval that always uses the most commonly used synset. We also implemented an advanced version for retrieving related objects that takes other domain model terms as context. It determines the best matching synset based on the context words. Figure 5.10 shows how senses of a noun and reference to synsets are modeled in WordNet RDF. The lexical entry *doctor* refers to two senses *"1-n"* and *"4-n"* (there is also a third sense, which is not shown in the picture). Every sense has a number and refers to the corresponding synset of WordNet.

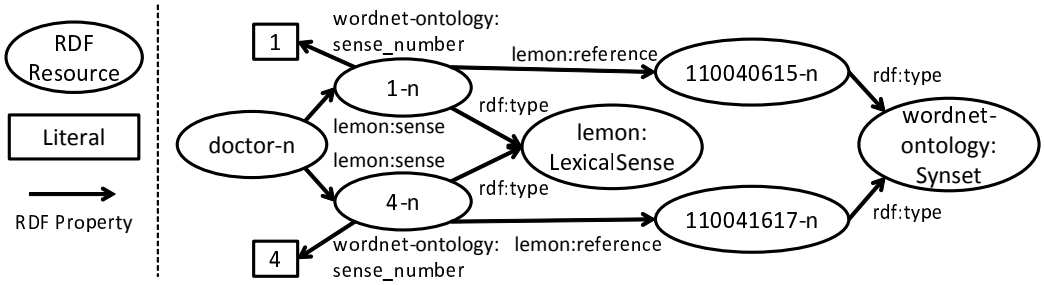


Figure 5.10: Multiple senses of lexical entries and their references to WordNet RDF synsets

In WordNet, the most common sense is always the first sense. Listing 5.2 shows a sample query that retrieves the first synset of the word "doctor". The query extends the previous query by lines 9-11, where all senses are matched and filtered by the sense number, and the reference to the synset is returned. The result of this query (line 14) is the synset *110040615-n*.

```
1  PREFIX lemon: <http://lemon-model.net/lemon#>
2  PREFIX wordnet: <http://wordnet-rdf.princeton.edu/>
3  PREFIX wordnet-ontology: <http://wordnet-rdf.princeton.edu/ontology#>
4
5  select ?synset where {
6   ?form lemon:writtenRep "doctor"@eng.
7   ?noun lemon:canonicalForm ?form.
8   ?noun wordnet-ontology:part_of_speech wordnet-ontology:noun.
9   ?noun lemon:sense ?sense.
10  ?sense wordnet-ontology:sense_number "1"^^<http://www.w3.org/2001/XMLSchema
        #integer>.
11  ?sense lemon:reference ?synset.
12 }
13 ------------------------------------------------------------------------
14 http://wordnet-rdf.princeton.edu/wn31/110040615-n
```

Listing 5.2: WordNet SPARQL query that retrieves the first synset for the keyword "doctor"

Related Object Retrieval. Once the synset of a keyword has been determined, we can now formulate queries for semantically related objects by matching the corresponding relationships of WordNet.

Figure 5.11 shows three aspects of a synset. Additional information of lexical sense is provided by a description (gloss) and by a sentence in which the term is used (sample). The synset member relationship links to all nouns (the synonyms) that are grouped in the synset. The figure also shows three more synsets connected with a hypernym relationship to the doctor's synset.

In Listing 5.3 we provide the query that matches all the hyponyms of the synset *110040615-n* (the doctor/physician synset). It can of course be combined with the previous query to directly retrieve all the hyponyms of a particular keyword. Lines 9-11 show part of the result.

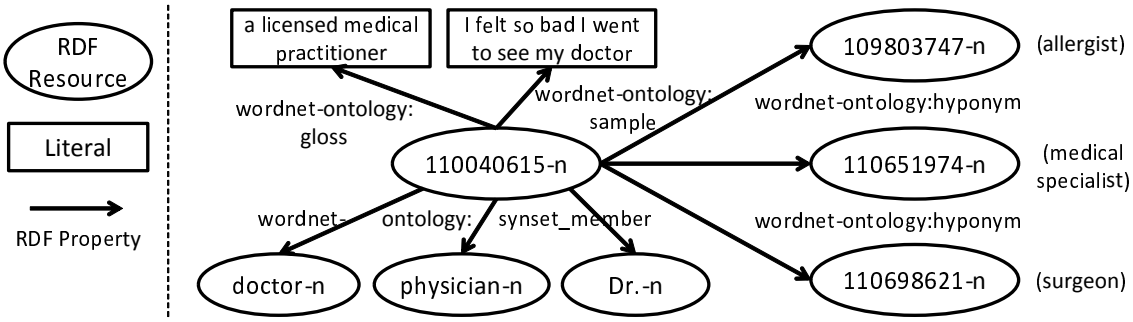


Figure 5.11: Synset members and hyponym relations in WordNet RDF

```
1  PREFIX lemon: <http://lemon-model.net/lemon#>
2  PREFIX wordnet: <http://wordnet-rdf.princeton.edu/wn31/>
3  PREFIX wordnet-ontology: <http://wordnet-rdf.princeton.edu/ontology#>
4
5  select ?hyponym where {
6   wordnet:110040615-n wordnet-ontology:hyponym ?hyponym
7  }
8  ------------------------------------------------------------------
9  http://wordnet-rdf.princeton.edu/wn31/109803747-n
10 http://wordnet-rdf.princeton.edu/wn31/110651974-n
11 http://wordnet-rdf.princeton.edu/wn31/110698621-n
12 ...
```

Listing 5.3: WordNet SPARQL query that retrieves all hyponyms of the doctor/physician synset

Term Retrieval. Determining the written representation for a synset can lead to several terms due to the synonym grouping in WordNet. Figure 5.12 shows the details of the surgeon's synset (*110698621-n*). Three lexical entries are members of the synset: *surgeon, operating surgeon, sawbones*. They all have a canonical form with a written representation (for space reasons we only show two of them in the picture).

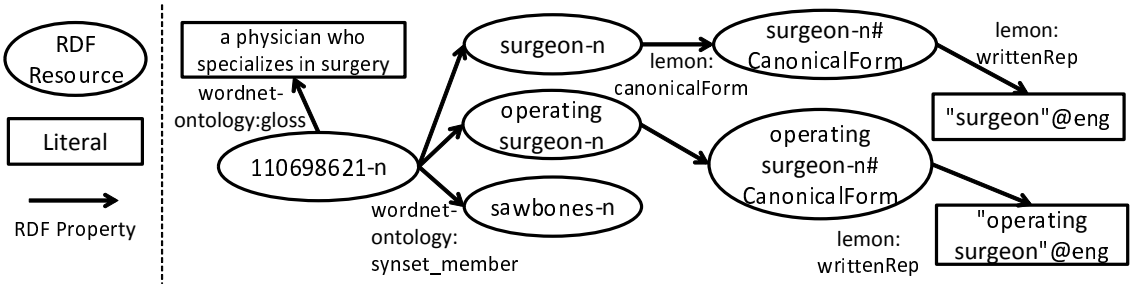


Figure 5.12: Canonical forms and their written representations of multiple synset members in WordNet RDF

Listing 5.4 extends the previous query. It also first retrieves the hyponyms of the doctor/-physician synset (line 6) and then follows all links to the synset members (line 7), their canonical forms (line 8), and the written representations (line 9). Lines 12-17 show part of the result (28 terms in total). No further filtering is required.

```
PREFIX lemon: <http://lemon-model.net/lemon#>
PREFIX wordnet: <http://wordnet-rdf.princeton.edu/wn31/>
PREFIX wordnet-ontology: <http://wordnet-rdf.princeton.edu/ontology#>

select ?term where {
 wordnet:110040615-n wordnet-ontology:hyponym ?hyponym.
 ?hyponym wordnet-ontology:synset_member ?noun.
 ?noun lemon:canonicalForm ?form.
 ?form lemon:writtenRep ?term.
}
-------------------------------------------------------------
resident
surgeon
intern
allergist
operating surgeon
sawbones
...
```

Listing 5.4: WordNet SPARQL query that retrieves the complete set of terms for all hyponyms of the doctor/physician synset

WordNet Relationship Mapping. In the previous paragraphs, we demonstrated how to retrieve narrower terms from WordNet RDF for noun keywords using a series of SPARQL queries. Because queries for other semantic relationships are implemented in a similar way, we present the mapping of all knowledge-base-independent queries (see Section 3.6) to the respective WordNet relationships in Table 5.1. Note that although WordNet contains many verbs, it neither provides noun-verb nor verb-noun relationships.

No.	Query Name	WordNet Relation
(1)	Get Broader Nouns	wordnet-ontology:hypernym
(2)	Get Narrower Nouns	wordnet-ontology:hyponym
(3)	Get Part Nouns	wordnet-ontology:member_meronym wordnet-ontology:part_meronym wordnet-ontology:substance_meronym
(4)	Get Whole Nouns	wordnet-ontology:member_holonym wordnet-ontology:part_holonym wordnet-ontology:substance_holonym
(5)	Get Related Nouns	*All wordnet-ontology relationships*
(6)	Get Related Verbs	*not supported by WordNet*

Table 5.1: Mapping of knowledge base independent queries to WordNet-specific relationships

5.6.3 OpenCyc Specific Queries

In Section 5.4.1 we described the data model of OpenCyc and gave an example of how terminology is represented and linked in the knowledge base. The OWL version of OpenCyc contains OWL classes that represent real world concepts. The classes have labels and aliases and they are linked with certain semantic relationships.

Keyword Search and Target Object Retrieval. OpenCyc does not model a separate vocabulary. Therefore, keyword search and target object retrieval are performed in one step. Figure 5.13 shows the concept *Medical_Doctor* and all its labels and part of the description. In OpenCyc, a concept always has exactly one RDF label and references zero or more alias labels with a property called *prettyString*. The illustration shows the human-readable version of OpenCyc OWL. That is, the concept has an English name and refers to its identifier via an additional link.

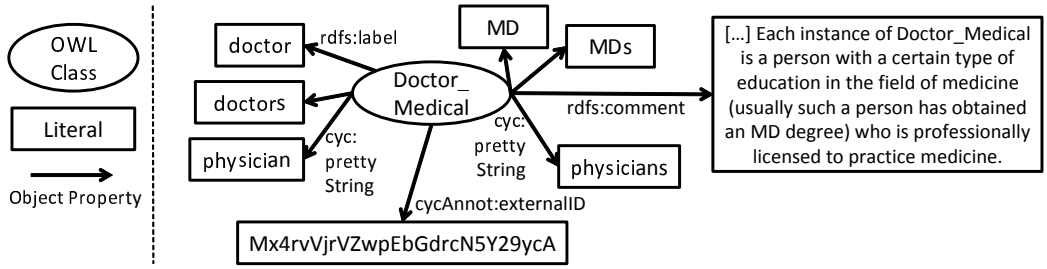


Figure 5.13: Labels and aliases in OpenCyc

Cycorp, Inc. has never directly supported SPARQL queries on OpenCyc, but OpenCyc's OWL data dumps can easily be loaded into a triple store. Consequently, we use the SPARQL syntax for knowledge base queries as we did for WordNet. Unfortunately, the OpenCyc web interface was discontinued in early 2017. Therefore, it is no longer possible to search OpenCyc using the HTML pages rendered from the knowledge base. Listing 5.5 shows how to identify all concepts that are OWL classes and labeled "doctor". The query searches RDF labels and OpenCyc prettyString labels. Line 13 shows the result of the query.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX owl: <http://www.w3.org/2002/07/owl#>
4  PREFIX cyc: <http://sw.opencyc.org/concept/>
5
6  select ?concept where {
7    {?concept rdfs:label "doctor"}
8    UNION
9    {?concept cyc:prettyString "doctor"}
10   ?concept rdf:type owl:Class.
11 }
12 ------------------------------------------------------------------------
13 http://sw.opencyc.org/concept/Doctor_Medical
```

Listing 5.5: OpenCyc query that retrieves concepts from OpenCyc that have the label "doctor"

Related Object Retrieval. To satisfy our knowledge base independent queries, we use OpenCyc's taxonomic relationships and instance relationships to second-order collections. OpenCyc does not support part-whole relationships or data related to verbs.

Figure 5.14 presents examples of more specific concepts of the *Doctor_Medical* concept. On the left, direct sub-concepts are shown using the RDF(S) subclass relationship. Listing 5.6 shows the appropriate query to retrieve these concepts. It extends the previous query at line 8 and returns sub-concepts. In order to determine super-concepts of a particular concept, only the subject and the object of line 8 need to be switched.

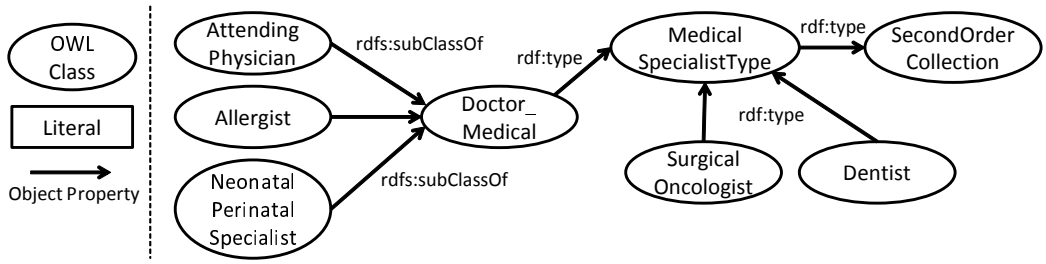


Figure 5.14: Taxonomic relations and references to second order collections in OpenCyc

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX owl: <http://www.w3.org/2002/07/owl#>
4
5  select ?subconcept where {
6   ?concept rdfs:label "doctor".
7   ?concept rdf:type owl:Class.
8   ?subconcept rdfs:subClassOf ?concept.
9  }
10 ------------------------------------------------------------
11 http://sw.opencyc.org/concept/AttendingPhysician
12 http://sw.opencyc.org/concept/Allergist
13 http://sw.opencyc.org/concept/NeonatalPerinatalSpecialist
14 ...
```

Listing 5.6: OpenCyc query that retrieves sub-concepts of concepts from OpenCyc that have the label "doctor"

The second aspect shown in Figure 5.14 is that *Doctor_Medical* is also an instance of *MedicalSpecialistType*. Other instances of this second order collection are shown in the picture on the right. Members of a second order collection are of the same kind and are therefore related terms. Listing 5.7 shows a query to find related concepts of the Doctor_Medical concept. It first follows all type associations (line 7) and filters them for second order collections (line 8). The second filter (line 9) excludes second order collections that are topics. Then all instances of the collection are retrieved (line 10). In some cases, the result of such a query overlaps with the result of the sub-concept query (e.g., the allergist is a more specific medical doctor, but also a medical specialist).

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX owl: <http://www.w3.org/2002/07/owl#>
4  PREFIX cyc: <http://sw.opencyc.org/concept/>
5
6  select ?relatedconcept where {
7   cyc:Doctor_Medical rdf:type ?secordcoll.
8   ?secordcoll rdf:type cyc:SecondOrderCollection.
9   ?secordcoll rdf:type cyc:FacetingCollectionType.
10  ?relatedconcept rdf:type ?secordcoll.
11 }
12 ------------------------------------------------------------
13 http://sw.opencyc.org/concept/AddictionMedicineSpecialist
14 http://sw.opencyc.org/concept/SurgicalOncologist
15 http://sw.opencyc.org/concept/Dentist
16 ...
```

Listing 5.7: OpenCyc query that retrieves related concepts of the Doctor_Medical concept

Term Retrieval. Having identified more general, more specific, or related concepts, we retrieve the terms associated with each concept. Listing 5.8 shows an example of retrieving all labels of the concept *AddictionMedicineSpecialist*.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX cyc: <http://sw.opencyc.org/concept/>

select ?terms where {
 {cyc:AddictionMedicineSpecialist rdfs:label ?terms.}
 UNION
 {cyc:AddictionMedicineSpecialist cyc:prettyString ?terms.}
}
-------------------------------------------------------------
addiction medicine specialist
addiction medicine specialists
addictionist
addictionists
```

Listing 5.8: OpenCyc query that retrieves all terms of a concept

Filtering. OpenCyc contains a variety of concepts that are built with existing concepts and functions (for example, a collection intersection function). These concepts cannot be used for proper terminology retrieval. We filter the results that are annotated with a function name. Listing 5.9 shows the extension of query 5.6, excluding concepts that contain "Fn".

```
...
  ?subconcept rdfs:subClassOf ?concept.
  ?subconcept <http://sw.cyc.com/CycAnnotations_v1#label> ?annot.
  FILTER regex(?annot, "^((?!Fn).)*$")
...
```

Listing 5.9: Query extension to filter OpenCyc functions

The result of the query in Listing 5.8 is just one example of how OpenCyc contains many synonymous labels, which are variations in capital letters or plural forms of the main term. These are filtered in the mapper component of the OntoConnector.

OpenCyc Relationship Mapping. Table 5.2 summarizes the mapping of all knowledge base independent queries (see Section 3.6) to the relationships used in OpenCyc. Only taxonomic relationships and related terms are supported by OpenCyc.

No.	Query Name	OpenCyc Relation
(1)	Get Broader Nouns	rdfs:subClassOf inverse
(2)	Get Narrower Nouns	rdfs:subClassOf
(3)	Get Part Nouns	*not supported by OpenCyc*
(4)	Get Whole Nouns	*not supported by OpenCyc*
(5)	Get Related Nouns	rdf:type of second-order collections
(6)	Get Related Verbs	*not supported by OpenCyc*

Table 5.2: Mapping of knowledge base independent queries to OpenCyc-specific relationships

5.6.4 ConceptNet Specific Queries

In Section 5.4.3 we described the data model of ConceptNet and gave a small example of how terminology is represented and linked in the knowledge base. Concepts are labeled nodes in a graph and are linked with labeled edges. Because of the simple data model we can perform keyword search, target object retrieval and related object retrieval in one step. Figure 5.15 shows examples of taxonomic relations in ConceptNet.

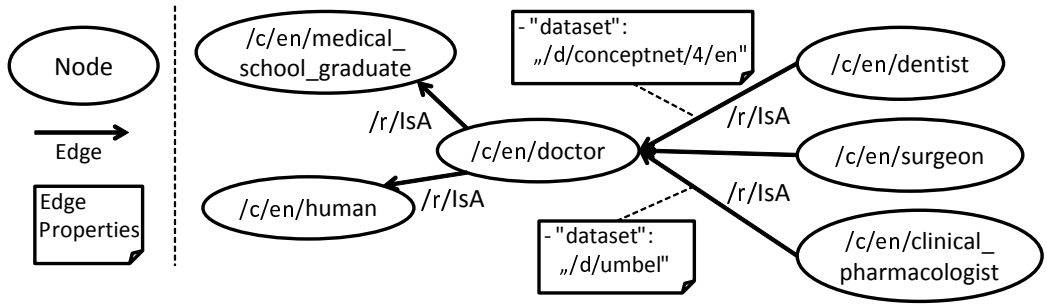


Figure 5.15: Taxonomic relationships in ConceptNet

Keyword Search, Target Object Retrieval and Related Object Retrieval. There are three methods to acquire information from ConceptNet that we use for our modeling suggestions. (1) Concept lookup: If you know the URI of a concept you can retrieve all the edges that are connected to the concept. (2) Concept queries: The search allows to query for start nodes, end nodes, and specific relationships, as well as most of the values of edge properties. (3) Related concepts: ConceptNet offers a feature to retrieve a ranked list of similar concepts for an input concept.

Listing 5.10 shows a lookup query for the concept *doctor* in the English language. The API URL is followed by the concept identifier and optional offset and limit. From the result we took one edge in which the *doctor* concept occurs (the list of sources was removed for space reasons). The edge describes the */r/IsA* relationship (line 22) between the start concept */c/en/surgeon* (line 28) and the end concept */c/en/doctor* (line 14). Concept lookup can be used to test the existence of a concept.

```
1  http://api.conceptnet.io/c/en/doctor?offset=0&limit=200
2  ------------------------------------------------------------------------
3  {
4    "@context": [
5      "http://api.conceptnet.io/ld/conceptnet5.6/context.ld.json"
6    ],
7    "@id": "/c/en/doctor",
8    "edges": [
9      {
10       "@id": "/a/[/r/IsA/,/c/en/surgeon/,/c/en/doctor/]",
11       "@type": "Edge",
12       "dataset": "/d/conceptnet/4/en",
13       "end": {
14         "@id": "/c/en/doctor",
15         "@type": "Node",
16         "label": "a doctor",
17         "language": "en",
18         "term": "/c/en/doctor"
19       },
20       "license": "cc:by/4.0",
21       "rel": {
```

```
        "@id": "/r/IsA",
        "@type": "Relation",
        "label": "IsA"
      },
      "sources": [ ... ],
      "start": {
        "@id": "/c/en/surgeon",
        "@type": "Node",
        "label": "a surgeon",
        "language": "en",
        "term": "/c/en/surgeon"
      },
      "surfaceText": "[[a surgeon]] is [[a doctor]]",
      "weight": 4.47213595499958
    },
    ...
```

Listing 5.10: ConceptNet web API lookup to retrieve all edges for the concept /c/en/doctor

In order to implement our knowledge base independent queries it is required to query for specific relationships that are connected to the respective concepts. Listing 5.11 shows a concept query for all *IsA* relationships that are starting from the */c/en/doctor* concept. The resulting target nodes are broader concepts of the *doctor* concept.

```
http://api.conceptnet.io/query?start=/c/en/doctor&rel=/r/IsA&offset=0&limit
    =100
-------------------------------------------------------------------------
{
  "@context": [
    "http://api.conceptnet.io/ld/conceptnet5.6/context.ld.json"
  ],
  "@id": "/query?rel=/r/IsA&start=/c/en/doctor",
  "edges": [ {
      "rel": { "@id": "/r/IsA", ... },
      "end": { "@id": "/c/en/man_of_science", ... },
      "start": { "@id": "/c/en/doctor", ... },
      "surfaceText": "[[A doctor]] is [[a man of science]]",
      "weight": 2.0
        ...
    }, {
      "rel": { "@id": "/r/IsA", ... },
      "end": { "@id": "/c/en/medical_practitioner/n", ... },
                        "start": { "@id": "/c/en/doctor", ... },
      "surfaceText": "[[doctor]] is a type of [[medical practitioner]]",
      "weight": 2.0
        ...
    },
    ...
```

Listing 5.11: ConceptNet web API query to determine all *IsA* edges in which the concept /c/en/doctor is the start node

ConceptNet includes a *relatedTo* relationship that explicitly connects concepts that are related. Consequently, related terms can be queried as shown in the previous query. Recently, ConceptNet started to offer another feature for retrieving related terms. This is made possible with ConceptNet Numberbatch[12], a dataset of pre-computed multilingual word embeddings created from ConceptNet, word2vec, GloVe, and OpenSubtitles. The API was extended to query similar concepts for a single concept and to determine relatedness between two concepts.

[12] `https://github.com/commonsense/conceptnet-numberbatch`

Listing 5.12 shows an example to retrieve the most related concepts of *doctor*. Each of the result entries includes the respective concept identifier accompanied with a measurement of how similar the concepts are.

```
http://api.conceptnet.io/related/c/en/doctor?filter=/c/en
-------------------------------------------------------------
{
  "@id": "/c/en/doctor",
  "related": [
    {"@id": "/c/en/doctor", "weight": 1.0 },
    {"@id": "/c/en/physician","weight": 0.877},
    {"@id": "/c/en/attending_physician","weight": 0.849},
    {"@id": "/c/en/dpt","weight": 0.845},
    {"@id": "/c/en/doctors","weight": 0.783},
    {"@id": "/c/en/cardiothoracic_surgeon","weight": 0.746},
    {"@id": "/c/en/physicians","weight": 0.707},
    {"@id": "/c/en/medico","weight": 0.7},
    ...
```

Listing 5.12: ConceptNet web API query to determine related concepts of /c/en/doctor filtered by English language

Term Retrieval. All previously shown JSON result sets have in common that the terms of linked nodes are specified using properties of the start and end nodes. In order to return sets of terms two steps are performed: (1) The JSON result is parsed for the respective start or end nodes. (2) The property *term* is extracted, concept and language prefixes are removed, and underscore characters are replaced with spaces.

Filtering. The concept search function of ConceptNet is not restrictive enough to only return edges that exactly match the specified URI. In some cases the result also contains concepts having longer names that start with the keyword or are followed by word variations. These will be removed from the result set. We also filter edges that have a weight of zero or a negative weight (indicating the statement is not true).

ConceptNet Relationship Mapping. Table 5.3 summarizes the mapping of all knowledge base independent queries (see Section 3.6) to the relationships used in OpenCyc. Only taxonomic relationships and related terms are supported by OpenCyc.

No.	Query Name	ConceptNet Relation
(1)	Get Broader Nouns	/r/IsA + start node
(2)	Get Narrower Nouns	/r/IsA + end node
(3)	Get Part Nouns	/r/PartOf + end node /r/HasA + start node
(4)	Get Whole Nouns	/r/PartOf + start node /r/HasA + end node
(5)	Get Related Nouns	/r/RelatedTo + start node /r/SimilarTo + start node *Numberbatch related query*
(6)	Get Related Verbs	*not supported by ConceptNet*

Table 5.3: Mapping of knowledge base independent queries to ConceptNet-specific relationships

5.7 Query Result Integration

In our mediator-wrapper architecture (see Figure 5.7 on page 122) each knowledge base is queried separately. The wrapper of each connected knowledge base delivers a list of terms to the mapper component depending on the query. The integration is performed straightforward in three steps. (1) **Normalization**: We lowercase all terms except for words that only contain uppercase letters (usually acronyms). Plural nouns are reduced to their singular form. Adjectives in multiword expressions are reduced to their normal form. Normalization is implemented using the morphology component of the Stanford CoreNLP toolkit[13]. (2) **Approximate String Matching**: In order to reduce further word variants (e.g., American and British English spelling differences) we apply fuzzy string search on the resulting list of terms. It is implemented using the Java implementation of FuzzyWuzzy[14]. (3) **Duplicate Detection**: In the final list of terms duplicates are eliminated, but it is preserved how often a term occurred. The frequencies are an important indicator for subsequent ranking. Terms found in many knowledge bases should receive more prominent positions than terms that occurred only once.

5.8 Templates for Knowledge Base Integration

Based on the exemplary integration of WordNet, OpenCyc and ConceptNet, we now derive generic templates in this section that allow us to integrate a variety of knowledge bases that use the same data models. The respective templates are already implemented in the OntoConnector component, thus the integration of further knowledge bases using the respective data models is possible without any effort.

5.8.1 Lemon-Based Lexical Resources

The first template (cf., Listing 5.13) is designed to retrieve related terms from knowledge bases that use the Lexicon Model for Ontology (lemon) for their terminology. It searches for keywords in the written representations of the canonical forms of all lexical entries (line 5-7). One or more possible senses reference the concept of the target knowledge base (lines 7-8). The concept is connected to a related concept using a specific relationship of the target knowledge base that must be inserted in line 9. Finally, lines 10-13 follow the properties to the lexical sense, lexical entry and retrieve related terms.

```
1  PREFIX lemon: <http://lemon-model.net/lemon#>
2  PREFIX tkb: <http://namespace.targetknowledgebase.org/ns#>
3
4  select DISTINCT ?relatedTerm where {
5   ?form lemon:writtenRep "<<<keyword>>>".
6   ?entry lemon:canonicalForm ?form.
7   ?entry lemon:sense ?sense.
8   ?sense lemon:reference ?ontologyConcept.
9   ?ontologyConcept <<<tkb:relationshipName>>> ?relatedConcept.
10  ?relatedConcept lemon:isReferenceOf ?relatedSense
11  ?relatedSense lemon:sense ?relatedEntry
12  ?relatedEntry lemon:canonicalForm ?relatedform.
13  ?relatedform lemon:writtenRep ?relatedTerm.
14 }
```

Listing 5.13: Template for a SPARQL query to retrieve related terms for a given keyword from a lemon-based knowledge base

[13] https://stanfordnlp.github.io/CoreNLP/
[14] https://github.com/xdrop/fuzzywuzzy

Currently, we are aware of the following resources that have been converted using the lemon-model for integration into the Linguistic Linked Open Data Cloud:

- WordNet RDF [299] – the lexical database for English
- BabelNet 2.0 [348] – a multilingual encyclopedic dictionary
- DBnary [349] – Wiktionary as a multilingual lexical resource in RDF
- UBY [350] – a linked lexical resource for English and German

5.8.2 OWL Ontology Schemata

The second template developed integrates OWL encoded ontology schemata. Normally, the predominant content of OWL schemata is taxonomic knowledge, thus the relationship name in line 9 of Listing 5.14 will be replaced by *rdf:subclassOf* for narrower terms and subject/object will be switched for broader terms retrieval. Anyway, if the knowledge base encodes other forms of knowledge that can be used for domain modeling (e.g., part-of relationships), these user-defined properties can be used in line 9 instead. In the template, resulting terms are filtered for the English language (line 11) as many knowledge bases provide labels in multiple languages.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX owl: <http://www.w3.org/2002/07/owl#>
4  PREFIX tkb: <http://namespace.targetknowledgebase.org/ns#>
5
6  select DISTINCT ?relatedTerm where {
7   ?concept rdfs:label "<<<keyword>>>"@en.
8   ?concept rdf:type owl:Class.
9   ?relatedconcept <<<tkb:relationshipName>>> ?concept.
10  ?relatedconcept rdfs:label ?relatedTerm .
11  FILTER (lang(?relatedTerm) = "en")
12 }
```

Listing 5.14: Template for a SPARQL query to retrieve related terms for a given keyword from OWL-based ontologies

There is a huge amount of OWL schemata and vocabularies that cannot be listed here in detail. The Linked Open Vocabularies (LOV) dataset[15] provides a good entry point to search for terms and domains and retrieve further information on published vocabularies.

5.8.3 SKOS Vocabularies

For reasons of space, we have not demonstrated the exemplary integration of vocabularies based on the Simple Knowledge Organization System (SKOS)[16] in Section 5.4. However, these types of knowledge sources are easy to integrate. Therefore, we provide a template for their integration (see Listing 5.15).

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
4
5  select DISTINCT ?relatedTerm where {
6   ?concept skos:prefLabel "<<<keyword>>>"@en.
7   ?concept rdf:type skos:Concept.
8   ?concept <<<skos relationship>>> ?relatedConcept.
```

[15] https://lov.linkeddata.es/dataset/lov/
[16] https://www.w3.org/2004/02/skos/

```
 {?relatedConcept skos:prefLabel ?relatedTerm.}
 UNION
 {?relatedConcept skos:altLabel?relatedTerm.}
 FILTER (lang(?relatedTerm) = "en")
}
```

Listing 5.15: Template for a SPARQL query to retrieve related terms for a given keyword from a SKOS-based vocabulary

As with OWL schemata, there are a variety of SKOS vocabularies and thesauri (e.g., EUROVOC, LCSH, SNOMED, DBpedia categories, just to name a few important ones), which cannot be detailed here. Suominen & Mader [351] provide a qualitative study with 24 vocabularies. More vocabularies can also be found on the W3C SKOS website[17].

5.8.4 JSON-LD APIs

Providing a generic template for proprietary JSON-LD RESTful APIs is a challenging task. API calls are not standardized, and the returned format can vary a lot depending on the context specification although semantically equivalent. In order to access these APIs in a similar way as shown before, we use SPARQL-LD [352, 353], an extension of SPARQL 1.1 Federated Query that allows to directly access and use linked data contained in arbitrary resources on the Web (e.g., embedded RDF, JSON-LD files, RDF files) in SPARQL queries. Listing 5.16 shows the respective template that uses the SERVICE operator for the API call and matches triples according to the relationships of the knowledge source.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX tkb: <http://namespace.targetknowledgebase.org/ns#>

select DISTINCT ?relatedTerm where {
  SERVICE <http://api.call.org/query/keyword> {
    ?concept rdfs:label "<<<keyword>>>"@en.
    ?concept rdf:type <<<tkb:typeName>>>
    ?relatedconcept <<<tkb:relationshipName>>> ?concept.
    ?relatedconcept rdfs:label ?relatedTerm .
  }
}
```

Listing 5.16: Template for a SPARQL-LD query to retrieve related terms from a JSON-LD-based API call

5.9 Summary

In this chapter, we presented the detailed approach of how conceptual terms and their lexical relationships are extracted from heterogeneous knowledge bases. We have analyzed a number of knowledge bases that use different data models with respect to the organization of lexical and conceptual content. We have implemented OntoConnector, a component that realizes a mediator-based knowledge base querying approach to provide unified access to information about terms and their relationships at the schema level, in proprietary intermediate data models, and at the instance level. Query results are mapped to modeling language-specific concepts to generate model element suggestions. From the exemplary integration of WordNet, OpenCyc, and ConceptNet, a number of generic templates for integrating knowledge bases have been derived that allow the simple integration of a variety of knowledge bases that expose either a lemon lexicon, an OWL schema, a SKOS vocabulary, or a JSON-LD API.

[17] https://www.w3.org/2001/sw/wiki/SKOS/Datasets

Chapter 6

DoMoRe: Implementation of the Recommender System

The thesis has so far developed the methodological foundations for domain modeling support and two methods for retrieving domain knowledge, namely SemNet, which provides access to a large number of semantically related terms derived from text datasets, and OntoConnector, which provides uniform access to structured lexical information and relationships from external knowledge bases. There is a huge amount of information available, but a modeling expert will have difficulty choosing information relevant to a project or domain model, even if suitable search tools are available. This chapter elaborates on how to provide the user with specific pieces of information and how to present the information to the user using automated recommendations. The implemented system presented here combines the developed results of the previous chapters under one roof and thus also addresses the first challenge of domain modeling: the high costs of acquiring domain knowledge.

6.1 Introduction

In this chapter, we describe DoMoRe, a domain modeling recommender system that suggests related model elements for domain models. It implements the developed semantic modeling support method of this thesis and integrates SemNet and OntoConnector to provide the user with context-aware information during model development. The recommender system extends a popular modeling environment, the Eclipse Ecore Diagram Editor, part of the Eclipse Modeling Project. We provide details on how the recommendations from SemNet and connected knowledge bases are generated and show two features that present the suggestions to the user.

Section 6.2 reviews related approaches and systems with similar objectives. In Section 6.3 we describe the modeling tool and the environment, which are extended by our recommendation system. Section 6.4 shows the architecture of the system. Section 6.5 describes how to derive recommendations for the respective types of model elements from SemNet and the connected knowledge bases. The implementation of the ranking strategy is given in Section 6.6. Details about the Semantic Autocompletion and the Model Advisor feature are presented in Section 6.7. The chapter is summarized in Section 6.8.

6.2 Related Recommender Systems

In this section we summarize the state of the art in modeling recommendation systems. We start with repository-based approaches and works that focus on single aspects of recommendation, and then compare our work with two recent recommendation systems.

6.2.1 Modeling Assistance Approaches

Modeling support systems provide additional information and functionality during the modeling process to help model development. They typically focus on two areas: (1) creating model libraries or similar content; and (2) developing assistance frameworks and functions using these libraries. The largest known *model repository* of UML models and meta models is the Lindholmen UML dataset [354]. It contains links to over 93,000 UML diagrams collected from GitHub repositories. A similar effort, the Gothenburg UML Repository[1] contains over 20,000 models crawled from the Internet, images and GitHub (only a collection of nearly 1,000 models is publicly searchable). Not surprisingly, most of the models are implementation models rather than domain models (for example, the search for "hospital" or "doctor" yielded 7 models, while a search for "interface" returned 90 models). Other important resources are ReMoDD [355], MOOGLE [356], the AtlanMod Metamodel Zoos[2] (containing a total of several hundred models). EMFStore[3] and the Eclipse Model Repository[4] are tools for maintaining model repositories. There are works that propose certain *recommendation features*: SmartEMF [233] uses reasoning in Prolog for consistency checking in DSL development. Kuhn proposes a concept for recommending method names in source code and UML models [357].

6.2.2 HERMES Recommender Project

The HERMES project addresses modeling support with a framework approach. It is a research prototype supporting the reuse of software models [358]. Its main objective is to provide tool support for creating model libraries [359] and creating recommenders that use the contents of these model libraries during model development [360].

The framework consists of four components: (1) Harvest: This component helps to identify reusable parts in existing models and stores them in a model library. (2) Evolve: This component is responsible for model management within the model library to evaluate and improve models according to quality guidelines. (3) Reuse: This component helps the developer to find suitable models in model libraries through recommendations. (4) Store: This component is the back end of the model library and stores models in a database and provides access to external model storage (e.g., Git or directory-based access).

At first glance, the HERMES project appears to be very similar to DoMoRe, but a detailed analysis shows the different strategies to achieve the modeling support. The main differences with regard to this thesis are as follows:

HERMES focuses on the reuse of models. Consequently, the framework focuses on a paradigm shift from copy & paste modeling [360] to model library recommendations. The assumption of HERMES is that model libraries exist and are maintained. The project also provides methods to manually create quality-assured model libraries [361]. In contrast, we have analyzed that existing model libraries (e.g., ReMoDD [355], MOOGLE [356], or Eclipse Model Repository [362]) are by no means sufficient to provide model recommendations for all types of domains. They contain only a small number of models[5]. In addition, many of them are specific to software implementations and cannot be used for domain modeling. The HERMES approach faces the chicken or egg dilemma. Without a large number of existing models, the recommendations are very limited. Without sophisticated recommendations, less support for creating domain models is achieved, resulting in fewer contributions to a model library. A review of the publications related to the HERMES project did not indicate the size of the model libraries that were collected or used for recommendations. In contrast, our work primarily addresses the bottleneck of existing models. We have built a large knowledge base for domain modeling ourselves, relying on preconfigured and

[1] `http://models-db.com/`

[2] `http://web.emn.fr/x-info/atlanmod/index.php?title=Zoos`

[3] `http://www.eclipse.org/emfstore/`

[4] `http://modelrepository.sourceforge.net`

[5] For example, on April 25, 2019, ReMoDD (`http://www.cs.colostate.edu/remodd/v1/`) contained only 60 models. Only 6 of them were class diagrams and 3 of them Ecore models. The MOOGLE search engine (discontinued) was based on approximately 150 EMF models.

extensible plug-in mechanisms for external knowledge sources, which contain many more elements for recommendations than existing model libraries.

The HERMES project is a framework approach [363]. It focuses on architectural foundations and provides programming skeletons to develop custom recommendations in a command-based model editor [364]. Consequently, concrete recommendation algorithms, presentation of modeling suggestions, connections to external model libraries must be implemented according to the needs of someone who wants to develop a recommender and are not part of HERMES. Nonetheless, there is documentation on how to implement a recommendation strategy based on WordWeb[6] (parsing the free version of this particular dictionary website). In contrast, this thesis provides a ready-to-use recommendation system that is able to provide modeling suggestions based on several already-linked knowledge sources. It implements a holistic recommendation algorithm based on statistical semantics to create model element suggestions in a uniform way.

6.2.3 EXTREMO Assistant

The EXTREMO Assistant [365] is an extensible tool for facilitating meta-model development. The approach is based on repositories with a common data model that can be uniformly queried. Repositories are created from heterogeneous file sources, supporting the EMF, OWL, and XSD technological spaces [366].

The main components are: (1) A common data model for storing information about model elements extracted from heterogeneous sources, (2) a persistence component that stores extracted information in a repository, (3) a query service that uses fuzzy and synonym search using WordNet, and (4) a user interface subsystem for search, repository exploration, and modeling tool interaction.

The assistant distinguishes itself from our DoMoRe recommendation system in the following respects: (1) Like in the HERMES project, EXTREMO has the main objective to provide a modeling assistance infrastructure. Its focus is on integrating multiple information sources into the Eclipse infrastructure. There are ready-to-use visualizations and connectors, but the system integrates only a few sample cases, such as: The Atlanmod Metamodel Zoo, some OMG specification meta-models (e.g., BPMN, SBVR), and two sets of US government OWL files. As a result, the effort to acquire knowledge is still high as appropriate sources of information must be discovered and analyzed to be programmatically integrated into the system. (2) The authors chose a common repository approach (see Section 5.2.3 Data Centralization) that requires full processing of each source of information as well as extraction and transformation of the relevant data into the EXTREMO target data model. This is problematic for large sources that require a timely conversion step. As the assistant only supports file-based sources, it cannot query online databases, for example, RDF data using SPARQL. (3) The user must actively search for information he is interested in or explore the repository. The query services help him to find related terms with WordNet, but no automatic recommendations are made based on the current content of a model.

6.3 Eclipse Modeling Environment

The Eclipse Modeling Project (EMP)[7] is a customized Eclipse distribution that provides a set of (meta) modeling tools and frameworks for developing domain-specific languages, modeling tools, and for model-driven software development. At the heart of EMP is the Eclipse Modeling Framework (EMF) that contains the Ecore Meta-Meta Model for the development of meta-models and domain models in general. EMF provides tools for developing meta-models, code-generation functions for deriving suitable implementation classes in Java, and a generic model editor for testing instances of meta-models. EMF is the starting point for the abstract syntax development of modeling languages. In addition, EMP contains several other tools: concrete syntax design (xText for textual modeling, Sirius and GMF for graphical notation), model-to-model transformations

[6] `http://wordweb.info/`
[7] `https://www.eclipse.org/modeling/`

(ATL, QVT, epsilon), model-to-text transformations (Acceleo, JET), and model storage and retrieval (CDO, Teneo).

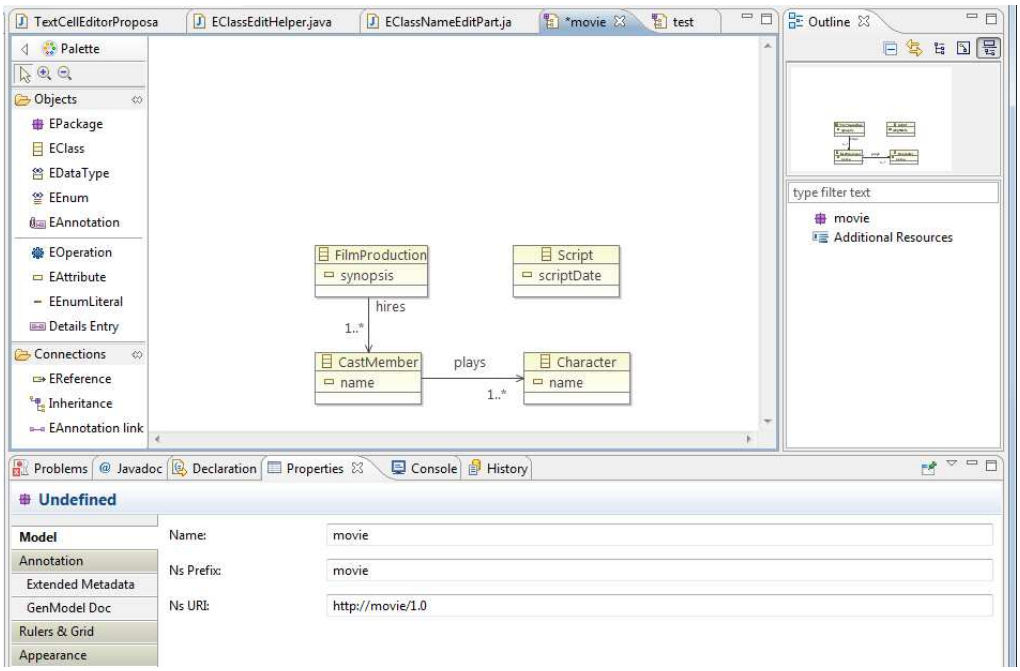


Figure 6.1: Ecore Diagram Editor

When launching a domain modeling project with Eclipse, one usually comes first into contact with the EcoreTools that include a graphical editor for EMF models, called Ecore Diagram Editor (see Figure 6.1). It provides a graphical user interface similar to other UML tools for designing classes, attributes, operations, data types, and references for Ecore models. On the left side of the editor, a palette with the available meta-model elements is offered. In the middle is the diagram area for creating, customizing and connecting model elements. In the lower area, properties of the elements can be manipulated. On the right side, an overview of the model and additional information are displayed.

The Ecore Diagram Editor, and also the complete Eclipse Modeling Framework is a set of Eclipse plug-ins. The Eclipse platform architecture is based on a plug-in model. *"Plug-ins are structured bundles of code and/or data that contribute functionality to the system. Functionality can be contributed in the form of code libraries (Java classes with public API), platform extensions, or even documentation. Plug-ins can define extension points, well-defined places where other plug-ins can add functionality."*[8].

The Eclipse Platform Architecture as shown in Figure 6.2 provides a flexible way to extend existing plug-ins and to integrate new functionality. It is used to build the required components that implement the semantic modeling support for a widely used domain modeling tool.

6.4 Architecture

The goal of implementing the recommender system is to extend the Ecore Diagram Editor and Eclipse environment to access the content of a model being developed, to respond to user interactions with the editor, to retrieve information from connected knowledge sources, and to recommend model elements when modeling. To implement the proposed support features, several plug-in types

[8] `https://help.eclipse.org/neon/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/arch.htm`

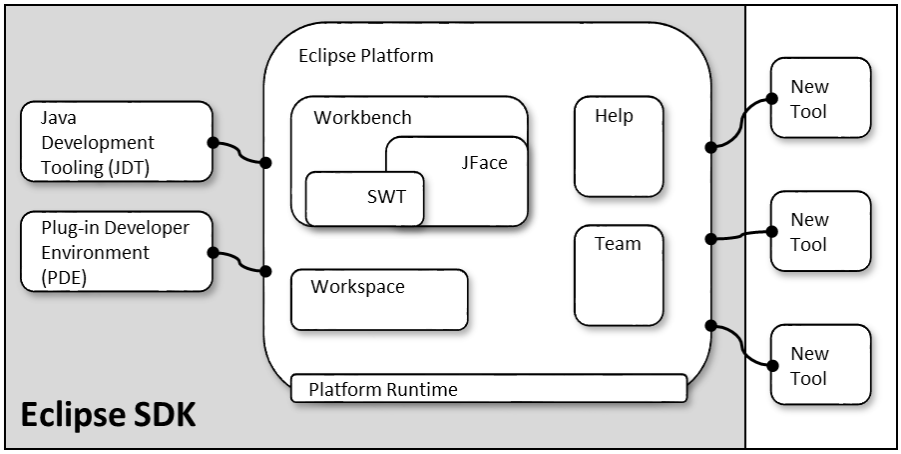


Figure 6.2: Eclipse Platform Architecture (from Eclipse Neon Help, Chapter "Platform Plug-in Developer Guide")

must be integrated into the Eclipse environment: The Eclipse environment has an event notification system for all types of component interactions with the platform. A background plug-in with two listener functions is attached to the platform core. It recognizes each time the diagram editor is started and used. If so, a second listener is attached to the editor to receive notifications of the user interactions. The contextual information display is implemented as an Eclipse view plug-in, a view pane that can be attached to any part of the workspace. A second plug-in that contributes to the user interface intercepts user input on model element names to suggest appropriate terms. Back end functionalities are implemented as Eclipse core runtime plug-ins.

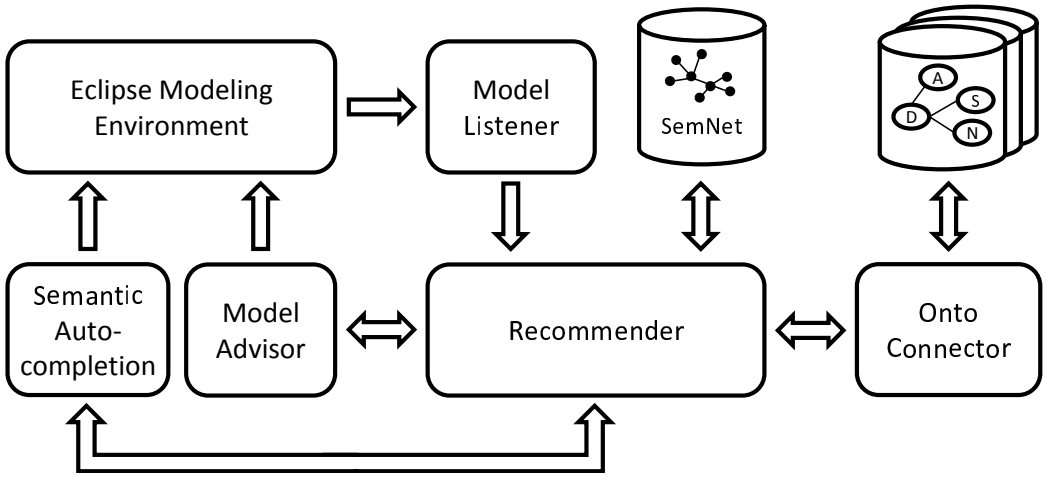


Figure 6.3: Architecture of the DoMoRe recommender system

Figure 6.3 depicts the architecture of the DoMoRe recommender system. The *Model Listener* observes changes in Ecore models that are developed with the Ecore diagram editor. Whenever a change in a model is made, the current content of the model is retrieved together with the newly added or changed model element and its relationships. The *Recommender* is notified and coordinates all subsequent steps of modeling suggestions. First, the domain model is transformed into a lexical-semantic representation using the domain-specific terms and semantic relationship mappings (cf., Section 3.5, Table 3.1). Based on this representation the *Semantic Network* is queried for related terms and directly delivers ranked lists of related terms to the recommender.

The *Ontology Connector* manages the set of connected knowledge bases and is queried as well. It incorporates the mediator and mapper (cf., Section 5.5, Figure 5.7) and operates the translation of the terminological queries to knowledge base specific queries and the integration of results. The recommender controls two components with which the user interacts. The *Model Advisor* is a view in the Eclipse environment that displays contextual information of the model elements. It shows possible generalizations, specializations, aggregations, associations, and related elements. The developer can use this view to easily add new content to a domain model by drag & drop of suggested elements to the diagram. The appropriate relationships will be created automatically. *Semantic Autocompletion* is triggered in case a new element in the model is named or the name of an existing element is changed. This feature behaves like a search engine. A context-sensitive pop-up list with names for the respective element is displayed, and the suggestions are filtered while typing.

6.5 Recommendation Generation

In this section we provide insights into the automated generation of model element recommendations by means of examples. We first show in Section 6.5.1 the noun term recommendation for class names that is applicable for our modeling support scenario 1 (except for associated class information), and scenarios 3 to 8 (cf., Section 3.4). These types of recommendations are mainly based on the binary relationships contained in SemNet and connected knowledge bases. The second type of recommendations is demonstrated in Section 6.5.2, the suggestion of verbs for association names, which is applicable for modeling support scenario 1 (only associated class information), scenario 2, and scenario 9. These types of recommendations are mainly based on the ternary relationships and binary noun-verb relationships contained in SemNet.

6.5.1 Class Name Recommendation

As an example for noun term recommendation we demonstrate the knowledge retrieval and recommendation generation for Scenario 3: A new, unconnected class is created, and names for that class are suggested. Figure 6.4a shows a small domain model example containing two classes connected with a named association. After creating the new class the model listener triggers the recommender, and the lexical representation of the domain model is generated (cf., Figure 6.4b). The information need depends on the model refinement step. In this case, noun terms are required that are semantically related to both *Hospital* and *Doctor* (cf., Figure 6.4c).

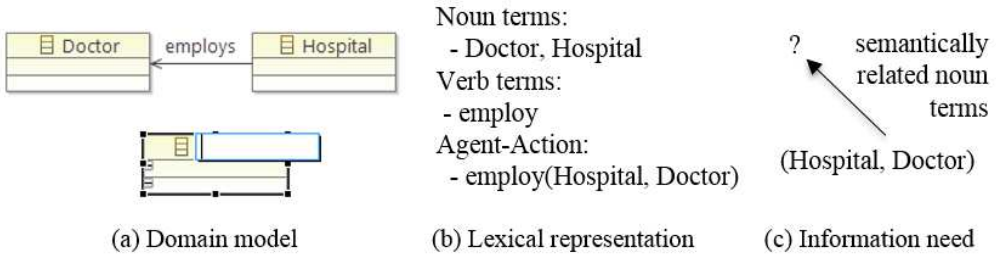


Figure 6.4: Lexical preparation in the procedure of the recommendation generation

In the following, the information need is broken down into separate lexical queries for each term and relationship type (see Figure 6.5d). The main reason for separately retrieving the information is that there is virtually no conceptual knowledge base that contains n-ary relationships. In contrast, our semantic network directly supports ternary relationships that allow more accurate results for term pairs, but for more than two terms, separate queries must be executed in each case. In the next step, the semantic network is queried for each term (cf., Figure 6.5e) and every connected knowledge base is queried for each term (cf., Figure 6.5f).

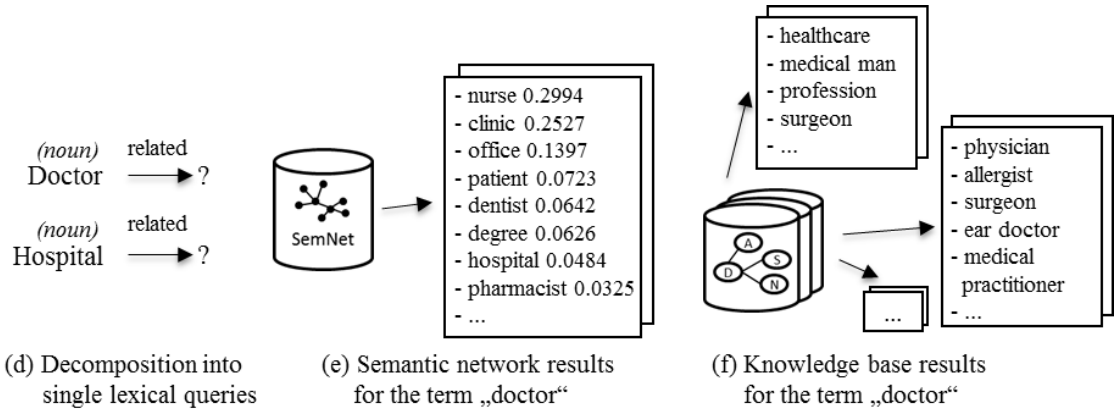


Figure 6.5: Retrieval in the procedure of the recommendation generation

Previously, separate lists of related terms were determined for each term of the original domain model, for each relationship type and for each knowledge source[9]. First, results from the knowledge bases are integrated based on the following principle. Per term the distinct union of all intermediate results (e.g., related terms of "doctor" from WordNet, BabelNet, ConceptNet, etc.) is created, and it is recorded how often each term had occurred. The resulting lists have a preliminary order, with terms with higher frequencies appearing first (cf., Figure 6.6g).

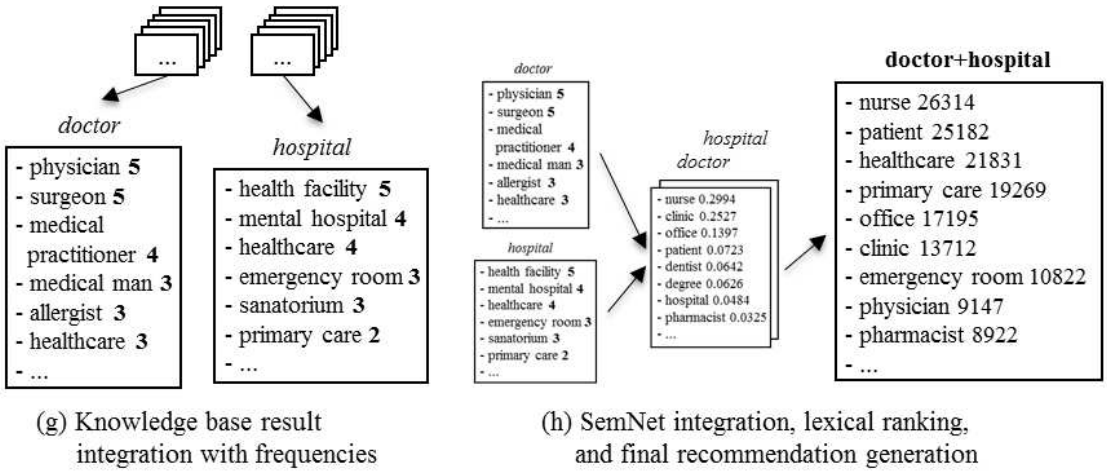


Figure 6.6: Integration and ranking in the procedure of the recommendation generation

In the final step, the presorted knowledge base results are integrated into the semantic network results. First, the knowledge base result and the respective semantic network result for each term are joined using the importance weight. It assures that terms found in many knowledge bases receive more prominent positions in the final ranking. Secondly, one final list of recommended terms is created. Separate results are intersected and relative frequencies of common terms are multiplied. The final list is divided into n segments: Terms that are related to n query terms appear first. After that, terms follow that are related to $n-1$ query terms, and so forth. Finally, a sorting by relevance is achieved by applying the pointwise mutal information score (cf., Figure 6.6h). This measurement is explained in Section 6.6.

[9]For example, if three classes exist in the model and broader and narrower relationships have to be retrieved from five knowledge sources, 30 intermediate result lists are generated.

Class Name Suggestions for Other Relationship Types. Besides the recommendation of related classes, as shown in the example, the recommender system suggests classes connected with specific relationship types (e.g., subclasses, aggregations) according to scenario 1 and scenarios 3 to 8. The procedure of recommendation generation is similar to the steps described above, but for this purpose primarily connected knowledge bases are queried and SemNet is mainly responsible for complementing and ranking the recommendations.

6.5.2 Association Name Recommendation

This section demonstrates the recommendation of verbs for association names, particularly the generation of suggestions for modeling support scenario 9: A new association is created between two classes. Figure 6.7a shows the respective example domain model, which contains two classes connected with a recently drawn association that is going to be named. The lexical representation of the model just contains two noun terms and an unnamed agent-action relationship (cf., Figure 6.7b). In this case, the information need is a verb representing the connection between the two nouns *Hospital* and *Patient* (cf., Figure 6.7c).

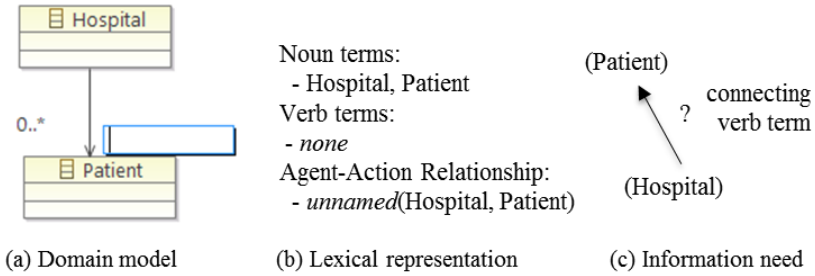


Figure 6.7: Lexical preparation for verb term recommendations

Decomposition of the information need is not required because a verb connecting two noun terms is a ternary relationship that can be directly retrieved from SemNet. The query that retrieves related verb terms for both *Hospital* and *Patient* (cf., Figure 6.8d) is executed and the lexical ranking with PMI is applied. The resulting list of verbs is directly used for autocompletion of the association name (cf., Figure 6.8e).

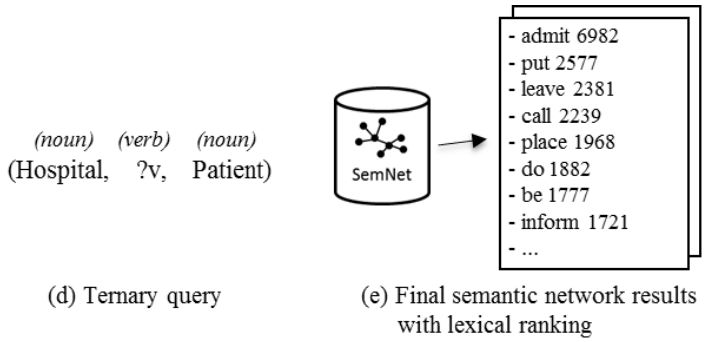


Figure 6.8: Retrieval and ranking for verb term recommendations

Further Uses of Verb Suggestions. Besides the recommendation of associations between two classes as shown above, SemNet provides data for modeling support scenario 1 (associated class information) and scenario 2 (context information for a selected association) using the binary and ternary noun-verb relationships. This allows to generate recommendations of associations plus the connected class by querying only one noun term.

6.6 Ranking Implementation

It is likely that queries to our semantic network and the connected knowledge bases deliver numerous related terms (up to a few thousand for each query). The ranking implemented in the recommender component is responsible for presenting the most relevant model elements first. Hence, if a list of related terms is retrieved, they will be ordered and the most important terms appear at the top. This is achieved by combining different relatedness measures.

From the construction of the semantic network we know *absolute frequencies* of co-occurring terms (cf., Section 4.3.8.2). For each term in the network we compute *relative frequencies* with respect to each connected related term. This normalization allows to compare the relatedness among different terms. Both measures allow a basic ranking of terms, but they have a shortcoming: Very general terms (e.g., time, man, year) that appear in almost all contexts are likely to be ranked on prominent positions.

To overcome this disadvantage we implement an information theoretic measurement: *Pointwise Mutual Information* (PMI) and its normalized form (cf., Equations 6.1). It measures the dependency between the probability of coinciding events and the probability of individual events (first introduced into lexicography by Church and Hanks [325]).

$$pmi(x,y) = \log\left[\frac{p(x,y)}{p(x)p(y)}\right] \quad npmi(x,y) = \frac{pmi(x,y)}{-\log\left[p(x,y)\right]} \tag{6.1}$$

Applying PMI to co-occurring terms means that x and y are terms, and PMI relates the probability of their coincidence $p(x,y)$ with the probabilities of observing both terms independently $p(x)p(y)$. PMI is an associativity score of two terms taking into account their individual corpus frequency, thus, very frequent and general terms receive lower scores. Unfortunately, this measurement also has a shortcoming: Although very general terms are ranked lower, very rare terms that co-occur only very few times with other terms tend to receive high scores.

Finally, in order to achieve a balanced ranking our recommender system uses the lexicographer's mutual information (LMI), which is the NPMI score multiplied with the absolute co-occurrence frequency [367].

6.7 Eclipse Plug-ins

Two extensions to the Eclipse Ecore Diagram Editor have been implemented that allow the user to interact with the recommender system. This section shows how context information is presented to the user and how the model element name suggestions work in the user interface.

6.7.1 Model Advisor Plug-in

The Model Advisor is a view plug-in that displays contextual information about the currently selected model element. When a model element is selected the recommender component queries the semantic network and knowledge bases for just one term, but with multiple relationship types. The information is aggregated and grouped into related elements, possible generalizations, specializations, aggregations, and associations (cf., Figure 6.9).

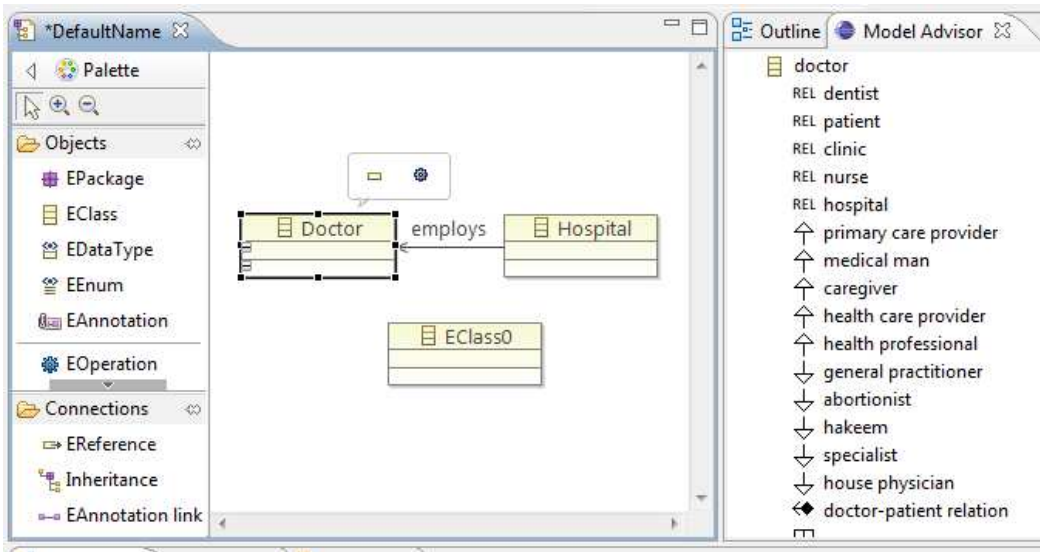


Figure 6.9: Model Advisor of the recommender system: suggestions for possible related classes, superclasses, subclasses, and aggregations

6.7.2 Semantic Autocompletion Plug-in

The Semantic Autocompletion plug-in directly extends the behavior of how the name of a class or association is typed. Whenever a name is edited, the user has the option to activate the feature with a Ctrl-Space keystroke as it is done for code completion when programming in the Eclipse environment. Generated lists of ranked terms are retrieved from SemNet depending on the current content of the model and a context-sensitive pop-up list of related terms is shown. It behaves like a search engine and is filtered while typing (cf., Figure 6.10).

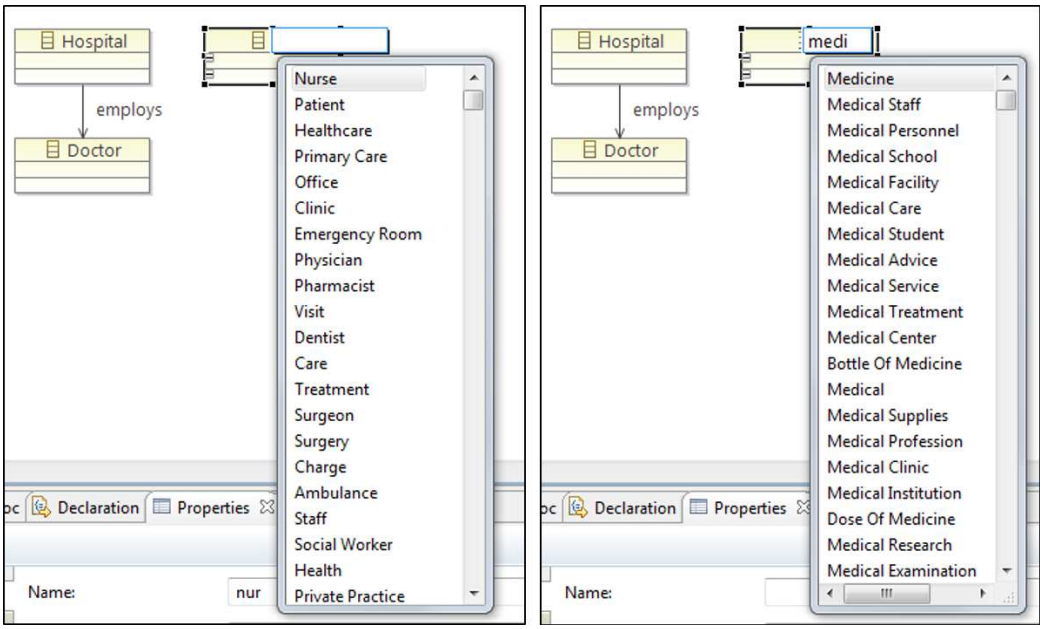


Figure 6.10: Semantic Autocompletion of the recommender system: Context-sensitive name predictions and infix search

6.8 Summary

In this chapter, we described how our semantic modeling support was implemented in a widely used modeling tool. The Eclipse Modeling Project and its EMF-based model editors have been enhanced with our Domain Modeling Recommender System (DoMoRe). It combines OntoConnector and SemNet under one roof to create context-aware modeling suggestions based on the connected knowledge bases and our large semantic network of terms. Using detailed examples, we showed how to automatically generate recommendations for class names and association names in domain models. Since the knowledge source queries typically return a large number of results, a ranking strategy based on pointwise mutual information has been implemented to present the most relevant model elements to the user, depending on the content of a model. Two features have been developed that allow the user to interact with the recommender system: Semantic Autocompletion enables search engine-like behavior when entering the names of model elements, and the Model Advisor displays contextual information grouped by modeling relationships.

Chapter 7

Practical Applications of Semantic Modeling Support

7.1 Introduction

This dissertation was conducted in the context of several research projects that were carried out in close cooperation with industry. This made it possible to continuously test and apply the developed methods and tools using real-world use cases from the projects. All projects had important domain modeling tasks. In this chapter we summarize the respective research projects that have taken place in different environments and domains, report on the results of the domain modeling and the experiences in the development and use of SemNet, OntoConnector and the DoMoRe recommendation system.

The ideas for this dissertation originally came from the BIZYCLE project [368] at TU Berlin, a research cooperation between academia and industry, to develop model-driven methods for software and data integration and their practical application in various industrial areas. The project demonstrated the potential of applying semantic technologies for model-based methods [369] and the need to further develop the link between information extraction and model-driven engineering. The research leading to this dissertation was carried out at two research institutions, the Database and Information Management Group (DIMA) of the TU Berlin, where the BIZWARE project (cf., Section 7.2) was conducted, and the Semantic Technologies Group at the Hasso Plattner Institute for IT Systems Engineering in Potsdam, where the dwerft project (cf., Section 7.3) and the AdA project (cf., Section 7.4) took place.

7.2 BIZWARE Research Project

Project Setting. BIZWARE [370] explored the potential of Domain-Specific Languages (DSLs) and Model-Driven Engineering for small and medium-sized enterprises (SMEs) in a variety of sectors, including healthcare, manufacturing / production, finance / insurance, publishing and facility management. It was a three-year research collaboration (2010-2013) of two academic partners with eight SMEs. The overall objective of the project was to develop a systematic and standardized process for building DSL-based software, including deployment, run-time and life cycle aspects, and operation of such domain software. Participatory modeling between software professionals and domain experts was to be enabled through dedicated (graphical and textual) languages in the specified domains. BIZWARE therefore focused on the development of domain-specific languages (DSLs) for the respective domains and a DSL framework including meta-DSL management, the so-called BIZWARE model and software factory.

Main Domain Modeling Results. While industry partners developed domain-specific languages in their respective domains, the academic partners developed methods, guidelines, and tools to support DSL development. Figure 7.1 gives an overview of the domain-specific languages and the model and software factory developed in the project. Originally, companies planned to use DSLs in customer projects. However, the project found that the use of DSLs to modernize their own software products and development infrastructures was more efficient. An important success story of the project was the development and application of a configuration DSL in a company offering software solutions for publishers. It allowed to generate large configuration artifacts in SQL language based on a textual DSL that provides a user-friendly syntax and editor for developers. This significantly reduced the manual customization of the software product for customers and made it easier to find inconsistencies. A second DSL, which has also become productive, was a combination of graphical workflow modeling in facility management and code generation for configuring software systems that have implemented these workflows.

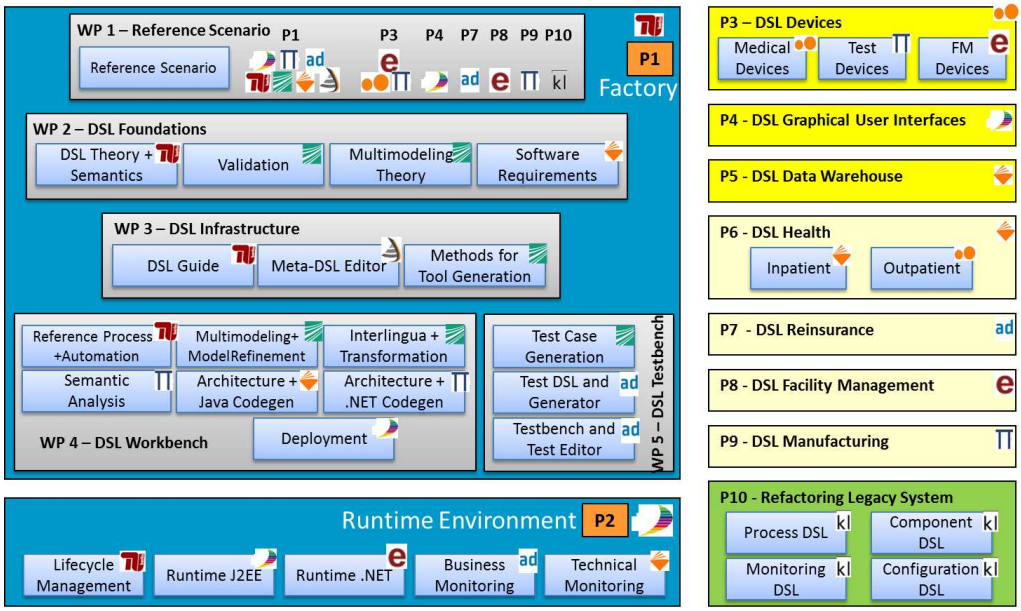


Figure 7.1: Overview of the BIZWARE model and software factory and implemented domain-specific languages

Progress of the Thesis Contributions. During the time at TU Berlin the concept of the dissertation and methodology of the Semantic Modeling Support was developed [371]. The first version of SemNet was created that was capable of relating noun terms consisting of up to three words (2.7 million terms and 37.5 million binary relationships) [372]. Based on SemNet a prototype of the semantic autocompletion feature was implemented and demonstrated in a healthcare use case with suggestions of single noun terms using probabilistic relationships [373]. Recommendations of model elements using specific relationships were based on a manual knowledge base integration [374].

Lessons Learned. Actors of the project were software developers from the respective companies on the one hand, who worked on the introduction of DSL-based workflows to improve their development tasks. On the other hand, modeling experts from academia participated in the project, who developed methods, guidelines, and tools to support domain-specific modeling. The modeling experts worked closely with the software engineers and accompanied the modeling process using the respective recommendation components. As the software engineers had little experience with

DSLs, the recommendations obtained from SemNet mainly supported the domain analysis phase to identify and agree on domain-specific terms that were later used in DSL's meta-models. In particular, the suggestions during the abstraction process helped to properly distinguish between the class and instance levels. The analysis of the modeling sessions and the respective suggestions showed that the ranking algorithms had to be improved (too general terms in the upper positions) and that the software engineers had missed suggestions for relationships (associations) between domain-specific terms. The extraction of verbal and ternary relationships was not available at that time and was implemented later based on the requirements.

7.3 dwerft Research Project

Project Setting. The dwerft project[1] was a three-year collaborative research effort (2014-2017) by three academic partners (one computer science group and two groups involved in media production) and eight companies involved in the production, distribution and archiving of film and television programs. Dwerft aimed to apply Linked Data principles for all metadata exchanges at every step of the media value chain. Starting with the initial idea of a script, all metadata is converted according to either existing or newly developed ontologies and reused in subsequent steps in the media value chain. Thus, metadata collected during media production becomes a valuable asset not only for each step from pre- and post-production, but also for distribution and archiving. The project successfully integrated a number of film production tools based on the Linked Production Data Cloud (LPDC), a technology platform for the film and television industry, to enable the interoperability of software in the production, distribution and archiving of audiovisual content.

Main Domain Modeling Results. The Linked Production Data Cloud stores and publishes semantic metadata under a unified ontology schema. One of the main tasks of the project was the development of this common data model, which conveys all the metadata originating from different sub-tasks of the film production process (e.g., screenplay, production planning, set information, post-production, distribution). The resulting **Film Ontology** [375] vocabulary was designed in collaboration with domain experts to create appropriate terminology that describes the various tasks of media production and all related metadata. The ontology schema is capable of representing film scripts (e.g., scenes, scene content, characters, sets, etc.), production planning metadata (e.g., film crews, departments, cast, filming locations, shooting schedule, equipment used, etc.), on-set information (such as shots, takes, and associated clips), post-production metadata (e.g., time codes, codecs, resolutions and formats of recorded and processed clips), rights management information, and quality assessment metadata of archived audiovisual material (e.g., surface damage, splices, bulges, glued areas, etc.). A visualization of the final project ontology is shown in Figure 7.2. It can be downloaded from `https://github.com/yovisto/dwerft/tree/master/tools/ontology`. An online version is available at `http://filmontology.org`.

Progress of the Thesis Contributions. During the time at HPI, SemNet was extended in two ways. On the one hand, the extraction process was refined to cover both binary and ternary noun-verb relationships. The context extension was implemented that allowed the recognition of terms and relationships an order of magnitude larger than in the first version (a total of 5.9 million terms and 355 million binary relationships). OntoConnector has been enhanced to include a template mechanism that enables the automated integration of knowledge bases with standardized vocabulary models and the low effort integration of proprietary models and access methods. Finally, the extended components were integrated into the DoMoRe recommendation system [320].

Lessons Learned. The contributing actors in the ontology design were domain experts of their respective roles (mostly non-technical) and modeling experts responsible for creating the domain

[1] `http://dwerft1.dwerft.de/`

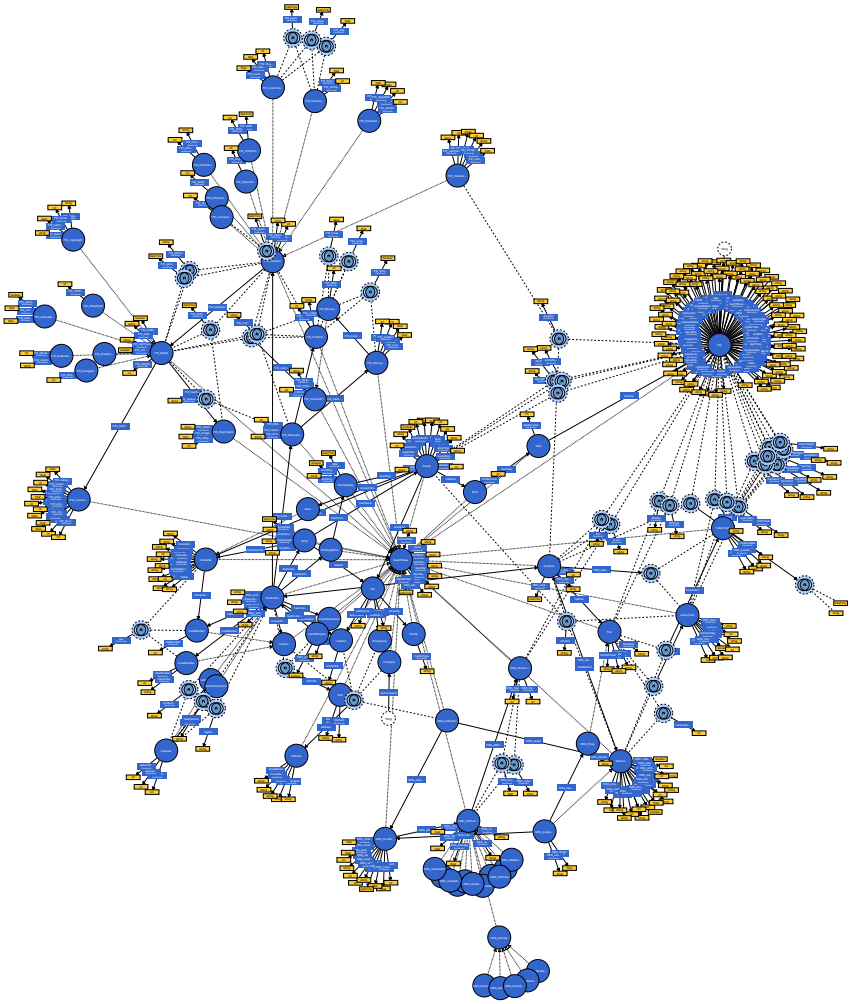


Figure 7.2: Visualization of the dwerft project ontology for film production tool integration

models and metadata schemata. The ontology was designed in two iterations. The first iteration was tool-oriented. Therefore, most domain knowledge was derived from data exported from production tools using proprietary XML files. Domain experts of the early process steps and the respective tool providers were consulted. The metadata integration, which was based on that first version, worked well for the first part of the process chain, but connecting the subsequent steps was difficult because mapping the data required more complex transformations and thus further knowledge engineering. At this point, the project delivered a realistic use case, with which the modeling suggestions of the DoMoRe recommendation system could be tested. For example, the recommendations revealed that props (elements used in a scene) such as costumes were required but could not be exported by the production planning tool. For this reason, an extension of the API and the export format of the manufacturer was requested.

Due to the required advanced knowledge engineering, domain experts from almost all phases of the media value chain actively participated in the second iteration of the ontology design. To accomplish this, many interviews with domain experts had to be conducted in order to gather domain-specific knowledge and discuss drafts of domain models. The extension of the ontology schema allowed to further test the SemNet extraction process and the modeling recommendations using terms from these additional areas of the media value chain. In a few experiments automated modeling suggestions were used to support post-meeting modeling and interview preparation with more extensive models that led to more efficient agreement on required metadata. Later in the

project, WebProtégé[2] was used to enable interactive, distributed ontology development with the help of employees from the partner companies that had more technical experience.

7.4 AdA Research Project

Project Setting. The interdisciplinary research group "Audio-Visual Rhetorics of Affect"[3] is an ongoing collaboration (2017-2020) in which film scholars collaborate with computer scientists to support empirical film studies using tool-based semantic video annotation and automated video analytics. Film scholars explore the hypothesis that TV news draw on audio-visual patterns of film productions to emotionally influence viewers by analyzing television reports, documentaries, and genre films of the topic "financial crisis." In a comprehensive corpus analysis, they identify and annotate low-to-high-level audio-visual patterns, such as shot duration, dominant colors, major-minor tonality and depicted visual concepts. By comparing different annotations from different scenes and genres, film scientists can analyze this opinion-forming level of reporting.

The aim of the project is to reduce the burden of elaborate, manual annotation routines in order to accelerate the film-scientific analysis of audio-visual motion patterns at the level of larger data sets. All annotation data and analysis results are published as Linked Open Data using the project's semantic vocabularies.

Main Domain Modeling Results. Film scientists carry out an in-depth corpus analysis by precisely describing feature films, documentaries, and television news using a film-analytical annotation method called eMAEX[4]. The description requires a lot of manual effort and results in hundreds of annotations per scene as ground truth data. One goal of the project is to publish this valuable data as Linked Open Data in order to make these annotations available to other film scholars as well as researchers from other fields.

Therefore, the AdA ontology [376], which is being developed, contains a vocabulary for fine-grained semantic video annotations using film-analytical concepts and terms. The vocabulary offers a number of categories under which a movie is analyzed (e.g., camera, image composition, acoustics). Each category includes the respective concepts used to annotate the segments of a movie (e.g., camera movement speed, field size). About 75% of the concepts have associated predefined values (e.g., long shot, medium shot, close-up and others for field size). Others are free text annotations, such as dialog transcriptions. Currently, the AdA ontology includes 9 categories (annotation level), 78 concepts (annotation types), and 436 predefined annotation values. Part of the vocabulary is visualized in Figure 7.3. It can be downloaded at github.com. An online version is available at `http://ada.filmontology.org/`.

Progress of the Thesis Contributions. The very specialized film-analytical vocabulary offered several use cases to test the DoMoRe recommendation system with rare domain-specific terms. Thus, the lexical ranking could be improved in the recommendation component [377].

Lessons Learned. The collaboration in domain modeling is similar to the dwerft project. The respective actors are domain experts (the film scholars) and modeling experts, who are responsible for creating the vocabulary and ontology. In contrast, film scholars were unable to directly participate in ontology design tasks using tools such as Protégé because they had no background in information modeling or Semantic Web technologies, although they are tech-savvy. The film-analytical method eMAEX was already very systematic in terms of the structuring of domain-specific knowledge. This allowed us to set up a set of spreadsheet-based forms in which film scholars were able to provide domain-specific concepts, terms, and descriptions using a familiar

[2] `https://webprotege.stanford.edu/`

[3] `http://www.ada.cinepoetics.fu-berlin.de/en/`

[4] eMAEX - Electronically-based Media Analysis of EXpressive movements — `https://empirische-medienaesthetik.fu-berlin.de/en/emaex-system/`

[6] `http://ada.filmontology.org/ontoviz/`

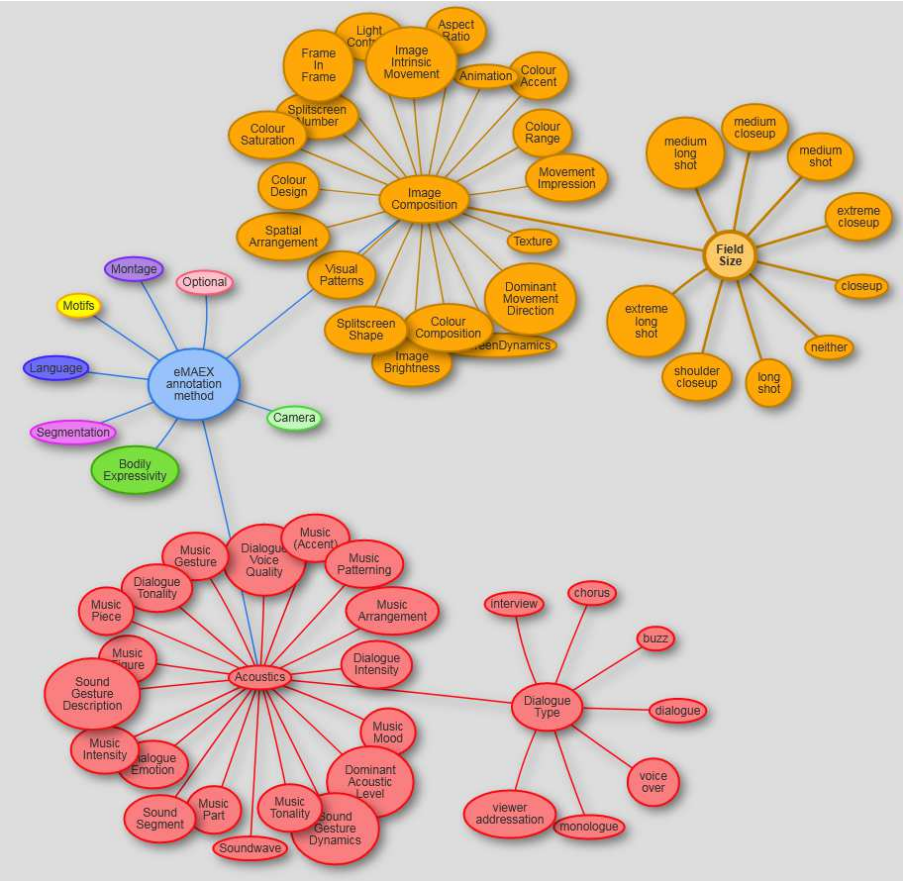


Figure 7.3: Visualization of parts of the AdA vocabulary for fine-grained semantic video annotation, created using our visualization application[6]

software environment. The RDF mapping language (RML) [378] and the RML tools were then used to automatically generate the project ontology from the semi-structured input data. As a result, an almost automated process of ontology generation has been implemented that allows for rapid response to concept and term changes. This was particularly helpful at the beginning of the project and allowed rapid prototyping of the ontology at the stage when the input data changed quite frequently.

The use of the recommendation system in this case was not so easy for the following reasons. First, the film scholars had fairly clear ideas about the naming of concepts and the use of their domain-specific terms. The agreement on the annotation types often led to longer internal discussions by the domain experts, so that the quick recommendation of appropriate sets of terms was not effective. Compared to the other projects, the structured domain knowledge was already well prepared, but not machine-readable. Consequently, it was not a problem to develop a semantic data model for the film analysis method to be used in a video annotation tool. Secondly, the field of film analysis is very specialized, so that many terms could not be found either in connected knowledge bases or in SemNet (e.g., dialog tonality, image-intrinsic movement). Other large corpora of this field were not available for analysis. Nonetheless, the vocabulary provided some interesting examples of annotation-concept-annotation-value relationships, most of which are specialization relationships (e.g., Recording Playback Speed - slow motion, timelapse, freeze). As the project progresses, we will look more closely at identifying these relationships in SemNet.

Chapter 8

Conclusions and Outlook

8.1 Key Research Results

In this thesis, novel methods and tools to support domain modeling through automated knowledge discovery and modeling recommendations were presented. Domain modeling is used to capture concepts and relationships of a particular application field in domain models. They are a key factor in improving communication and understanding in software development and a foundation for generating software. We motivated the importance of domain modeling and the use of model-driven engineering in industrial and academic software and data integration projects, as recent studies have confirmed an increasing and mature dissemination of these model-driven practices.

Domain modeling is a challenging task, because it requires multidisciplinary collaboration and information gathering from different groups of people, documents and other sources of knowledge. One challenge is that gathering the necessary information and implementing it in domain models is often very time-consuming. The methods developed here, which provide domain information directly during modeling, are designed to prevent the initial effort from being considered too great a hurdle, thereby omitting domain modeling. Two other challenges arise directly from the need to obtain domain information from a variety of sources. First, the heterogeneity of the information in terms of file formats, access methods, protocols, and schemata prevents consistent access to a large amount of structured domain knowledge. In addition, the proportion of structured data is very low compared to unstructured data such as textbooks, manuals or specifications. Relevant documents containing the required information must be located and manually inspected to extract and model the concepts and relationships of a domain. Second, although there are some useful knowledge bases for domain modeling, most of them were created manually and are not extensive enough. Approaches to automating the creation of knowledge bases from text are primarily focused on factual knowledge at the instance level that cannot be used for domain modeling at the conceptual level. The three challenges were addressed as follows.

The high cost of acquiring domain knowledge (Challenge 1) is approached by the context-aware provision of domain knowledge. On the one hand, we have developed the **Semantic Modeling Support**, a methodological foundation for providing modeling recommendations (Contribution 3). For this, an iterative procedure was defined in which a user makes a specific change to a model (refinement operation), domain information is queried based on the current state of the model (knowledge acquisition), and suggestions are made to the user (element recommendation). A number of modeling scenarios have been defined based on the types of operations a user can perform to change a model. The semantic relationships used in domain models have been mapped to lexical and knowledge modeling relationships as the basis for extracting information from text datasets and structured knowledge sources. On the other hand, these methods were implemented in a **Domain Modeling Recommender System** (Contribution 4) by enhancing the widely used Ecore Diagram Editor of the Eclipse Modeling Project. Recommendations for class names and association names in domain models are automatically generated using the knowledge resources

connected with the system. The recommendations are presented to the user with the Semantic Autocompletion feature, which provides search engine-like behavior when entering the names of model elements, and with the Model Advisor component, which displays contextual information for the domain model.

The heterogeneity of available knowledge bases (Challenge 2) is addressed by an extensible knowledge base query component. We have analyzed a number of existing sources of knowledge regarding contained lexical and conceptual knowledge and how this knowledge can be used for domain modeling. The knowledge bases were exemplarily integrated by developing a query procedure and the corresponding SPARQL queries for terms and their relationships and by implementing result integration and the corresponding mapping to modeling language constructs. These examples were the foundation for building our **Mediator-Based Querying Architecture with Generic Templates** (Contribution 2) for knowledge base integration. It provides unified access to information about terms and their relationships contained in knowledge bases at the schema level, in intermediate data models, and at the instance level. It also allows the near-automated integration of a wide range of knowledge bases that use a Lemon lexicon, OWL schema, or SKOS vocabulary, as well as the integration of other proprietary knowledge bases that provide SPARQL endpoints or JSON-LD APIs.

The unavailability of large conceptual knowledge resources (Challenge 3) is addressed through the automated extraction of semantically related terms from large-scale natural language datasets. A text analysis process has been developed that is capable of extracting terms and their relationships directly from N-gram corpus data. This type of extraction has the distinct advantage of eliminating the need to process the complete sentences of the original corpus, which is several orders of magnitude larger. Our approach relies on a set of syntactic part-of-speech patterns to recognize single-word terms and multiword expressions by applying hierarchical pattern matching to preprocessed 5-grams and generated 6-grams of the Google Books N-gram dataset. The 6-grams were obtained with a specially designed context extension algorithm using some effective heuristics to produce only useful 6-grams. This has increased the number of recognized terms by more than 60% and the number of recognized relationships by more than an order of magnitude. Extraction explicitly focused on conceptual terms, which is why proper nouns and named entities are ignored. As the extraction is based only on syntactic properties of sentences and statistical features of text corpora, it is completely domain-independent and can be easily transferred to other languages. Relationships between terms are determined using their co-occurrences and N-gram frequencies as input to methods of distribution semantics. As a result, we were able to create a **Semantic Network of Related Terms** (Contribution 1) consisting of nearly 6 million nouns, noun phrases and verbs, and over 355 million binary and ternary relationships that directly allows to answer top-N queries. SemNet is made publicly available with accompanying interfaces.

Table 8.1 summarizes the challenges of the thesis, how they have been approached, and what contributions the dissertation has achieved. The methods and tools developed in this work have been continuously applied and improved in various contexts. Three research projects have been conducted with industry and academia, where semantic modeling support and the recommendation system have helped to improve and make domain modeling tasks more efficient in several domains.

At the beginning of the PhD project the question was raised *"How to improve the development of domain models through automated knowledge acquisition?"*. It has been shown that a recommender system implementing context-sensitive modeling suggestions is a well-working tool to support domain modeling. It was further asked *"Where does the required knowledge come from?"*. This work provided an in-depth analysis of several structured knowledge sources and a large unstructured text dataset to enable the acquisition of lexical knowledge for model element recommendations. It has been questioned *"How can the necessary knowledge be acquired automatically?"*. The thesis developed appropriate extraction methods and components that enable transparent on-demand access to terms and their relationships in knowledge bases and text document collections. It was also asked *"How can the acquired knowledge be used to improve modeling?"*. The different representations of domain models, knowledge bases and textual content have been connected using semantic relationship mappings that enable the automated transformation of the acquired knowledge into modeling recommendations. Finally, it was asked *"How*

Challenge	Objective	Approach	Contribution
Cost of domain knowledge acquisition	Develop methodological foundations	Iterative procedure with modeling support scenarios	Semantic modeling support (Chapter 3)
	Automatically provide domain information	Context-sensitive suggestions of model elements	DoMoRe recommendation system (Chapter 6)
Heterogeneity of knowledge bases	Transparent access to lexical knowledge	Mediator-based knowledge base querying	OntoConnector with knowledge base templates (Chapter 5)
Lack of conceptual knowledge resources	Automated construction of semantic networks of related terms	N-Gram based text analysis and distributional semantics	SemNet knowledge base with detailed extraction process (Chapter 4)

Table 8.1: Summary of the contributions of this thesis

does model development affect the acquisition of knowledge?". The iterative process of model refinement, knowledge acquisition, and element recommendation allows to query and provide the necessary domain knowledge at any given time for each development state of the domain model, keeping the human in the loop.

The results of my dissertation have been published in the following core research and application-oriented papers, which are briefly described in the following paragraphs.

Core Research Papers. *"Supporting Software Language Engineering by Automated Domain Knowledge Acquisition"* by H. Agt [371] is the first paper in the course of the thesis published and presented at the **Doctoral Symposium of the MODELS 2011 conference**, the premier international conference series on model-driven engineering languages and systems. The paper describes the PhD proposal, concept of the dissertation, reviews related work, and presents first results of the proposed solution. The paper was assigned a mentor, Bran Selic, a recognized pioneer in the application of model-driven engineering in industrial settings. He was responsible for reviewing and discussing the paper during the symposium[1]. The paper was selected as **best paper** [379] together with a second paper for publication (acceptance rate 20%).

"Guidance for Domain Specific Modeling in Small and Medium Enterprises" by H. Agt, R.-D. Kutsche, T. Wegeler [374] is the second paper which was accepted as full paper at the **Workshop on Domain-Specific Modeling at the OOPSLA 2011 conference**. The DSM workshop series has a long history and is a premier forum for a domain-specific language and modeling solutions. The paper describes the concept and first results of modeling guidance for domain-specific language development. It includes the description of the implementation of modeling suggestions based on querying multiple knowledge bases (part by Henning Agt). The paper was co-authored with Timo Wegeler (a project colleague from Fraunhofer – responsible for the framework part).

"SemAcom: A System for Modeling with Semantic Autocompletion" by H. Agt [373] is a demonstration accepted at the **MODELS 2012 conference**. The demo paper describes the semantic autocompletion function for domain modeling based on the semantic network of terms. The

[1] `http://ecs.victoria.ac.nz/Events/MODELS2011/Symposia`

demonstration at the conference included a booth with a poster throughout the conference and a short presentation of the paper.

"Automated Construction of a Large Semantic Network of Related Terms for Domain-Specific Modeling" by H. Agt, R.-D. Kutsche [372] is the fourth paper that was accepted as full paper at the **main research track of the CAiSE 2013 conference**. CAiSE is a CORE A ranked conference and the premier conference series for information systems engineering. The paper describes the detailed construction of SemNet, the extraction of semantically related terms from text datasets, based on natural language processing and statistical semantics, and its application in a domain modeling context (acceptance rate 16.6%).

"DoMoRe – A Recommender System for Domain Modeling" by H. Agt-Rickauer, R.-D. Kutsche, H. Sack [320] is the fifth full paper in the context of the dissertation. It was published and presented at the **MODELSWARD 2018 conference** (acceptance rate 25.1%). The conference is the leading European conference series on Model-Driven Engineering. The paper describes the recommender system as a whole with a focus on support scenarios, extensions of SemNet, recommendation generation and practical application of the system. The paper was **selected for extended publication** by the program committee.

"Automated Recommendation of Related Model Elements for Domain Models" by H. Agt-Rickauer, R.-D. Kutsche, H. Sack [377] is the sixth full paper in the course of the thesis and an extended version of [320]. The article was published in a **special issue of the Springer CCIS series 2019** and provides more detailed descriptions and examples of the semantic network, additional details on the recommendation generation and on the implementation of the recommender system, as well as additional related work.

Application-Oriented Papers. *"SemAcom: A System for Modeling with Semantic Autocompletion"* by H. Agt [373] was not only a demo paper at the **MODELS 2012 conference** but also describes the first application of SemNet in the medical domain in the context of the BIZWARE research project.

"Quantitative Analysis of Art Market Using Ontologies, Named Entity Recognition and Machine Learning: A Case Study." by D. Filipiak, H. Agt-Rickauer, C. Hentschel, A. Filipowska, H. Sack [380] was a full paper at the **BIS 2016 conference**. The paper was developed in cooperation with Poznan University and describes a data-oriented approach to art market analysis. I was responsible for the knowledge base, ontology and data collection part. The recommender system was applied during the development of the art market data model. The paper received the **best paper award**.

"Data Integration for the Media Value Chain" by H. Agt-Rickauer, J. Waitelonis, T. Tietz, H. Sack [375] was a poster paper at the **ISWC 2016 conference**. The paper describes the successful integration of a set of film production tools based on the Linked Production Data Cloud, a technology platform for the film and TV industry to enable software interoperability used in production, distribution, and archiving of audiovisual content. The recommender system was used during the development of the project ontology that is used for data integration.

"Semantic Annotation and Automated Extraction of Audio-Visual Staging Patterns in Large-Scale Empirical Film Studies" by H. Agt-Rickauer, C. Hentschel, H. Sack [376] was a poster paper at the **SEMANTiCS 2018 conference**. The paper describes the development of a vocabulary for fine-grained semantic video annotation and its application in tool-supported empirical film studies. The recommender system was used during the development of the project vocabulary.

8.2 Future Work

This dissertation contributes to the fields of domain modeling, knowledge acquisition, and information integration. In the course of the work, the main emphasis had to be placed on certain languages, methods, and resources for developing a fully functional recommendation system. For example, UML class diagrams, count-based distributional models, and the English language were selected. During the research it became evident that certain limitations could not be addressed in this thesis. Hence, we suggest possible directions of how future research could extend the semantic modeling support to a wider scale, of how further challenges of knowledge extraction could be tackled, thus addressing the discovered limitations.

Support for More Languages The adaptation of the semantic modeling support and the recommender system to more languages is twofold: (1) On the one hand, we refer to more *modeling languages* (e.g., ER Diagrams, RDFS/OWL schema) and respective modeling tools that can be supported with the methods and resources developed in this thesis. The required steps to achieve modeling support for these languages and tools are: (a) The extension and customization of the respective modeling tool to integrate the Model Advisor and Semantic Autocompletion feature. (b) The development of additional semantic relationship mappings for the respective modeling language to be able to transform domain knowledge into appropriate modeling recommendations. A good candidate for demonstrating the recommender system in the area of ontology design is the recently published new version of WebProtégé, a cloud-based ontology editor [381].

(2) On the other hand, support for more *natural languages* is required to suggest domain-specific terms and relationships for models formulated in other languages than English. That would have been very helpful, for example, in the dwerft project, where many discussions with domain experts were conducted in German. The extraction methods developed in this thesis rely for the most part on statistical features of large text corpora that can be applied across languages. Only the used POS tagging is language-dependent because a fine-grained lexical categorization is required for conceptual term detection. Consequently, the construction of semantic networks can be transferred to many other segmented Indo-European languages (e.g., German, French, Spanish). The following steps are necessary for the implementation: (a) The use of corpora of the respective language and their derived N-gram datasets. For example, the Google Books N-gram dataset provides downloads[2] for French, German, and Italian. However, the required N-gram counts for the extraction procedure can be obtained from any other corpus. (b) The use of language-specific part-of-speech taggers and respective tagsets and the adaptation of the syntactic extraction patterns to the respective language-specific tagset.

Creating semantic networks for each single language is one option, as it was recently done, for example, in the field of word embeddings [382]. In contrast, several approaches have been proposed in the last few years to create *bilingual* and *multilingual* word embeddings, which combine several languages in the same vector space, based either on multilingual encyclopedias and knowledge bases or based on parallel and comparable corpora (for an overview, see the survey by Ruder et al. [383]). Building a multilingual SemNet requires generating parallel N-gram data, which is challenging due to the different word orders and use of compounds in each language. To some extent this has been addressed by N-Gram based machine translation [384], but is still subject to research.

Conceptual Knowledge Extraction SemNet has taken a significant step in extracting conceptual knowledge. However, there is still room for improvement and development of new methods, as shown by recent activities in the research community. A major disadvantage of vector space models is that they establish relationships using distances or information-theoretical measurements (continuous values), but cannot extract specific types of relationships. Therefore, we have combined SemNet with lexical information from knowledge bases on the application side. Extracting specific conceptual relationships is important in building large conceptual and common sense knowledge

[2] `http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`

bases. An example is the field of taxonomy extraction (e.g., as defined in SemEval-2016 Task 13 [385]). This is a more complex task than detecting hypernym relationships as suggested by Hearst more than 25 years ago [120]. In this particular evaluation, participants had great difficulty delivering extracted taxonomies that were qualitatively superior to a simple string subsequence baseline. For this reason, the difficulty of the task has been reduced in the following years [123]. Recently, taxonomy extraction was tackled through the combination of supervised learning and word embeddings [386]. We also suggest that such a combination of methods should be pursued not only for taxonomic relationships but also for other conceptual relationships, in order to advance the construction of large common sense knowledge bases [387, 388]. This is also confirmed by the fact that in the field of distributed semantic representations work is being done to integrate word meanings into the corresponding models [389, 390].

Further Possibilities for Improvement In addition to the future work on additional language support and advanced conceptual knowledge acquisition, this paragraph discusses at which points of the work limitations exist and how they can be tackled.

In Chapter 3 the iterative procedure of Semantic Modeling Support has been presented. The approach assumes that all sources of knowledge can be queried directly, which has led to the decision that SemNet is generated in an offline extraction process. On the one hand, the effort of connecting new knowledge bases to OntoConnector is minimal, and the extraction of semantic networks from completely new text records is fully automatic. On the other hand, such analyses take too long to be used for ad hoc knowledge acquisition because certain characteristics of the entire corpus must be determined when using distributional approaches. This research challenge also exists for machine learning approaches that require a computationally intensive training phase. Ad hoc extraction is achieved to some extent by declarative extraction systems and stream processing, such as those implemented in recent versions of SystemT [391] and Apache Flink [392]. To enable ad hoc extraction, the architecture of the DoMoRe system would have to be fundamentally changed. One option is to develop a component for in-depth analysis (in the sense of grammatical analysis) of short and medium length technical documents that are often part of software projects.

In Chapter 4 the extraction methods for building SemNet have been presented. A design choice was to optimize the process for single-machine in-memory processing to efficiently perform some of the frequency aggregations. This limits the approach to the size of the main memory (64 GB was sufficient for all steps in this thesis). Alternatively, the extraction can be performed distributed using, for example, Apache OpenNLP and / or NLTK with Apache Spark. For most operations, the overhead of a distributed framework would not have been justified except for the required POS tagging, which took the longest compared to the rest of the extraction. However, there is no alternative to distributed processing for larger N-gram datasets than the Google Books N-gram dataset, which can contain longer, and therefore much more, N-grams.

Identification of conceptual terms has been implemented similar to the robust extraction of technical terms proposed in the literature, using a small set of syntactic patterns that have been verified using a number of existing knowledge base schemas. However, this limits the flexibility to adapt to differently structured N-Gram datasets (e.g., in the case of not tokenized hyphenated words) because manual adjustments are necessary. Alternatively, pattern learning may be implemented with a distant supervision approach using seeds of concept terms from ontology schemas. This would probably have allowed patterns to be used that were discarded because of too many false positives (e.g., terms containing a gerund), but such an approach is still difficult to implement if a context of only five or six words is analyzed.

The evaluation of SemNet was carried out by quantitatively comparing with existing knowledge databases and by measuring term and relationship coverage. Two complementary qualitative approaches should be addressed in future work: (1) A ranking-based evaluation that measures the performance of top-N queries to SemNet. The steps required for the implementation are: (a) The compilation of a set of query terms and the generation of a dataset of corresponding ranked lists of related terms for each knowledge source. (b) A user study that provides relevance judgments by directly comparing rankings presented in random order. (c) The assessment of the user ratings.

(2) A controlled experiment to measure the overall efficiency of the recommendation system. The steps required for implementation are: (a) The participants are introduced to the modeling tool and asked to perform multiple domain modeling tasks. (b) Subjects are randomly subdivided into a treatment group using the tool with recommendations of related model elements and a control group modeling without recommendations. (c) Results from both groups are then compared to predefined solutions from domain experts to measured the outcome variables *time on task* and *model completeness*.

In Chapter 5, mediator-based knowledge base querying was developed as part of the OntoConnector component. The approach requires a permanent Internet connection and is to some extent prone to error if connected SPARQL endpoints cannot be reached. In contrast, SemNet is always installed locally with the modeling tool. One possible solution to the unavailability of knowledge bases is the implementation of a caching mechanism and the prefetching of data if the domain of interest can be determined by some example terms. Using data dumps is also possible, but a more balanced way is to use Linked Data Fragments [393]. OntoConnector does not yet have a user-friendly interface to help complete the provided SPARQL templates. An important feature to implement is the automatic discovery of suitable relationship names in newly connected knowledge bases for mapping to our lexical queries.

Chapter 6 presented the implemented recommendation system DoMoRe. Recommendations are mainly provided using element names in either the Model Advisor or the Semantic Autocompletion feature. An important research task is how visual representations of the collected knowledge (e.g., in the form of diagram fragments) can be generated. Integrating the recommendation functionality into the Ecore Diagram Editor has been difficult as the tool is partially made up of generated code and does not provide the necessary interfaces. This resulted in a very tightly coupled implementation that prevented the use of automatic updates of the editor.

Finally, semantic modeling support cannot only be used to suggest what to include in a model and to recommend the most relevant names of model elements. Knowledge bases and semantic networks can also be used to implement an evidence-based verification of conceptual models to automatically identify semantically incorrect modeling constructs. This includes, for example, the detection of misused relationship types (e.g., "doctor" and "surgeon" connected with an aggregation relationship), the localization of directed relationships pointing in the wrong direction (e.g., "doctor" *subclassOf* "surgeon"), and finding concepts that have too much semantic distance to the rest of the model (e.g., "doctor", "surgeon", and "plywood").

Bibliography

[1] Dijkstra, E.W.: The humble programmer. Commun. ACM **15**(10) (1972) 859–866

[2] Bauer, F.L.: Software engineering — wie es begann. Historische Notizen zur Informatik (2009) 72–75

[3] Shaw, M.: Abstraction techniques in modern programming languages. IEEE software (4) (1984) 10–26

[4] Selic, B.: Personal reflections on automation, programming culture, and model-based software engineering. Automated Software Engineering **15**(3-4) (2008) 379–391

[5] Cuadrado, J.S., Izquierdo, J.L.C., Molina, J.G.: Applying model-driven engineering in small software enterprises. Science of Computer Programming **89** (2014) 176–198

[6] Bruel, J.M., Combemale, B., Ober, I., Raynal, H.: Mde in practice for computational science. Procedia Computer Science **51** (2015) 660–669

[7] Booch, G., Rumbaugh, J., Jacobson, I.: Unified modeling language (uml). World Wide Web: http://www. rational. com/uml/(UML Resource Center) **94** (1998)

[8] Petre, M.: Uml in practice. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press (2013) 722–731

[9] Kelly, S., Lyytinen, K., Rossi, M.: Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In: Advanced Information Systems Engineering, Springer (1996) 1–21

[10] Whittle, J., Hutchinson, J., Rouncefield, M.: The state of practice in model-driven engineering. Software, IEEE **31**(3) (2014) 79–85

[11] Whittle, J., Hutchinson, J.: Mismatches between industry practice and teaching of model-driven software development. In: Models in Software Engineering. Springer (2011) 40–47

[12] Mussbacher, G., Amyot, D., Breu, R., Bruel, J.M., Cheng, B.H., Collet, P., Combemale, B., France, R.B., Heldal, R., Hill, J., et al.: The relevance of model-driven engineering thirty years from now. In: Model-Driven Engineering Languages and Systems. Springer (2014) 183–200

[13] Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press (March 2008)

[14] Fowler, M.: Domain Specific Languages. Addison-Wesley, Boston (2010)

[15] Hermans, F., Pinzger, M., Van Deursen, A.: Domain-specific languages in practice: A user study on the success factors. Springer (2009)

[16] Selic, B.: The pragmatics of model-driven development. IEEE software **20**(5) (2003) 19

[17] Andersson, H., Herzog, E., Johansson, G., Johansson, O.: Experience from introducing unified modeling language/systems modeling language at saab aerosystems. Systems Engineering **13**(4) (2010) 369–380

[18] Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., Heldal, R.: Industrial adoption of model-driven engineering: Are the tools really the problem? In: Model-Driven Engineering Languages and Systems. Springer (2013) 1–17

[19] Reggio, G., Leotta, M., Ricca, F., Clerissi, D.: What are the used uml diagrams? a preliminary survey. In: EESSMOD@ MoDELS. (2013) 3–12

[20] Burden, H., Heldal, R., Whittle, J.: Comparing and contrasting model-driven engineering at three large companies. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM (2014) 14

[21] Hebig, R., Bendraou, R., Völter, M., Chaudron, M.R.: Model-driven development processes and practices: Foundations and research perspectives. In: MD2P2@ MoDELS. (2014) 2–6

[22] Hutchinson, J., Whittle, J., Rouncefield, M.: Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Science of Computer Programming **89** (2014) 144–161

[23] Petre, M.: "no shit" or "oh, shit!": responses to observations on the use of uml in professional practice. Software & Systems Modeling **13**(4) (2014) 1225–1235

[24] Gorschek, T., Tempero, E., Angelis, L.: On the use of software design models in software development practice: An empirical investigation. Journal of Systems and Software **95** (2014) 176–193

[25] Reggio, G., Leotta, M., Ricca, F.: Who knows/uses what of the uml: a personal opinion survey. In: Model-Driven Engineering Languages and Systems. Springer (2014) 149–165

[26] Whittle, J., Rouncefield, H.B., Heldal, R., Kristoffersen, S.: How industry uses mde. (2015)

[27] Vetro, A., Bohm, W., Torchiano, M.: On the benefits and barriers when adopting software modelling and model driven techniques - an external, differentiated replication. In: Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on, IEEE (2015) 1–4

[28] Sobernig, S., Hoisl, B., Strembeck, M.: Extracting reusable design decisions for uml-based domain-specific languages: A multi-method study. Journal of Systems and Software **113** (2016) 140–172

[29] Chaudron, M.R.: Empirical studies into uml in practice: pitfalls and prospects. In: 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering (MiSE), IEEE (2017) 3–4

[30] Budgen, D., Burn, A.J., Brereton, O.P., Kitchenham, B.A., Pretorius, R.: Empirical evidence about the uml: a systematic literature review. Software: Practice and Experience **41**(4) (2011) 363–392

[31] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G.: Preliminary findings from a survey on the md state of the practice. In: Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, IEEE (2011) 372–375

[32] Broy, M.: Domain modeling and domain engineering: Key tasks in requirements engineering. In: Perspectives on the Future of Software Engineering. Springer (2013) 15–30

[33] Prieto-Díaz, R.: Domain analysis: An introduction. ACM SIGSOFT Software Engineering Notes **15**(2) (1990) 47–54

[34] Iscoe, N., Williams, G.B., Arango, G.: Domain modeling for software engineering. In: Software Engineering, 1991. Proceedings., 13th International Conference on, IEEE (1991) 340–343

[35] Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software & Systems Modeling **7**(3) (2008) 345–359

[36] Bera, P., Evermann, J.: Guidelines for using uml association classes and their effect on domain understanding in requirements engineering. Requirements Engineering **19**(1) (2014) 63–80

[37] Evermann, J., Wand, Y.: Ontology based object-oriented domain modeling: representing behavior. Theoretical and Practical Advances in Information Systems Development: Emerging Trends and Approaches: Emerging Trends and Approaches (2011) 37

[38] Wand, Y., Weber, R.: Research commentary: information systems and conceptual modeling — a research agenda. Information Systems Research **13**(4) (2002) 363–376

[39] Evans: Domain-Driven Design: Tacking Complexity In the Heart of Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)

[40] Mylopoulos, J.: Conceptual modelling and telos 1 (2008)

[41] Larman, C.: Applying UML and patterns: an introduction to object oriented analysis and design and interactive development. Pearson Education India (2012)

[42] Lukyanenko, R., Parsons, J.: Is traditional conceptual modeling becoming obsolete? In: International Conference on Conceptual Modeling, Springer (2013) 61–73

[43] Hoisl, B., Sobernig, S., Strembeck, M.: A catalog of reusable design decisions for developing uml/mof-based domain-specific modeling languages. (2015)

[44] Landre, E., Wesenberg, H., Olmheim, J.: Agile enterprise software development using domain-driven design and test first. In: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, ACM (2007) 983–993

[45] Nilsson, J.: Applying domain-driven design and patterns: with examples in C# and. NET. Pearson Education (2006)

[46] Vernon, V.: Implementing domain-driven design. Addison-Wesley (2013)

[47] Stahl, T., Völter, M., Bettin, J., Haase, A., Helsen, S.: Model-driven software development - technology, engineering, management. Pitman (2006)

[48] Gamma, E., Johnson, R., Helm, R., Vlissides, J.: Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. Pearson Deutschland GmbH (2011)

[49] Kosar, T., Bohra, S., Mernik, M.: Domain-specific languages: A systematic mapping study. Information and Software Technology **71** (2016) 77–91

[50] Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: A survey. Software Engineering, IEEE Transactions on **39**(6) (2013) 869–891

[51] Braun, R.: Towards the state of the art of extending enterprise modeling languages. In: Model-Driven Engineering and Software Development (MODELSWARD), 2015 3rd International Conference on, IEEE (2015) 1–9

[52] Tairas, R., Cabot, J.: Corpus-based analysis of domain-specific languages. Software & Systems Modeling **14**(2) (2015) 889–904

[53] Erickson, J., Siau, K.: Can uml be simplified? practitioner use of uml in separate domains. In: proceedings EMMSAD. Volume 7. (2007) 87–96

[54] Reggio, G., Leotta, M., Ricca, F., Clerissi, D.: What are the used activity diagram constructs? a survey. In: Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on, IEEE (2014) 87–98

[55] Jacobson, I.: Taking the temperature of uml. WEB site blog. ivarjacobson. com/taking-the-temperature-of-uml (2009)

[56] Linthicum, D.S.: Enterprise application integration. Addison-Wesley Professional (2000)

[57] Lheureux, B., Pezzini, M., Thompson, J., Altman, R., Sholler, D., Schulte, W., Malinverno, P., Knipp, E.: Predicts 2013: Application integration. Gartner Report (2012)

[58] Hohpe, G., Woolf, B.: Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional (2004)

[59] Ziegler, P., Dittrich, K.R.: Three decades of data integration—all problems solved? In: Building the Information Society. Springer (2004) 3–12

[60] Seacord, R.C., Plakosh, D., Lewis, G.A.: Modernizing legacy systems: software technologies, engineering processes, and business practices. Addison-Wesley Professional (2003)

[61] Force, A.T.: Architecture-driven modernization scenarios. OMG, USA (2006)

[62] Erlikh, L.: Leveraging legacy system dollars for e-business. IT professional **2**(3) (2000) 17–23

[63] Erl, T.: Service-oriented architecture: concepts, technology, and design. Pearson Education India (2005)

[64] Jamshidi, P., Ahmad, A., Pahl, C.: Cloud migration research: a systematic review. IEEE Transactions on Cloud Computing **1**(2) (2013) 142–157

[65] Bergmayr, A., Bruneliere, H., Izquierdo, J.L.C., Gorronogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarria, L., Pezuela, C., et al.: Migrating legacy software to the cloud with artist. In: Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on, IEEE (2013) 465–468

[66] Rugaber, S., Stirewalt, K.: Model-driven reverse engineering. IEEE software **21**(4) (2004) 45–53

[67] Bruneliere, H., Cabot, J., Dupé, G., Madiot, F.: Modisco: A model driven reverse engineering framework. Information and Software Technology **56**(8) (2014) 1012–1032

[68] Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. Sigplan Notices **35**(6) (2000) 26–36

[69] Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37** (December 2005) 316–344

[70] Van Deursen, A., Klint, P.: Domain-specific language design requires feature descriptions. CIT. Journal of computing and information technology **10**(1) (2002) 1–17

[71] Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design guidelines for domain specific languages. arXiv preprint arXiv:1409.2378 (2014)

[72] Kantor, P.B., Rokach, L., Ricci, F., Shapira, B.: Recommender systems handbook. Springer (2011)

[73] Burke, R.: Hybrid recommender systems: Survey and experiments. User modeling and user-adapted interaction **12**(4) (2002) 331–370

[74] He, C., Parra, D., Verbert, K.: Interactive recommender systems: a survey of the state of the art and future research challenges and opportunities. Expert Systems with Applications **56** (2016) 9–27

[75] Robillard, M.P., Maalej, W., Walker, R.J., Zimmermann, T.: Recommendation systems in software engineering. Springer Science & Business (2014)

[76] Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing knowledge about information systems. ACM Transactions on Information Systems (TOIS) **8**(4) (1990) 325–362

[77] Störrle, H.: Structuring very large domain models: experiences from industrial mdsd projects. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, ACM (2010) 49–54

[78] Prakash, N., Rolland, C., Pernici, B.: Use of domain knowledge for requirements validation. In: Information System Development Process: Proceedings of the IFIP WG8. 1 Working Conference on Information System Development Process, Como, Italy, 1-3 September, 1993. Volume 30., Elsevier (1993) 99

[79] Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling **13**(3) (2014) 941–962

[80] Dong, X.L., Srivastava, D.: Big data integration. In: Data Engineering (ICDE), 2013 IEEE 29th International Conference on, IEEE (2013) 1245–1248

[81] Embley, D.W., Liddle, S.W.: Big data—conceptual modeling to the rescue. In: International Conference on Conceptual Modeling, Springer (2013) 1–8

[82] Gruber, T.R.: The acquisition of strategic knowledge. Elsevier (1988)

[83] Sutcliffe, A., Sutcliffe, A.: The Domain Theory: Patterns for knowledge and software reuse. CRC Press (2002)

[84] Kidd, A.: Knowledge acquisition for expert systems: A practical handbook. Springer Science & Business Media (1987)

[85] Reinhartz-Berger, I.: Towards automatization of domain modeling. Data & Knowledge Engineering **69**(5) (2010) 491–515

[86] Tacla, C.A., Freddo, A.R., Paraiso, E.C., Ramos, M.P., Sato, G.Y.: Supporting small teams in cooperatively building application domain models. Expert Systems with Applications **38**(2) (2011) 1160–1170

[87] Reinhartz-Berger, I., Cohen, S., Bettin, J., Clark, T., Sturm, A.: Domain engineering. Springer (2013)

[88] Buitelaar, P., Cimiano, P., Magnini, B.: Ontology Learning from Text: Methods, Evaluation and Applications. Volume 123 of Frontiers in Artificial Intelligence and Applications Series. IOS Press, Amsterdam (7 2005)

[89] Davidov, D.: Classification of semantic relationships between nominals using pattern clusters. In: Proceedings of ACL 2008. (2008) 227–235

[90] Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M.A., Nordmoen, B., Fritzsche, M.: Where does model-driven engineering help? experiences from three industrial cases. Software & Systems Modeling **12**(3) (2013) 619–639

[91] Modoni, G.E., Caldarola, E.G., Terkaj, W., Sacco, M.: The knowledge reuse in an industrial scenario: A case study (2015)

[92] Omoronyia, I., Sindre, G., Stålhane, T., Biffl, S., Moser, T., Sunindyo, W.: A domain ontology building process for guiding requirements elicitation. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer (2010) 188–202

[93] Ionita, D., Wieringa, R., Bullee, J.W., Vasenev, A.: Tangible modelling to elicit domain knowledge: an experiment and focus group. In: Conceptual Modeling. Springer (2015) 558–565

[94] Damian, D., Helms, R., Kwan, I., Marczak, S., Koelewijn, B.: The role of domain knowledge and cross-functional communication in socio-technical coordination. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE (2013) 442–451

[95] Koskinen, K.U., Pihlanto, P., Vanharanta, H.: Tacit knowledge acquisition and sharing in a project work context. International journal of project management **21**(4) (2003) 281–290

[96] Ryan, S., O'Connor, R.V.: Acquiring and sharing tacit knowledge in software development teams: An empirical study. Information and Software Technology **55**(9) (2013) 1614–1624

[97] Smith, E.A.: The role of tacit and explicit knowledge in the workplace. Journal of knowledge Management **5**(4) (2001) 311–321

[98] Ferrucci, D., Lally, A.: Uima: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering **10**(3-4) (2004) 327–348

[99] Busse, S., Kutsche, R.D., Leser, U., Weber, H.: Federated information systems: Concepts, terminology and architectures. Forschungsberichte des Fachbereichs Informatik **99**(9) (1999)

[100] Manning, C.D., Raghavan, P., Schütze, H., et al.: Introduction to information retrieval. Volume 1. Cambridge university press Cambridge (2008)

[101] Leser, U., Naumann, F.: Informationsintegration. dpunkt, Heidelberg (2007)

[102] Gandon, F., Sabou, M., Sack, H.: Weaving a web of linked resources. Semantic Web **8**(6) (2017) 767–772

[103] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: 6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007), Berlin, Heidelberg, Springer-Verlag (2007)

[104] Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: 16th international World Wide Web conference (WWW 2007), New York, NY, USA, ACM Press (2007)

[105] Nastase, V., Strube, M., Börschinger, B., Zirn, C., Elghafari, A.: Wikinet: A very large scale multi-lingual concept network. In: LREC. (2010)

[106] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, AcM (2008) 1247–1250

[107] Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM **57**(10) (2014) 78–85

[108] Fellbaum, C.: WordNet : An Electronic Lexical Database. The MIT Press, Cambridge, MA (1998)

[109] Ruppenhofer, J., Ellsworth, M., Petruck, M.R., Johnson, C.R., Scheffczyk, J.: FrameNet II: Extended theory and practice. Institut für Deutsche Sprache, Bibliothek (2016)

[110] Schuler, K.K.: Verbnet: A broad-coverage, comprehensive verb lexicon. (2005)

[111] Navigli, R., Ponzetto, S.P.: Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence **193** (2012) 217–250

[112] Lenat, D.B.: Cyc: A large-scale investment in knowledge infrastructure. Communications of the ACM **38**(11) (1995) 33–38

[113] Speer, R., Havasi, C.: Representing General Relational Knowledge in ConceptNet 5. In: Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), Istanbul, Turkey (2012)

[114] Weikum, G., Theobald, M.: From information to knowledge: harvesting entities and relationships from web sources. In: Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems of data. PODS '10, New York, NY, USA, ACM (2010) 65–76

[115] Clark, A., Fox, C., Lappin, S.: The handbook of computational linguistics and natural language processing. John Wiley & Sons (2013)

[116] Moens, M.F.: Information extraction: algorithms and prospects in a retrieval context. Volume 21. Springer Science & Business Media (2006)

[117] Banko, M., Cafarella, M.J., Soderland, S., Broadhead, M., Etzioni, O.: Open information extraction from the web. In: IJCAI. Volume 7. (2007) 2670–2676

[118] Fader, A., Soderland, S., Etzioni, O.: Identifying relations for open information extraction. In: Proceedings of the conference on empirical methods in natural language processing, Association for Computational Linguistics (2011) 1535–1545

[119] Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al.: Never-ending learning. Communications of the ACM **61**(5) (2018) 103–115

[120] Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational linguistics - Volume 2. COLING '92, Stroudsburg, PA, USA (1992)

[121] Snow, R., Jurafsky, D., Ng, A.Y.: Semantic taxonomy induction from heterogenous evidence. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Association for Computational Linguistics (2006) 801–808

[122] Gupta, A., Piccinno, F., Kozhevnikov, M., Pasca, M., Pighin, D.: Revisiting taxonomy induction over wikipedia. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, Osaka, Japan, December 11-17 2016. Number EPFL-CONF-227401 (2016) 2300–2309

[123] Camacho-Collados, J.: Why we have switched from building full-fledged taxonomies to simply detecting hypernymy relations. arXiv preprint arXiv:1703.04178 (2017)

[124] Group, O.M.: Introduction to omg's unified modeling language (2016)

[125] Kühne, T.: What is a model? In: Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2005)

[126] da Silva, A.R.: Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures **43** (2015) 139–155

[127] Muller, P.A., Fondement, F., Baudry, B., Combemale, B.: Modeling modeling modeling. Software & Systems Modeling **11**(3) (2012) 347–359

[128] Stachowiak, H.: {Allgemeine Modelltheorie}. (1973)

[129] Liddle, S.W.: Model-driven software development. In: Handbook of Conceptual Modeling. Springer (2011) 17–54

[130] Jouault, F., Bézivin, J., Kurtev, I.: Tcs:: a dsl for the specification of textual concrete syntaxes in model engineering. In: Proceedings of the 5th international conference on Generative programming and component engineering, ACM (2006) 249–254

[131] Brown, A.W., Conallen, J., Tropeano, D.: Introduction: Models, modeling, and model-driven architecture (mda). In: Model-Driven Software Development. Springer (2005) 1–16

[132] Rodriguez-Priego, E., García-Izquierdo, F., Rubio, Á.: Modeling issues: a survival guide for a non-expert modeler. Model driven engineering languages and systems (2010) 361–375

[133] Gasevic, D., Djuric, D., Devedzic, V.: Model Driven Architecture and Ontology Development. Springer-Verlag New York, Inc. (2006)

[134] Bézivin, J.: In search of a basic principle for model driven engineering. Novatica Journal, Special Issue **5**(2) (2004) 21–24

[135] Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: International Conference on the Unified Modeling Language, Springer (2001) 19–33

[136] Henderson-Sellers, B.: Bridging metamodels and ontologies in software engineering. J. Syst. Softw. **84** (February 2011) 301–313

[137] Aßmann, U., Zschaler, S., Wagner, G.: Ontologies, meta-models, and the model-driven paradigm. In: Ontologies for software engineering and software technology. Springer (2006) 249–273

[138] Object Management Group (OMG): Meta Object Facility (MOF) Core Specification. OMG Document Number formal/2016-11-01 (`http://www.omg.org/spec/MOF/2.5.1/`) (2016)

[139] Favre, J.M.: Towards a basic theory to model model driven engineering. In: 3rd Workshop in Software Model Engineering, WiSME. (2004) 262–271

[140] Favre, J.M.: Foundations of meta-pyramids: Languages vs. metamodels–episode ii: Story of thotus the baboon1. In: Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2005)

[141] Atkinson, C., Kühne, T.: Profiles in a strict metamodeling framework. Science of Computer Programming **44**(1) (2002) 5–22

[142] Gasevic, D., Djuric, D., Devedzic, V.: Model Driven Engineering and Ontology Development. 2nd edn. Springer Publishing Company, Incorporated (2009)

[143] Gronback, R.C.: Eclipse modeling project: a domain-specific language (DSL) toolkit. Pearson Education (2009)

[144] Jacobson, I., Rumbaugh, J., Booch, G.: The unified modeling language user guide. Addison Wesley (1999)

[145] Kühne, T.: Matters of (meta-) modeling. Software and Systems Modeling **5**(4) (2006) 369–385

[146] Karagiannis, D., Kühn, H.: Metamodelling platforms. In: EC-Web. Volume 2455. (2002) 182

[147] Selic, B.: The theory and practice of modeling language design for model-based software engineering—a personal perspective. In: Generative and Transformational Techniques in Software Engineering III. Springer (2011) 290–321

[148] Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Softw. **20** (September 2003) 36–41

[149] Seidewitz, E.: What models mean. IEEE software **20**(5) (2003) 26–32

[150] Harel, D., Rumpe, B.: Meaningful modeling: what's the semantics of "semantics"? Computer **37**(10) (2004) 64–72

[151] Oestereich, B., Bremer, S.: Analyse und Design mit der UML 2.5: objektorientierte Softwareentwicklung. Oldenbourg verlag (2012)

[152] Object Management Group (OMG): Unified Modeling Language (OMG UML). OMG Document Number formal/2015-03-01 (`http://www.omg.org/spec/UML/2.5`) (2015)

[153] Harsu, M.: A survey on domain engineering. Tampere University of Technology (2002)

[154] Kang, K.C., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE software **19**(4) (2002) 58–65

[155] Embley, D.W., Thalheim, B.: Handbook of Conceptual Modeling. Springer (2014)

[156] Wieringa, R.: Real-world semantics of conceptual models. In: The evolution of conceptual modeling. Springer (2011) 1–20

[157] Vallecillo, A.: On the combination of domain specific modeling languages. In: European Conference on Modelling Foundations and Applications, Springer (2010) 305–320

[158] Strembeck, M., Zdun, U.: An approach for the systematic development of domain-specific languages. Software: Practice and Experience **39**(15) (2009) 1253–1292

[159] Kleppe, A.: A language description is more than a metamodel. In: Fourth International Workshop on Software Language Engineering, Nashville, USA. (2007) 1–9

[160] Object Management Group (OMG): MDA Guide rev. 2.0. OMG Document Number ormsc/2014-06-01 (`http://www.omg.org/cgi-bin/doc?ormsc/14-06-01`) (2014)

[161] Miller, J. and Mukerji, J.: MDA Guide Version 1.0.1. OMG Document Number omg/2003-06-01 (`http://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf`) (2003)

[162] Omg, Q.: Meta object facility (mof) 2.0 query/view/transformation specification. Final Adopted Specification (November 2005) (2008)

[163] Schreiber, A.T., Schreiber, G., Akkermans, H., Anjewierden, A., Shadbolt, N., de Hoog, R., Van de Velde, W., Shadbolt, N.R., Wielinga, B.: Knowledge engineering and management: the CommonKADS methodology. MIT press (2000)

[164] Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific american **284**(5) (2001) 34–43

[165] Hitzler, P., Krotzsch, M., Rudolph, S.: Foundations of semantic web technologies. CRC press (2009)

[166] Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: principles and methods. Data & knowledge engineering **25**(1) (1998) 161–197

[167] Staab, S., Studer, R.: Handbook on ontologies. Springer Science & Business Media (2013)

[168] Pellegrini, T., Blumauer, A.: Semantic web. Wege zur vernetzten Wissensgesellschaft. Berlin [ua] Springer (2006)

[169] Sowa, J.F.: Principles of semantic networks: Explorations in the representation of knowledge. Morgan Kaufmann (1991)

[170] Guarino, N., Oberle, D., Staab, S.: What is an ontology? In: Handbook on ontologies. Springer (2009) 1–17

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

[172] Genesereth, M.R., Nilsson, N.J.: Logical foundations of artificial. Intelligence. Morgan Kaufmann **58** (1987)

[173] van Harmelen, F., Patel-Schneider, P.F., Horrocks, I.: Reference description of the daml+ oil ontology markup language. Contributors: T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, LA Stein (2001)

[174] Yu, L.: A developer's guide to the semantic Web. Springer Science & Business Media (2011)

[175] Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited, (2016)

[176] Hogan, A.: Linked data & the semantic web standards. (2014)

[177] Berners-Lee, T.: Design issues: Linked data (2006). URL http://www.w3.org/DesignIssues/LinkedData.html (2006)

[178] Berrueta, D., Phipps, J., Miles, A., Baker, T., Swick, R.: Best practice recipes for publishing rdf vocabularies. Working draft, W3C (2008)

[179] Masood, A., Hashmi, A.: Text analytics: The dark data frontier. In: Cognitive Computing Recipes. Springer (2019) 189–224

[180] Indurkhya, N., Damerau, F.J.: Handbook of natural language processing. Volume 2. CRC Press (2010)

[181] Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval. Volume 39. Cambridge University Press (2008)

[182] Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics **19**(2) (June 1993) 313–330

[183] Petrov, S., Das, D., McDonald, R.: A universal part-of-speech tagset. arXiv preprint arXiv:1104.2086 (2011)

[184] Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: Proceedings of the NAACL '03, Stroudsburg, PA, USA, Association for Computational Linguistics (2003) 173–180

[185] Jurafsky, D., Martin, J.: Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall series in artificial intelligence. Prentice Hall (2000)

[186] Grineva, M., Grinev, M., Lizorkin, D.: Extracting key terms from noisy and multitheme documents. In: Proceedings of the 18th international conference on World wide web. WWW '09, New York, NY, USA, ACM (2009) 661–670

[187] Salton, G., Yang, C.S.: On the specification of term values in automatic indexing. Journal of documentation **29**(4) (1973) 351–372

[188] Mihalcea, R., Tarau, P.: Textrank: Bringing order into text. In: Proceedings of the 2004 conference on empirical methods in natural language processing. (2004)

[189] Evert, S.: The statistics of word cooccurrences: word pairs and collocations. (2005)

[190] Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. Volume 999. MIT Press (1999)

[191] Finkel, J.R., Grenager, T., Manning, C.: Incorporating non-local information into information extraction systems by gibbs sampling. In: Proceedings of the 43rd annual meeting on association for computational linguistics, Association for Computational Linguistics (2005) 363–370

[192] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360 (2016)

[193] Danilevsky, M., Li, Y., Reiss, F., Zhu, H., et al.: Systemt: Declarative text understanding for enterprise. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers). Volume 3. (2018) 76–83

[194] Clark, K., Manning, C.D.: Improving coreference resolution by learning entity-level distributed representations. arXiv preprint arXiv:1606.01323 (2016)

[195] Palmer, M., Gildea, D., Kingsbury, P.: The proposition bank: An annotated corpus of semantic roles. Computational linguistics **31**(1) (2005) 71–106

[196] Chiticariu, L., Li, Y., Reiss, F.R.: Rule-based information extraction is dead! long live rule-based information extraction systems! In: Proceedings of the 2013 conference on empirical methods in natural language processing. (2013) 827–832

[197] Roller, S., Kiela, D., Nickel, M.: Hearst patterns revisited: Automatic hypernym detection from large text corpora. arXiv preprint arXiv:1806.03191 (2018)

[198] Pawar, S., Palshikar, G.K., Bhattacharyya, P.: Relation extraction: A survey. arXiv preprint arXiv:1712.05191 (2017)

[199] Jiang, J., Zhai, C.: A systematic exploration of the feature space for relation extraction. In: Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference. (2007) 113–120

[200] Agichtein, E., Gravano, L.: Snowball: Extracting relations from large plain-text collections. In: Proceedings of the fifth ACM conference on Digital libraries, ACM (2000) 85–94

[201] Xu, F., Uszkoreit, H., Krause, S., Li, H.: Boosting relation extraction with limited closed-world knowledge. In: Proceedings of the 23rd International Conference on Computational Linguistics: Posters, Association for Computational Linguistics (2010) 1354–1362

[202] Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant supervision for relation extraction without labeled data. In: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2, Association for Computational Linguistics (2009) 1003–1011

[203] Min, B., Grishman, R., Wan, L., Wang, C., Gondek, D.: Distant supervision for relation extraction with an incomplete knowledge base. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. (2013) 777–782

[204] Hasegawa, T., Sekine, S., Grishman, R.: Discovering relations among named entities from large corpora. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics (2004) 415

[205] Akbik, A., Visengeriyeva, L., Herger, P., Hemsen, H., Löser, A.: Unsupervised discovery of relations and discriminative extraction patterns. Proceedings of COLING 2012 (2012) 17–32

[206] Mausam, M.: Open information extraction systems and downstream applications. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, AAAI Press (2016) 4074–4077

[207] Lenci, A.: Distributional semantics in linguistic and cognitive research. Italian journal of linguistics **20**(1) (2008) 1–31

[208] Harris, Z.S.: Distributional structure. Word **10**(2-3) (1954) 146–162

[209] Firth, J.: A synopsis of linguistic theory 1930-1955. Studies in linguistic analysis (1957) 1–32

[210] Hindle, D.: Noun classification from predicate-argument structures. In: Proceedings of the 28th annual meeting on Association for Computational Linguistics, Association for Computational Linguistics (1990) 268–275

[211] Turney, P.D., Pantel, P.: From frequency to meaning: vector space models of semantics. J. Artif. Int. Res. **37**(1) (January 2010) 141–188

[212] Potts, C.: Distributional approaches to word meanings (2013)

[213] Lenci, A.: Distributional models of word meaning. Annual review of Linguistics **4** (2018) 151–171

[214] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)

[215] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. (2013) 3111–3119

[216] Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. Journal of machine learning research **3**(Feb) (2003) 1137–1155

[217] Bakarov, A.: A survey of word embeddings evaluation methods. arXiv preprint arXiv:1801.09536 (2018)

[218] Mandera, P., Keuleers, E., Brysbaert, M.: Explaining human performance in psycholinguistic tasks with models of semantic similarity based on prediction and counting: A review and empirical validation. Journal of Memory and Language **92** (2017) 57–78

[219] Bobadilla, J., Ortega, F., Hernando, A., Gutiérrez, A.: Recommender systems survey. Knowledge-based systems **46** (2013) 109–132

[220] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge & Data Engineering (6) (2005) 734–749

[221] Lops, P., De Gemmis, M., Semeraro, G.: Content-based recommender systems: State of the art and trends. In: Recommender systems handbook. Springer (2011) 73–105

[222] Heitmann, B., Hayes, C.: Using linked data to build open, collaborative recommender systems. In: AAAI spring symposium: linked data meets artificial intelligence. Volume 2010. (2010)

[223] Figueroa, C., Vagliano, I., Rocha, O.R., Morisio, M.: A systematic literature review of linked data-based recommender systems. Concurrency and Computation: Practice and Experience **27**(17) (2015) 4659–4684

[224] de Gemmis, M., Lops, P., Musto, C., Narducci, F., Semeraro, G.: Semantics-aware content-based recommender systems. In: Recommender Systems Handbook. Springer (2015) 119–159

[225] Di Noia, T., Ostuni, V.C.: Recommender systems and linked open data. In: Reasoning Web International Summer School, Springer (2015) 88–113

[226] Tietz, T., Jäger, J., Waitelonis, J., Sack, H.: Semantic annotation and information visualization for blogposts with refer. In: VOILA@ ISWC. (2016) 28–40

[227] Robillard, M., Walker, R., Zimmermann, T.: Recommendation systems for software engineering. IEEE software **27**(4) (2010) 80–86

[228] Bruch, M., Monperrus, M., Mezini, M.: Learning from examples to improve code completion systems. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM (2009) 213–222

[229] Holmes, R., Walker, R.J., Murphy, G.C.: Approximate structural context matching: An approach to recommend relevant examples. IEEE Transactions on Software Engineering (12) (2006) 952–970

[230] Cubranic, D., Murphy, G.C., Singer, J., Booth, K.S.: Hipikat: A project memory for software development. IEEE Transactions on Software Engineering **31**(6) (2005) 446–465

[231] Ye, Y., Fischer, G.: Reuse-conducive development environments. Automated Software Engineering **12**(2) (2005) 199–235

[232] Sen, S., Baudry, B., Vangheluwe, H.: Domain-specific model editors with model completion. In: International Conference on Model Driven Engineering Languages and Systems, Springer (2007) 259–270

[233] Hessellund, A., Czarnecki, K., Wasowski, A.: Guided development with multiple domain-specific languages. In: International Conference on Model Driven Engineering Languages and Systems, Springer (2007) 46–60

[234] Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Model repositories: Will they become reality? In: CloudMDE MoDELS. (2015) 37–42

[235] Nalepa, G.J., Baumeister, J.: Synergies Between Knowledge Engineering and Software Engineering. Springer (2018)

[236] Kühne, T.: Unifying explanatory and constructive modeling: towards removing the gulf between ontologies and conceptual models. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, ACM (2016) 95–102

[237] Henderson-Sellers, B., Gonzalez-Perez, C., Eriksson, O., Ågerfalk, P.J., Walkerden, G.: Software modelling languages: A wish list. In: 7th IEEE/ACM International Workshop on Modeling in Software Engineering, MiSE 2015, Florence, Italy, May 16-17, 2015. (2015) 72–77

[238] Guizzardi, G.: Ontological foundations for structural conceptual models. CTIT, Centre for Telematics and Information Technology (2005)

[239] Guizzardi, G., Wagner, G., Almeida, J.P.A., Guizzardi, R.S.: Towards ontological foundations for conceptual modeling: the unified foundational ontology (ufo) story. Applied ontology **10**(3-4) (2015) 259–271

[240] Carvalho, V.A., Almeida, J.P.A., Fonseca, C.M., Guizzardi, G.: Multi-level ontology-based conceptual modeling. Data & Knowledge Engineering **109** (2017) 3–24

[241] Tairas, R., Mernik, M., Gray, J.: Using ontologies in the domain analysis of domain-specific languages. In: International Conference on Model Driven Engineering Languages and Systems, Springer (2008) 332–342

[242] Thonggoom, O., Song, I.Y., An, Y.: Semi-automatic conceptual data modeling using entity and relationship instance repositories. In: Proceedings of the 30th international conference on Conceptual modeling. ER'11, Berlin, Heidelberg, Springer-Verlag (2011) 219–232

[243] Gomes, P., Gandola, P., Cordeiro, J.: Helping software engineers reusing uml class diagrams. In: Proceedings of the 7th international conference on Case-Based Reasoning. ICCBR '07, Berlin, Heidelberg, Springer-Verlag (2007) 449–462

[244] Walter, T., Parreiras, F.S., Staab, S.: An ontology-based framework for domain-specific modeling. Software and Systems Modeling (2014) 1–26

[245] Ojamaa, A., Haav, H.M., Penjam, J.: Semi-automated generation of dsl meta models from formal domain ontologies. In: Model and Data Engineering. Springer (2015) 3–15

[246] Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How do practitioners use conceptual modeling in practice? Data & Knowledge Engineering **58**(3) (2006) 358–380

[247] Halpin, T.: Object-role modeling (orm/niam). In: Handbook on architectures of information systems. Springer (1998) 81–103

[248] Schreiber, G., Wielinga, B.J., Akkermans, H., Velde, W.V.d., Anjewierden, A.: Cml: The commonkads conceptual modelling language. In: Proceedings of the 8th European Knowledge Acquisition Workshop on A Future for Knowledge Acquisition. EKAW '94, London, UK, UK, Springer-Verlag (1994) 1–25

[249] Atkinson, C., Kühne, T.: In defence of deep modelling. Information & Software Technology **64** (2015) 36–51

[250] Langer, P., Mayerhofer, T., Wimmer, M., Kappel, G.: On the usage of uml: Initial results of analyzing open uml models. In: Modellierung. Volume 19. (2014) 21

[251] Atkinson, C., Kiko, K.: A detailed comparison of uml and owl. (2008)

[252] Landhäußer, M., Körner, S.J., Tichy, W.F.: Synchronizing domain models with natural language specifications. In: Proceedings of the First International Workshop on Realizing AI Synergies in Software Engineering. RAISE '12, Piscataway, NJ, USA, IEEE Press (2012) 22–26

[253] Chen, P.P.: English sentence structure and entity-relationship diagrams. Inf. Sci. **29**(2-3) (1983) 127–149

[254] Storey, V.C.: Understanding semantic relationships. The VLDB Journal **2**(4) (1993) 455–488

[255] Maroto García, N., Alcina, A.: Formal description of conceptual relationships with a view to implementing them in the ontology editor protégé. Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication **15**(2) (2009) 232–257

[256] Chaffin, R., Herrmann, D.J.: The similarity and diversity of semantic relations. Memory & Cognition **12**(2) (1984) 134–141

[257] Miller, G.A.: Wordnet: A lexical database for english. Commun. ACM **38**(11) (November 1995) 39–41

[258] Olivé, A.: Conceptual Modeling of Information Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)

[259] (OMG), O.M.G.: Ontology definition metamodel (odm). version 1.1. (2014)

[260] Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A.: Ontology engineering in a networked world. Springer Science & Business Media (2012)

[261] Brickley, D., Guha, R.V.: Rdf schema 1.1. W3C Recommendation **25** (2014)

[262] Almeida, M., Souza, R., Fonseca, F.: Semantics in the semantic web: a critical evaluation. Knowledge organization **38**(3) (2011) 187–203

[263] Huang, C.r.: Ontology and the lexicon: a natural language processing perspective. Cambridge University Press (2010)

[264] Van Assem, M., Menken, M.R., Schreiber, G., Wielemaker, J., Wielinga, B.: A method for converting thesauri to rdf/owl. In: The Semantic Web–ISWC 2004. Springer (2004) 17–31

[265] Miles, A., Bechhofer, S.: Skos simple knowledge organization system reference. W3C recommendation **18** (2009) W3C

[266] Winston, M.E., Chaffin, R., Herrmann, D.: A taxonomy of part-whole relations. Cognitive science **11**(4) (1987) 417–444

[267] Aitchison, J., Gilchrist, A., Bawden, D.: Thesaurus construction and use: a practical manual. Psychology Press (2000)

[268] Zesch, T.: Study of semantic relatedness of words using collaboratively constructed semantic resources. PhD thesis, TU Darmstadt (2010)

[269] Pomikálek, J., Jakubícek, M., Rychlỳ, P.: Building a 70 billion word corpus of english from clueweb. In: LREC. (2012) 502–506

[270] Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The stanford corenlp natural language processing toolkit. In: ACL (System Demonstrations). (2014) 55–60

[271] Popescu, A.M.: Information extraction from unstructured web text. PhD thesis (2007)

[272] Banko, M.: Open Information Extraction for the Web. PhD thesis, University of Washington (2009)

[273] Bhagat, R., Hovy, E., Patwardhan, S.: Acquiring paraphrases from text corpora. In: Proceedings of the fifth international conference on Knowledge capture, ACM (2009) 161–168

[274] Leigh, H.: The patient: Biological, psychological, and social dimensions of medical practice. Springer Science & Business Media (1980)

[275] Brown, K.: Encyclopedia of language and linguistics. (2006)

[276] Baroni, M., Lenci, A.: Distributional memory: A general framework for corpus-based semantics. Computational Linguistics **36**(4) (2010) 673–721

[277] Turney, P.D.: Distributional semantics beyond words: Supervised learning of analogy and paraphrase. arXiv preprint arXiv:1310.5042 (2013)

[278] Kiela, D., Clark, S.: Detecting compositionality of multi-word expressions using nearest neighbours in vector space models. In: EMNLP. (2013) 1427–1432

[279] Ramisch, C.: Multiword Expressions Acquisition. Springer (2014)

[280] Lin, D., Church, K.W., Ji, H., Sekine, S., Yarowsky, D., Bergsma, S., Patil, K., Pitler, E., Lathbury, R., Rao, V., et al.: New tools for web-scale n-grams. In: LREC. (2010)

[281] Lin, Y., Michel, J.B., Aiden, E.L., Orwant, J., Brockman, W., Petrov, S.: Syntactic annotations for the google books ngram corpus. In: Proceedings of the ACL 2012 system demonstrations, Association for Computational Linguistics (2012) 169–174

[282] Justeson, J.S., Katz, S.M.: Technical terminology: some linguistic properties and an algorithm for identification in text. Natural language engineering **1**(01) (1995) 9–27

[283] Feldman, R., Fresko, M., Kinar, Y., Lindell, Y., Liphstat, O., Rajman, M., Schler, Y., Zamir, O.: Text mining at the term level. In: Principles of Data Mining and Knowledge Discovery. Springer (1998) 65–73

[284] Williams, S.: An analysis of pos tag patterns in ontology identifiers and labels. Technical report, Technical Report TR2013/02, Department of Computing, The Open University, UK (2013)

[285] Nakashole, N., Weikum, G., Suchanek, F.: Patty: a taxonomy of relational patterns with semantic types. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics (2012) 1135–1145

[286] Hitzler, P., Krötzsch, M., Rudolph, S., Sure, Y.: Semantic Web: Grundlagen. Springer-Verlag (2007)

[287] Group, W.O.W., et al.: {OWL} 2 web ontology language document overview. (2009)

[288] Cyganiak, R., Wood, D., Lanthaler, M.: Rdf 1.1 concepts and abstract syntax. W3C Recommendation. Feb (2014)

[289] Vatant, B., Wick, M.: Geonames ontology (2012)

[290] Prud'Hommeaux, E., Seaborne, A., et al.: Sparql query language for rdf. W3C recommendation **15** (2008)

[291] Guarino, N.: Formal ontology, conceptual analysis and knowledge representation. International journal of human-computer studies **43**(5) (1995) 625–640

[292] Guarino, N.: Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy. Volume 46. IOS press (1998)

[293] Wong, W., Liu, W., Bennamoun, M.: Ontology learning from text: A look back and into the future. ACM Computing Surveys (CSUR) **44**(4) (2012) 20

[294] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia-a crystallization point for the web of data. Web Semantics: science, services and agents on the world wide web **7**(3) (2009) 154–165

[295] Hepp, M.: Possible ontologies: How reality constrains the development of relevant ontologies. Internet Computing, IEEE **11**(1) (2007) 90–96

[296] Niles, I., Pease, A.: Towards a standard upper ontology. In: Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001. FOIS '01, New York, NY, USA, ACM (2001)

[297] McCrae, J., Aguado-de Cea, G., Buitelaar, P., Cimiano, P., Declerck, T., Gómez-Pérez, A., Gracia, J., Hollink, L., Montiel-Ponsoda, E., Spohr, D., et al.: Interchanging lexical resources on the semantic web. Language Resources and Evaluation **46**(4) (2012) 701–719

[298] Chiarcos, C., McCrae, J., Cimiano, P., Fellbaum, C.: Towards open data for linguistics: Linguistic linked data. In: New Trends of Research in Ontologies and Lexical Resources. Springer (2013) 7–25

[299] McCrae, J., Fellbaum, C., Cimiano, P.: Publishing and linking wordnet using lemon and rdf. In: Proceedings of the 3rd Workshop on Linked Data in Linguistics. (2014)

[300] Siemoneit, B., McCrae, J.P., Cimiano, P.: Linking four heterogeneous language resources as linked data. ACL-IJCNLP 2015 (2015) 59

[301] Roget, P.M.: Roget's Thesaurus of English Words and Phrases... TY Crowell Company (1911)

[302] Summers, E., Isaac, A., Redding, C., Krech, D.: Lcsh, skos and linked data. Universitätsverlag Göttingen (2008) 25

[303] Caracciolo, C., Stellato, A., Morshed, A., Johannsen, G., Rajbhandari, S., Jaques, Y., Keizer, J.: The agrovoc linked dataset. Semantic Web **4**(3) (2013) 341–348

[304] Labra Gayo, J.E., McCrae, J.P., Windhouwer, M., de Melo, G.: Lexvo. org: Language-related information for the linguistic linked data cloud. Semantic Web **6**(4) (2015) 393–400

[305] Manaf, N.A.A., Bechhofer, S., Stevens, R.: The current state of skos vocabularies on the web. In: The Semantic Web: Research and Applications. Springer (2012) 270–284

[306] McCrae, J.P., Chiarcos, C., Bond, F., Cimiano, P., Declerck, T., de Melo, G., Gracia, J., Hellmann, S., Klimek, B., Moran, S., et al.: The open linguistics working group: Developing the linguistic linked open data cloud

[307] Shvaiko, P., Euzenat, J.: Ontology matching: state of the art and future challenges. IEEE Transactions on knowledge and data engineering **25**(1) (2013) 158–176

[308] Hasan, K.S., Ng, V.: Automatic keyphrase extraction: A survey of the state of the art. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Volume 1. (2014) 1262–1273

[309] Beliga, S., Meštrović, A., Martinčić-Ipšić, S.: An overview of graph-based keyword extraction methods and approaches. Journal of information and organizational sciences **39**(1) (2015) 1–20

[310] Tandon, N., de Melo, G., Weikum, G.: Deriving a Web-Scale Common Sense Fact Database. In: AAAI. (2011)

[311] Nulty, P., Costello, F.: Using lexical patterns in the Google Web 1T corpus to deduce semantic relations between nouns. In: Proceedings of the Workshop on Semantic Evaluations. DEW '09, Stroudsburg, PA, USA (2009) 58–63

[312] Lin, D., Church, K., Ji, H., Sekine, S., Yarowsky, D., Bergsma, S., Patil, K., Pitler, E., Lathbury, R., Rao, V., et al.: Unsupervised acquisition of lexical knowledge from n-grams: Final report of the 2009 jhu clsp workshop. In: Proceedings of Workshop at the Center for Language and Speech Processing at Johns Hopkins University in. Volume 2010. (2009) 89

[313] Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In: Proceedings of the 20th international joint conference on Artifical intelligence. IJCAI'07, San Francisco, CA, USA (2007)

[314] Zesch, T.: Study of Semantic Relatedness of Words Using Collaboratively Constructed Semantic Resources. PhD thesis, TU Darmstadt (Februar 2010)

[315] Zhang, Z., Gentile, A.L., Ciravegna, F.: Recent advances in methods of lexical semantic relatedness–a survey. Natural Language Engineering **19**(4) (2013) 411–479

[316] Pennington, J., Socher, R., Manning, C.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). (2014) 1532–1543

[317] Levy, O., Goldberg, Y.: Dependency-based word embeddings. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Volume 2. (2014) 302–308

[318] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A.: Advances in pre-training distributed word representations. In: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018). (2018)

[319] MacAvaney, S., Zeldes, A.: A deeper look into dependency-based word embeddings. arXiv preprint arXiv:1804.05972 (2018)

[320] Agt-Rickauer, H., Kutsche, R.D., Sack, H.: Domore – a recommender system for domain modeling. In: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD, INSTICC, SciTePress (2018) 71–82

[321] Frantzi, K., Ananiadou, S., Mima, H.: Automatic recognition of multi-word terms:. the c-value/nc-value method. International Journal on Digital Libraries **3**(2) (2000) 115–130

[322] Michel, J.B., Shen, Y.K., Aiden, A.P., Veres, A., Gray, M.K., Team, T.G.B., Pickett, J.P., Hoiberg, D., Clancy, D., Norvig, P., Orwant, J., Pinker, S., Nowak, M.A., Aiden, E.L.: Quantitative Analysis of Culture Using Millions of Digitized Books. Science **331**(6014) (January 2011) 176–182

[323] Goldberg, Y., Orwant, J.: A dataset of syntactic-ngrams over time from a very large corpus of english books. (2013)

[324] Harris, Z.: Distributional structure. Word **10**(23) (1954) 146–162

[325] Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. Computational linguistics **16**(1) (1990) 22–29

[326] Pibiri, G.E., Venturini, R.: Efficient data structures for massive n-gram datasets. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM (2017) 615–624

[327] Euzenat, J., Shvaiko, P.: Ontology Matching, Second Edition. Springer (2013)

[328] Algergawy, A., Cheatham, M., Faria, D., Ferrara, A., Fundulaki, I., Harrow, I., Hertling, S., Jiménez-Ruiz, E., Karam, N., Khiat, A., et al.: Results of the ontology alignment evaluation initiative 2018. In: CEUR Workshop Proceedings. Volume 2288., RWTH (2018) 76–116

[329] Faria, D., Pesquita, C., Balasubramani, B.S., Tervo, T., Carriço, D., Garrilha, R., Couto, F.M., Cruz, I.F.: Results of aml participation in oaei 2018. In: Ontology Matching: OM-2018: Proceedings of the ISWC Workshop. (2018) 125

[330] Otero-Cerdeira, L., Rodríguez-Martínez, F.J., Gómez-Rodríguez, A.: Ontology matching: A literature review. Expert Systems with Applications **42**(2) (2015) 949–971

[331] Suchanek, F.M., Abiteboul, S., Senellart, P.: Paris: Probabilistic alignment of relations, instances, and schema. Proceedings of the VLDB Endowment **5**(3) (2011) 157–168

[332] Nguyen, V., Bodenreider, O., Sheth, A.: Don't like rdf reification?: making statements about statements using singleton property. In: Proceedings of the 23rd international conference on World wide web, ACM (2014) 759–770

[333] Rouces, J., de Melo, G., Hose, K.: Framebase: Enabling integration of heterogeneous knowledge. Semantic Web **8**(6) (2017) 817–850

[334] Rouces, J., de Melo, G., Hose, K.: Heuristics for connecting heterogeneous knowledge via framebase. In: European Semantic Web Conference, Springer (2016) 20–35

[335] Bayerl, S., Granitzer, M.: Linked data warehousing. In: Linked Enterprise Data. Springer (2014) 177–192

[336] Leser, U.: Query planning in mediator based information systems. (2000)

[337] Quilitz, B., Leser, U.: Querying distributed rdf data sources with sparql. In: European semantic web conference, Springer (2008) 524–538

[338] Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngonga Ngomo, A.C.: A fine-grained evaluation of sparql endpoint federation systems. Semantic Web **7**(5) (2016) 493–518

[339] Wylot, M., Hauswirth, M., Cudré-Mauroux, P., Sakr, S.: Rdf data storage and query processing schemes: A survey. ACM Computing Surveys (CSUR) **51**(4) (2018) 84

[340] Hartig, O.: Querying a web of linked data: foundations and query execution. Volume 24. Ios Press (2016)

[341] McCrae, J., Spohr, D., Cimiano, P.: Linking lexical resources and ontologies on the semantic web with lemon. In: Extended Semantic Web Conference, Springer (2011) 245–259

[342] McCrae, J., Aguado-de Cea, G., Buitelaar, P., Cimiano, P., Declerck, T., Gómez Pérez, A., Gracia, J., Hollink, L., Montiel-Ponsoda, E., Spohr, D., et al.: The lemon cookbook. Online. Google Scholar (2010)

[343] Grassi, M., Piazza, F.: Towards an rdf encoding of conceptnet. In: International Symposium on Neural Networks, Springer (2011) 558–565

[344] Najmi, E., Malik, Z., Hashmi, K., Rezgui, A.: Conceptrdf: An rdf presentation of conceptnet knowledge base. In: Information and Communication Systems (ICICS), 2016 7th International Conference on, IEEE (2016) 145–150

[345] Chen, H., Trouve, A., Murakami, K.J., Fukuda, A.: A concise conversion model for improving the rdf expression of conceptnet knowledge base. In: Artificial Intelligence and Robotics. Springer (2018) 213–221

[346] Leser, U., Naumann, F.: Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen. dpunkt Verlag (2012)

[347] Wiederhold, G.: Mediators in the architecture of future information systems. Computer **25**(3) (1992) 38–49

[348] Ehrmann, M., Cecconi, F., Vannella, D., Mccrae, J.P., Cimiano, P., Navigli, R.: Representing multilingual data as linked data: the case of babelnet 2.0. In: LREC. (2014) 401–408

[349] Sérasset, G.: Dbnary: Wiktionary as a lemon-based multilingual lexical resource in rdf. Semantic Web **6**(4) (2015) 355–361

[350] Eckle-Kohler, J., McCrae, J.P., Chiarcos, C.: Lemonuby–a large, interlinked, syntactically-rich lexical resource for ontologies. Semantic Web **6**(4) (2015) 371–378

[351] Suominen, O., Mader, C.: Assessing and improving the quality of skos vocabularies. Journal on Data Semantics **3**(1) (2014) 47–73

[352] Fafalios, P., Tzitzikas, Y.: Sparql-ld: a sparql extension for fetching and querying linked data. In: International Semantic Web Conference (Posters & Demos). (2015)

[353] Fafalios, P., Yannakis, T., Tzitzikas, Y.: Querying the web of data with sparql-ld. In: International Conference on Theory and Practice of Digital Libraries, Springer (2016) 175–187

[354] Hebig, R., Quang, T.H., Chaudron, M.R., Robles, G., Fernandez, M.A.: The quest for open source projects that use uml: mining github. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, ACM (2016) 173–183

[355] France, R.B., Bieman, J.M., Mandalaparty, S.P., Cheng, B.H., Jensen, A.: Repository for model driven development (remodd). In: Software Engineering (ICSE), 2012 34th International Conference on, IEEE (2012) 1471–1472

[356] Lucrédio, D., Fortes, R.P.d.M., Whittle, J.: Moogle: a metamodel-based model search engine. Software & Systems Modeling **11**(2) (2012) 183–208

[357] Kuhn, A.: On recommending meaningful names in source and uml. In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, ACM (2010) 50–51

[358] Ganser, A., Lichter, H., Roth, A., Rumpe, B.: Staged model evolution and proactive quality guidance for model libraries. Software Quality Journal (2015) 1–34

[359] Ganser, A., Lichter, H.: Engineering model recommender foundations. In: Modelsward 2013, proceedings of the 1st international conference on model-driven engineering and software development, Barcelona, Spain. Volume 19. (2013) 135–142

[360] Dyck, A., Ganser, A., Lichter, H.: On designing recommenders for graphical domain modeling environments. In: MODELSWARD. (2014) 291–299

[361] Ganser, A., Lichter, H., Roth, A., Rumpe, B.: Proactive quality guidance for model evolution in model libraries. In: ME 2013–Models and Evolution Workshop Proceedings. (2013) 50

[362] Elinson, S., Hanns, M., Kirstein, M., Kronseder, S., Köhler, S.: Eclipse model repository (2010)

[363] Dyck, A., Ganser, A., Lichter, H.: A framework for model recommenders requirements, architecture and tool support. In: Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on, IEEE (2014) 282–290

[364] Dyck, A., Ganser, A., Lichter, H.: Model recommenders for command-enabled editors. MDEBE'2013 (2013)

[365] Segura, Á.M., Pescador, A., de Lara, J., Wimmer, M.: An extensible meta-modelling assistant. In: Enterprise Distributed Object Computing Conference (EDOC), 2016 IEEE 20th International, IEEE (2016) 1–10

[366] Ángel, M.S., de Lara, J., Neubauer, P., Wimmer, M.: Automated modelling assistance by integrating heterogeneous information sources. Computer Languages, Systems & Structures **53** (2018) 90–120

[367] Milajevs, D., Sadrzadeh, M., Purver, M.: Robust co-occurrence quantification for lexical distributional semantics. ACL 2016 (2016) 58

[368] Agt, H., Bauhoff, G., Cartsburg, M., Kumpe, D., Kutsche, R., Milanovic, N.: Metamodeling foundation for software and data integration. In: International United Information Systems Conference, Springer (2009) 328–339

[369] Agt, H., Bauhoff, G., Widiker, J., Milanovic, N., Kutsche, R.D.: Model-based semantic conflict analysis for software- and data-integration scenarios. Technical Report 2009/7, Technische Universität Berlin (2009)

[370] Agt, H., Kutsche, R.D., Natho, N., Li, Y.: The bizware research project. In: Model Driven Engineering Languages and Systems-Exhibition Track, 15th International Conference, MODELS. (2012)

[371] Agt, H.: Supporting Software Language Engineering by Automated Domain Knowledge Acquisition. In: MODELS 2011 Workshops. Volume 7167 of LNCS., Wellington, New Zealand, Springer (2012)

[372] Agt, H., Kutsche, R.D.: Automated construction of a large semantic network of related terms for domain-specific modeling. In: Advanced Information Systems Engineering, 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Volume 7908 of Lecture Notes in Computer Science (LNCS)., Springer (2013) 610–625

[373] Agt, H.: SemAcom: A System for Modeling with Semantic Autocompletion. In: Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Demo Track, Innsbruck, Austria (2012)

[374] Agt, H., Kutsche, R.D., Wegeler, T.: Guidance for Domain Specific Modeling in Small and Medium Enterprises. In: SPLASH '11 Workshops. Proceedings of the compilation of the co-located workshops on DSM'11, Portland, OR, USA (2011)

[375] Agt-Rickauer, H., Waitelonis, J., Tietz, T., Sack, H.: Data integration for the media value chain. In: Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016. (2016)

[376] Agt-Rickauer, H., Hentschel, C., Sack, H.: Semantic annotation and automated extraction of audio-visual staging patterns in large-scale empirical film studies. In: Proceedings of the Posters and Demos Track of the 14th International Conference on Semantic Systems (SEMANTiCS 2018), Vienna, Austria. (2018)

[377] Agt-Rickauer, H., Kutsche, R.D., Sack, H.: Automated recommendation of related model elements for domain models. Communications in Computer and Information Science. CCIS **991** (2019) 1–25

[378] Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: Rml: A generic language for integrated rdf mappings of heterogeneous data. In: LDOW. (2014)

[379] Evermann, J., Porres, I.: Doctoral symposium at models 2011. In: Models in Software Engineering - Workshops and Symposia at MODELS 2011, Wellington, New Zealand, October 16-21, 2011, Reports and Revised Selected Papers. (2011) 1–3

[380] Filipiak, D., Agt-Rickauer, H., Hentschel, C., Filipowska, A., Sack, H.: Quantitative analysis of art market using ontologies, named entity recognition and machine learning: A case study. In: Business Information Systems - 19th International Conference, BIS 2016, Leipzig, Germany, July, 6-8, 2016, Proceedings. (2016) 79–90

[381] Horridge, M., Gonçalves, R.S., Nyulas, C.I., Musen, M.A.: Webprot\'eg\'e: A cloud-based ontology editor. arXiv preprint arXiv:1902.08251 (2019)

[382] Grave, E., Bojanowski, P., Gupta, P., Joulin, A., Mikolov, T.: Learning word vectors for 157 languages. arXiv preprint arXiv:1802.06893 (2018)

[383] Ruder, S., Vulić, I., Søgaard, A.: A survey of cross-lingual word embedding models. arXiv preprint arXiv:1706.04902 (2017)

[384] Marino, J.B., Banchs, R.E., Crego, J.M., de Gispert, A., Lambert, P., Fonollosa, J.A., Costa-Jussà, M.R.: N-gram-based machine translation. Computational linguistics **32**(4) (2006) 527–549

[385] Bordea, G., Lefever, E., Buitelaar, P.: Semeval-2016 task 13: Taxonomy extraction evaluation (texeval-2). In: Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016). (2016) 1081–1091

[386] Sarkar, R., McCrae, J.P., Buitelaar, P.: A supervised approach to taxonomy extraction using word embeddings. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018). (2018)

[387] Tandon, N., Varde, A.S., de Melo, G.: Commonsense knowledge in machine intelligence. ACM SIGMOD Record **46**(4) (2018) 49–52

[388] Jebbara, S., Basile, V., Cabrio, E., Cimiano, P.: Extracting common sense knowledge via triple ranking using supervised and unsupervised distributional models. Semantic Web (Preprint) (2018) 1–20

[389] Camacho-Collados, J., Pilehvar, M.T.: From word to sense embeddings: A survey on vector representations of meaning. Journal of Artificial Intelligence Research **63** (2018) 743–788

[390] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. arXiv preprint arXiv:1802.05365 (2018)

[391] Chiticariu, L., Danilevsky, M., Li, Y., Reiss, F., Zhu, H.: Systemt: Declarative text understanding for enterprise. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers). (2018) 76–83

[392] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **36**(4) (2015)

[393] Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-scale querying through linked data fragments. In: LDOW. (2014)