Dissertation submitted to the Graz University of Technology, Faculty of Computer Science, for the attainment of the degree of Doctor of Engineering Sciences (Dr. techn.)

Automatic Support for Ontology Evaluation

Review of Entailed Statements and Assertional Effects for OWL Ontologies

Viktoria Pammer

February 24, 2010

Reviewed and Examined by:

Prof. Dr. Klaus Tochtermann
Knowledge Management Institute
Graz University of Technology

Prof. Dr. Rudi Studer
Institute of Applied Informatics and Formal Description Methods
Karlsruher Institute of Technology

Advised by:

Dr. Stefanie N. Lindstaedt
Luciano Serafini

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 9. Februar 2010 ……

……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

February 9, 2010
……
date

……..
(signature)

Acknowledgements

Writing this doctoral thesis and doing all the research that led up to it was a lot of work and took considerable time. There are also some other people, besides myself, who have invested time and energy into this PhD. My advisors, Stefanie Lindstaedt and Luciano Serafini, dedicated both time and patience to me and this work. They encouraged my ideas and pushed me to go on and work on them, for which I am truly grateful. Luciano specifically has read some parts of my thesis in many more versions than is human, and gave me useful feedback every time. I also shared interesting discussions with colleagues at the Knowledge Management Institute and the Know-Center, as well as with people at conferences, and I am glad for every input I got over the years from everyone. Specific honours and thanks in this regard go to my colleague and 'PhD-sister' Barbara Kump, with whom I've shared quite a few ups and downs. Speaking of the Knowledge Management Institute and the Know-Center, the last but not the least thanks go to Klaus Tochtermann who heads both, and in doing so manages to provide a research environment which I experienced both as relaxed and motivating.
Throughout all of this I had friends, my mother and sister and the man of my life. Especially the latter had to suffer through some intense periods, with higher frequency towards the end of this thesis, and has yet never failed to brighten up my mood. These people have made up, and still do, the best part of my life after all.

Abstract

The typical goal of representing knowledge formally is to make knowledge accessible to machines. Hence, knowledge representation formalisms are chosen to accommodate well to the strengths of machines, often at the expense of being less easy to understand for humans. At the same time, formal representations are most often created or maintained by humans. This discrepancy provides the motivation for the central idea of this thesis, namely to facilitate evaluating the conceptual correctness of ontologies by means of reasoning services. For instance, when a logical axiom is added to an ontology, this addition may lead to logically consistent but conceptually wrong inferences. In the context of this thesis, specifically description logics with a view on those underlying the W3C recommended web ontology language OWL are considered as knowledge representation language in which ontologies are described.
First, it has been investigated both by analysis and in practice how the review of automatically inferred statements can serve the purpose of evaluating an existing ontology. In a second venture, the usage of instance data to give ontology engineers concrete examples of the implications of general statements, i.e. terminological and role axioms in description logic languages, has been studied. Indeed, the effects of terminological and role axioms on instance data have not been studied in literature so far, although from a mathematical point of view the question of conservative extensions, i.e. the logical difference between two sets of terminological and role axioms, is very close. Thus it was necessary to provide a formal definition for effects that capture the intension of "knowledge lost or gained about data", as well as find and prove a decision procedure. Both research ventures have led to concrete implementations of ontology evaluation functionalities within MoKi, a wiki-based ontology engineering environment. As a consequence, MoKi is now, to the best of the author's knowledge, the only state-of-the art ontology engineering environment that explicitly supports conceptual ontology evaluation rather than ontology debugging in the sense of finding logical contradictions.

Kurzfassung

Die Motivation Wissen formal zu repräsentieren besteht üblicherweise darin es für Maschinen verarbeitbar zu machen. Daher werden Wissensrepräsentationsformalismen sinnvollerweise so gewählt, daß sie gut automatisch verarbeitbar sind. Darunter leidet allerdings häufig die Verständlichkeit der Repräsentation für Menschen. Auf der anderen Seite sind es meistens Menschen, die Wissen formalisieren oder formalisiertes Wissen aktuell halten müssen. Diese Diskrepanz ist die zentrale Motivation der vorliegenden Doktorarbeit, Möglichkeiten zu untersuchen wie die Evaluierung der konzeptionellen Richtigkeit von Ontologien automatisch unterstützt werden kann. Wenn zum Beispiel ein logischer Satz zu einer Ontologie, einem formalen Wissensmodell über Konzepte und Beziehungen zwischen Konzepten, hinzugefügt wird, kann es passieren daß die Ontologie zwar logisch weiterhin widerspruchsfrei ist aber konzeptionell falsch. Als spezieller Formalismus werden im Rahmen dieser Doktorarbeit Beschreibungslogiken betrachtet.
Zuerst wurde sowohl analytisch als auch anhand einer Benutzerstudie betrachtet, inwieweit die systematische Begutachtung von Inferenzen den Ontologieevaluierungsprozess unterstützen kann. Als weitere Forschungstätigkeit wurde die Verwendung von konkreten Daten zur Erstellung von Beispielen für die Bedeutung von logischen Sätzen über Konzepte und Relationen, terminologische und Rollenaxiome, untersucht. Vor der vorliegenden Doktorarbeit wurden in der Literatur den Auswirkungen von terminologischen und Rollenaxiomen auf konkrete Daten kaum Bedeutung geschenkt. Es war also notwendig, zuerst eine formale Definition zu finden die dem Ziel entspricht, “Wissen über Daten das neu hinzukommt oder verloren geht” auszudrücken. Darauf aufbauend werden die Entscheidbarkeit des Problems gezeigt und ein Entscheidungsalgorithmus beschrieben. Beide Forschungsarbeiten führten zu einer Erweiterung des MoKi, eines wiki-basierten Ontologieeditors, um Funktionalitäten die Ontologieevaluierung unterstützen. MoKi ist somit, nach bestem Wissen der Autorin, zu diesem Zeitpunkt der einzige State-of-the-Art Ontologieeditor der konzeptionelle Ontologieevaluierung unterstützt.

Contents

List of Figures

List of Tables

1

Introduction

Just as testing is an integral part of software engineering, so is ontology evaluation an integral part of ontology engineering. Typically, the knowledge representation formalisms in which ontologies are represented are chosen to accommodate well to the strengths of machines rather than to the strengths of humans, such as expressive logics. This puts humans who create ontologies at a disadvantage and makes it difficult for them (all of us!) to assess the quality of the created ontologies. Second, ontology evaluation is typically carried out as an activity separate of modelling itself, so that at the end of the ontology engineering process the complete ontology is evaluated given a set of evaluation criteria. Especially in ontology engineering settings where contributors are spatially distributed, or where an ontology is intended to evolve over time, this falls short of the actual necessity to execute continuous quality control. The central research question addressed within this thesis is how to evaluate ontologies and how to support this automatically, ideally within an ontology engineering environment. A focus has been put on investigating the **assessment of implications of formal axioms**, i.e. knowledge which is implicit in the explicitly stated ontology, **in order to ensure alignment between the interpretation of the ontology by machines and humans**. To this purpose, the benefits of reviewing inferences for ontology evaluation purposes are discussed, and a reasoning service that computes specific inferences, namely assertional effects, has been created. The latter allows contributors to assess their modelling choices as soon as they have been taken. Through this, ontology engineering and ontology evaluation activities become more tightly integrated, which in the end is a prerequisite for an evolutionary execution of the ontology engineering process.
Where necessary, theoretical discussions are limited to the knowledge representation formalism of description logics, and implementation-oriented discussions to ontologies expressed in OWL, the W3C recommended web ontology language that is based on description logics. Furthermore, ontology engineering is considered to take place during ontology engineering. From a methodological viewpoint, this thesis contributes at a conceptual level to automated support for ontology evaluation by providing theoretical discussions and consequent implementation of promising ideas. User evaluations were outside the scope of this thesis.

1.1 Motivation

Ontologies have received rising attention especially through the interest in the Semantic Web, although they have a long tradition in the scientific fields of artificial intelligence, database theory, library and life sciences, where however not always the term "ontology" is used. An ontology is now most commonly understood to be "a formal, explicit specification of a shared conceptualization" [123], although many slightly differing definitions of what an ontology precisely is still exist. More casually put, a formal ontology consists of an agreed-upon list of concepts, their inter-relations and the concepts' formal definitions in a given knowledge representation formalism as for instance logic. In computer science, ontologies are seen as enabling technology for a variety of purposes. First, they can be used to describe content in a machine-intelligible way (which is one of the milestones towards realising the Semantic Web [12]). Also the recent trend to publish data, e.g. about products or about people, on the web instead of processed content such as full-blown texts[1], falls together well with the goals of the Semantic Web. Second, ontologies serve to pull knowledge out of applications and explicate this knowledge, thus making it easier to build reusable applications. In the same vein, interoperability between applications (ontology merging in order to produce a common viewpoint on a domain) can be increased for instance. Finally, ontologies can also be used to simply document a precise, shared understanding of a domain between humans as happens when a terminology needs to be fixed. In this case however, ontologies sometimes are not heavily formalised. Although ontologies range from very informal to simple to strictly formal, in the context of computer science, formal ontologies, i.e. machine-readable and machine-interpretable ontologies, are naturally at the focus of attention.

Creating and maintaining a formal ontology requires a significant quantity and quality of human effort. On the one hand, the goal of representing knowledge formally is to make knowledge accessible to machines. Hence, knowledge representation formalisms are chosen to accommodate well to the strengths of machines, often at the expense of being less easy to understand for humans. For instance, highly expressive logics have been chosen to represent knowledge on the Semantic Web rather than pictures. At the same time, the formal representation is often created or maintained by humans and not by machines. This discrepancy is one of the main reasons why knowledge engineering is such a cumbersome task for humans.

In order to create a formal ontology, typically some variation of the following ontology engineering process is executed: First, the domain to be represented is defined, i.e. what shall be represented and what shall be left out of the representation. Knowledge which shall be expressed formally is then acquired from different data sources such as domain experts, documents, databases etc. This knowledge needs to be organised, and different world views or conflicts of knowledge need to be sorted out. Finally, the acquired and structured knowledge needs to be encoded in the chosen knowledge representation formalism. Attempts have been made to automatically create ontologies, for instance by learning ontologies from natural language text through applying machine-

[1] The trend to publish data is part of the paradigm change named "Web 2.0" by O'Reilly [94] and the Linked Data initiative [73] is its Semantic Web incarnation.

learning techniques and prior knowledge about natural language. However, such ontology learning approaches provide only partial support since state-of-the-art ontology learning techniques is at the level of term and relation extraction when learning from natural language text [17, 77]. On the other hand, they currently usefully serve as part of knowledge acquisition activities. Furthermore, even if ontology learning was completely successful, ontology engineering can not always be seen as merely "re-writing" already known knowledge in a formal language. Sometimes, it is precisely this act of formal specification in which implicit knowledge is made explicit or new knowledge is generated. So there may not actually be natural language texts or other prior documentation of the knowledge to be formalised at hand. For this reason I argue that some manual interference with ontologies will always be necessary where a certain quality is expected from these formal models.
However, as becomes apparent already from the short description of the ontology engineering process above, ontology engineering is a challenging task also for humans. Already explicating knowledge which is implicitly available to domain experts frequently presents difficulties, and is often not possible at all. Structuring explicit knowledge is the next task, conceptually difficult as well. The complexity of the knowledge representation formalisms significantly adds to these difficulties (see e.g. [75] for a similar statement). In this work, I address the latter problem in the context of description logics, which underlies the W3C recommended web ontology language OWL. In short, ontology engineers shall review inferences, which can be thought of as the interpretation given by machines to the explicitly stated ontology. Thus, ontology engineers can ascertain that machines understand the ontology (and reason on it) as intended by the ontology engineers.

Through understanding the implications of formal statements, ontology engineers are able to judge the statements' adequacy and to detect modelling mistakes. Modelling mistakes may occur for instance because the ontology in question is large, complex or heterogeneous if multiple people were involved in modelling, or because the used formalism is too little understood, naturally besides simple errors of negligence. Sometimes mistakes manifest themselves as logical inconsistencies[2], at other times as undesired inferences or less obviously via the implemented ontology's models. The latter means that either there are unintended models which satisfy the given ontology or there are intended models which do not satisfy the modelled ontology[3]. One fundamental difficulty for humans is to anticipate the implications formal statements may have if reasoning is applied to the whole ontology (see also [75] and specifically for

[2] Note that logical inconsistencies are always modelling mistakes. In the case where inconsistent facts, as for instance conflicting world views as "God exists" vs. "God does not exist", shall be modelled this must still be done in a logically consistent manner, for instance by qualifying these statements as opinions of different persons. Nevertheless, it can be argued that particularly in a distributed and collaborative knowledge engineering setting it is reasonable to assume that logical inconsistencies will arise from time to time. Following this argument, the question of how ontology engineering tools, and foremost reasoners, should deal with inconsistencies in case they do arise are studied, e.g. in [11, 53].

[3] The latter two modelling mistakes are inspired by the discussion by Guarino in [50] of conceptualizations, ontologies and intended models, where mismatches between intended models and implemented ontologies are discussed as problem in the context of ontology integration.

OWL 1 DL [109]). The following are examples of factors that complicate the correct and efficient judgement of implications of their own statements by ontology engineers.

Model Size and Overview

The formal model may be so large that an ontology engineer can not easily maintain and/or get an overview of the complete ontology. Thus, it becomes impossible to judge the implications of single statements that are added, deleted or modified.

Complexity of Inference Rules

Inference rules in the chosen knowledge representation language may be so complex that an ontology engineer can not easily judge the implications of single statements that are added, deleted or modified.
This can be assumed to be true for most logics, including description logics. The precise reason may be a limited computational ability of the ontology engineer. Additionally, it may even be impossible to compute all consequences because infinitely many exist. In the latter case also an automated support can only present a subset of all inferences. The most relevant challenge for an automated approach then is to find relevant consequences which can be presented. In most logics and description logics, infinitely many consequences may exist.

Control

In a collaborative setting, the formal model is not under the control of a single knowledge engineer. Effects of modelling may result from parts that the current ontology engineer had no hand in and is not aware of. This is closely related to the two previous points, i.e. collaborative modelling becomes more of a problem only when the inference rules as well as the formal model are ”sufficiently complex”, so that one ontology engineer has a hard task of keeping an overview of the formal model.

1.2 Structure of this Thesis

Chap. 2 gives an overview of the theoretical foundations on which this thesis builds. First, ontologies and their purpose in various areas of research are described (Chap. 2.1). An introduction to description logics (DL) gives an insight into the formalism that underlies the W3C recommended web ontology language OWL (Chap. 2.2). The ontology engineering process from beginning to end is described in Chap. 2.3, before ontology evaluation is put into focus in Chap. 2.4. The sequence of the four foundational chapters can be regarded as leading step-by-step into the core research area of this thesis, namely evaluating manually created ontologies in DL. This zoom view on the foundational topics ontologies, description logics, ontology engineering and ontology evaluation is depicted in Fig. 1.1. Following this, MoKi, a wiki-based ontology engineering environment, is presented as technical basis for the implementation of ontology evaluation functionalities in the scope of this thesis (Chap. 3). There, the goals of MoKi, the challenges on the way to meeting them and MoKi’s overall design are described.

Chapters 4- 6 contain the contributions that were made by the author in the scope of this thesis. All contributions circle around presenting implications of formal statements (inferences) with the purpose to enable ontology engineers to review whether what they explicitly stated corresponds to what they intended to express. First, the benefits and limitations of reviewing entailed statements as part of an evaluation procedure are studied analytically based on a list of common modelling errors and empirically based on usage of a suitable tool in five application cases (Chap. 4). Second, a particular kind of entailed statements, namely the effects of ontology editing activities on instance data (assertional effects) have been studied from a theoretical point of view (Chap. 5). The purpose of assertional effects is to provide ontology engineers with concrete examples of the implications of logical axioms. Chapter 6 is concerned with the integration of the insights gained from the two theoretical ventures into MoKi as concrete ontology evaluation functionalities. The thesis concludes with a summary of the most relevant insights and a reflection on the contribution of this thesis to the research field of automatic support for ontology evaluation (Chap. 7).
Throughout this thesis, many examples are given to illustrate complex issues. In the foundational chapters (Chaps. 2 and 3), examples are taken from a variety of subjects to underline the broad applicability of what is described. In the contribution chapters (Chaps. 4- 6), examples are consistent throughout each chapter. In Chap. 4, examples are inspired by an ontology that describes core concepts relevant in an Austrian innovation management company. This exemplary ontology was chosen because it is also one of the ontologies used in the experimental study described in Chap. 4, and therefore gives an insight into a practical ontology engineering setting. In Chap. 5, a very small example about Greek islands and philosophers is used to make clear complex logical issues. In Chap. 6, a large ontology about the academic world (publications, university structures, etc.) is used to provide examples. This choice was made in order to use the same example as the demonstration MoKi at [83].

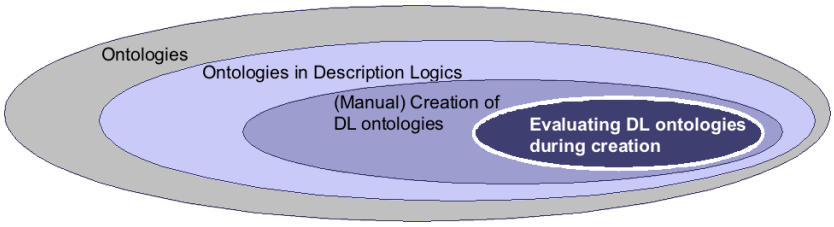


Fig. 1.1. The research field to which this thesis contributes can be narrowed down from a multitude of research fields to the research area of **evaluating DL ontologies during and after manual creation**, and in particular (hence the necessity to focus on a particular formalism) **the usage of reasoning services to automatically support evaluation**.

1.3 Contributions of this Thesis

Chapter 2 represents an overview of ontologies, description logics, ontology engineering and ontology evaluation. The overview is given based on research carried out by

the author in the field of knowledge engineering in collaboration with, mostly, Chiara Ghidini, Barbara Kump, Stefanie Lindstaedt, Marco Rospocher and Luciano Serafini. Selected own publications from the field of knowledge engineering are:

– **Viktoria Pammer**, Barbara Kump, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Stefanie Lindstaedt. Revision Support for Modelling Tasks, Topics and Skills. In *Proceedings of I-SEMANTICS'09, 5th International Conference on Semantic Systems*, pages 501-508, Graz, Austria, September 2-4, 2009.

  This paper discusses ontology evaluation when interpreted as formative evaluation which takes place repeatedly during ontology engineering. An implementation of such a formative evaluation approach in the scope of the Integrated Modelling Methodology, a methodology for modelling tasks, topics and skills, is described. All methodological steps and tools presented in this paper have been applied and evaluated in practice during the APOSDLE project [3, 71][4], and some of them have been integrated in MoKi afterwards (models checklist, ontology questionnaire, see Chap. 6).

– Chiara Ghidini, Marco Rospocher, Luciano Serafini, Barbara Kump, **Viktoria Pammer**, Andreas Faatz, Andreas Zinnen, Joanna Guss and Stefanie Lindstaedt. Collaborative Knowledge Engineering via Semantic MediaWiki. In *Proceedings of the Third International Conference on Semantic Systems (I-Semantics 2008)*, pages 134-141, Graz, Austria, September 3-5, 2008.

  This paper describes early efforts to bridge the gap between knowledge acquisition and formal modelling through the usage of Semantic MediaWiki and lessons learned from this. The work describes a forerunner of MoKi (see Chap. 3).

Chap. 3 presents MoKi, a wiki-based ontology engineering environment that aims to accommodate both informal modelling and formal modelling activities. MoKi has been designed in the scope of the APOSDLE project [3, 71], initially by Chiara Ghidini, Marco Rospocher and Gaetano Calabrese. In Chap. 3 I thus describe work to which I have contributed, but which has been led by others.

– Marco Rospocher, Chiara Ghidini, **Viktoria Pammer**, Luciano Serafini, and Stefanie Lindstaedt. MoKi: the MOdelling WIki. In *Proceedings of the Forth Semantic Wiki Workshop (SemWiki 2009), co-located with 6th European Semantic Web Conference (ESWC 2009)*, volume 464 of *CEUR Workshop Proceedings*, 2009.

  The above is a recent publication which outlines the central concepts of MoKi, although it is by necessity shorter than Chap. 3. For instance, it does not contain detailed descriptions of all model element templates in MoKi and the formal meanings given to fields in them. Also the description of related work in Chap. 3.2 is

[4] APOSDLE (www.aposdle.org) has been partially funded under grant 027023 in the IST work programme of the European Community.

more extensive than in the above publication.

In Chap. 4, the systematic review of entailed statements is studied as part of the ontology evaluation process. The benefits and limitations of this approach have been investigated analytically and through an experimental study. Although inferences and often also explanations for inferences are available in state-of-the-art ontology engineering tools, the procedure of reviewing inferences for ontology evaluation purposes is little studied in literature. This thesis thus adds to state-of-the-art by showing analytically which kinds of modelling errors can be detected through such a procedure, and which not. Moreover, the application of this review procedure in practice indicates the feasibility of this approach, but also the need for better than existing support in removing undesired inferences. On a meta-level, my contribution therefore consists of bringing relevant description logic founded work into the field of knowledge engineering.

– **Viktoria Pammer** and Stefanie Lindstaedt. Ontology Evaluation Through Assessment of Inferred Statements: Study of a Prototypical Implementation of an Ontology Questionnaire for OWL DL Ontologies. In *Knowledge Science, Engineering and Management, Third International Conference, KSEM 2009*, pages 394-405, Vienna, Austria, November 25-27, 2009.

  This paper corresponds closely to Chap. 4, but misses the analysis of benefits and limitations based on common modelling errors.

In Chap. 5, the extension of reasoning services computing logical differences between terminologies (description logic TBoxes) to consider also data is described. While the latter is well-studied under the name of conservative extensions in description logics, the impact of changes in terminology on data has not been studied so far. However, there is certainly merit in doing so: When an ontology is used to define the semantics of data, thus forming a knowledge base, the ontology needs not only to describe correctly a shared view of the world, but more importantly the ontology and the data must both correspond to the same view of the world. Furthermore, presenting the effects of formal statements on concrete data serves to illustrate the general truths by concrete examples. Research in this direction has been targeted foremost at the theoretical side , i.e. on formal definitions of such effects, decision procedures for finding them and on issues that arise when bringing these to practical implementation. (Chap. 5). This part of the thesis provides the basis for supporting evaluation already during formal model authoring as opposed to after formal model authoring in practice.

– **Viktoria Pammer**, Luciano Serafini and Stefanie Lindstaedt. Highlighting Assertional Effects of Ontology Editing Activities in OWL. In *Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD 2009), co-located with the 8th International Semantic Web Conference (ISWC 2009)*, volume 519 of *CEUR Workshop Proceedings*, 2009.

This paper corresponds closely to Chap. 5.

– **Viktoria Pammer**, Peter Scheir and Stefanie Lindstaedt. Ontology Coverage Check: Support for Evaluation in Ontology Engineering. In *Proceedings of FOMI 2006 - 2nd Workshop on Formal Ontologies Meet Industry, December 14-15*, pages 123-134, Trento, Italy, December 14-15, 2006.

  Effects of terminological axioms on data have first been described in this paper. The central idea is that an ontology can be tested through test individuals similar to testing software by exemplary executions of the code, and the concept of what it means to cover an ontology with test individuals is discussed.

In Chap. 6, four validation modules that support ontology evaluation in MoKi are described. Two of these are based directly on the above two ventures, namely the ontology questionnaire that provides inferences to support ontology evaluation in a separate evaluation step (Chap. 4), and the assertional effects functionality that provides effects of knowledge base modifications on individuals to support evaluation directly during authoring (Chap. 5). The other two modules, the models checklist and the quality indicator stem from research done in collaboration with Barbara Kump, Chiara Ghidini and Marco Rospocher [41, 97]. Together, the MoKi validation modules support ontology evaluation both during modelling as well as during a separate evaluation activity. As a consequence, MoKi is now, to the best of my knowledge, the only state-of-the art ontology engineering environment that explicitly supports conceptual ontology evaluation.

2

Foundations

2.1 Ontologies

What is an ontology?

In the field of computer science, an ontology is now commonly understood as an engineering artefact [50] which is a "formal, explicit specification of a shared conceptualisation" [123] of a domain of discourse.

> **Example:** A domain of discourse could be "medicine", "life sciences", "requirements engineering", "everything which a company X needs to know in order to be successful" etc.

Formal essentially means that an ontology is expressed in a machine-interpretable way. This is not meant in the technological sense, i.e. something hand-written can be formal although current character recognition technology would not be able to interpret it. Rather, formal means the language in which the ontology is expressed, i.e. the meaning of the language's symbols and syntactical constructs is clearly defined.

> **Example:** Natural languages are not formal languages, since the meanings of their words are not clearly defined. Meanings change, words (symbols) are ambiguous, etc.

Explicit means that in an ontology the meaning of its elements, such as for instance concepts, relations or individuals, must be described unambiguously and in a definite manner, leaving as little space for interpretation as required/possible.

> **Example:** The word (symbol) "Jaguar" stands for various concepts such as "a wild feline predator" or "a racy car" . If the symbol "Jaguar" is used in an ontology, it must be clear for which concept it stands. Additionally, the concept's relations with other concepts must be made clear, e.g. "Jaguar, the cat" hunts other "Animals" such as "Deer" in order to eat, and it eats only "Animals" as opposed to "Plants".

In practice, a rigorous definition of a concept is often dispensed with if the effort to reach such a definition (e.g. little benefit in definition, a complex or disputed topic) outweighs the benefits of having such a definition. Therefore practical ontologies only

"approximate intended models" [50]. Very often this concerns very common-sense concepts such as "Person" or "Animal" or "Event"[1] [2].

> **Example:** Coming back again to the above introduced ontology about jaguars (the wild cats), it may be that "Animal" is only known to be opposed to "Plants". In such an ontology, nothing would be stated about which properties qualify something as an animal rather than a plant.

A *conceptualisation* is an "intensional semantic structure" [51]. It is a particular, often simplified, view of the world that contains relevant objects and relevant relationships between these objects. A conceptualisation is not an engineering artefact but rather an abstract idea inside a human's mind: It is what we actually "mean" or intend to say. Thus, a conceptualisation is also language-independent, i.e. could be expressed as an ontology in different knowledge representation languages. More formally expressed, a conceptualisation is the set of intended worlds given a particular domain.

> **Example:** Given the particular domain of *persons* and the *parent*-relationship, a typical conceptualisation would be that in every admissible (intended) world, a person has exactly one female and one male person as parents. This is not something which is given a priori by calling the objects "parents" and the relationships "parents", but this is simply the usual meaning. This set of intended worlds is only a subset of all possible worlds. Among all possible worlds there are for instance also worlds in which a person has three parents, another has no parents, and yet another person has two male parents and so on.

Consequently, an engineering artefact such as an ontology typically only approximates a conceptualisation. In rare, simple cases it may be possible to completely capture a conceptualisation in an ontology. An ontology's completeness and precision then depend on how close the set of modelled worlds (worlds which satisfy the explicitly stated ontology) matches the set of intended worlds. It can be said that an ontology is complete when all intended worlds are among the modelled worlds, and that an ontology is more precise as there are less unintended worlds are among the modelled worlds[3]. This is illustrated in Fig. 2.1, and a good discussion of intended versus modelled worlds is

[1] A good practice would be of course to reuse existing formal definitions from so-called upper level ontologies, as e.g. DOLCE [27, 79] or SUMO [87, 124]

[2] This seems to point to a major limitation of symbolic knowledge representation: There are definitely concepts which are hard to define formally. Is then the goal to create intelligent agents achievable when based on such limited representations? In a philosophical sense this is seriously a debatable question. In practice, it seems that representing knowledge symbolically goes a long way to building *more* intelligent systems than exist nowadays. However, none of them will probably win a Turing test as long as they lack inherently human skills such as e.g. understanding natural language. Naturally, the Turing test itself is something which is hotly argued about. For instance, it is debatable whether an artificial intelligence must be undistinguishable from human intelligence in the first place.

[3] This terminology does not as far as I know come up directly in ontology-related literature. Instead, I borrow here from the notions of precision and recall as known in information retrieval and am consistent with the general informal understanding of the terms "completeness" and "precision".

given in [50]. While a conceptualisation underlies essentially all human-created artefacts such as free natural language text or software, the underlying conceptualisation is not always made explicit. When reading a natural language text, it is left to the (human) reader to disambiguate between multiple meanings of words or to bring along the necessary background knowledge required to understand the text. In software, the conceptualisation may be made more explicit, e.g. as UML diagram, through class structures or database schemata. In these instances however, the goal is not to explicate a conceptualisation but to implement a program that does something or a database that holds a certain kind of data and responds well to specific queries. Additionally, factual knowledge is frequently mixed with problem-solving knowledge (e.g. in software programs) and implementation-specific artefacts (e.g. database schemata optimised for specific queries). Nonetheless, for concrete UML diagrams or database schemata etc. it may well be argued that they constitute ontologies.
Shared refers to the fact that an ontology should represent the view of a community on a domain of discourse, or a view to which multiple agents (human or machine) are expected to commit. This part of the definition is motivated partly from the intended usage of ontologies to facilitate knowledge sharing and reuse.

An ontology typically describes concepts, relations between concepts, and sometimes individuals. At a first level, all concepts, relations and individuals that appear in an ontology (the engineering artefact, e.g. the digital file one can open) are represented by symbols. These symbols in turn stand for real-world objects, which in turn evoke some response in an agent (neural activity in case the agent is a human). Individuals are elements in an ontology which indeed refer to things in the real world. Concepts and relationships on the other hand are closer to the German "Begriff" or what Plato called "ideas".

> **Example:** The idea of a house is an ideal or simplification which no real house can ever attain. Nevertheless, all real houses are somehow similar to this ideal.

Concepts and relationships therefore can be thought of as referring, instead of to a single real-world element, to sets of real-world elements.

> **Example:** The concept "house" can thus be understood as a symbol which refers to the set of all things which are houses and which evokes for instance the picture of a stylized house plus some memories of real houses in a human agent.

However, an ontology is not necessarily a complex construct. This has been illustrated in [80] by using the metaphor of "ontology spectrum". Within this spectrum, the degree of detail and the level of logical expressivity rises from "controlled vocabulary" over "taxonomy" to complex logic theories. A glossary for instance represents a controlled vocabulary, and if it is shared by a certain community it may already qualify as ontology. Classification systems for books such as the Dewey Decimal Classification system [24] or online directory systems such as for instance the dmoz Open Directory Project [26] are examples of taxonomies. In general however, an ontology in computer science is understood to contain at least a specialisation hierarchy (is-a hierarchy). This excludes glossaries, many taxonomies and online directories (whenever they are

hierarchically structured but not according to a single hierarchical relation such as subsumption of mereonomy). On the other hand this definition includes for instance a classification schemata of different biological species. Based on a specialisation hierarchy, it is already possible to do simple reasoning. For example, properties can propagate down the specialisation hierarchy. In other fields of computer science, this principle is known under the name "inheritance".

> **Example:** "An Online Store is a Store." expresses that the concept of an "online store" is more specific than the concept "store". If a store sells products, then it can be inferred that an online store also sells products.

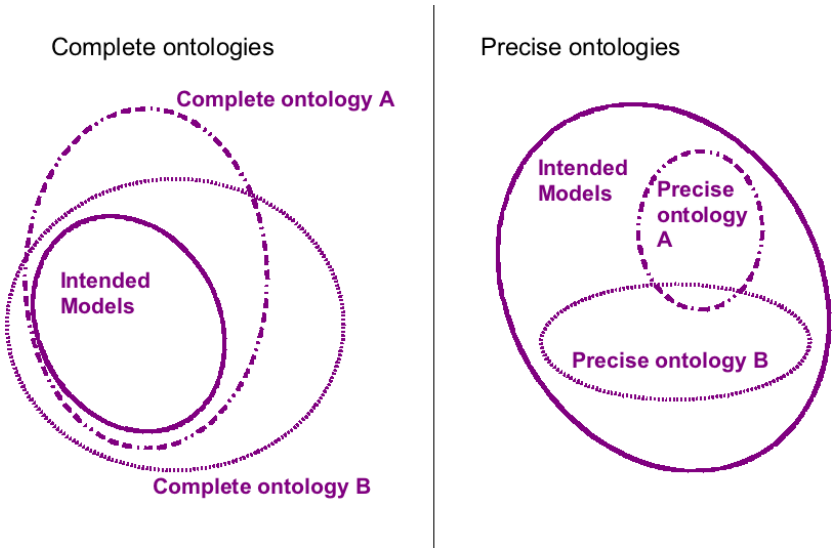


Fig. 2.1. Complete ontologies are satisfied by all intended worlds. The more precise an ontology is, the less unintended worlds satisfy the ontology (the figure shows very precise but incomplete ontologies). A complete and precise ontology is satisfied only and by all intended worlds. The sets labelled "ontology" in the figure are actually sets of worlds that satisfy an ontology.

What are ontologies useful for?

Ontologies are being perceived as useful for different reasons in different research areas which are actually *application areas* from the point of view of ontologies.

In the field of *artificial intelligence*, knowledge representation is studied primarily with the goal to empower machines to act intelligently based on some kind of reasoning over the represented knowledge (see e.g. [122] for some scenarios). As clear-cut as this may sound at first, extremely divergent approaches have been put forward to represent knowledge. For instance, knowledge may be represented as a logic theory, as a set of rules which capture case-based experiences of domain experts, as a set of values and preferences on which decisions can be based, as probabilities or as a complex system as for instance a neural network [23]. Ontologies, because of their rather general nature

of specifying concepts and their interrelations, have been most often represented in frame-based (e.g. F-Logic), graph-based (e.g. semantic networks, topic maps), or logic-based (e.g. description logics) knowledge representation formalisms. In the context of artificial intelligence, an ontology serves to set up a common *vocabulary*, formally defined, based on which discussions between agents can take place.

> **Example:** In this scenario it is assumed that all necessary resources are described in a machine-readable way, so that an artificial agent can process them: Alice organises a trip to Graz. She specifies the desired arrival and departure dates, her preferred way of travel and her preferred type of accommodation together with a price range. Her personal intelligent and automatic travel agent carries out the search for appropriate offers instead of Alice. When the agent searches for accommodation, it finds information about rooms in Bed & Breakfeast establishments, in hotels or in youth hostels. In order to intelligently process these offers, the internal knowledge structure of the agent needs to contain these concepts, so that it can filter out the offers which correspond to types of rooms as desired by Alice.
> Note that in order to rank all offers which satisfy Alice's requirements, Alice's preferences should be taken into account. These preferences and in particular the method for applying the preferences to rank the received offers are typically not part of the ontology, even though it makes sense that the intelligent agent has ways to encode both preferences and a ranking method.

> **Example:** An ontology can define the kinds of objects a robot can expect to find in its environment.

In the field of *database theory*, ontologies are perceived as useful for database design and data modelling to provide an implementation independent specification of the database entities and their relations at knowledge level [50]. Given an ontology, mappings to schemata for different types of databases such as relational, object-oriented etc. can be constructed. In this case, ontologies provide the technical basis for reuse across application or enterprise boundaries. Of course, in order to reuse data across boundaries, applications and enterprises still need to commit to one ontology, which is to a large part not a technical but rather a conceptual difficulty. It is generally agreed upon, that database schemata can not be directly seen as ontologies, since database schemata mostly (i) are not intended to be shareable but rather describe data as required by one specific application or within one specific enterprise setting, (ii) do not give a formal definition of the schema's semantics, (iii) serve to constrain which data are valid, but are not used at query time to generate inferences and (iv) are implementation-specific [121].

> **Example:** An ontology defines an employee as a "person who is employed by a company". A database at a social insurance company $Dest$ represents insured persons in a table which includes the persons' names and their respective employer. A database at company Src represents its employees in a table including the employees' names and their salary. In order for an application *Register-Employees-with-Insurance-Company* at company Src to successfully

register new employees with the social insurance company $Dest$, *Register-Employees-with-Insurance-Company* needs to map the schema of company Src's databases somehow to company $Dest$'s database schema. This is possible if both, company Src and company $Dest$, commit to the same ontology. Note that such a commitment could of course be hardwired into the code of *Register-Employees-with-Insurance-Company*. Hardwiring has the disadvantage that valuable information is not made explicit. So if a database designer at company Src changes for instance the meaning of the column "employee" in company Src's database to hold not the full employees' names but just a String identifier without knowing how *Register-Employees-with-Insurance-Company* exactly works, the wrong data will be registered with the social insurance company $Dest$. If the database schema of Src's database however were conceptually described in an ontology, the database designer would not even need to know about the existence of *Register-Employees-with-Insurance-Company* Thus, an ontology also serves as explicit documentation underlying conceptualisations and consequently rationale underlying complex software systems.

The research field around the vision of the *Semantic Web* is comparatively young. In short, the Semantic Web aims to make Web content accessible to machines by adding formal semantics to the current Web. It is expected that this will significantly reduce efforts and alleviate a variety of activities on which humans nowadays spend a lot of time and efforts, such as searching for information, extracting information or maintaining facts consistent throughout different publication resources [32]. Research around the Semantic Web is strongly influenced both by artificial intelligence and database theory. This becomes clear when one considers how web content can actually be made more accessible to machines: First, by describing content formally, which relates directly to issues around knowledge representation formalisms and the design of agents acting intelligently upon such formalisms. Second, by directly publishing data instead of processed content, which relates immediately to issues of data modelling, data sharing and reuse, and querying over data bases. Semantic descriptions of knowledge on the web are considered in three forms: As general knowledge about some domain of discourse (i.e. upper ontologies, domain specific ontologies), as specific knowledge about instances in the world (i.e. knowledge bases containing facts, in which ontologies provide the means to understand the facts)[4] and as metadata of other content (i.e. semantic annotations whose meaning is defined by underlying ontologies). Thus, the Semantic Web will be not a separate web, but be an extension of the existing Web, and published data and formally described Web content will coexist with conventional Web content. The problem addressed by the Semantic Web is that content on the WWW is laid out for human consumption. A typical HTML page, or even worse (from a machine-understanding point of view) a page with some flash animations etc. can be easily

[4] The boundaries between the first two (general knowledge vs. specific knowledge about instances) sometimes blur: On the one hand in terminology, where an artefact which contains facts is also called an ontology, and on the other hand when an ontology formalisms contains the notion of "nominals". A clearer definition and discussion of these subtleties is given in Section 2.2.

understood by human consumers if designed with a minimum of sensibility. Unfortunately this makes it hard for machines to understand content, and difficult for humans to tell machines in a conceptual manner how to interpret content.

> **Example:** On a typical newspaper website, as for instance on the website of the Austrian newspaper "der Standard", `http://derstandard.at`, a human reader can easily distinguish between the different articles from the type of font used. A machine agent however needs to painfully parse the webpage in order to find out different phrases in different fonts, and is then not guaranteed to find only articles instead of advertisements. Additionally, a machine agent is susceptible to changes in layout of the website. On the Semantic Web, a newspaper could publish its articles directly as data and offer different dynamically generated versions for a variety of user-end-point devices, or the newspaper could publish its articles in a rather traditional HTML page but enriched with some semantic markup to indicate for machine-agents what each HTML/XML element actually means.

The transition from the current WWW to a Semantic Web is already taking place (see e.g. [133]). Languages like XML and XHTML [143] provide the possibility to include content markup through self-defined tags in webpages in addition to layout information. New languages are currently being developed for more semantic markup, i.e. where the content markup is given well-defined semantics through underlying ontologies and by being based on a semantically defined language. An example of such a language is for instance RDFa [1, 107], which defines its own small vocabulary (in the RDFa DTD schema) based on which RDF [78] statements can be embedded in XHTML and unambiguously interpreted. Additionally, there are many websites on the current web whose graphical appearance is already driven directly by data storages, as for instance websites of many online stores. On the Semantic Web, such data could also be published directly as data, interpretable based on an underlying ontology.
A machine-intelligible Web can in many aspects make the Web a better place to be for humans. For instance, information overload is an often-heard term which describes a real problem. It is now widely acknowledged that human agents on the web need better support than what currently exists for finding relevant information. Additionally, complex issues like trust, security and identity-management are currently being attacked at a fairly technical level. By this I mean that the technical prerequisites for trust relationships over the web, and the mathematical foundations of cryptography etc. are being laid down. However, the issue of trust raises more fundamental questions of what trust is, how it works and how it influences people's interactions with (digital) artefacts. Additionally, the challenge of embedding complex technical solutions to trust into easy-to-use user interfaces is still open: In state-of-the art systems, users need to go into a lot of technical details in order to find configurations which suit their needs, e.g. for defining which sources to trust, for signing emails, for choosing between identities etc. In a word, users need to translate their needs into some machine-language instead of telling computers in natural language what they want. While the Semantic Web does not immediately provide natural language understanding (indeed, the current approach to realising a machine-intelligible web is less ambitious than that), current se-

mantic technologies aim to enable humans and machines to communicate at knowledge level [86][5]. Thus, automatic intelligent agents with an internal knowledge representation at knowledge level can not only navigate the web better but also communicate better with humans.

> **Example:** Taking up the above example of articles from newspapers, in the current web a user who wants get an overview over all *articles* from the newspaper *der Standard* from a specific *day X*, the user can not just tell an automatic agent just to search for *articles* from the newspaper *der Standard* published on *day X*. Instead, the user needs to tell the agent the specific web address of *der Standard* and additionally which HTML elements the agent should consider as articles.

Technologies that enable the Semantic Web are called *semantic technologies*. Semantic technologies are prone to change, and the currently proposed semantic technologies are only a proposal of how to achieve the vision of a machine-readable web. To date, semantic technologies encompass most prominently RDF [78], a graph-based language for relating resources, as a quasi-standard on which the Semantic Web will be based as it currently seems. In addition, logic-based knowledge representation formalisms are being used, with OWL 2 [96] being the current W3C recommendation and de facto standard representation language. There are however a plethora of other languages available that could serve as alternatives, or inform a continued development of the current standards. Two examples are topic maps and conceptual graphs. Additionally, it is widely recognised that the current languages are missing the possibility to express and process rules.
Concluding this short excursion into the Semantic Web, I point out that the issue of deciding how to semantically represent knowledge on the Semantic Web, i.e. choosing a suitable knowledge representation formalism, should not be confused with the goals and arguments for the Semantic Web in general. The latter points into the desirable direction, whereas various semantic technologies try to provide solutions. Furthermore, if e.g. the challenge of natural language understanding would be solved, then the whole paradigm of realising the Semantic Web vision through formal representation of knowledge would need to be revisited.

In addition to the above stated uses for ontologies like representing knowledge on which machines can act intelligently (AI perspective), providing implementation-independent data model specifications (database theory perspective) which can facilitate data sharing, representing web content in a machine-readable way in order to support intelligent automatic web agents and applications and to support data sharing on the web (Semantic Web perspective), some other uses can be argued for ontologies.

[5] In [86], knowledge level is informally defined as a level of a system description. At this level of system description, every system is an agent that has goals, is capable performing actions that may or may not lead to goals and has a body that exists in the real world and that actually performs these actions. When communicating at knowledge level, humans could tell machines what goals should be achieved, and ideally machines would know of or find actions that lead to achieving them. Similarly, machines could explain their results to humans in terms of the goal that should have been achieved and explain why they "believed" that certain actions would lead to the achievement of goals.

Ontologies as explicit description of knowledge can be used to document the shared understanding of a community of people, to facilitate knowledge reuse (think e.g. of patterns) or finally can also serve as a tool for conceptual analysis, e.g. to analyse a domain and separate domain knowledge from operational knowledge [89].

Since I consider in this thesis solely application scenarios in the context of the Semantic Web, I restrict discussions from now on to ontologies represented in the knowledge representation formalism of *description logics*. This is because the proposed standard ontology language for the web, OWL 2, is based on a specific description logic, namely $\mathcal{SROIQ}$. In this sense I consider an ontology as a logical theory which should capture the conceptualisation of its creator(s) of a particular domain. Thus, I keep important properties which actually define what an ontology is, such as "shared", "typically describing factual knowledge rather than problem-solving knowledge" in mind only as background on which I work. This view on ontologies as engineering artefacts expressed in description logics then also illuminates clearly the key motivation for this work: While machines deal well with logic theories, humans do less so, and yet humans have interest in providing formal descriptions of their knowledge in the form of ontologies.

2.2 Description Logics

Description logics (DL) are a family of logics which are usually decidable and most often but not always fragments of first-order logics[6]. In the context of the Semantic Web, important description logic languages are $\mathcal{SHOIN}$, on which OWL 1 DL [10] is based, and $\mathcal{SROIQ}$, on which OWL 2 DL [96] is based.
Although this chapter contains a complete introduction into description logics and gives all language constructs that appear in $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$, I nevertheless refer the interested reader to [8] for a thorough introduction to description logics. More specific references are of course given in the text below. The notation used for DL in this chapter and throughout the rest of this thesis follow the standard notation for DL, which is also used e.g. in [8].

2.2.1 Description Logic Basics and $\mathcal{ALC}$

Vocabulary and DL Elements

Each DL representation uses a vocabulary Σ which consists of a set of concept names N_C, a set of role names N_R and a set of individual names N_I. These three sets are required to be disjoint[7].

[6] Modern description logics for instance often contain the notion of datatypes, which is not exactly part of first-order logics.

[7] In OWL 2 a feature called *punning* has been introduced. It allows an ontology engineer to use a single name (URI) for e.g. a concept and a role. This is a technical construct however: When interpreting an OWL 2 ontology, one must interpret the concept and the role as two different entities.

Example: $Island, Sea \in N_C$, $locatedIn, hasInhabitant \in N_R$ and $crete, mediterranean \in N_I$ are exemplary concept, role and individual names when one wants to talk about geographic knowledge. Note that the names themselves do not carry formal meaning but for usability of an ontology it makes sense to give concepts, roles and individual meaningful names.

In DL languages, *complex concepts* and *complex roles* are inductively defined by a set of *constructors* operating on the concept, role and possibly individual names of Σ. Different description logic languages are distinguished by the set of constructors and properties of elements they provide. The set of complex concepts therefore depends on the chosen description language $\mathcal{DL}$ and the chosen vocabulary Σ and is abbreviated $\mathcal{C}(\Sigma, \mathcal{DL})$. Concepts (resp. roles) who are described just by a concept (resp. role) name are also called *primitive* or *named concepts* (resp. roles). Usually I will use $A, B \in N_C$ to denote primitive concepts, $C, D \in \mathcal{C}(\Sigma, \mathcal{DL})$ to denote possibly complex concepts, $R, S \in N_R$ to denote roles and lower-case letters $a, b, x, y \in N_I$ to denote individuals. Although the formal definition of various constructors will only be defined later on, the following example already illustrates some, thus clarifying the difference between primitive and complex concepts.

Example: The concept $UninhabitedIsland$ can be defined by using the primitive concepts $Island$ and $Person$ and the primitive role $hasInhabitant$ as $UninhabitedIsland \doteq Island \sqcap \neg\exists hasInhabitant.Person$, i.e. an uninhabited island is an island which is not inhabited by any person. If $UninhabitedIsland$ should denote the sets of islands on which nothing at all (not even animals, say) lives, the definition could look like

$$UninhabitedIsland \doteq Island \sqcap hasInhabitant.\bot$$

This means that it is impossible (logically inconsistent) that something or someone inhabits this island. In both cases, $UninhabitedIsland$ is a complex concept.

Concepts can be seen as unary predicates, such that each concept describes a set of entities for which the unary predicate is true. Roles can be seen as binary predicates, such that each role describes a set of entity-pairs for which the binary predicate is true. Figure 2.2 illustrates this model-theoretic semantics of concepts and roles (see also below for interpretations and models). N-ary predicates are usually not built-in in DL languages but can be expressed based on unary and binary predicates (see e.g. [90]).

Example: "Alice travelled from Athens to Crete in summer 2009" is a 4-ary relation. One cannot simply separate this sentence into binary relations by saying "Alice started her travel in Crete", "Alice travelled to Crete" and "Alice travelled in summer 2009". What one would like to do is write $travel$ as a 4-ary predicate e.g. as $travel(alice, athens, crete, summer2009)$. One frequently used workaround is to formulate the predicate $travel$ as a concept $Travel$ and assign different positions in the 4-ary predicate $travel$ to different relations like for instance $participant$, $duration$, $startingPoint$ and $visitedLocation$. A reformulation of the above statement in unary and

binary predicates could then read for instance as $Travel(alicesHoliday)$, $participant(alicesHoliday, Alice)$, $startingPoint(alicesHoliday, athens)$ and $visitedLocation(alicesHoliday, crete)$.

The Basic DL $\mathcal{ALC}$

The DL language $\mathcal{ALC}$[8] knows the concept constructors shown in Table 2.1, namely complex negation $\neg$ (both primitive and complex concepts can be negated), conjunction $C \sqcap D$ and the universal restriction $\forall R.C$. $\top$ denotes a tautology and is a valid $\mathcal{ALC}$ concept, together with all concepts $A \in N_C$ and all concepts that can be constructed using the above-mentioned concept constructors. The above given list of constructors is minimal: In addition, the following expressions are used as abbreviations. $\bot$ is used to abbreviate $\neg\top$, i.e. it denotes a contradiction. $C \sqcup D$ abbreviates $\neg(\neg C \sqcap \neg D)$, i.e. it denotes disjunction. $\exists R.C$ abbreviates $\neg\forall R.\neg C$, i.e. it denotes existential restriction. So $\mathcal{ALC}$ contains all boolean constructors plus qualified universal and existential quantification. $\mathcal{ALC}$ provides the basis for a lot of more expressive DL languages. All discussions in this work consider languages which at least contain $\mathcal{ALC}$.

Logical Axioms

Throughout this work the term *statement* is used interchangeably with *axiom*, but the first is used rather when talking about ontology engineering issues and the latter is used rather when talking about logic/theoretical issues.

Terminological axioms describe relation between concepts. A concept inclusion axiom is an expression of the form $C \sqsubseteq D$ and is often expressed verbally as "D subsumes C" or "C is a subclass of D". A concept equality axiom is an expression of the form $C \doteq D$ and can be seen as abbreviation for writing the two inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. A set of terminological axioms constitutes a *terminological box*, the so called TBox. *Role axioms* describe relations between roles. In analogy to terminological axioms, there are role inclusion axioms and role equality axioms. Additionally, roles may have properties like functional, transitive, reflexive, irreflexive or antisymmetric, and it may be possible to define roles as inverses of each other depending on the expressivity of the chosen DL language. Such statements about properties of roles are also sometimes called "role assertions", e.g. in [57]. I will call them *role property axioms* instead in order not to mix them up with role assertions as defined further below. A set of role axioms (role inclusion, equality and property axioms) constitutes a *role box*, the so called RBox. Terminological and role axioms express general truths within the domain of discourse. Note that $\mathcal{ALC}$ does not know role axioms.

Example: $Sea \sqsubseteq WaterArea$ means that every sea is a water-area, and thus is more specific. This allows stating common properties for everything which is a water area, as are ponds, rivers etc. For instance, one can state that "All fish live in water areas".
$locatedIn \sqsubseteq partlyLocatedIn$ means that if something is (completely) located somewhere, then it is also partly located there. This describes a certain view on what completely located and partly located means.

[8] Attributive Language with complex negation.

Assertional axioms describe knowledge about individuals. A *concept (resp. role) assertion* is a statement of the form $C(x)$ resp. $R(x, y)$. These are often expressed verbally as "x is of type C" or "x is an instance of C" respectively "x is related to y via R". Concept and role assertions constitute an *assertional box*, the so called ABox. Assertional axioms express truths about specific individual entities. Note that $\mathcal{ALC}$ does not allow equality and inequality assertions.

> **Example:** Assertional axioms about Crete and the Mediterranean Sea are for instance $Island(crete)$ and $locatedIn(crete, mediterranean)$. The first states that every "Crete is an island" while the second states that "Crete is located in the Mediterranean Sea". Subscribing again to the set-view of concepts and roles, this means that $crete$ belongs to the set of all islands, and $(crete, mediterranean)$ is an entity-pair in the set of all things where the first is located in the second (see Figure 2.2).

An overview over terminological, role and assertional axioms is given in Table 2.2, where already a number of axioms which are not available in $\mathcal{ALC}$ but available in $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$ are listed.
By *ontology* a TBox and an RBox $(\mathcal{T}, \mathcal{R})$ is understood, while by *knowledge base* a TBox, an RBox and an ABox $\mathbf{KB} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ is meant. From this point of view, an ontology (resp. a knowledge base) is simply a logical theory, in contrast to the more philosophical definition of an ontology being a "formal, explicit specification of a shared conceptualisation" given above (Chap. 2.1). In the course of this thesis, I will use the term ontology when it is clear from the context that I speak about a logical theory that does not reference individual entities. I will use the term knowledge base to indicate that the logical theory references individual entities, or in general to indicate that the logical theory may as well reference individual entities. In particular with very expressive DL languages such as $\mathcal{SHOIN}$ or $\mathcal{SROIQ}$ as described below, the borders between ontologies and knowledge bases begin to blur as individuals can become part of terminological axioms.

Semantics of Description Logics

So far, the basic syntactical elements of DLs, in particular of $\mathcal{ALC}$, have been discussed, and with the knowledge described above one should be able to construct syntactically valid $\mathcal{ALC}$ concepts and logical axioms. What is missing so far, is how to interpret such concepts and axioms, in a word the semantics (or meaning) is missing.

An *interpretation* $\mathcal{I}$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the domain of interpretation, and an interpretation function that assigns to every primitive concept $A \in \mathcal{P}$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every primitive role $R \in \mathcal{R}$ a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to any instance $x \in \mathcal{X}$ an element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation satisfies an inclusion axiom $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, a concept assertion $C(x)$ iff $x^{\mathcal{I}} \in C^{\mathcal{I}}$ and a role assertion $R(x, y)$ iff $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$. Note that the domain of interpretation can be infinite. An interpretation that satisfies a TBox $\mathcal{T}$ (resp. an RBox $\mathcal{R}$ or an ABox $\mathcal{A}$) is said to be a *model of* $\mathcal{T}$ (resp. of $\mathcal{R}$ or of $\mathcal{A}$).

> **Example:** One possible model for $Island \sqsubseteq WaterArea$ is $\Delta^{\mathcal{I}} = \{1, 2\}$ and $Island^{\mathcal{I}} = \{1\}$ and $WaterArea^{\mathcal{I}} = \{1, 2\}$. The following is for instance not

a model of $Island \sqsubseteq WaterArea$: $\Delta^{\mathcal{I}} = \{1, 2\}$ and $Island^{\mathcal{I}} = \{1\}$ and $WaterArea^{\mathcal{I}} = \{2\}$, since not everything which is an island (1) is interpreted as water area ($WaterArea^{\mathcal{I}}$ does not contain the element 1).

The domain of interpretation can be understood as the set of all "real world" objects that are described by the logical axioms in a knowledge base. It is important to remember here that the logical axioms in a knowledge base use a given vocabulary, and only the mapping from concept, role and individual names (symbols) to objects in some world gives these statements a meaning. In the end it does not matter then whether the described world is the real world or an imaginary world.
An ontology (resp. knowledge base) is satisfiable iff there is an interpretation that satisfies all its axioms, i.e. if it has a model. An ontology (resp. knowledge base) entails an axiom α if and only if all its models satisfy α. This is written as $\mathcal{T} \models \alpha$ resp. $\mathbf{KB} \models \alpha$. In practice, most ontologies and knowledge bases have multiple models, and indeed this is most often the desired behaviour.

Example: An ontology that defines $Island \sqsubseteq WaterArea$ has infinitely many models. For instance, in one world there is maybe only a single island $island_1 \in \Delta^{\mathcal{I}}$ which is element both of $Island^{\mathcal{I}}$ and $WaterArea^{\mathcal{I}}$. In another world there are two islands, $island_1, island_2 \in \Delta^{\mathcal{I}}$ and both of them are elements of both $Island^{\mathcal{I}}$ and $WaterArea^{\mathcal{I}}$. This is the intended behaviour - with the inclusion axiom $Island \sqsubseteq WaterArea$ one after all wants to express that in every conceivable world, whatever is an island is also an area of water, no matter how many islands exist.

As already mentioned above however (see Fig. 2.1), it may happen that not all of the models that satisfy a knowledge base correspond to intended models.

Open World Assumption and the Unique Name Assumption

Two distinguishing features of description logics are the open world assumption (OWA) and the non-Unique Name Assumption (UNA). The first means that for reasoning one must assume that the available information may be incomplete. The second means that for reasoning it is not assumed that two names denote different entities just because the names are different. These two issues are inherently linked together, but I start by explaining the Open World Assumption.

Example: Assume that $likes(alice, bob)$ and $Female(alice), Male(bob)$ is known, and that females and males are disjoint. Then following the OWA, $\forall likes.Male(alice)$, i.e. that Alice likes only male persons, is not entailed, since there could be a female whom Alice likes about which the knowledge base has no knowledge.

Consequently, one can never conclude from data alone a universal restriction.

Example: Assume that $likes(alice, bob)$ is known. In a closed world, one could then conclude that $\forall likes.Male(alice)$, i.e. Alice likes only male persons. In an open world, there may be other people who Alice likes and which are female.

Clearly, this feature is particularly well suited to a web environment where, in contrast to a closed environment, one cannot assume to have all existing data available. Closed world environments are for instance local database applications, such as e.g. information about train schedules by one train company. Already in data exchange it may be necessary to switch to the open world assumption, for instance if not all fields in a target database can be filled given the data available in a source database [70].
Under the Unique Name Assumption (UNA), a reasoner can assume that if it encounters two different symbols, such as $crete$ and kos, that these denote two different entities. This means that in every model which satisfies a knowledge base, two different elements of the domain of interpretation must be used to interpret $crete$ and kos. If the UNA is not made, a reasoner can make no prior assumption about the (in)equality of entities: It could be that $crete$ and kos denote two different entities, but it could also be that they denote the same entity. In languages that do not make the UNA, it is typically possible to explicitly assert equality, such as $crete = kos$, or inequality, such as $crete \neq kos$, between entities[9]. The non-adoption of the UNA is also very well suited to a web environment where one cannot assume that everyone will give the same names to the same things.
The difference between the closed world and the open world assumption and between making the unique name assumption and not making it, is really a shift of paradigm similar to the shift between data retrieval and information retrieval. It marks a transition from a (web) environment in which a few people control a limited amount of information to an open environment with many people contribute content ("produsers") in a very heterogeneous and often incomplete manner. Unfortunately, reasoning under the open world assumption minus the unique name assumption is sometimes perceived as counterintuitive by users, who have grown up in a digital world of closed-world-reasoning applications based on unique names.

> **Example:** If a source, such as a website, lists all neighbouring countries of Greece following the pattern $isNeighbourOf(greece, turkey)$ etc. the statement $isNeighbourOf(greece, austria)$ can not be identified as wrong. As far as any reasoner knows, the information from the source could simply be incomplete.

This counterintuitive reasoning (the absence of an inference that seems "logic" to most people in the example above) that follows from the open world assumption is aggravated by the fact that the Unique Name Assumption is not made in the available Semantic Web languages RDF(S) and OWL.

[9] Note that (in)equality can only be asserted between individuals. Assume that one wants to state that the concepts $Islander$ and $Icelander$ are really different concepts. This is not directly possible in most description logic languages, as long as concept names cannot be used as individuals. It is also not directly possible in most DL languages to express "inequivalence" between concepts, i.e. that the can share individuals, but need not share all. However, it is possible to express disjointness, i.e. that two concepts can not share any domain individuals that interpret them, and equivalence, i.e. that two concepts are interpreted always by the same domain elements.

Example: A less than ideal solution, but "obvious" at first glance, is to use a negative role assertion such as $\neg isNeighbourOf(greece, austria)$. This solution has two disadvantages however: First, from a practical viewpoint, many DL languages do not support negated role assertions. For instance, in the web ontology language OWL negated role assertions have only been introduced very recently with the OWL 2 specification. Second, even if it is possible to assert $\neg isNeighbourOf(greece, austria)$, this negative assertion needs to be made for all countries that are not neighbours of Greece. This is unfortunate if there are more countries that are not neighbours of greece than there are neighbours. It is also conceptually unfortunate, since if any new country comes into being (e.g. when a country splits like Yugoslavia, the UdSSR etc.) some new assertions need to be made about these countries relating them negatively go Greece. This is counterintuitive, since if Slovenia has nothing to do with Greece, a user would probably also not think of explicitly stating that Slovenia is not a neighbour of Greece.

Example: A good way to solve the issue addressed by the example above is to model a concept $NeighbourOfGreece$, and describe it as an exhaustive set of different individuals[108]. First, it must be stated for all involved country names that they denote different real world entities[10]. Then, the concept $NeighbourOfGreece$ would be defined as the disjunction of all neighbouring countries, i.e. $NeighbourOfGreece \doteq (albany \sqcup \ldots \sqcup turkey)$. Note that such an expression requires a DL language which contains nominals, and thus is already fairly expressive. In OWL, $NeighbourOfGreece$ would be represented as an enumeration class, which has the same meaning as the above-described disjunction.

Name	Syntax	Semantics
Universal concept	$\top$	$\Delta^{\mathcal{I}}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Universal restriction	$\forall R.C$	$\{v \in \Delta^{\mathcal{I}} \mid (v,w) \in R^{\mathcal{I}} \rightarrow w \in C^{\mathcal{I}}\}$

Table 2.1. Syntax and semantics of $\mathcal{ALC}$ concepts.

2.2.2 $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$

$\mathcal{SHOIN}$ extends $\mathcal{ALC}$ with role hierarchies, i.e. role inclusion and hence also role equality, inverse and transitive roles, cardinality restrictions and nominals (see Tables 2.2 and 2.3 for an overview, [58] for a detailed description of an only slightly

[10] This is necessary because the UNA is not made. Without stating that the different country-names denote different entities, a reasoner would again not detect an inconsistency with $NeighbourOfGreece(austria)$. Instead, it would simply assume that "austria" is another name for one of the neighbouring countries of Greece.

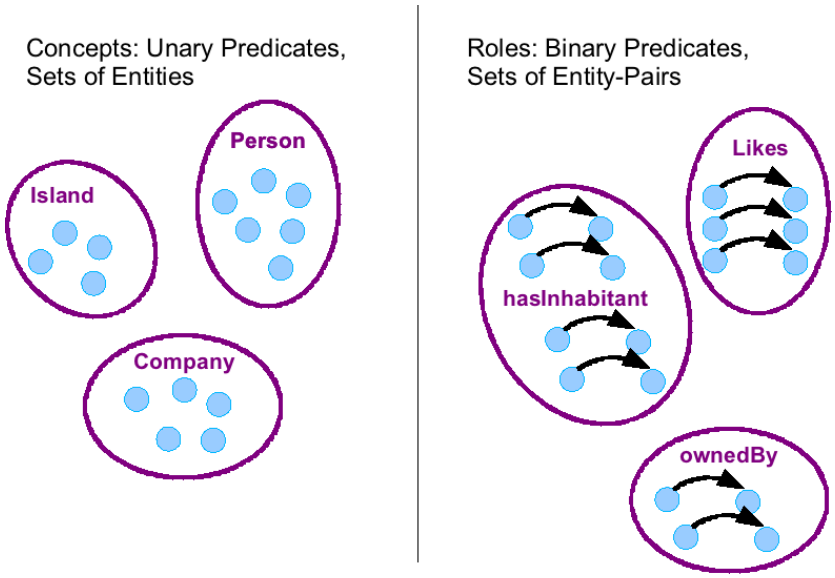


Fig. 2.2. Concepts are unary predicates and interpreted as sets of entities. Roles are binary predicates and interpreted as sets of entity-pairs. Typically, for concept names the singular is used, i.e. the concept $Island$, although it denotes a set of many islands, is named in singular.

more expressive logic, and [10] for the corresponding constructs in OWL 1). Since no role constructors are provided in $\mathcal{SHOIN}$, only very simple role hierarchies can be expressed.

A functional role allows only one value per individual, i.e. if an individual a relates to two individuals b and c via a functional role R, then b and c must be the same individual. Functional roles are often also called features. The inverse of a functional role is inverse functional.

> **Example:** The role $hasCapital$ is functional since per definition a country has exactly one capital.

> **Example:** The inverse role $isCapitalOf = hasCapital^{-}$ is inverse functional, i.e. everything can have at most one "incoming" relation of type $isCapitalOf$.

A transitive role describes a relation that given $R(a, b)$ and $R(b, c)$ entails $R(a, c)$. Typically, ordering relations such as $\leq, \geq, < . >$ and derived ordering relations based on numeric values such as "has more inhabitants" or "is heavier than" are transitive.

> **Example:** If $locatedIn(Heraklion, crete)$ and $locatedIn(crete, greece)$ then $locatedIn(Heraklion, greece)$ can be inferred if $locatedIn$ is a transitvite role.

A cardinality restriction describes individuals who appear in a certain number of (or in more or less than a certain number of) relations via a specified role R. In addition to $\leq nR$ (shown in Table 2.3), $\geq nR$ abbreviates $\neg \leq (n-1)R$ and $= nR$ abbreviates $\leq nR \sqcap \geq nR$. Functional roles can be expressed using a number restriction as $\top \sqsubseteq \leq$

Name	Syntax	Semantics
Terminological Axioms		
Concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subset D^{\mathcal{I}}$
Concept equality	$C \doteq D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Disjointness	$C \sqcap D \sqsubseteq \bot$	$C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$
Role Axioms		
Role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subset S^{\mathcal{I}}$
Role equality	$R \doteq S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
Role disjointness	$Dis(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
Inverse role	$R = S^{-}$	$R^{\mathcal{I}} = \{(u, v) \in \Delta \times \Delta \mid (v, u) \in S^{\mathcal{I}}\}$
Functional role	$Func(R)$	$\forall u, v, w \in \Delta^{\mathcal{I}} : (u, v) \in R^{\mathcal{I}} \wedge (u, w) \in R^{\mathcal{I}} \rightarrow v = w$
Inverse functional role	$InvFunc(R)$	$\forall u, v, w \in \Delta^{\mathcal{I}} : (v, u) \in R^{\mathcal{I}} \wedge (w, u) \in R^{\mathcal{I}} \rightarrow v = w$
Symmetric role	$Sym(R)$	$(u, v) \in R^{\mathcal{I}} \rightarrow (v, u) \in R^{\mathcal{I}}$
Antisymmetric role	$Asym(R)$	$(u, v) \in R^{\mathcal{I}} \rightarrow (v, u) \notin R^{\mathcal{I}}$
Reflexive role	$Ref(R)$	$\{(u, u) \mid u \in \Delta^{\mathcal{I}}\} \subset R^{\mathcal{I}}$
Irreflexive role	$Irr(R)$	$\{(u, u) \mid u \in \Delta^{\mathcal{I}}\} \notin R^{\mathcal{I}}$
Transitive role	$Trans(R)$	$\forall u, v, w \in \Delta^{\mathcal{I}} : (u, v), (v, w) \in R^{\mathcal{I}} \rightarrow (u, w) \in R^{\mathcal{I}}$
Assertional Axioms		
Concept assertion	$C(x)$	$x^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertion	$R(x, y)$	$(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$
Equality assertion	$x = y$	$x^{\mathcal{I}} = y^{\mathcal{I}}$
Inequality assertion	$x \neq y$	$x^{\mathcal{I}} \neq y^{\mathcal{I}}$

Table 2.2. Terminological, role and assertional axioms. Not all description logics support all kinds of axioms. In particular not all DL languages support role axioms. Role disjointness must be expressed as a separate binary operator as long as boolean role constructors are not available in the DL language of choice. This is the case for most currently available DL languages. There is no standard syntax for defining roles as functional, inverse functional etc. for description logics in general. Functionality and inverse functionality can be expressed as $\top \sqsubseteq \leq 1.R$ and $\top \sqsubseteq \leq 1.R^{-}$ respectively in DL languages that support number restrictions.

$1R$, i.e. wherever the role R occurs, there can be at most one value for any individual. In analogy, inverse functional roles can be expressed as $\top \sqsubseteq \leq 1R^{-}$.

> **Example:** $\leq 3\ hasNeighbourCountry$ is the set of all countries that have at most three neighbouring countries. $= 3\ hasNeighbourCountry$ is the set of all countries that have exactly three neighbouring countries.
> $\geq 3\ hasNeighbourCountry$ is the set of all countries that have more than or equal to three neighbouring countries.

A nominal $o \in N_O$ is a concept whose interpretation set has exactly one element. Nominals can also be seen as individuals that are used like concepts. Where nominals are allowed in a DL language, the sets N_I and N_C are therefore not disjoint since $N_O \subseteq N_I$ and $N_O \subseteq N_C$ and consequently $N_I \cap N_C = N_O$.

> **Example:** Let $christianGod$ be the individual denoting the god Christians believe in, and $believesIn$ the role that models in which god a person believes in. Then one can define a Christian as follows: $Christian \doteq \ldots \sqcap \exists believesIn.\{christianGod\} \sqcap \ldots$. The dots express that there may be also

other requirements on being a Christian. The existential restriction expresses that every Christian believes in $christianGod$. Given this definition it is not excluded that a Christian believes also in other things, such as e.g. the Holy Mary or something else entirely.

In addition, the availability of inverse roles in $\mathcal{SHOIN}$ allows expressing both domain and range restrictions on roles. Although these are seen as restrictions on roles for practical purposes in OWL, both actually can be encoded as terminological axioms and are consequently not listed among role axioms, nor as a specific kind of terminological axioms. A domain restrictions forces each subject of a role to be of a certain type and corresponds to the terminological axiom $\exists R.\top \sqsubseteq C$. A range restriction forces each object of a role to be of a certain type and corresponds to the terminological axiom $\exists R^{-}.\top \sqsubseteq C$.

Example: The domain of $isCapitalOf$ can be restricted to cities, and its range can be restricted to countries. This is encoded as the two terminological axioms $\exists isCapitalOf.\top \sqsubseteq City$ and $\exists isCapitalOf^{-}.Country$.

$\mathcal{SROIQ}$ further adds a lot of expressivity regarding roles. Roles can now be defined as reflexive, irreflexive, symmetric and antisymmetric. Furthermore, the universal role U which subsumes all other roles, role composition $R \circ S$ and negative role assertions $\neg R(x, y)$ are added. As concerns concept constructors, cardinality restrictions are now enhanced to qualified cardinality restrictions ($\leq nR.C$) and the Self-Constructor ($\exists R.Self$) is added. (see Table 2.4 for an overview of $\mathcal{SROIQ}$ language constructs, [57] for a detailed description of $\mathcal{SROIQ}$ plus a tableaux-based decision procedure, and [96] for the corresponding constructs in OWL 2). Some restrictions as to which roles can appear in role inclusion axioms and in role property assertions apply. In short, role inclusion axioms must not lead to a cyclic role box (called a "regular" role hierarchy), and role property axioms are only allowed on "simple" roles (see Tables 2.5 and 2.6 or [57] for a detailed definition of simple roles and regular role hierarchies).

Qualified cardinality restrictions generalise both unqualified cardinality restrictions as available e.g. in $\mathcal{SHOIN}$ and quantified restrictions such as the universal and the existential restriction available from $\mathcal{ALC}$ on. $\leq nR$ thus becomes an abbreviation of $\leq nR.\top$, $\exists R.C$ an abbreviation for $\geq 1R.C$ and $\forall R.C$ an abbreviation of $= 0R.\neg C$. The $Self$-constructor describes sets of individuals that are all related to themselves via a specific relation. Role composition means chaining together roles. Role composition in conjunction with role hierarchies can also be used to express transitivity. Thus the role property assertion $Trans(R)$ can also be expressed as the role inclusion axiom $R \circ R \sqsubseteq R$.

Example: I give here two typical examples for the $Self$-constructor. The first is $Suicide = \exists kills.Self$, i.e. suicides are people who kill themselves. On a more joyful note, $Narcissist = \exists loves.Self$ i.e. narcissists are people who love themselves. The latter example does have some philosophical difficulties however, since the question arises whether everyone who loves him- or herself is already a narcissist.

Example: Through role composition one can express that if a person is citizen of a city, and this city belongs to a country, then the person in question is also a citizen of the country. $isCitizenOf \circ isGeopoliticalPartOf \sqsubseteq isCitizenOf$ expresses the general truth that if a person is a citizen of something which in turn geopolitically belongs to something else, then the person is also a citizen of the "something else"

Example: Role composition can also be used to address the traditional example of why rules are necessary in addition to description logics[11]: $isDaughterOf \circ isSisterOf \sqsubseteq isNieceOf$ expresses that if A is the daughter of B and B is the sister of C, then A is the niece of C.

Name	Syntax	Semantics
Cardinality restriction	$\leq nR$	$\{v \in \Delta^{\mathcal{I}} \mid \#\{w \mid (v,w) \in R^{\mathcal{I}}\} \leq n\}$
Nominals	$o \in N_O$	$\lvert o^{\mathcal{I}} \rvert = 1$

Table 2.3. Syntax and semantics of $\mathcal{SHOIN}$ concepts, only constructs which extend $\mathcal{ALC}$ are shown.

Name	Syntax	Semantics
Qualified cardinality restriction	$\leq nR.C$	$\{v \in \Delta^{\mathcal{I}} \mid \#\{w \mid (v,w) \in R^{\mathcal{I}} \wedge w \in C^{\mathcal{I}}\} \leq n\}$
$Self$-constructor	$\exists R.Self$	$\{v \in \Delta^{\mathcal{I}} \mid (v,v) \in R^{\mathcal{I}}\}$
Universal role	U	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Role composition	$R \circ S$	$\{(u,w) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (u,v) \in R^{\mathcal{I}} \wedge (v,w) \in S^{\mathcal{I}}\}$

Table 2.4. Syntax and semantics of $\mathcal{SROIQ}$ concepts and roles, only constructs which extend $\mathcal{SHOIN}$ are shown.

2.2.3 Standard Reasoning Problems

One of the goals of a knowledge representation (KR) formalism is to provide inference mechanisms. Typically, together with a knowledge representation formalism comes a specification of what can be inferred from explicitly stated ("already known") knowledge [23]. In description *logics*, as the name already suggests, the chosen inference mechanism is logical deduction based on the semantics given to a DL language.

[11] There are still other examples where rules might be necessary. In general, variables can be used in rules, and thus more complex logical formulae are possible. One simple example is "If $Country(x)$ and $Country(y)$ and $numberInhabitants(x, xNum)$ and $numberInhabitants(y, yNum)$ and $xNum > yNum$ then $largerThan(x, y)$", i.e. if country x has more inhabitants than y, we want to infer that x is larger than y. This can not be expressed in description logic languages. Finally, operational rules (actions triggered by events) can not directly be formulated in logic in general, where the sequence in which axioms are considered does not matter [54].

Given a role box $\mathcal{R}$, a role R is simple iff: – R does not occur on the right hand side of a role inclusion axiom, or – R^- is simple, or – $S \sqsubseteq R \in \mathcal{R}$ and S is a simple role A role box $\mathcal{R}$ is simple iff all roles that appear in role property axioms are simple.

Table 2.5. Simple roles in $\mathcal{SROIQ}$. Role property axioms in $\mathcal{SROIQ}$ can be asserted on simple roles only.

Let $\prec$ be a regular order on roles. A role inclusion axiom $w \sqsubseteq R$ is $\prec$-regular iff R is a role name, i.e. a primitive role, and: – $w = RR$, or – $w = R^-$, or – $w = S_1 \circ \ldots \circ S_n$ and $S_i \prec R$ for all $1 \leq i \leq n$, or – $w = R \circ S_1 \circ \ldots \circ S_n$ and $S_i \prec R$ for all $1 \leq i \leq n$, or – $w = S_1 \circ \ldots \circ S_n \circ R$ and $S_i \prec R$ for all $1 \leq i \leq n$ A role box $\mathcal{R}$ is regular iff an order $\prec$ exists such that all role inclusion axioms in $\mathcal{R}$ are regular.

Table 2.6. A regular role hierarchy in $\mathcal{SROIQ}$. A role hierarchy in $\mathcal{SROIQ}$ must be regular.

The existence of software tools that automatically compute inferences is extremely important for knowledge engineers who want to use a specific KR formalism. In the case of description logics, a number of DL reasoners, tools that perform logical inferences based on a DL knowledge base, are available. I list here the most prominent: Pellet [99] is an open-source Java reasoner, Fact++ [30] an open source C++ reasoner, and Racer [105] is a commercial reasoner.

Satisfiability

The notion of knowledge base satisfiability has already briefly been defined above but is reviewed here in some more detail. A concept C is satisfiable iff there is an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}}$ is non-empty, i.e. C is non-contradictory.

> **Example:** The concept $Island \sqcap \neg Island$ is unsatisfiable since nothing can be both, an $Island$ and not an $Island$. Formally, this means that no domain element $w \in \Delta^{\mathcal{I}}$ can be both in $Island^{\mathcal{I}}$ and $(\neg Island)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus Island^{\mathcal{I}}$.

A knowledge base $\mathbf{KB}$ is satisfiable iff it has a model, i.e. there is an interpretation $\mathcal{I}$ that satisfies $\mathbf{KB}$. Sometimes this is also called consistency.

> **Example:** A knowledge base $\mathbf{KB} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ with $\mathcal{T} = \{C \doteq Island \sqcap \neg Island\}$ and $\mathcal{A} = \{IslandA(crete), \neg Island(greece)\}$ is satisfiable. In one possible interpretation $\mathcal{I}$, $C^{\mathcal{I}}$ is empty, $crete^{\mathcal{I}} \in Island^{\mathcal{I}}$, $greece^{\mathcal{I}} \notin A^{\mathcal{I}}$. C is of course still unsatisfiable.

One can also consider the satisfiability of a concept C with respect to a knowledge base $\mathbf{KB}$. This means whether there is an interpretation that satisfies both C and $\mathbf{KB}$.

Example: The concept $Island \sqcap WaterArea$ by itself is satisfiable, since one can build an interpretation $\mathcal{I}$ in which there is a domain element $w \in \Delta^{\mathcal{I}}$ such that $w \in Island^{\mathcal{I}}$ and $w \in WaterArea^{\mathcal{I}}$. However, given the TBox $\mathcal{T} = \{Island \sqsubseteq LandArea, LandArea \sqcap WaterArea \sqsubseteq \bot\}$, $Island \sqcap WaterArea$ becomes unsatisfiable: In order to satisfy $\mathcal{T}$, every $w \in \Delta^{\mathcal{I}}$ that interprets $Island$ also interprets $LandArea$ and cannot interpret $WaterArea$. Therefore, the set $(Island \sqcap WaterArea)^{\mathcal{I}}$ is empty in all models of $\mathcal{T}$ and $Island \sqcap WaterArea$ is unsatisfiable with respect to $\mathcal{T}$.

For ABox consistency one can differentiate, in analogy to concept satisfiability , between whether an ABox is consistent "by itself" and whether an ABox is consistent with respect to an ontology $(\mathcal{T}, \mathcal{R})$.

Example: While the ABox $\mathcal{A} = \{Island(crete), WaterArea(crete)\}$ is consistent by itself, it is inconsistent with respect to the TBox $\mathcal{T} = \{Island \sqsubseteq LandArea, LandArea \sqcap WaterArea \sqsubseteq \bot\}$.

Naturally, knowledge base consistency and concept satisfiability are fundamentally interesting reasoning services for knowledge engineers. For instance, when one creates a knowledge base it is crucial to know whether the knowledge base allows a model at all or whether there are concepts that cannot be instantiated in any case[12].

Concept satisfiability and ABox consistency (with or without respect to an ontology $(\mathcal{T}, \mathcal{R})$) can easily be reformulated as knowledge base satisfiability.

- A concept C is satisfiable w.r.t. a knowledge base $\mathbf{KB}$ if and only if, given a new individual x_{new} that does not occur yet in $\mathbf{KB}$, $\mathbf{KB} \cup \{C(x_{new})\}$ is satisfiable.
- An ABox $\mathcal{A}$ is consistent with respect to an ontology $(\mathcal{T}, \mathcal{R})$ if and only if the knowledge base $\mathbf{KB} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ is satisfiable.

Actually, the distinction between these cases (concept satisfiability, concept satisfiability with respect to an ontology, knowledge base satisfiability) disappears in many DL languages, at least as regards computational purposes. Conceptually, it may still make sense to differentiate.

Summarising, concept satisfiability and knowledge base satisfiability are central reasoning problems. Below I discuss some more reasoning problems of interest to ontology engineers that can be reduced to knowledge base satisfiability, i.e. they can be regarded as special cases of knowledge base satisfiability.

Subsumption

Is C more specific than D, i.e. does $C \sqsubseteq D$ "always" hold? Similar to the distinction between concept and knowledge base satisfiability, subsumption can be asked with respect to no ontology or with respect to an ontology. In the first case, one actually asks whether $C \sqsubseteq D$ holds regardless of any ontology. If this is true, one also says $\models C \sqsubseteq D$. In the second case one asks whether an ontology entails $C \sqsubseteq D$, i.e.

[12] A knowledge base which contains unsatisfiable concepts is sometimes also called "incoherent" (see e.g. [104]). In the example above, the knowledge base is, according to this terminology, satisfiable but incoherent.

$\mathcal{T} \models C \sqsubseteq D$.
The below reduction of subsumption to (un)satisfiability holds in DL languages that are closed under negation[13].

- $C \sqsubseteq D$ holds if and only if $C \sqcap \neg D$ is unsatisfiable.
- $\mathcal{T} \models C \sqsubseteq D$ holds if and only if $C \sqcap \neg D$ is unsatisfiable with respect to $\mathcal{T}$.

Equivalence

Is $C \doteq D$? Since equivalence is only an abbreviation for the two general inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$, i.e. for mutual subsumption, also equivalence can be reduced to satisfiability.

Disjointness

Are C and D disjoint? Since disjointness can be expressed as the general inclusion axiom $C \sqcap D \sqsubseteq \bot$, also disjointness can be reduced to satisfiability.

Instance checking

Does $C(x)$ "always" hold, i.e. does a knowledge base $\mathbf{KB}$ entail $C(x)$? Instance checking is always performed with respect to a knowledge base $\mathbf{KB} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$. However, single elements such as $\mathcal{T}$, $\mathcal{R}$ or $\mathcal{A}$ may be empty.

- $\mathbf{KB} \models C(x)$ is true if and only if $\mathbf{KB} \cup \{\neg C(x)\}$ is unsatisfiable.

Instance retrieval

Find all individuals $x \in N_I$ that are instances of C. This can be done by checking for all individuals $x \in N_I$ whether x is an instance of C. Instance retrieval is mentioned as a separate reasoning problem not so much because it is logically distinct from instance checking but because this is the basis for knowledge base querying.

2.3 Ontology Engineering

Knowledge engineering is the activity of creating a formal description of knowledge. This activity typically encompasses studying a particular domain, excerpting the relevant concepts in this domain and the inter-relationships of these concepts, and representing these in a chosen knowledge representation formalism (see e.g. [112]). If the outcome of the process is specifically an ontology, this activity is also called ontology

[13] "Closed under negation" means that if $C \in \mathcal{C}(\Sigma, \mathcal{DL})$ then also $\neg C \in \mathcal{C}(\Sigma, \mathcal{DL})$. This is true for all DL languages that are discussed in this section since they all contain complex negation.

engineering (OE)[14].
In the course of this thesis, I use the term ontology engineering rather than knowledge engineering in order to emphasize that my research considers specifically the construction of ontologies and ontology-based knowledge systems. However, many activities are shared between general knowledge engineering and specifically engineering an ontology, and a lot of arguments are more generally true for any knowledge engineering process which involves a formal representation of knowledge.

2.3.1 Ontology Engineering Activities

In the following, a list of typical ontology engineering activities is given. For each of the activities, the description contains the main goals pursued in the activity and the main challenges that this activity presents. Despite the linear presentation however, the ordering does not mean to imply a specific ontology engineering process. Typically, knowledge bases and ontologies are built following an evolutionary lifecycle model [33, 101], which means that it is possible to always go back to any other phase in the process in case some knowledge is found to be missing. Such a structuring of the ontology engineering tasks is also depicted in Figure 2.3.

Scope Definition

In this phase, the scope of the domain of discourse is determined. Factors which help determine the scope are typically the purpose of the knowledge base and the available resources (time, money, domain experts, digital resources, etc.) The outcome of this phase provides the baseline against which decisions of how to represent knowledge, and which knowledge to represent will be taken in the rest of the process.

Knowledge Acquisition

The goal of this phase is to collect as much knowledge about the domain of discourse as possible from a variety of sources, e.g. domain experts, specific problem instances (e.g. case studies) or digital resources such as documents, multimedia files, etc. The outcome of this phase is an unsorted, largely non-prioritised collection of knowledge and knowledge sources from the domain of discourse. The collection typically contains inconsistencies as to vocabulary or different viewpoints. The focus of this stage is to come up with as complete knowledge as possible.

[14] I remind the reader here of an earlier distinction between the terms ontology and knowledge base, where the first denoted a terminology $(\mathcal{T}, \mathcal{R})$ and the second denoted a terminology plus data $(\mathcal{T}, \mathcal{R}, \mathcal{A})$. When I talk about ontology engineering in this thesis I do not strictly exclude the possibility however that the ontology engineer also creates assertional axioms and thus an ABox $\mathcal{A}$. However, it is typically the case that the hard part is designing the structure of data and that this structure is expressed as $\mathcal{T}$ and $\mathcal{R}$. Concrete data are typically filled automatically through an application or at least not according to a conceptually difficult process such as ontology engineering. On the other hand, especially in DL languages that contain nominals and hence also in OWL 2, the borders between TBoxes and ABoxes get fuzzier through nominals, and thus also an ABox is created in the ontology engineering process as described below. In a wider sense, the outcome may still be called a knowledge base then.

The main difficulty in this stage consists in identifying all and only relevant sources of information, and in motivating domain experts to share their knowledge, and to invest time to do so. Research targeted at this stage is typically concerned either with knowledge elicitation from experts (see e.g. [21]), with ontology learning from text (see e.g. [17, 77]) or integration of structured information from heterogeneous sources.

Informal Modelling

The goal of this phase is to present the knowledge collected in the knowledge acquisition phase in an informal but structured manner. At this stage, a vocabulary is fixed. Concepts and inter-relationships are described verbally as unambiguously as possible. However, constraints are not yet formalised in the chosen knowledge representation formalism.
Challenges at this stage are of conceptual nature, and lie for instance in deciding on how to consistently accommodate differing viewpoints and representations or in deciding which knowledge is in scope and which is not.

Formal Modelling

At this stage, the outcome of the informal modelling phase is encoded in a selected knowledge representation formalism. Typically, this activity does not only mean rewriting the acquired knowledge in a different formalims (i.e. simple transfer from verbal encoding to formal encoding) but requires asking more detailed questions, and thinking in more detail about what actually is true. At this stage also decisions as to the degree of detail with which knowledge is to be represented are taken.
The difficulty in this stage lies in finding an adequate formal representation of the knowledge available so far. Another difficulty pertains to the evaluation of formally encoded knowledge: While in the informal modelling phase, domain experts could still directly be asked to verify stated knowledge, this is not necessarily possible with complex formal expressions. Research targeted at this stage is mostly concerned with knowledge representation. On the one hand, knowledge representation formalisms are researched, for instance regarding their computational properties such as decidability or computational complexity or their suitability to different domains is researched. On the other hand, representing patterns of knowledge in a given KR formalism is also researched.

Evaluation

As all engineering artefacts, the knowledge base can contain errors. In the evaluation phase, such errors are systematically sought and remedied. Naturally, evaluation may lead to revision of the knowledge base. A detailed discussion of ontology evaluation is given below in Chap. 2.4.
Evaluation activities are carried out in all ontology engineering phases but with regard to different aspects of the ontology. Research targeted at this stage concerned both with evaluation methodologies and with the automation of this task.

In all phases, research revolves around human-computer interaction issues when software is built that shall support the corresponding task.

The naming of the above activities follows roughly the naming of activities in the first version of the Integrated Modelling Method (IMM) [42, 43] specified for developing a Domain Model, a Task Model and a Skill Model in the context of the APOSDLE project [3, 40, 71][15]. The essentially same ontology engineering tasks have been described by many authors in a similar way however, sometimes under a different naming, and sometimes to a different degree of detail.

Russell and Norvig [112] talk about knowledge engineering in general and call the single activities differently. For instance what is called informal modelling above is called "Decide on a vocabulary or predications, functions and constants" by Russell and Norvig, but otherwise describe the very same activities. Additionally, Russell and Norvig include a task called "Pose queries to the inference procedure and get answers" between formal modelling and evaluation. This activity fits well (i) into the category of evaluation as described within this work, if queries are asked to detect errors or (ii) into the category of application of the knowledge base. Russell and Norvig describe the ontology engineering process as linear.

Specifically for ontologies, Pinto and Martins [101] describe the ontology creation process as consisting of "Specification" (= scope definition), "Conceptualization", "Formalization", "Implementation" and "Maintenance". Conceptualization here means describing an ontology's concepts and their interrelationships, and thus corresponds approximately to informal modelling as described above. Pinto and Martins differentiate between formalising the conceptualization and representing this formalisation in a specific knowledge representation language, and thus formalization and implementation correspond to formal modelling as described above. The task of maintenance is left out in the above described general ontology engineering process, which is a point open for argumentation in each concrete case. Pinto and Martins see knowledge acquisition, evaluation and documentation as ontology engineering activities which need to be present in all phases. Pinto and Martins relate their ontology creation process specifically to three more ontology engineering methodologies, namely TOVE ([48, 49], ENTERPRISE ([138]) and METHONTOLOGY ([33, 34]).

Noy and McGuinness [89] describe the ontology creation process on a very fine granular scale. Roughly, the following steps are carried out in any ontology engineering process: First, "Determine the domain and scope of the ontology", which corresponds to the scope definition task described above. Second, "Consider reusing existing on-

[15] Note that in the final version of the IMM [41], the phases informal modelling, formal modelling and evaluation have been integrated into more abstract tasks for creating the Domain and Task Model, and for creating the Skill Model (Learning Goals) The reason for this abstraction was that the final version of the IMM realised an inherent order in which the three models for domain, domain tasks and domain skills should be modelled. The final version of the IMM is now more modular, in that it prescribes an overall knowledge engineering process for modelling all three aspects, with single modules for creating the single aspects. Although the IMM also prescribes a specific way of modelling the domain, the domain tasks and the domain skills, it is now easily possible to substitute any other knowledge/ontology engineering methodology specific for such a model in the whole IMM.

tologies", which is only implicitly suggested by other methodologies if at all. Then the authors proceed to list "Enumerate important terms in the ontology", "Define classes and the class hierarchy" (classes correspond to concepts), "Define the properties of classes - slots" (properties and slots roughly correspond to relations between classes, but the term "slot" stems from the knowledge representation formalism of frames), "Define the facets of slots" (i.e. define the formal properties of relationships, for instance which values they can take) and "Create instances". The final step assumes that a knowledge base about concrete instances is to be built, instead of an ontology holding only generic truth (which may serve to hold data which is automatically created for example).

An ontology engineer is a person carrying out these tasks. Note that in the ontology engineering process there may be (i) multiple ontology engineers at work and (ii) ontology engineers may have different fields of expertise, i.e. ranging from knowledge acquisition to specific KR formalisms.
In most ontology engineering methodologies, the ontology engineer is assumed to be an expert, a person specifically trained for the above ontology engineering activities and usually trained in some knowledge representation formalisms. In this thesis however I argue in many places that the success of intelligent knowledge based systems depends on enabling also persons less proficient in ontology engineering to create ontologies adequate quality.

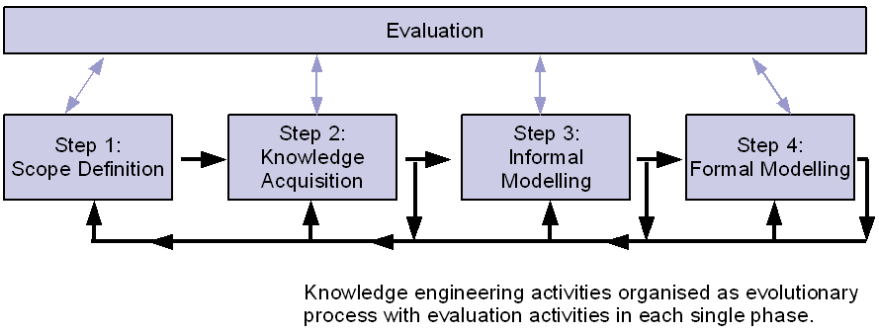


Fig. 2.3. Ontology engineering activities, organised as evolutionary process with evaluation in each single phase.

2.3.2 Tool Support for Ontology Engineering

Tool support for ontology engineering can be given at different levels and for different ontology engineering phases. Tools that support the knowledge acquisition phase are typically not seen as ontology engineering tools, especially if they support the rather creative and unstructured process of knowledge acquisition from humans (domain experts). Nevertheless, some computer-based support can be given. For instance, mind-mapping tools can be used to document the output of creativity techniques

such as brainstorming. Better technical support is available for knowledge acquisition from digital resources. For text, there are ontology learning tools like for instance Text2Onto [20] or tools for general knowledge discovery from text like the KnowMiner [65]. Informal modelling is rarely directly supported by tools, and if so mostly through the availability of graphical modelling or the attachment of rich media content to formal models. In this category fall mind-mapping tools and some ontology editors such as for instance WebODE [22], which provides a graphical designer, or myOntology [119], which supports the attachment of rich media resources and hyperlinks. I point here also to MoKi (see Chap. 3.2), since it particularly aims to address informal modelling. In general, it is unclear how these two creative phases, knowledge acquisition and informal modelling, can be supported automatically. This in turn can be assumed to be the main reason why there is no convention to call such tools ontology engineering tools.

Tools that support formal modelling are called ontology engineering tools and sometimes more specifically ontology editors. Also, requirements on ontology editors are more clear than requirements on tools supporting the knowledge acquisition and informal modelling stages. Per definition, an ontology editor must be able to read and write ontologies in a recognised format for ontologies. To date, this includes mostly formats for languages from RDF(S) to OWL, but also editors for other formalisms like topic maps or conceptual graphs could be considered.I focus in the following discussion on tools that support OWL. Of course, it is desirable that an ontology editor supports multiple languages or at least multiple formats for a single language (since e.g. RDF(S) and OWL typically know more than one format). Additionally, since OWL is a web ontology language, tool support at a technical level includes the editors capability to load ontologies from the web. An ontology editor should also support the creation of OWL ontologies at a logical level, i.e. interface with a reasoner, so that basic questions such as satisfiability can be answered directly within the ontology editor, and if the editor contains a query mechanism to systematically find model elements. The requirements presented so far are supported by a lot of editors such as e.g. Swoop [126], Protégé [102] or the NeOn toolkit [130][16]. Other requirements on ontology editors concern for instance versioning, collaboration, or visualization. These are met very differently, if at all, by different ontology editors. In Chap. 3 I discuss in particular the challenge to support collaboration, and together with this the issue of ontology versioning. Regarding visualization, I only point out that a lot of visualization approaches actually visualize the graph described by RDF(S), or the RDF-graph representation of an OWL ontology. Tool support for ontology evaluation is discussed below, after a more thorough discussion of ontology evaluation itself.

2.4 Ontology Evaluation

Despite the fact that ontology evaluation is a crucial task in ontology engineering, there are relatively few systematic approaches to ontology evaluation and none that can be

[16] A longer list of available ontology editors can be found e.g. in Wikipedia: `http://en.wikipedia.org/wiki/Ontology_editor`

called standard. Ontology evaluation turns out to be a non-trivial task, as is the whole of the ontology engineering process, and additionally one that can be generalised only with difficulty. Therefore, while I start off with some general considerations, I quickly narrow down a discussion of ontology evaluation to evaluation by an ontology engineer during the phase of formal modelling, and relate to my own work where appropriate.

2.4.1 Categories in Ontology Evaluation

What Is Evaluated?

Different authors have come up with different categorisations for relevant dimensions which to consider when evaluating an ontology. I follow the categorisation of [38] when I divide the relevant dimensions into usability-related, structure-related and functionality-related. Usability-related features are for instance a naming scheme (good if it is consistent), documentation of the ontology (good if it exists and is intelligible), structure (good if it is conceptually clear), information about licensing (good if it exists), etc. Structure-related features concern an ontology's graph nature and logical properties, such as the existence of cycles, the depth, logical consistency, etc. Valuating these features is often dependent on the purpose of the ontology but logical consistency is always a required feature in ontologies expressed in logic-based formalisms. Functionality-related features finally concern the adequacy of an ontology with respect to the goal for which the ontology will be used. For instance, it is considered whether an ontology contains all and only relevant information, whether it is modular enough, whether it is agreed upon by the relevant people (i.e. domain experts, participants in a community). I call the latter conceptual correctness, in order to express that this property depends on the conceptualisations of people. However, correctness may be unattainable in practice, so one may settle for adequacy instead of correctness in any real-world situation. Obviously, the functional features are typically the ones that one finds hard to measure. In this thesis I am mainly concerned with the functional dimension of an ontology, and here specifically with conceptual correctness, i.e. whether the relevant people agree with what is modelled. In my work I focus on agreement between people and the ontology, and do not consider problems of agreement between people.

Summary "What Is Evaluated":
– Usability
– Structure
– Functionality

Table 2.7. Categories in Ontology Evaluation: What Is Evaluated?

When Does Evaluation Take Place?

On the other hand, the situation in which an ontology is evaluated must be considered. Each situation has specific requirements on ontology evaluation methodologies and on

desired outcomes, and additionally determines who the evaluators are (see e.g. [46] for a discussion of a similar categorisation). Three situations can be distinguished: ontology reuse, ontology engineering, and application. When reusing ontologies it is necessary to find out which ontology fits best to one's own purpose. One can differentiate between reuse with the purpose of integrating the reused ontology with own work[17] on the one hand and reusing an existing ontology without further modification on the other hand. The latter situation shares its requirements on an evaluation methodology with the requirements of evaluation during application as will be seen. During application, the ontology is a part of a larger system. Evaluation or testing of different components may be part of a required procedure before deployment, or may be part of a debugging process. In any case, the situation in which ontology evaluation takes place influences which ontology evaluation methodologies are appropriate. First, this is because the people who carry out the evaluation are different. In the case of evaluating an ontology during modelling, the evaluator is an ontology engineer and the creator (or at least part of a team) of the ontology. In the case of reuse for the purpose of modification, the evaluator is typically an ontology engineer too, but not the creator of the ontology. In the case of reuse for the purpose of direct application and in the case of evaluation during application, the evaluator is not necessarily an ontology engineer, and is not the creator of the ontology either. Then, each specific situation puts different requirements on an ontology evaluation methodology. I start here by going backwards with the three situations, since I continue the rest of this chapter with ontology evaluation during ontology engineering. If an ontology is evaluated during application or reused directly within an application, the evaluator just wants to know whether the ontology is adequate or not for the application. This can mean that the evaluation does not take place directly on the ontology but that the ontology is evaluated purely within the context of an application. Such a procedure is also called blackbox, since the content of the ontology is not directly observed. Conceptual correctness for instance is not important per se in this situation but only if it influences the behaviour of the application in a negative way. If an ontology is reused for further modification, the evaluator needs to have more fine-grained information about the ontology content (glassbox procedure), but again in the end the evaluator only wants to know whether the ontology is adequate. In these two cases, given more than one ontology that is considered for reuse, the evaluator benefits from a ranking of ontologies. When an ontology is evaluated during ontology engineering, the needs of the evaluator change in that the evaluator does not only want to have feedback on whether the ontology is adequate or not, or a measurement of how adequate it is, but in addition wants to have feedback on how to modify it in order to improve it.

> **Example:** I assume for this simple example that there is a direct correlation between the existence of verbal descriptions of concepts and usability. This assumption does not consider other aspects of usability, and neither that an

[17] In this case, different relations between the reused ontology and the product of the new ontology engineering activities are possible: For instance, an existing ontology may be imported into a new ontology to reuse vocabulary and formal definitions, an existing ontology may be refined or modified, or an existing ontology may serve simply as a template from which design is borrowed, as is the purpose of ontology design patterns [37, 92].

existing verbal description may not be intelligible or may simply be wrong. If the evaluator wants to reuse the ontology, (s)he would probably be satisfied with an evaluation that outputs "70 percent of the ontology's elements are not documented". If the evaluator is an ontology engineer and wants to improve this, (s)he would in addition want to know which elements are not documented.

Since the goal of this thesis is to investigate support for ontology engineering, I discuss in more detail evaluation in different phases of the ontology engineering process. Depending on the phase in which evaluation takes place, the evaluator deals with different kinds of artefacts and different goals against which to evaluate. In scope definition, there is not yet something which can actually be called ontology, but nevertheless the evaluation of the scope definition is valuable in order to find out whether the purpose and scope of the ontology are well-aligned, and if the scope is precisely enough defined.

Example: If two primary applications which shall use the ontology are identified within the scope definition phase, the evaluator should check that these two applications do not have conflicting requirements or assumptions on the ontology.

In knowledge acquisition, the evaluator needs to find out for instance whether all relevant data sources (sources can be domain experts or data sources like text corpora) have been considered, or whether there are conflicting views on the knowledge that shall be described between different sources. In informal modelling, the evaluator needs in particular to check whether relevant parts of the domain have been left out of the model, whether verbal descriptions are already precise enough to start formalising, etc. Formal modelling finally needs to consider in particular whether the formalisation corresponds to the informal model, whether the formalisation is logically correct, i.e. a consistent ontology has been formalised, etc. I note here, that I and colleagues have published a paper on formative evaluation throughout an ontology engineering process in which a more detailed discussion of evaluation in different phases is given [97].

What Is the Reference?

Finally, the reference against which evaluation is performed can be considered to categorise evaluation methods [14]. In the end, the chosen reference depends very much on the ontology evaluation situation, i.e. how the ontology has been created, for what purpose it has been created and at which stage in the ontology engineering process evaluation takes place. First, the reference can be a gold-standard, i.e. an ontology of which one knows already that it is good (enough). This approach is typically taken when an ontology has been automatically learned from text documents or from any other data source. One could argue, that here the ontology evaluation serves to actually evaluate the ontology learning algorithm. Second, the reference can be directly another data source, where the data is typically expressed in a different formalism as the ontology. For instance, the data source may be a collection of text documents or the content of a database. This approach is useful if the ontology shall serve to describe already existing data. Two examples are that a knowledge base shall directly contain data, or because the ontology is to be used to annotate digiatl resources. The obvious difficulty

Summary "When Does Evaluation Take Place":

- *Reuse*
  - *Reuse with subsequent modification*
    The evaluator is an ontology engineer but not the creator of the ontology.
  - *Reuse for direct application*
    The evaluator is possibly not an ontology engineer and possibly not the creator of the ontology.

  In case there are multiple ontologies to choose from, the evaluator benefits from a ranking.
- *Ontology Engineering*
  The evaluator is an ontology engineer, the creator of the ontology, and benefits from hints on how to improve the ontology.
- *Application*
  The evaluator is possibly not an ontology engineer and possibly not the creator of the ontology, and benefits from a binary judgement (adequate / not adequate) on the ontology in the context of a particular application.

Table 2.8. Categories of Ontology Evaluation: When Does Evaluation Take Place? This influences who evaluates and what requirements are put on the evaluation methodology.

with this approach lies in how the ontology should be compared with data in a possibly completely different format. To a certain extent, this is one of the principles on which the investigation of assertional effects, as described within this thesis, is based. The display of assertional effects also serves to enable ontology engineers to check the compliance of their ontology with respect to already existing data. For the purpose of assertional effects, the existence of data in a description logic formalism is assumed. Third, the reference can be the knowledge of domain experts, who may or may not evaluate an ontology systematically given a set of criteria. This is the kind of evaluation that is investigated within this thesis, since I always assume that it is humans who review inferences and ascertain their correctness. Fourth, the frame of reference can be the performance of an application environment in which the ontology is deployed. The comparison can be either with the same application environment fed with other ontologies, as well as an application environment that performs the same tasks but based on a different architecture. In the latter case, one can again argue that it is not the ontology itself that is evaluated but rather the architecture of the ontology-based system.

2.4.2 Ontology Evaluation Methodologies

As already mentioned, there is not really a standard set of methodologies or evaluation guidelines available. In practice, it seems to be the case that depending on the evaluation situation (one from the above), the evaluator chooses a set of relevant properties the ontology should have and tries to evaluate these. This can be supported for instance through a catalogue of ontology properties that one can evaluate, as implicitly given in [38] or explicitly given in the Ontometric framework [128]. Ontometric is applicable particularly in a situation of reuse, and therefore I do not consider it in more depth in the following discussion. Modelling methodologies are often accompanied by guidelines on what to pay attention to during modelling. From this point of view, also many

Summary "What Is the Reference":

- *Gold Standard*
  The reference is an already existing ontology. This is useful if the ontology has been created automatically.
- *Data Source*
  The reference is any data source, and data is typically not expressed in the same formalism as the ontology. This approach is useful if the ontology is intended to describe the data.
- *Domain Experts*
  The reference is the knowledge of domain experts. This appraoch is useful during ontology engineering.
- *Application*
  The ontology is evaluated with an application.

Table 2.9. Categories of Ontology Evaluation: What Is the Reference? The right reference depends on how the ontology was created, in which situation ontology evaluation takes place, and on the purpose of the ontology. The first three references indicate a glassbox evaluation procedure, while the last leads to a blackbox evaluation procedure.

modelling guidelines can be used to evaluate an ontology during ontology engineering. The investigation of typical modelling errors for a specific knowledge representation formalism can lead to catalogues of modelling mistakes and solutions to them. For OWL in particular, there are at least two such studies that I am aware of [109, 111]. Currently, such lists are not widely available as practical guidelines to ontology engineers, and they are rarely incorporated into ontology engineering tools.

Among existing ontology evaluation methodologies, OntoClean [52] stands out as the most formal methodology. It is based on metaproperties of concepts like essence, identity and unity of concepts, inspired by philosophical considerations about the nature of concepts. After having identified the relevant metaproperties of each concept, the ontology's taxonomic relations can be evaluated with regard to their conceptual correctness. The main drawback of this method is that it is difficult to apply in practice, and thus will probably be applied only for high-quality or top-level ontologies. This method directly targets conceptual correctness, is applicable during ontology engineering and in the phase of formal modelling.

A more practical approach to evaluation is the usage of so called competency questions, first mentioned in [137]. Competency questions are questions the ontology should be able to answer and can be posed either informally e.g. as natural language questions, or formally as queries or logical axioms. This approach serves very well for using competency questions as requirements on the ontology against which it can at different stages be tested. Competency questions are formulated repeatedly during the ontology engineering process and refined as well. A first set of competency questions should be formulated already during the scope definition phase. This method targets functionality of the ontology with regard to a specific purpose, and is applicable during ontology engineering in knowledge acquisition (is the required knowledge that can answer the competency questions already collected?), in informal modelling and in formal modelling.

There is also a proposal to express expected inferences and unexpected inferences as logic formulae [140] similar to unit tests in software engineering. Although this approach has not been applied in practice by the authors to date, an implementation of unit tests for WSML has been described in [127].

2.4.3 Tool Support for Ontology Evaluation

The Ontometric framework [128], which supports choosing between ontologies, can be supported through decision making tools. Such tools must be given a weighting of available evaluation features plus an assessment of different ontologies with respect to all features. The provision of this input may not be very practical given the large number of features that can be considered. OntoClean can also be partly automated [139]. A correct labelling of concepts in the ontology with the metaproperties of essence, identity and unity is required beforehand. Reasoning services can be used to verify the logical consistency of an ontology. Thus, they support ontology evaluation during ontology engineering in the formal modelling phase, and evaluate the structural feature of logic consistency. Since this thesis is subsequently concerned with the review of logical inferences for evaluation purposes and detecting inferred knowledge lost or gained about data, I go into some more detail at this point.

As has already been pointed out, logical inconsistency of an ontology as well as single unsatisfiable concepts constitute a modelling error. Reasoning services that find axioms responsible for unsatisfiable concepts and then suggest ways to repair the ontology have been researched under the names of ontology debugging and repair (see e.g. [63, 118]). Inference explanation, e.g. [56, 61], is based on the same theoretical foundation, but with a different focus in application, as is already suggested through the naming. Research in belief revision investigates how a knowledge base must be modified in order to account for a change in beliefs on a broader basis. Changes are either the addition of new knowledge (new axioms are added) or the removal of prior knowledge (axioms are deleted). Central difficulties treated by research in this regard are (i) how to revise the existing knowledge base if directly adding α to $\mathbf{KB}$ would lead to an inconsistent knowledge base and (ii) how to revise $\mathbf{KB}$ such that α is no longer entailed ([2], and [104] specifically for description logics). While ontology debugging and repair is specifically concerned with resolving unsatisfiability (or equivalently: with removing an arbitrary inference), research in belief revision deals more generally with computational properties of these actions. Therefore, belief revision provides the theoretical foundation for knowing how an undesirable inference can be dealt with.
Strategies for belief revision are useful once a problematic inference has been identified. The problematic inference is obvious when it consists of an unsatisfiable concept or the inconsistency of a whole knowledge base. In other situations, an update (adding or removing a particular axiom) may be explicitly asked for, which means that the problematic inference has already been defined in some way. This can happen for instance when a logic theory contains some working hypotheses and one of them has been proven wrong in some world external to the ontology. In a general ontology engineering setting however, the question is also how to find out that there are wrong statements in the ontology, or wrong conclusions that can be derived. This is where

my research on reviewing inferences in general (Chap. 4) as well as particularly on data (Chap. 5) comes in. Such a review makes ontology engineers aware of inferences, which is the prerequisite of identifying wrong inferences.

3

MoKi- A Wiki-based Ontology Engineering Environment

There is a gap in tool support for ontology engineering between the activities seen as creative, such as knowledge acquisition and informal modelling and formal modelling, which is seen as more technical. Typically, different methods and tools support knowledge acquisition, informal modelling and formal modelling. Support for knowledge acquisition is given on the one hand by knowledge elicitation techniques for knowledge acquisition from experts, like card-sorting, laddering or structured interviews, and text mining tools for knowledge acquisition from natural language text on the other. Tool support for informal modelling is given for instance by mind-mapping tools, which enable loosely structuring knowledge. Ontology editors like Swoop, Protégé or the NeOn toolkit support formal modelling. The advantage of a single integrated modelling environment in which all these phases can be carried out is clear. Every switch between tools involves some work additional work, no matter how well importing and exporting between tools works, and people tend to avoid unnecessary work. A single modelling environment thus has the potential to make people execute more easily a dynamic and agile ontology engineering process. By the latter, I mean a process in which iterations through knowledge acquisition, informal modelling, formal modelling, evaluation and consequent refinement and revision are short and frequent. Such dynamics and agility have indeed been seen as inherent in ontology engineering from the start [101], and they become more central as ontology engineering moves into strongly collaborative and spatially as well as temporally distributed environments. This is often the case nowadays, where a modelling team can be distributed over the globe, or when it is a virtual community that builds up its own knowledge base. Additionally, results from different ontology engineering activities can be documented in one place as well as discussions which have led to these results. Thus every actor involved in some stage of the ontology engineering process can easily check up on what has become of her contribution over time, or where the ontology part she is currently working on has actually come from (traceability).

MoKi attacks precisely the gap between modelling tools supporting different modelling activities, in that it aims to be and to become an ontology engineering environment which accommodates informal modelling and formal modelling activities. The following challenges for a computational environment to achieve this goal have been identified as central.

3.1 Challenges for a Computational Environment to Accommodate Informal and Formal Modelling Activities

Collaboration Between Actors with Different Expertise

First, people with a variety of skills must be addressed as users. This addresses not only the dichotomy of domain experts vs. (expert) ontology engineers, but a much finer-granular shading of skills in people participating in ontology engineering. Users may be experts in different domains, and regarding different parts of ontology engineering, for instance some ontology engineers may have more experience with knowledge acquisition and others with a particular knowledge representation formalism.

> **Example:** In the context of APOSDLE [3, 71], learning topics, business processes and human skills were modelled in an integrated manner. Various knowledge engineers were involved in each application domain, where some were more involved with the overall modelling process, and integrating the several aspects, while others concentrated for instance on modelling business processes or on formalising the collected knowledge and on technical issues related to putting the formal models to use in the APOSDLE system [42].

While ontology engineering has always been seen as a collaborative activity, typically the focus has been put on dealing with the quantity of persons interacting with common resources when dealing with support for collaboration, instead of dealing with their quality, i.e. their different competencies. In the few instances where collaboration between people with different skills, typically expert ontology engineers and domain experts, is considered, the focus is often put on capturing and structuring discussions about content [129, 141]. In the traditional sense of collaboration, the following are features of tools[1] which support the collaborative creation of a common engineering artefact: Distributed (synchronous or asynchronous) access to the shared content is essential. Following this, typically version control is required in order to deal with vandalism. Along the same line, user identity management in order to keep track of who is responsible for which information (provenance of information, the possibility to punish vandalism or to reward "good" behaviour), and further also access control are often provided. Interconnecting discussions and content helps document the work process and rationales for the outcome[2]. Depending on the environment in which collaborative work tools are used, support for officially imposed workflows is also sometimes required. For instance, in a productive environment, there may be an official chain of contributors who need to agree to changes before they can be effectuated and published.

[1] There is no standard naming convention for such tools. In literature they are called for instance collaborative editing tools, collaborative authoring tools, computer-supported collaborative work tools etc.

[2] Such a documentation of the process is for instance among the obligatory features according to [74] in order for a tool to be called a collaborative writing tool (The authors consider only writing, in contrast to authoring images/multimedia/structured-data etc). It is also argued in [28] to be a key feature in collaborative ontology engineering since ontology engineering is a creative process similar to writing and designing, and for both such activities the necessity to interlink created content and discussion has been shown in earlier work.

Technically, implementing a workflow in a tool requires at least user identity management and rating or voting (in the widest sense, to include also approval) mechanisms. Specifically for collaborative ontology engineering environments, similar features have been elicited as requirements as discussed in [88, 135]. MoKi provides most of these features by design, as will be discussed at the end of this chapter.
The requirements for supporting collaboration between users with different skills, and thus different requirements on the work environemt are less clear. During development of MoKi, the main assumption was that depending on skill (and task at hand), users need different views of the knowledge contained in the working environment. Views may differ in formality and in content, i.e. which parts of the model(s) a user is interested in.

Coherence Between Content Expressed At Different Levels of Formality

Second, it must be possible to store content at different levels of formality and in different knowledge representation formalisms. Content must be as coherent as possible across different levels of formality and different formalisms. This way, the computational environment can become useful both during conceptual and creative activities such as knowledge acquisition and informal modelling as well as during technical and rigorous activities such as formal modelling and evaluation.
Coherence is an essential basic requirement in order to meet the first goal, namely to support collaboration among people with different skills and therefore different needs. Consider that in a traditional ontology engineering process, domain experts would express their knowledge and structure it informally, not making use of any KR formalism. Domain experts could then go through a validation procedure and finalise their informal version of the ontology. Ontology engineers, experts in some chosen formalism such as e.g. description logics in the form of OWL, would then start with this informal yet "final" ontology and formalise and implement it. During this translation from informal knowledge representation to formal knowledge representation, collaboration between domain experts and ontology engineers would typically take place. During such collaboration, new knowledge might be created, or some precisions to the informally expressed knowledge may take place. While such changes would be expressed in the resulting formal ontology, these changes would not necessarily be documented in the final informal ontology. The formal ontology however would not, conceptually, be easily accessible to the domain experts. In contrast to this, within an ontology engineering environment which is able to keep informal and formal descriptions of knowledge coherent, such changes would be translated back from the formal ontology to the informal ontology.
The main difficulty is that it is little understood what support for keeping coherence is required. An automatic translation between essentially differently formal representations is a huge challenge, and is in its extreme the core challenge of ontology learning. However, related to this is the slightly easier challenge of providing import and export functionalities from and to a number of standard KR formalisms and standard formats for informal models (like for instance mind-maps). A more lightweight approach to support coherence could be for instance a notification of ontology engineers when changes at one level of formality occur.

Integration of Different Kinds of Models

Third, the formalisation of different kinds of models such as for instance general domain knowledge, business processes, a system design etc, must be supported. The first rationale behind this is that during informal modelling it makes sense not to spend too much efforts on deciding what kind of knowledge a particular piece of knowledge actually is. Such decisions should be taken care of mostly during formalisation where a differentiation between different kinds of knowledge is reasonable. On the one hand, this is because particular kinds of models, such as process models, skill models in human resources development etc., have particular well-known properties which may have been studied well and particular modelling and evaluation guidelines may exist. On the other hand, because some knowledge representation formalisms especially suitable for a particular kind of models may exist, such as for instance OWL for general domain knowledge, BPMN (Business Process Modelling Notation, [19]) for business process modelling or UML (Unified Modelling Language, [136]) for system and software modelling etc. The second rationale behind the goal to support the formalisation of different kinds of models is that in cases where such different aspects about a single domain need to be modelled, it makes sense to model as much as possible together to profit from overlaps as well as to document interconnections between these different aspects.

> **Example:** For the APOSDLE system, domain knowledge and business processes were modelled. It was desirable to keep interconnections between single tasks and single topics, which would then form the basis for a more detailed skill model (see e.g. [40] for a description of the underlying knowledge model). "*To Hold a Creativity Workshop* requires knowledge about *Creativity Techniques*" is an interconnection between a task and a topic from the general domain model. The more detailed skill would then read "Know how to apply a *Creativity Technique*".

Such an integration can be realised through the implementation of robust import and export functionalities. Models contained in an integrated environment must be easily exported to specialised knowledge representation formalisms, and imported again after manipulation in external, possibly specialised, tools. In case a corresponding import mechanism does not exist, this may present a direct trade-off with the requirement for coherence as discussed above however. For instance, if a BPMN model is exported from an integrated modelling environment, and in an external tool unsupported BPMN features are included, the BPMN model can not be fully re-imported.

3.2 Related Work

Naturally there are tools which address to some extent the same challenges as MoKi. Where these tools have already been discussed above in Chap. 2.3 the discussion below focuses in particular on how these tools target the above described challenges of collaboration, coherence and integration.

Collaborative ontology engineering tools

Early web-based ontology engineering environments like for instance the Ontolingua Server [31] and WebODE [22] supported collaboration simply in that they give web access to a single resource to multiple persons. Tadzebao [28] adds to this by supporting discussions on all kinds of ontology components in a web based environment, with the added bonus that discussions could contain pictures as well as text. Also Swoop claimed from the beginning to support collaborative ontology engineering, e.g. in [62]. The relevant features are limited to supporting versioning and publishing annotated changes on an annotation server, and Swoop is still a stand-alone desktop application. On the other hand, the versioning supported by Swoop is very fine-grained. For instance, it is possible to show the revision log of a single entity, of a single ontology or of the complete workspace, which may span multiple ontologies. Changes in the ontology can be annotated based on the Annotea framework [59], and are external to the annotated resource, i.e. changes in the ontology are annotated and published on an Annotea server. Collaborators can retrieve annotated changes from the Annotea server and update their local ontologies accordingly. Thus, Swoop does not enforce the development of a single coherent ontology in the case of collaboration, but instead supports the exchange of fine-grained ontology changes. Also discussion of content is only indirectly supported via ontology changes and their annotations. Quite recently, Collaborative Protégé [135] was released, and it contains a multitude of more advanced features to support collaboration. It is also implemented as a client-server architecture, where contributors can simultaneously access the same remote ontology. Collaborative Protégé enhances this basic collaboration support through the possibility to annotate all kinds of ontology elements. The primary intended use of such annotations are threads of discussions, but annotations also capture provenance information, i.e. which user has effectuated a change.

Semantic wikis for ontology engineering

A number of ontology engineering environments are wiki-based, and fall into the broader category of semantic wikis. These tools typically offer per default, per definition of being a wiki that is, functionalities like synchronous access to the shared ontology, user identity management, tracking of provenance of information and annotation of single model elements, although the granularity at which annotation is possible varies in some details between the tools. OntoWiki [7, 91] offers the possibility to annotate, comment on and rate all its elements such as concepts, instances, literals, statements, or files. myOntologyy [85, 119] allows discussions on concepts, and additionally supports a consensus finding process. Both OntoWiki and myOntology have been built from scratch, i.e. they do not build on an existing wiki platform, and both are typical wiki-style environments in other respects. BOWiki [9, 13] on the other hand is an example of a domain-specific environment in which biological information, specifically about the links between genotypes and phenotype in the form of gene annotations, can be stored. BOWiki is an extension of MediaWiki and Semantic MediaWiki (see below), but it offers features supporting collaboration in addition to those provided by MediaWiki. At the time of writing, no working version of BOWiki was to

be found. AceWiki [69], a Java-based wiki built from scratch. It stands a little bit apart from the previous tools in that its focus lies on expressing logics in Attempto Controlled English (ACE, [6]), which is a subset of the natural languge English. So while at the user front-end only ACE is displayed, the information can be (mostly) translated to OWL and thus Pellet can be used to reason on it. AceWiki definitely helps non-expert ontology engineers express their knowledge formally, since AceWiki users are able to from near-natural language sentences for doing so [68]. However, in contrast to the above discussed wiki-based ontology engineering environments it does not provide more support for collaboration than giving synchronous access to a shared ontology over the web.

Semantic wikis as platforms

Semantic wikis such as Semantic MediaWiki (SMW) [67] and KiWi [64, 116] aim to be general platforms on which traditional, human-readable content such as text, images and multimedia can co-exist with structured data. In contrast to environments for ontology engineering, structured data are primarily seen as metadata for the traditional content. KiWi also supports light-weight content structuring paradigms such as tagging and related features such as tag recommendation, and aims to include social software features as well.

Coherence Between Content Expressed At Different Levels of Formality

A pragmatic approach for an ontology engineering tool to support coherence between content expressed at different levels of formality is of course to provide the possibility to attach annotations to all kinds of ontology components. This enables at least keeping informal and formal (parts of the) models in one place. Many of the above ontology environments can contain only textual annotations (Ontolingua, WebODE, Swoop, Protégé). In contrast to this, Tadzebao supports attaching pictures as annotations as well, and most wiki-based ontology engineering environments support annotation with everything which can be held in a webpage.
Research concerning translations between different KR formalisms (or concerning imports and exports from and to different KR formalisms) have traditionally focussed on the comparison between languages which already are formal, such as a comparison of the description logic $\mathcal{ALC}$ and the logic multi-modal $\mathbf{K}$ which have been found to be equivalent [117], or the attempts to bidirectionally map RDF and Topic Maps [39, 84, 100, 134].

3.3 Design and Implementation of MoKi

The description below holds for Version 1.2 of MoKi, with the ontology questionnaire and the assertional effects functionality (two of MoKi's evaluation functionalities) only being available from Version 1.5Beta on. An online demonstration system, a user manual, screencasts and various other resources can be found on MoKi's website [83]. Source code and binaries are expected to be released in the near future.

3.3.1 MoKi as a MediaWiki plugin

With the above challenges in mind, it was decided to implement MoKi based on the Semantic MediaWiki. Semantic MediaWiki is a plugin to MediaWiki [81] which allows storing data as RDF and thus enriching textual and multimedia content of MediaWiki pages by structured knowledge. MoKi is itself a plugin to MediaWiki, and it requires the plugins Semantic MediaWiki and Semantic Forms to be installed in order to run correctly.
The choice to implement MoKi as a semantic wiki was taken for several reasons. First, a wiki environment was chosen since the wiki principles of giving access to content (both read and write) to all as well as to make that access as easy as possible for everyone are well-aligned with the goals of MoKi. Furthermore, wikis are inherently web-based and allow textual as well as multimedia content , i.e. everything which can be published on the web can be published in a wiki. Most support versioning and standard collaboration (in the traditional sense) features like discussion threads or comments. Finally, most potential users of a system such as MoKi can be expected to know how a wiki looks and feels, and a large portion of these people also know how to actively contribute to wiki content. This is partly due to the large success of the online encyclopaedia Wikipedia [142], but as well to the arrival of wiki (and other Web 2.0) technology in the corporate world, as observed e.g. in [45, 113]. Second, a semantic wiki [115] already has the basic infrastructure to deal with structured data in addition to traditional human-readable content-types like text or multimedia. Thus, it is technically well suited to accommodate the results from informal as well as formal modelling activities. In choosing a particular semantic wiki upon which to build MoKi, the main requirements were (i) that it be under active development and that it could reasonably be expected to stay thus for some time in the future, (ii) that it be easy to extend and (iii) that it have a reasonable implementation of the semantic infrastructure. Although several semantic wikis exist that are under ongoing development, but still relatively few are suitable as a platform for extension. Nowadays, KiWi would be a serious alternative to consider besides Semantic MediaWiki. KiWi is a comparatively recently developed semantic wiki, although it goes back to an earlier system called IkeWiki [114]. The KiWi system is programmed in Java and has been newly developed from scratch, but the user interface follows the well-known wiki paradigm. At the time the MoKi development was started however, KiWi did not yet exist, and the continued development of IkeWiki was too unsure to be considered as basis for MoKi. Most semantic wikis, like for instance AceWiki or OntoWiki simply promised too little security in terms of continued development, too little benefits for reuse like features outside semantic content creation or, as e.g. SweetWiki [16, 125] a too ad-hoc semantic infrastructure. The choice therefore fell on Semantic MediaWiki first because it is based on MediaWiki, which is widely-spread, has a large community of developers and users, and is easily extensible through plugins. Second, Semantic MediaWiki itself has a sound, straightforward implementation of a semantic infrastructure and it is generic insofar as it is not adapted to any specific application scenario.

3.3.2 Every Model Element Is a Wiki Page

The basic design principle of MoKi is that every model element, i.e. every basic component of a model, corresponds to a wiki page. Speaking of ontologies, the relevant model elements are concepts, individuals and roles. etc. Different model elements are differentiated through the use of MediaWiki categories, such that for instance a wiki page describing a concept has the wiki category "Domain model"[3] , a wiki page describing an individual has the category "Individuals" and a wiki page describing a role has the category "MokiProperty"[4]. Each model element is internally given a formal meaning. For instance, a concept is given the meaning of a description logic concept, i.e. it is a unary predicate and can be interpreted as a set of entities for which the unary predicate holds. It is essential to be clear about this internal interpretation of model elements, since this forms the basis of technically dealing with imports from various knowledge representation formalisms and exports to various knowledge representation formalisms. Table 3.1 shows a synopsis of categories available in the current implementation of MoKi, their formal interpretation as well as in which kind of model the corresponding model element is expected to be useful.

Category	Model Element	Interpretation	Type of Model
"Domain model"	Concept	DL Concept	Domain ontology
"MokiProperty"	Property / Relation	DL Role	Domain ontology
"Individuals"	Individual	DL Nominal	Domain ontology
"Process model"	Process	BPMN Process	Process/Task model

Table 3.1. Category names in MoKi for designating different kinds of model elements in MoKi. "Type of Model" refers to the type of model in which such a model element is expected to occur.

For every kind of model element supported by MoKi, a template exists. Figure 3.1 shows an excerpt of an already filled-out concept description. The implementation of templates in MoKi is based on the Semantic Forms [36] extension to Semantic MediaWiki, which makes it easy to create templates. From a user perspective, a template is displayed as a list of fields which to be filled out in order to describe the model element. Fields can differ between model elements. Conceptually, a template asks the ontology engineer for information which is typically needed for a specific kind of model element.

> **Example:** When describing a domain concept, it is typical to ask "What is a superconcept?", i.e. what are more general notions than the currently described concept, in which categories does it fall?

[3] Wiki categories could have been used as well to represents the concepts of the domain model. However, when we started developing the tool, the support for categories in Semantic MediaWiki was rather preliminary, so we decided to represent domain concepts using standard pages.

[4] Semantic MediaWiki also knows relations (properties and types), but since these are interpreted as RDF properties and not as OWL properties, it was decided to not change the interpretation of built-in Semantic MediaWiki construct but overlay SMW with extra features. However, every MoKi property is also an SMW Property.

Naturally, most given fields are just possibilities and the users are not necessarily required to fill all of them when describing a specific model element. Additionally, there is in some templates the possibility for users to add custom fields.

> **Example:** An ontology engineer may want to describe the concept "Project", and then express that a project is typically managed by a person. In this case, the ontology engineer can add a new field to the concept "Project" which is called "managed-by" and fill it with the concept "Person".

The use of templates, which can easily be constructed using Semantic Forms, means that it is easy to extend MoKi to hold other kinds of models apart from generic domain ontologies and business processes. Although such an extension can not yet be done solely at the user interface (some programming in PHP is required in order to define the formal meaning of the fields in a new template and to add import and export support for the new model elements), it is easy to program.
Templates for all kinds of model elements are split conceptually into two different parts, namely one for informal modelling and one for formal modelling. Furthermore, most fields in a template are given a formal meaning, such that they can be imported from and exported to knowledge representation formalisms. Tables 3.2- 3.3 show the existing templates in MoKi with their fields and the formal meaning given to fields.

Informal Description

Every model element can be described informally. The purpose of this part is to document the model and to clarify what this element is meant to describe. This serves on the one hand as a starting point for modelling, during informal modelling activities, where formal descriptions are not yet aimed for. On the other hand, this part serves as "backwards"-interface to users not trained and/or not interested in reading and understanding the formal representation.
Part of the informal description is kept in the formal models as annotation elements if the formal language allows for this during export. For instance, OWL allows including verbal descriptions as `rdfs:comment` and labels (e.g. synonyms for concept names) as `rdfs:label`. In direct correspondence to these two annotation properties, templates for domain ontology elements have the fields "Description" and "Synonyms". Additionally, there is the possibility to add free notes, in which all content which MediaWiki allows can be added. This is a good possibility to reference source documents, document modelling choices and keep notes about open problems, etc. Free notes are per definition not meant to appear in the formal representation of the model. First, this would be technically impossible since most formal languages do not offer possibilites to embed rich content, such as e.g. a picture. Second, this is also conceptually undesirable since the free notes should be a place reserved for notes and knowledge which is not (yet) meant to be formalised.

Formal Description

Every model element can also be described formally. The central distinguishing characteristic of this formal description is, that each field is given a formal meaning by MoKi.

Some fields may be predefined, as for instance the fields "Is a" and "Is part of" for domain concepts (see Fig. 3.1), which are translated to description logic subsumption and an existential quantification over a dedicated "part-of" relationship[5] respectively (see Table 3.2). Depending on the field of application, it may also be decided to predefine fields which interconnect different kinds of models. Such predefined fields correspond to the metamodel which underlies a particular knowledge engineering scenario.

> **Example:** In APOSDLE a field called "required knowledge" was predefined in the template for tasks. The meaning of this field was to express that a task requires knowledge about some topic. Thus, users could connect tasks immediately to concepts of the learning domain ontology [40, 41].

> **Example:** In BOWiki, parts of the Gene Ontology [4] and the Ontology of Functions [18] provide the metamodel for the knowledge captured in BOWiki. For instance, the relation "hasFunction" is used to describe the function of a gene [9].

Other fields are user defined, as for instance the field "financedBy" in the example depicted in Fig. 3.1, which is translated through giving the subject (the currently edited concept) as `rdfs:domain` and the object (the filler of the corresponding field) as `rdfs:range` of the OWL object property corresponding to "financedBy". Such user defined fields give the MoKi users the possibility to create custom relations between model elements. Although this description is entered in wiki style or in form fields, it is necessary to remember that this still constitutes a formal description, since MoKi gives a very definite meaning to the values in these fields in relation to the edited page. This definite meaning is used during import from and export to various knowledge representation formalisms.

[5] MoKi defines dedicated has_direct_part and inverse $is\_direct\_part\_of$ relations for this purpose. It is planned to map these MoKi relations to corresponding relations in existing upper ontologies such as DOLCE [27].

Fig. 3.1. Excerpt of a filled-out concept template in MoKi, shown in the figure for a concept called "Workshop". The fields in the *Annotation*, *Hierarchical structure* and *Notes* boxes are available for all domain concepts. The fields in the *Properties* box are added by the ontology engineer specifically for each domain concept, as e.g. "hasParticipant" and "isOrganizedBy" for the concept "Workshop".

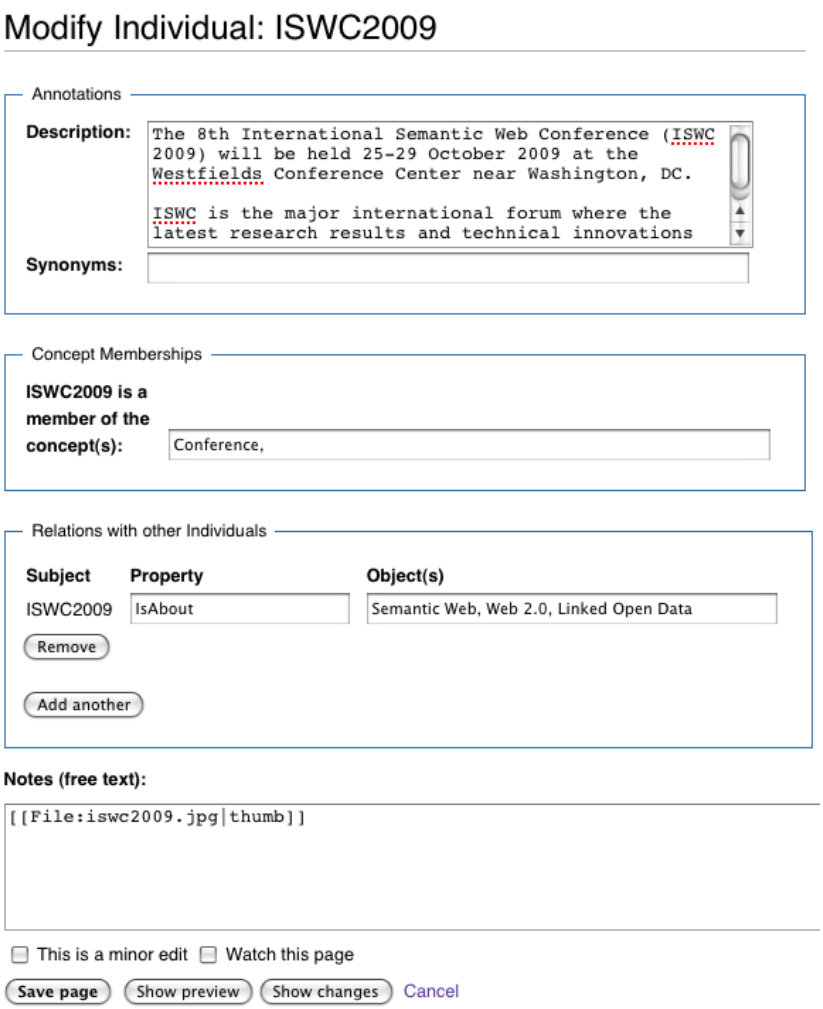


Fig. 3.2. Filled-out individual template in MoKi, shown in the figure for an individual called "ISWC2009" (the International Semantic Web Conference 2009). The fields in the *Annotation*, *Concept Memberships* and *Notes* boxes are available for all individuals. The statements in the *Relations with other Individuals* box are added by the ontology engineer specifically for each individual, as e.g. "isAbout" for the individual "ISWC2009".

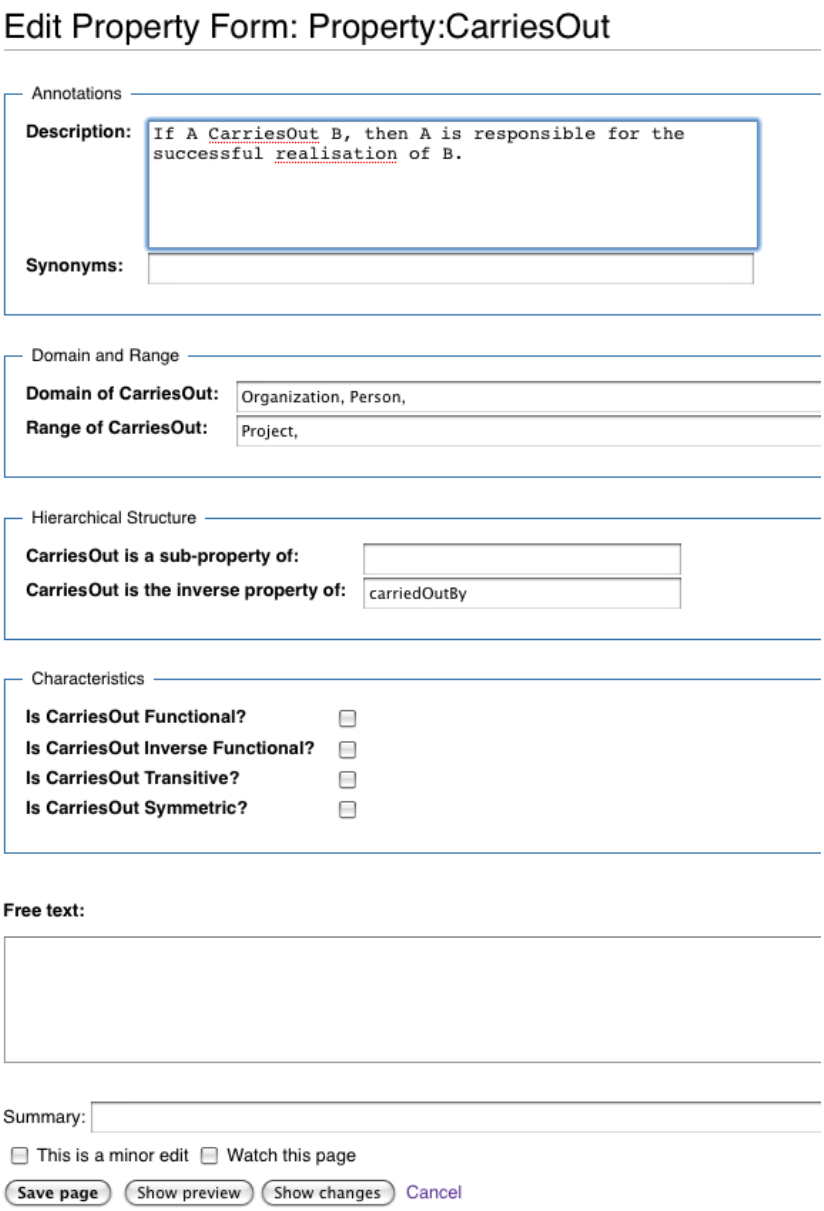


Fig. 3.3. Filled-out property template in MoKi, shown in the figure for a property called "CarriesOut". All fields (fields in the boxes *Annotation*, *Domain and Range*, *Hierarchical Structure*, *Characteristics* and *Free text*) are available for all properties.

Concept template in MoKi	
Description	
Formal interpretation:	`rdfs:comment`(C, v)
Synonym(s)	
Formal interpretation:	`rdfs:label`(C, v)
Usage:	Values in the field are comma-separated
Is a	
Formal interpretation:	$C \sqsubseteq D$
Is part of	
Formal interpretation:	$D \sqsubseteq \exists has\_direct\_part.C$
Comments:	An inverse role $is\_direct\_part\_of = has\_direct\_part^{-1}$ also exists. has_direct_part, $is\_direct\_part\_of$ are MoKi-defined roles.
R **(user defined)**	
Formal interpretation:	$C \sqsubseteq \forall R.D$
Free notes	
Comments:	Intended for rich content, not intended to be formalised.

Table 3.2. Concept template in MoKi, fields and their formal meaning. The formal meaning is expressed either as DL formula or as RDF triple (subject predicate object). C is the concept which the template describes, D is the concept with which a field is filled, or v is the data value with which a field is filled.

Property template in MoKi	
Description	
Formal interpretation:	rdfs:comment(C, v)
Synonym(s)	
Formal interpretation:	rdfs:label(C, v)
Usage:	Values in the field are comma-separated
Domain	
Formal interpretation:	$\exists R.\top \sqsubseteq D$
Usage:	Comma-separated values in the field denote a disjunction.
Range	
Formal interpretation:	$\exists R^{-}.\top \sqsubseteq D$
Usage:	Comma-separated values in the field denote a disjunction.
Is a sub-property of:	
Formal interpretation:	$R \sqsubseteq S$
Is an inverse property of:	
Formal interpretation:	$R = S^{-}$
Further properties:	Functional, inverse functional, transitive, symmetric.
Formal interpretation:	As usually defined, see e.g. Table 2.2.
Free notes	
Comments:	Intended for rich content, not intended to be formalised.

Table 3.3. Property template in MoKi, fields and their formal meaning. The formal meaning is expressed either as DL formula or as RDF triple (subject predicate object). R is the role which the template describes, D is the concept, S the role, y the individual or v the data value with which a field is filled.

Individual template in MoKi	
Description	
Formal interpretation:	rdfs:comment(C, v)
Synonym(s)	
Formal interpretation:	rdfs:label(C, v)
Usage:	Values in the field are comma-separated
Is a member of	
Formal interpretation:	$D(x)$
Usage:	Comma-separated values in the field denote a conjunction.
R **(user defined)**	
Formal interpretation:	$R(x, y)$
Free notes	
Comments:	Intended for rich content, not intended to be formalised.

Table 3.4. Individual template in MoKi, fields and their formal meaning. The formal meaning is expressed either as DL formula or as RDF triple (subject predicate object). x is the individual which the template describes and y is the individual or v the data value with which a field is filled.

3.3.3 MoKi Functionalities

The functionalities within MoKi support activities like importing and exporting models to and from MoKi, navigating through the MoKi content, authoring and editing the models contained in MoKi at an informal as well as formal level, and validating the models contained in MoKi. MoKi functionalities are accessed in the typical wiki-style through a menu with hyperlink items (see Fig. 3.4). Alternatively, the functionalities can also be accessed through directly typing in the corresponding URL into the address bar of the browser. Since the objective of this dissertation is not to provide a user manual, no screenshots with step-by-step explanations on how to use MoKi will be given. Instead the functionalities and their inner workings are explained. As already pointed out above, the online MoKi resources at [83] contain an excellent user manual with this information. All functionalities described below are additions to the existing semantic infrastructure provided by the Semantic MediaWiki, except if especially noted. For communication between MoKi users, no extra support is given so far. Instead, the discussion feature of MediaWiki is used. Versioning and keeping track of changes is also supported via the standard MediaWiki features.

Import/Export

MoKi can export the models it contains according to the formal meaning given to fields in templates for model elements. To date, this means that MoKi can export the domain ontology it contains into OWL 2, RDF/XML Rendering. For export, MoKi first exports its contents to RDF using the Semantic MediaWiki built-in functionality, and then reinterprets the generated RDF as OWL 2 according to the interpretation rules given to fields as explained in Tables 3.2 - 3.3 for domain concepts. Of course MoKi also supports the inverse functionality and is able to import OWL 2 ontologies in any rendering supported by the Jena Semantic Web and insofar as the imported ontology does not contain features which exceed MoKi's expressivity[6]. All constructs which MoKi does not know are ignored at import, so importing and exporting again can lead to a loss of data[7].

Apart from this straightforward support for import and export, MoKi also supports importing knowledge from less structured sources. First, it is possible to easily import hierarchies of concepts. This can be done by writing down the hierarchy as a simple ASCII list of terms, and indentation indicates the hierarchy (for a simple example see Fig. 3.5). The hierarchy can be interpreted either as specialisation hierarchy (description logic subsumption) or as mereological hierarchy (part-of relationships as interpreted by MoKi are created between super- and subordinate concepts). Second, MoKi supports the import of knowledge from text documents by including a term extraction functionality. This functionality uses at the backend the KnowMiner framework, a Java-based framework for knowledge discovery [47, 65]. Extracted terms can be taken over with one click as (candidate) concepts into the MoKi-contained knowledge base.

[6] MoKi's expressivity is the sum of features listed in Tables 3.2- 3.3. The goal of MoKi is to support OWL 2.

[7] At the time of writing, e.g. negative property assertions can not be made ("Alice does not like Bob"). This means that if an ontology with a negative property assertion is imported and exported again, this negative property assertion will be missing from the exported ontology.

Navigation / Model Access

In any information or knowledge management system, navigation through content is vital to its success. The best content is useless if it cannot be easily accessed. MoKi content can be accessed through standard MediaWiki functionalities like search, or through typing in the URL of a single page in the address bar of the browser. Apart from this, there is on the one hand the possibility to get lists of model elements in a tabular style (e.g. "List Concepts", "List Individuals", "List Properties" entries in the menu, see Fig. 3.6). These lists are by default sorted alphabetically according to concept names, but can be sorted according to any other column as well. The lists are created simply by using Semantic MediaWiki's "Ask" mechanism [5]. On the other hand, MoKi provides more visual access to the model(s) it contains. For domain ontologies, the obvious visualisations are tree visualisations of different hierarchies. Different to most existing ontology editors, MoKi displays not only the specialisation hierarchy as a tree ("IsA Browser") but also the mereological hierarchy in the "IsPartOf Browser" (see Fig. 3.7). Among the other ontology editors, there is only Protégé with its Outline/Existential View [29] plugin which offers a similar feature, namely the possibility to create hierarchical visualisations according to existential restrictions along arbitrary object properties. The model contained in MoKi can be directly interacted with via the tree visualisations: The subsumption hierarchy between model elements can be manipulated by dragging and dropping model elements, model elements can be added, deleted or renamed, and finally the full description of model elements (a wiki page in MoKi) can be accessed via a right-click. The trees are implemented based on the Javascript DHTMLx-Tree library [25].

Editing

Editing activities relate to single model elements and concern the creation of model elements, their repeated editing (among this renaming the model element, or even changing its type), and the deletion of model elements. These activities are supported mainly through standard MediaWiki functionalities related to pages, which is easy since every model element is represented as single wiki page. Changing the type of a model element (i.e. from domain concept to individual) can be done by changing the wiki category of the page. Note that information may be lost, and no advanced support is available for this operation.

Evaluation

Whatever purpose models are created for, they need to be evaluated in order to ensure that they will serve their intended use. MoKi expressly aims to support ontology evaluation during modelling, and the focus of this thesis has been to make progress in this direction. As a result, MoKi supports ontology evaluation through a models checklist, a quality indicator, the ontology questionnaire and through displaying assertional effects. While all evaluation functionalities are described in detail in the following chapter (Chap. 6), I shortly describe each here to complement the general overview of MoKi functionalities. The models checklist is accessible via the MoKi menu ("Models Checklist" in the menu, see Fig. 3.4). It is a list of characteristics that typically point to

oversights and modelling guidelines, and automatically retrieves elements that fit the characteristics. For instance, one point on the checklist is "Orphaned concepts", i.e. concepts that have no super- or subconcepts, have no parts and are not part of anything. These are often concepts left-over from brainstorming or another earlier modelling iteration. The quality indicator is displayed on the page of all elements and visualises the completeness and sharedness of the corresponding element. Both completeness and sharedness are heuristic measures, where the first captures how much information (verbal, structural) about the element is available while the second captures how many people have contributed to the description of the element. The ontology questionnaire corresponds roughly to the ontology questionnaire described above in Chap. 4, and is also accessible via the MoKi menu ("Inferences: Do You Agree?" in the menu, see Fig. 3.4). It displays inferred knowledge, i.e. statements that can be derived from the models contained within MoKi, provides explanations for them, as well as the possibility to remove them. In case explicitly made statements are deleted in order to remove an undesired inference, the ontology questionnaire also displays side-effects, i.e. all inferences that will be lost alongside. Assertional effects (Chap 5) are displayed on concept and property pages directly after an ontology edit that causes one or more assertional effects.

Maintenance

A data cleaner functionality allows the batch deletion of concepts, properties, individuals, the complete domain model, the complete process model or the complete MoKi content. It is currently available only through direct access of the corresponding special page `Special:DataCleaner`.

Fig. 3.4. Access is given to MoKi functionalities via a wiki-style menu with hyperlinked items. Functionalities in MoKi support activities like importing and exporting models, navigating through MoKi content, authoring and editing the models contained in MoKi at an informal and a formal level, and validating the models contained in MoKi.

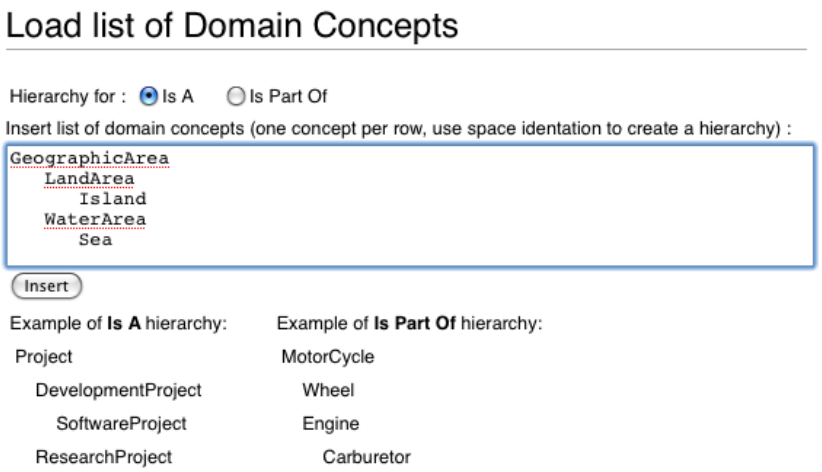


Fig. 3.5. Import a hierarchy into MoKi. Due to the selected radio button "Is a" this hierarchy will be interpreted as specialisation hierarchy, and the statements $LandArea \sqsubseteq GeographicArea$, $WaterArea \sqsubseteq GeographicArea$, $Island \sqsubseteq LandArea$, $Sea \sqsubseteq WaterArea$ will be inserted into MoKi.

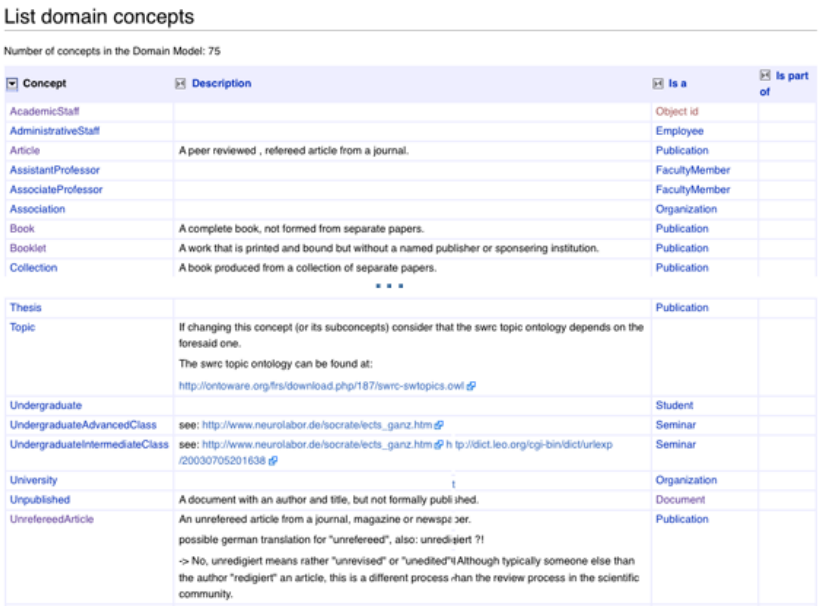


Fig. 3.6. Excerpt of the list of concepts in MoKi. MoKi displays all concepts and some relevant properties like verbal description, superconcept or containing component in a tabular form. By default the list is sorted alphabetically by concept names, but the user can sort it according to other table columns as well.

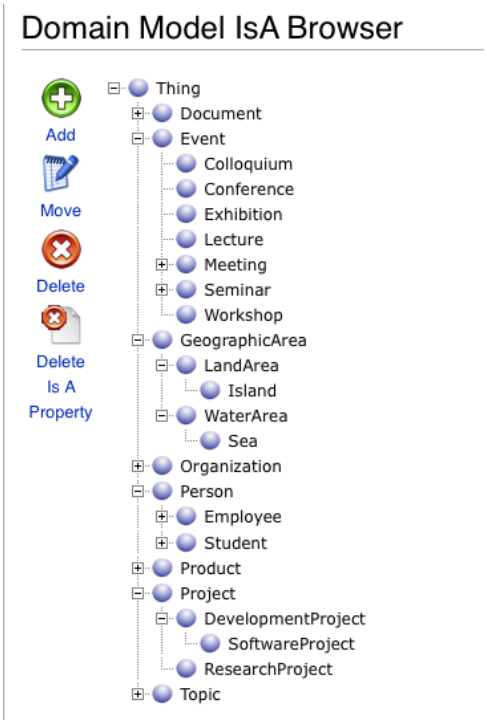


Fig. 3.7. Tree visualisation of the specialisation hierarchy (IsA Browser) in MoKi. Also the mereological hierarchy (part-of relationships) can be visualised as tree in MoKi. The model can also be directly manipulated via the visualisation.

3.3.4 PHP and Java

MediaWiki, MoKi's basis, as well as Semantic MediaWiki and Semantic Forms on which MoKi strongly builds are implemented purely in PHP. MoKi is therefore naturally implemented to a large part in PHP as well. However, MoKi uses JavaScript and Java in addition. JavaScript is used for the tree visualisations (DHTMLx Tree Library, as indicated above) following the AJAX paradigm in order to provide a rich and reactive user interface. Java is used at the backend for a variety of functionalities. The main reason is for using Java at the backend is that many good libraries are available in Java. Therefore, import and export functionalities and the term extraction functionality are implemented in Java. Java-based functionalities are implemented as main methods which take commandline arguments. The MoKi's PHP code executes the Java-based functionalities in the commandline then via the PHP command `exec()`. The export functionality writes as result of its call an owl-file in a temporary directory, which is then opened for download by the PHP code. The import functionality translates a given OWL file into an XML file, which is written to a temporary directory. This XML files can be imported by a semantic page import functionality of MoKi as the required wiki pages. The term extraction functionality writes the extracted terms to a file which is then read by MoKi's PHP code and output to the user interface.
Executing Java functionality through commandline calls of main methods and communicating over temporary files is an easy way to access Java from PHP, and serves its purpose well where only a few arguments need to be passed from one to the other.

3.4 Discussion

MoKi aims at supporting a variety of ontology engineering activities, such as knowledge acquisition, informal modelling, formal modelling, and evaluation of models at various stages. Knowledge acquisition is supported through the term extraction functionality. The term extraction functionality is state-of-the-art, which unfortunately means that it supports only a limited number of languages (English and German currently) and that the quality of results depends a lot on the corpus it is given. Informal modelling is supported via the possibility to import simple hierarchies, via the prominently placed possibility to verbally describe and document ("Description", "Synonym(s)") all kinds of model elements, as well as via the possibility to richly document ("Free notes") all kinds of model elements in all formats which can be held in a webpage. Furthermore, also evaluation of informal aspects of the models contained in MoKi is supported, in that for instance elements with no verbal description are explicitly pointed out to MoKi users. Formal modelling is supported from a user perspective by providing form fields with auto-fill functionality to ontology engineers. Fields are given a formal meaning, which is the basis of technically supporting formal modelling. However, on the formal modelling side, MoKi does not (yet) support the full expressivity of OWL 2. Most importantly, it does not yet support complex concepts. Evaluation finally is supported by checklists which list model elements with qualities which indicate potential modelling mistakes and which thus should be reviewed.
In order to accommodate the integration of informal and formal modelling within one

computational environment, the issues of collaboration between actors with different skills, keeping coherence between content expressed at different levels of formality and of integrating different kinds of models have been investigated during development of MoKi. Collaboration between people with varying skills in the domain, ontology engineering, and description logics as knowledge representation formalism is supported by providing a medium (a wiki) which can hold both informal and formal knowledge. Note also, that most requirements which are discussed in [88] for collaborative knowledge construction tools and in [135] for Collaborative Protégé are easily met by MoKi, merely by its being implemented on top of MediaWiki. The requirements satisfied by MoKi are distributed access to a shared ontology, version control, user identity management and tracking the provenance of information and discussion on model elements. Fine-grained access control is not possible in MoKi, and collaborative protocols that involve rating or voting are not supported in MoKi either. Coherence is ensured mostly through keeping the informal and formal model element descriptions in one place, i.e. in one wiki page. Inconsistencies between the natural language text or rich content on the one hand and the formal descriptions on the other hand are not detected. Indeed, this would exceed state-of-the-art in natural language understanding, and even more so in multimedia understanding. However, coherence is supported in another slightly roundabout way, namely through the "watch" functionality of MediaWiki. Through this functionality, users can be notified if changes at a wiki page, in MoKi that is changes concerning a model element, occur. Like this, both domain experts and ontology engineers can easily detect changes to parts of the ontology in which they hold an interest. MoKi is currently able to integrate the creation of domain models, business process models and skill models. An extension to other kinds of models is easily possible since templates, as provided by Semantic Forms, are used to create the specific model element pages. While MoKi naturally does not aim to substitute specialised modelling tools which have been developed for all kinds of knowledge representation formalisms, it does aim to be a central ontology engineering environment for all, such that sufficiently developed formal models can exported into standard KR formalisms from MoKi.

The goals that MoKi targets (supporting knowledge acquisition, informal and formal modelling, integration of informal and formal modelling through supporting collaboration between people of different skills, providing coherence across levels of formality and integrating different kinds of models) have been partly reached. The following are directions of ongoing and future work to come still closer. First, MoKi does not yet fully support OWL 2 but this is underway. The challenge hereby is less technical than related to user-interaction design. Care must be taken not to trade off knowledge representation complexity with ease-of-use, since MoKi was in particular designed to be used also by non-expert knowledge engineers. Second, MoKi did not originally integrate a reasoner. This has been addressed by myself in the context of integrating feedback on modelling through assertional effects and on evaluation through review of entailed statements into MoKi and is discussed in more detail below in Chap. 6. In general, the support for integrating informal and formal modelling in MoKi could go further. As an example, some back-translation of formally expressed knowledge into natural language could be provided (logic verbalisation).

4

Ontology Evaluation Through Review of Entailed Statements

Through reviewing entailed statements, ontology engineers can verify whether what they modelled is what they meant. The trick hereby is that entailed statements as well as simple solutions to get rid of unwanted entailments can be generated automatically. Ontology engineers benefit from reviewing inferred statements in two ways: First, they get an insight into knowledge implicit in the ontology, and second they can review the ontology without getting bored by having to go through what they have explicitly modelled yet another time.
My contribution consists of bringing relevant description logic founded work into the field of knowledge engineering. Diagnosing and repairing inconsistent logical theories has for long been researched in the logic community, and it will turn out that the developed methodologies will serve as a starting point to handle undesired inferences. It will also be seen however, that this can only be a starting for supporting ontology engineers in their inherently creative activities. I start my discussions from the ontology engineering viewpoint, namely that reviewing what has already explicitly been stated in slightly different terms is beneficial. Only from there will I discuss how this is related to the description logic problems of repairing inconsistencies. Relevant contributions are then the analytic discussion of the potential and limitations of reviewing entailed statements in order to evaluate an ontology, and experimental study in which this approach was executed based on a prototypical implementation, called the ontology questionnaire. Insights gained from this investigation finally have led to a more advanced support for ontology evaluation in MoKi (see Chapter 6).

4.1 Related Work

A variety of tools could be used to review entailed statements, since functionalities for inference explanation and ontology repair are available in most modern ontology editors, for instance in Swoop, in Protégé 4 using the explanation plugin [56, 103] and in the NeOn toolkit through the RadOn plugin [106]. Because of a different focus, the access to the inferred statements and to their explanations is often not very direct, and the approach to repair is more technical than conceptual. Although Protégé 4 includes

a list of inferred statements [1], inferred concept inclusions are most often displayed implicitly through a tree visualisation of the subsumption hierarchy. This implicit display of inferences may not be enough, for instance in order to detect misleading concept names. Although concept names do not change the formal meaning of an ontology, it is nonetheless important to convey as much information as possible via the name to human users of the ontology in order to enhance the ontology's usability. Usability in turn is relevant for ontologies since ontologies are often applied at the interface between humans and computers.

> **Example:** In order to distinguish between information sources accessible only from within a company and public information sources, in a tree visualisation it is easy to build a hierarchy with $InformationSource$ at the root, and $Internal$ and $External$ as child-concepts. When displayed as explicit statements however, it is easy to see that "Internal is an Information Source" does not make sense as statement.

Furthermore, the review of entailed statements during an ontology evaluation procedure can be seen as complementary to other systematic approaches for evaluation. Among the few systematic evaluation strategies (see Chapter 2.4), I argue that the usage of competency questions is particularly well suited to complement the review of entailed statements as ontology evaluation practice. While the first validates the ontology's conformity to requirements, the latter validates the agreement of domain experts and ontology engineers with what they have implicitly modelled.

4.2 Ontology Questionnaire: Tool Support for Reviewing Entailed Statements

The following procedure for systematic evaluation of a formal ontology is investigated: The ontology engineer reviews entailed statements, i.e. statements that logically follow from the ontology but are not explicitly stated. In case of agreement, a statement "passes" this review. In case of disagreement, the ontology engineer asks for an explanation for the statement in question. An explanation for an inferred statement is a minimal set of logical, explicitly made, statements that cause the inferred statement. In the ontology questionnaire, a tool to support exactly this procedure, partial support is given to the decision of how to act. The name "ontology questionnaire" expresses that this process can be seen as going through an automatically generated questionnaire for assessment of the ontology under review.
Within a complete ontology engineering process as depicted above in Fig. 2.3, such an evaluation procedure would take part within the "Formal Modelling" phase.

4.2.1 Walkthrough

Starting point for using the ontology questionnaire therefore is a content-wise arbitrary ontology. The ontology questionnaire displays inferred statements, as for instance

[1] In the "Active Ontology" tab, as "Inferred Axioms".

"Imaginary brainstorming is a creativity technique" for an ontology about the domain of innovation management (see Fig. 4.1). For each inferred statement, the ontology questionnaire offers a *justification*, i.e. a reason why the statement is entailed by the ontology. While reviewing the inferred statements, the user of the questionnaire (domain expert or ontology engineer) might disagree with an inferred statement and wish to remove it. This is now the interesting part: Removing an inferred statement is not directly possible precisely because it is inferred and not explicitly stated. The ontology questionnaire finds one or more explanations for an inferred statement, and the user can choose to remove the reason for the inference.

> **Example:** The statement "Imaginary brainstorming is a creativity technique" is inferred because of the explicit statements "Imaginary brainstorming is a brainstorming technique", "Brainstorming is an intuitive-creative creativity technique" and "An intuitive-creative creativity technique is a creativity technique"[2].

Usually there is more than one option (see Fig. 4.2), and sometimes it is also necessary to remove more than one statement. There is of course always the option to react more indirectly and change the structure and design of the ontology instead of simply removing some statements. Returning to the example from above, if the user disagreed with the inferred statement that "Imaginary brainstorming is a creativity technique" she can either delete the statement "Imaginary brainstorming is a brainstorming technique" or delete the statement "Brainstorming is a creativity technique" in the ontology in order to get rid of the unwanted inference. Note that side-effects are possible, i.e. it may be that such a deletion causes other inferences besides "Imaginary brainstorming is a creativity technique" to be removed as well.

4.2.2 Implementation

A prototypical implementation of an ontology questionnaire, the outcome of a bachelor thesis [132], was used for the experimental study described below. It is realised as a client-server application, and consequently users need only a browser to access the application. For reasoning, it accesses the open-source OWL DL reasoner Pellet [99, 120]. Justifications are computed using the built-in explanation functionalities in the Swoop [62, 126] libraries. Several decisions regarding the presentation to the user were made. First, in order to remove a selected axiom, the user needs to make a conscious choice. A minimal hitting set is suggested by the ontology questionnaire, but in case of multiple hitting sets of the same size, this suggestions is purely random and does not follow any further heuristics. Second, subsumption axioms of the form $C \sqsubseteq \top$, i.e. subsumption axioms stating that a concept is satisfiable, were excluded from display. Apart from this, no heuristics were applied for display. An improved version of the ontology questionnaire is integrated in MoKi [83, 110] and described in detail in Chap. 6.

[2] In this ontology there are two kinds of creativity techniques, one of which are called intuitive-creative and the other are called systematic-analytic.

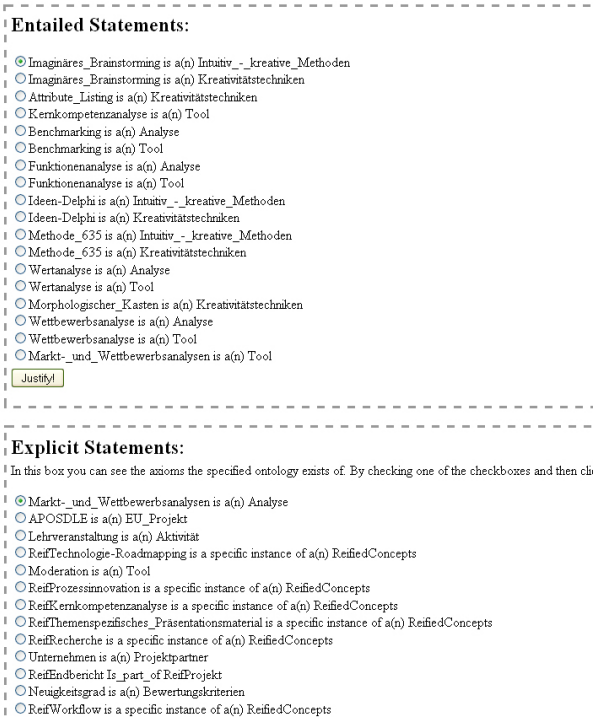


Fig. 4.1. The ontology questionnaire displays inferred subsumption axioms (upper box) as well as explicitly stated axioms (lower box).

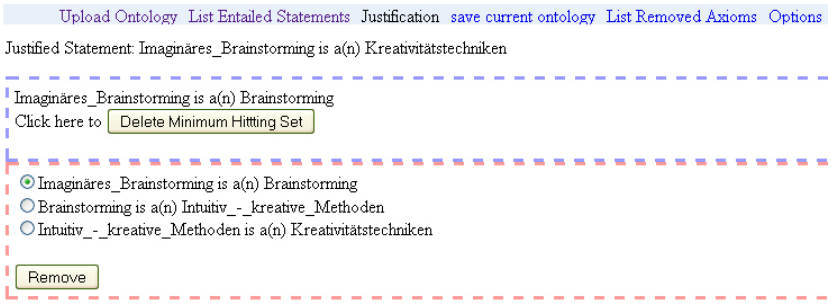


Fig. 4.2. Why is imaginary brainstorming a creativity technique? The ontology questionnaire retrieves an explanation for the inference. In case of disagreement, the user can choose to remove the suggested statement in the upper (blue) box, either of the three statements in the lower (pink) box, or deal more indirectly with it by changing the ontology's structure.

4.3 Formulation of Relevant Problems in DL

Some notions from the field of description logics are necessary in order to formally express the fundamental questions which need to be answered in an implementation of the ontology questionnaire: Which statements are entailed by an ontology? How must an ontology be modified in order to no longer entail a statement?

Given an ontology $\mathcal{O} = \{C_1 \sqsubseteq D_1 \ldots C_n \sqsubseteq D_n\}$, explicitly stated axioms can be removed from $\mathcal{O}$ by simply deleting them. For any inferred axiom α, i.e. $\mathcal{O} \models \alpha$ and $\alpha \notin \{C_i \sqsubseteq D_i | i = 1 \ldots n\}$, a more complex solution is obviously necessary. The task is formalised as follows: Modify $\mathcal{O}$ such that it results in a modified ontology $\mathcal{O}'$, for which $\mathcal{O}' \not\models \alpha$ is true. This modification is called "removing an inferred axiom from $\mathcal{O}$" throughout this paper, although clearly not the inferred axiom itself can be removed but only its causes. Since OWL DL is based on a monotonic logic, it can be seen at this point already that $\mathcal{O}' \subset \mathcal{O}$, i.e. some statements need to be removed from $\mathcal{O}$ in order to remove an inferred axiom α.

Two questions are fundamental to realising the ontology questionnaire as sketched conceptually above. Given an ontology $\mathcal{O} = \{C_1 \sqsubseteq D_1, \ldots C_n \sqsubseteq D_n\}$:

1. Apart from the explicitly modelled general inclusion axioms $C_i \sqsubseteq D_i, i = 1 \ldots n$, which further general inclusion axioms are entailed by $\mathcal{O}$?
2. Given a general inclusion axiom α such that $\mathcal{O} \models \alpha$, and $\alpha \notin \mathcal{O}$, how must $\mathcal{O}$ be modified into $\mathcal{O}'$ such that $\mathcal{O} \not\models \alpha$?

It is strictly necessary to answer the first question in order to construct the list of inferred statements which the ontology engineer shall review at all. It is not necessary in the technical sense but clearly desirable to be able to answer the second question in order to give hints on how to revise the ontology to the ontology engineer. As will be seen in the discussion later on however, the kind of support given is limited under a certain aspect.

4.3.1 Limitation to Explicitly Mentioned Concepts

In general, OWL DL ontologies entail an infinity of subsumption axioms. Thus, the number of entailed statements that are computed and displayed needs to be limited to a finite number for every practical purpose. Within the scope of the ontology questionnaire, such a limitation is achieved by considering only statements including concepts that are actually mentioned within the ontology in question. An arbitrarily complex concept C is *mentioned* within an ontology iff there exists an explicit statement involving C.

> **Example:** To illustrate this limitation consider an ontology in which $Human$ and $Mammal$ are concepts and $parent$ is a role. The axiom $Human \sqsubseteq Mammal$ entails an infinity of axioms like $\forall parent.Human \sqsubseteq \forall parent.Mammal$, $\forall parent.(\forall parent.Human) \sqsubseteq \forall parent.(\forall parent.Mammal)$ etc. If the complex concepts $\forall parent.Human$

and $\forall parent.Mammal$ are not mentioned explicitly in the ontology, the entailment $\forall parent.Human \sqsubseteq \forall parent.Mammal$ is not displayed in the questionnaire.

Then, listing entailed but not explicitly stated general inclusion axioms simply amounts to comparing concepts pair wise and checking which of the two is subsumed by the other, or if no subsumption is entailed either way. If any subsumption is found which is not explicitly stated, it counts as an entailed general inclusion axiom.

4.3.2 Justifications in OWL

The second question has been treated with slightly varying focus under the names of ontology debugging, ontology repair and ontology explanations (for a short discussion see Chapter 2.4). For the first implementation of the ontology questionnaire, I decided to follow the explanation mechanism that is available Swoop libraries [126] and since recently also in the OWL Api [131] which is based on justifications [61].
First, a justification of an entailment, which correspond to what has also been called "the reason for an entailed axiom" in this paper, is defined as follows.

Definition 1 (Justification). *Let $\mathcal{O} \models \alpha$ where α is an axiom and $\mathcal{O}$ an ontology. A set of axioms $\mathcal{O}' \subseteq \mathcal{O}$ is a justification for α in $\mathcal{O}$ if $\mathcal{O}' \models \alpha$ and $\mathcal{O}'' \not\models \alpha$ for every $\mathcal{O}'' \subset \mathcal{O}'$. (e.g. [61, p269])*

In other words, a justification is a minimal set of axioms from which the statement α follows. There may be more than one justification for any particular α. Clearly, in order to remove the entailment α, at least one axiom from *each* justification must be removed. Such a set of axioms (at least one from each justification) HS can be formally defined $\forall s \in S : HS \cap s \neq \emptyset$ where S is the set of all justifications. The technical term for HS is *hitting set*. This explains why sometimes multiple statements need to be deleted from an ontology in order to remove an inferred statement. Several algorithms to find all justifications for OWL entailments are described in [61].

4.4 Analysis of Benefits and Limitations Based on Typical Modelling Errors in OWL

Based on a study of typical modelling errors in OWL [109], I analysed which typical modelling errors can be detected through assessing inferred statements, and which would need further support. I note that the following analysis is of course valid for all ontology editing tools which give support by somehow displaying inferences. The following list of error patterns is directly taken from [109].
I count modelling errors which lead to erroneous inferences as amenable to the suggested support, and modelling errors which cause the lack of inferences as not amenable to the suggested support. The reason is that in order to detect *missing inferences*, the ontology engineer already needs to be sufficiently advanced.

Inclusion Axioms as Implication

Inexperienced ontology engineers sometimes understand the subsumption hierarchy as a vague, informal structure like for instance a folder hierarchy. However, subsumption has a very specific meaning, namely $A \sqsubseteq B$ means that everything that is of type A is also of type B.

> **Example:** $ReviewedArticle \sqsubseteq Article$ can be verbalised as "The concept $ReviewedArticle$ is subsumed by the concept $Article$", i.e. $ReviewedArticle$ is more specific than $Article$. It could also be expressed as implication "if something is a reviewed article then it is an article".

Although subsumption is one of the more simple to understand features of OWL, it is yet often not understood, and especially lost sight of in deep hierarchies.

> **Example:** From
>
> $$ConferenceProceedings \sqsubseteq ReviewedArticle$$
> $$ReviewedArticle \sqsubseteq Article$$
>
> it can be inferred that
>
> $$ConferenceProceedings \sqsubseteq Article$$
>
> Looking at this inference, it is unclear whether $ConferenceProceedings$ really denotes conference proceedings. In this case the explicitly stated hierarchy would be wrong. Alternatively, $ConferenceProcedings$ could mean something like "article published in conference proceedings" in which case the concept should stay where it is in the hierarchy but should be renamed.

A wrong specialisation hierarchy can easily be detected through the review of entailed statements.

Range and Domain Constraints

Range and domain constraints of relations in OWL are used for reasoning, while many people seem to expect range and domain constraints to actually restrict the values one can use relations with. However, these constraints work in the other direction: Whenever a relation is used in a role assertion, the subject and object are inferred to conform to the range and domain constraints.

> **Example:** If the relation $organises$ shall be used to specify that events are organised by persons, it makes sense to define $Person$ as the domain of $organises$, and $Event$ as its range. If someone then states $ProjectWorkshop \sqsubseteq Workhop \sqcap \exists organises^{-}.Project$, i.e. a $ProjectWorkshop$ is a workshop that is organised by a project, then $Project$ will be inferred to be a specific kind of $Person$.

Such surprising inferences can easily be detected through the review of entailed statements.

Difference Between Necessary *and* Necessary and Sufficient *Conditions*

A necessary condition corresponds to an inclusion axiom, while a necessary and sufficient condition corresponds to an equality axioms.

> **Example:** $Workshop \sqsubseteq \exists organises^{-}.Person$ essentially means that "a workshop is organised by someone" but not "everything which is organised by someone is a workshop".

These two often get mixed up (which may to some extent have something to do with the user interface in ontology editing tools). In order to exploit reasoning it is advantageous to find necessary and sufficient conditions even though this may be difficult, since nothing will be inferred to be subsumed under a class defined only by necessary conditions.
Assessing inferences can partly support detecting such an error, in case either (i) something is stated wrongly as necessary and sufficient condition and a second concept, or an instance, exists which is then classified (wrongly) or (ii) an expected inference is not made.

Errors in Understanding Logical Constructs

Finally, also common logical constructs may pose problems to ontology engineers. Reasons for this range from too little formal training to the fact that it is sometimes inherently difficult to sort out exactly what one actually wants to state. Particular difficulties seem to be the meanings of "and" (conjunction) and "or" (disjunction), which are slightly different than the less formal meaning given to these words in daily use of language. Additionally, it is sometimes not obvious in which order to put existential/universal restrictions, conjunction or disjunction, and negation.

> **Example:** For instance, it is wrong to define "A project funded by both an EU Funding Program and a National Program" as
>
> $$EUCofundedProject \doteq \exists fundedBy(EUFundingProgram \sqcap NationalFundingProgram)$$
>
> The above means that a $EUCofundedProject$ is funded by something which is both an $EUFundingProgram$ and a $NationalFundingProgram$. Instead, the following definition is closer to the intended meaning :
>
> $$EUCofundedProject \doteq \exists fundedBy.EUFundingProgram \sqcap \exists fundedBy.NationalFundingProgram$$

Such modelling mistakes can only partly be detected by reviewing entailed statements, depending on whether the wrong definitions lead to any inferences at all.

Open World Assumption

Directly related to the issues pertaining to existential and universal restrictions, is the difficulty of understanding open world reasoning. Most people nowadays are socialised with database systems, as e.g. getting train times from a website. If some piece of information does not exist in a database, it is assumed not to exist at all (closed world reasoning). This contrasts with the open world assumption made in description logics and in OWL.

Example: In order to differentiate between EU projects which are solely funded by a European funding program and EU co-funded projects which are funded jointly by a European funding program plus some other funding program (e.g. a national program), it does not suffice to define a EU Project as a project which is funded by a EU funding program:

$$EUProject \doteq \exists fundedBy.EUFundingProgram$$

In this case, given the definition that a project co-funded by the EU is a project which is funded by the EU and additionally by some national funding program:

$$EUCofundedProject \doteq \exists fundedBy.EUFundingProgram \sqcap \exists fundedBy.NationalFundingProgram$$

$EUCofundedProject$ would be inferred to be a specific kind of $EUProject$. This is undesirable if we wanted to express that a $EUProject$ is *only* funded by a European funding program. Such an undesired inference, originating from misunderstanding open world reasoning, can be detected through a review of entailed statements.
On the other hand, if the definition of a EU project is correctly extended to

$$EUProject \doteq \exists fundedBy.EUFundingProgram \sqcap \forall fundedBy.EUFundingProgram$$

then an FP7 project defined as $FP7Project \doteq \exists fundedBy.\{FP7\}$, where $FP7 \sqsubseteq EUFundingProgram$ is known, would still not be known to be a EU project because a reasoner would assume that maybe there is some additional funding by a national program that is not known to the reasoner. Hence, in this case no inference is made and this problem related to open world reasoning would only indirectly be visible through the lack of a probably expected result.

Therefore, modelling errors related to open world reasoning can be detected partly through the review of entailed statements.

Universal vs. Existential Restriction

In this error pattern I include also what is discussed as "trivial satisfiability of universal restriction" in [109]. This error pattern describes the difficulty non-expert ontology engineers have with differentiating between the two.

Example: Stating
$Workshop \sqsubseteq \exists organises^{-}.Person$ means that every workshop is organised by at least one person, but additionally there could be more people, or other things, e.g. werewolves who help organising.

In order to exclude this possibility, an additional universal restriction along the lines $Workshop \sqsubseteq \forall organises^{-}.Person$ is required.

Example: Stating only that $Workshop \sqsubseteq \forall organises^{-}.Person$ means that everyone who organises a workshop is a Person, but it could be that a workshop is organised by no one (trivial satisfiability).

Both omissions are therefore not easily visible as inferences, apart through the lack of expected inferences.

Failure to Make Information Explicit

Inexperienced ontology engineers often assume that properties of concepts are obvious to everyone, including a reasoner, because of their background knowledge about the concepts that are usually denoted by a given symbol (word). It is a learning process to realise that humans need good verbal descriptions, and machines need good formal descriptions. Additionally, being more precise is the whole purpose of creating an ontology.

Example: The similarity between the concepts "Brainstorming" and "Imaginary Brainstorming" is available to humans because of the given names. It is a mistake however to not connect them formally, for instance through a subsumption relation.

Example: Very often, disjointness statements are missing from an ontology. For instance, something that is a "Creativity Technique" is not a "Project Partner" in any intended world.

Missing information is not systematically detected through assessing inferred statements.

In general it can be seen that through the study of entailments, *missing information can not be detected*. Naturally this points strongly into a direction for future research, namely to find potentially missing information and suggest axioms for addition. Related to this is the problem that expected inferences that do not appear are not directly revealed by reviewing inferred statement. Instead, the ontology engineer can only remember expected inferences and try to find ways to produce them. Nonetheless, some modelling errors can be directly unveiled through reviewing inferred entailments. The following experimental study sheds some more light on potentials of the proposed procedure.

4.5 Experimental Study

4.5.1 Application Setting

In the scope of the APOSDLE [3, 71] project, five learning domains were modelled. For each learning domain an ontology about relevant topics for learning (domain model), an ontology about work processes in which learning should take place (task model) and an ontology about skills required by these tasks (skill model) were created. These models were created following a modelling methodology, the Integrated Modelling Methodology (IMM), which is described in detail in [41, 71]. The IMM has been specifically designed to support non-expert knowledge engineers and consequently offers revision support at all its steps [97]. Therefore all ontologies which "reached" the point of being reviewed in the ontology questionnaire had already gone through several different reviews, as is sketched in Fig. 4.3.

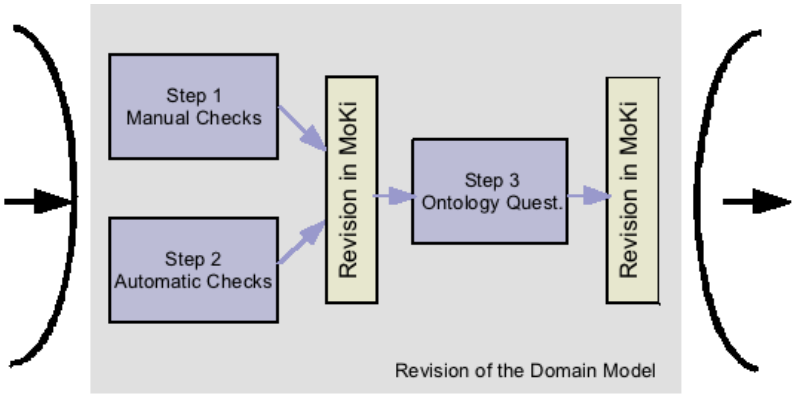


Fig. 4.3. A zoom into the revision process of the domain model in the Integrated Modelling Methodology (IMM). All revision steps imply, where the necessity of changes in the models is identified, an iteration loop going back to revising the domain model in MoKi and another entry into the revision process.

The five learning domains that were created concern the topics of consulting on industrial property rights (ontology with 95 concepts), the Information Technology Infrastructure Library (ontology with 101 concepts), electromagnetic simulation (ontology with 116 concepts), innovation management (ontology with 134 concepts) and statistical data analysis (ontology with 71 concepts). Each domain model configures an APOSDLE installation at an application partner of the project [72], and each domain model except the model about RESCUE was reviewed using the prototypical ontology questionnaire. The review was performed by employees of the application partner companies, who had knowledge both of the domain and of ontology engineering, without necessarily being an expert in either field. These persons are called "responsible ontology engineers" in the following description of the study.

4.5.2 Evaluation Procedure and Results

At each application partner, the responsible ontology engineer completed the ontology questionnaire. When this was done, I conducted a telephone interview with each ontology engineer to elicit information about the invested effort, the perceived benefit and consequences of reviewing the ontology via the ontology questionnaire. In order to guarantee some confidentiality to the involved application partners, the following analysis does not refer to the domain ontologies by their domain.
The questions were:

- **Invested effort:** How long did it take to go through the questionnaire? Was this a difficult task?
- **Benefit:** Did the ontology engineers benefit from going through the questionnaire?
- **Effect:** Did going through the questionnaire trigger any changes in the model? If yes, which?

The effort to go through the questionnaire ranged from half an hour to three hours. In two cases, the domain ontology was a simple taxonomy with only two levels. Therefore they did not contain any implicit subsumptions besides the asserted subsumptions, and the ontology questionnaire did not offer an additional benefit. In two cases, no change in the underlying domain model was triggered by its application. In a third case, which took three hours, a discussion about the organisation of domain concepts was triggered. Finally it was decided to split one concept into three more specific concepts in order to obtain a more valid hierarchy. This indicates the usefulness of the ontology questionnaire to support *reflection* of the created model.
All ontology engineers stated that they benefited from going through the questionnaire, since this allowed them to review in a slightly different form (i.e. minus everything they explicitly stated anyway) the domain ontology they had created. Interestingly, also the two knowledge engineers who did not change their ontology due to review in the ontology questionnaire nevertheless noted explicitly that they liked the tool, but had already revised thoroughly their ontology before applying it in the scope of the IMM. This reflects well on the IMM, in that it encourages sufficient incentives and ideas for revision, but unfortunately does not give more constructive feedback on the ontology questionnaire.
Concerning the usability, the ontology engineers were satisfied in general with the rather straight-forward presentation, but dissatisfied with the presentation of statements as lists without prioritisation. Both inferred statements and the originally explicated ontology were presented as such lists.

4.5.3 Discussion of the Experimental Study

Although it is difficult to derive general conclusions from just five use cases, such an experience nevertheless gives useful indications. One, the effort to use the ontology questionnaire is manageable for ontologies containing around 100 concepts. Two, it is desirable to develop a prioritised list of statements to review. Such a prioritisation will become more important if larger ontologies are to be reviewed. Three, all involved persons judged such a review as being worth the time spent. This is indeed highly

relevant to keep up the motivation of people in carrying out the ontology evaluation. Together, this feedback indicates that further research into the proposed direction of ontology evaluation is worthwhile. One point which definitely influenced the given feedback however was that the ontology questionnaire was applied in situations where the creation of the models was embedded in a thorough modelling methodology. From some statements given by the ontology engineers it can be suspected that the small number of changes triggered is due to the rigour of revisions which happened *before* the ontology questionnaire was even applied.
Furthermore, I have seen in some more informal experiments than the above described application study, that removing existing statements was not actually the method of choice to deal with an undesired inference. Frequently, more subtle actions were taken, which ranged from simple renaming to removal/addition of concepts and relations, and to a complete restructuring of the representation.

4.6 Discussion

Based on the analysis and the experimental study given a first prototype of the ontology questionnaire, the following improvements were made during integration of the ontology questionnaire within MoKi: A first improvement of the ontology questionnaire concerned the inclusion of assertional axioms in the ontology questionnaire. This means that assertional axioms such as for instance “Alice is a project leader” and “Alice works together with Bob” also show up as inferred statements. This does not present a major innovation, and such functionality is already integrated for instance in Swoop. In the context of the experiment carried out so far, not displaying inferred assertional axioms did not represent a limitation since the ontologies in question only contained concepts. Directly following user feedback, prioritisation of inferred statements was also taken up in MoKi’s ontology questionnaire. A simple heuristic is to prioritise all subsumption axioms which do *not* stem from the simple transitivity of subsumption. However, such axioms shall nevertheless be displayed in order to make explicit the meaning of a hierarchy in description logic, which contrasts with hierarchies frequently encountered in other settings such as folder hierarchies or keyword hierarchies for classifying books. Another issue is the representation of side-effects of removing axioms. When deleting asserted axioms, it is possible that not only the undesired entailment but also some actually desired inferences are also “removed”. Such information was not presented in the ontology questionnaire used in the experimental study described above, but is present in MoKi’s ontology questionnaire. However, there are also directions for further improvement that require more in-depth research and thus cannot be said to have been solved in MoKi’s ontology questionnaire either. One issue concerns the representation of axioms themselves. Although a simple attempt at expressing logical axioms in a close-to-natural-language form was made, no sophisticated verbalisation functionality was employed. that a more closely natural language representation of the logical axioms would push the ontology questionnaire more into the direction of the knowledge experts, i.e. more towards the source of knowledge. A critical point is that although work on English verbalisation for OWL exists (e.g. [60]), this seems to be lacking for most other languages.

An inherently more complex open issue is how to improve the hints given on how to modify the ontology in order to improve it. Currently, hints can be given as to which axioms might be removed via explanations of undesired inferences. The ultimate goal however should be to provide also hints on which (kinds of) axioms then to add. The rationale behind this goal is of course that if many undesired inferences are removed by removing explicit statements, in the end only the empty ontology with no undesired inferences remains. A first step in this direction could be based on the catalogue of modelling mistakes in [109] and take the form of simple links to modelling guidelines or specific examples in tutorials. In general however it is not obvious how such complex ontology revision behaviour could be supported automatically.

To conclude, the hypothesis that reviewing inferred statements is a valuable part of an ontology evaluation procedure which contributes to evaluating an ontology's correctness is supported by the analytic discussion and experimental study which have been carried out. Reviewing inferred statements can be tightly integrated with modelling, and lends itself naturally to being integrated in a modelling environment.

5

Assertional Effects of Ontology Editing Activities

Disregarding very simple ontologies, it is not trivial even for expert ontology engineers to grasp the effects of changes to an ontology. Given a knowledge base, which includes not only terminological and role axioms but also data, I propose to capitalize on the existence of data in order to support contributors during ontology editing. The goal is to make the effects of removing or adding terminological axioms to a knowledge base visible in terms of *knowledge lost or gained about data*. On the one hand, the formulation of effects in terms of assertions about instance data is expected to be easily intelligible also for contributors with little knowledge engineering expertise. On the other hand, when an ontology is used to define the semantics of data as it happens in a knowledge base, the ontology and the data must both correspond to the same view of the world. Presenting contributors to a knowledge base with knowledge lost or gained about data allows checking for this correspondence.
The contribution of this part of the thesis lies first in motivating the presentation of knowledge lost or gained about data to knowledge base contributors, second in the presentation of a formal characterization of the problem, and third in putting down conditions on the underlying description logic under which the problem is then decidable. From this theoretical discussion, a decision algorithm immediately follows. An approximation of the decision algorithm for assertional effects of ontology editing activities has been integrated in MoKi (Chap. 6).

5.1 Motivation

Consider the following excerpt from a knowledge base about common geographic knowledge and historical persons: It is known that "Crete and Kos are islands" (5.1,5.2), "Crete is located in Greece" (5.3) and "Crete is located in the Mediterranean Sea" (5.4) These facts are embedded in a knowledge base where a set of key concepts such as islands, areas of land vs. areas of water, nations etc. are defined along a hierarchy as illustrated in Figure 5.1, and individuals are assigned in general to sensible concepts as their types (e.g. "The Mediterranean Sea is a water area"). A contributor wants to formalise the concept $Island$ and adds an axiom stating that all islands are located only in areas of water (5.5).

$$Island(crete) \tag{5.1}$$
$$Island(kos) \tag{5.2}$$
$$locatedIn(crete, greece) \tag{5.3}$$
$$locatedIn(crete, mediterranean) \tag{5.4}$$
$$Island \sqsubseteq \forall locatedIn.WaterArea \tag{5.5}$$

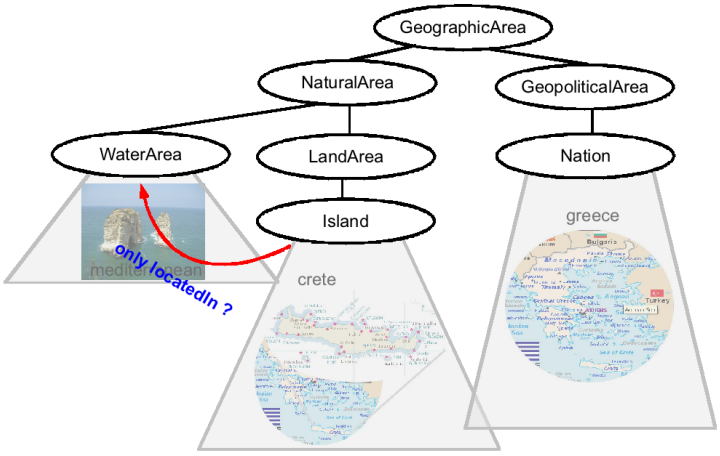


Fig. 5.1. The knowledge base given in (5.1) - (5.5) is a knowledge base about common geographic knowledge and historical persons. It is defined on top of an ontology which expresses that islands are a specific kind of land areas, both water and land areas are natural areas, nations are geopolitical areas, and everything mentioned so far is a geographic area. In correspondence with common knowledge, the Mediterranean Sea is seen as a water area, kos and crete as islands, and greece is a nation. The figure illustrates the tentative addition of the constraint "Every island is only located in water areas" (5.5).

The most obvious consequence of the additional axiom is that $greece$ will then be inferred to be a $WaterArea$, which is surely not intended. As a further example consider that the universal restriction $(\forall locatedIn.WaterArea)(kos)$ does not immediately imply the existence of a relation $locatedIn$ for kos. This has been pointed out in Section 4.4 above under the name of "trivial satisfiability of universal restrictions". Such a consequence, or rather the lack of such a consequence, can become visible by considering how the additional axiom affects the models of the knowledge base. Among the constraints added to the models of the original knowledge base are the additional type $WaterArea$ to $greece$ and $mediterranean$, but no additional relation $locatedIn$ for kos. In order to remedy this omission, an existential restriction such as for instance $Island \sqsubseteq \exists locatedIn.WaterArea$ should be added.
From the logical point of view, the new knowledge base is consistent. However, a conceptual inconsistency exists due to the fact that the ontology entails a statement that is inconsistent with the world that the modeller has in mind, namely $WaterArea(greece)$. Hence, the resulting theory will represent the point of view of the modeller inadequately or incorrectly. It must be pointed out that this problem is of conceptual nature,

in contrast to more formal problems like an unsatisfiable concept or a logically inconsistent knowledge base. Apart from inconsistent vocabulary use, sources of conceptual inconsistency also include differing design preferences by contributors, divergent to incompatible underlying views on the domain or modelling errors originating in (too) little experience with formal knowledge representation.
Considerations of this kind led to the study of terminological axioms on the instance data in a knowledge base. In particular I consider *inferred facts caused by additional terminological axioms* as effects. Where terminological axioms are removed from a knowledge base, *inferences that are lost* and thus "not known anymore" by the knowledge base are considered as effects. By ontology editing activities I understand in the scope of this work the explicit addition or removal of terminological axioms to/from an ontology. Such an activity is called an *ontology edit*. This work is not concerned with the manipulation of role axioms or facts.

Giving immediate feedback on the effect of ontology edits in terms of concrete individuals gives the contributing users an easy means to review their actions in the light of effects on the whole. This addresses the observation that an inherent difficulty in ontology engineering is that such effects are not obvious. By contrast, in software engineering the consequences of one's changes are immediately executable and thus visible (see also [75]). It is precisely this point that a reasoning service computing the effects of axioms on instance data changes. From this perspective, effects in terms of instance data serve as examples of how the terminological axioms will "work" on the knowledge base's data.
A reasoning service that computes effects of terminological axioms on concrete data is therefore especially relevant for knowledge bases maintained in the spirit of Web 2.0, where contributors add not only content, as e.g. in since Wikipedia [142], or facts, as e.g. in the Semantic Wikipedia envisioned in [67], to a knowledge base but also add terminological axioms to the underlying ontology. In such a setting, (i) frequent ontology edits are expected, (ii) the group of contributors is expected to be heterogeneous in terms of knowledge engineering expertise, but also in terms of views on the knowledge itself and (iii) it follows that such a knowledge base is in danger of becoming chaotic if not each contributor is able to judge the effects of her actions correctly and efficiently.

5.2 Assertional Effects of Ontology Editing Activities

The following definition restricts the meaning of assertional effects to *concept assertions*. Possible extensions with regard to these limitations are part of our ongoing research and discussed in Section 5.4.

Definition 2. (Assertional effects) *Let* $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$ *and* $\mathbf{KB}_T = (\mathcal{T}_0 \cup \mathcal{T}, \mathcal{R}, \mathcal{A})$. *Let* $\Sigma = (N_C, N_R, N_I)$ *be the vocabulary in which* $\mathbf{KB}_T$ *is formulated and let* $\mathcal{DL}$ *be the DL in which the assertional effects shall be formulated.*

– $C(x)$ *such that* $C \in \mathcal{C}(\Sigma, \mathcal{DL})$ *is an assertional effect of* $\mathcal{T}$ *on* $\mathbf{KB}$ *iff* $\mathbf{KB} \not\models C(x)$ *and* $\mathbf{KB}_T \models C(x)$.

- $\mathcal{T}$ *affects an individual* $x \in N_I$ *in* $\mathbf{KB}$ *iff an assertional effect* $C(x)$ *of* $\mathcal{T}$ *on* $\mathbf{KB}$ *exists.*
- $\mathcal{T}$ *affects a knowledge base* $\mathbf{KB}$ *iff an assertional effect* $C(x)$ *of* $\mathcal{T}$ *on* $\mathbf{KB}$ *exists.*

$\mathbf{KB}_T$ is assumed to be consistent and N_I is assumed to be non-empty, i.e. the knowledge base knows about at least one individual. Individuals can occur in $\mathcal{A}$ or, if the DL allows for nominals as $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$, also in $\mathcal{T}$.

Example: Assume $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$ is the knowledge base described above in (5.1) - (5.4). It contains the facts that Crete and Kos are islands, that Crete is located in Greece, and that Crete is located in the Mediterranean Sea. $\mathcal{T}_0$ and $\mathcal{R}$ are empty. Then, (5.5), which states that islands are located only in water areas, is added. The resulting knowledge base is called $\mathbf{KB}_T = (\mathcal{T}_0 \cup \mathcal{T}, \mathcal{R}, \mathcal{A})$. $WaterArea(greece)$ is an assertional effect of $\mathcal{T}$ on $\mathbf{KB}$ since $\mathbf{KB} \not\models WaterArea(greece)$, but $\mathbf{KB}_T \models WaterArea(greece)$.

The above definition can be applied to both ontology editing activities. If $\mathcal{T}$ is added to $\mathbf{KB}$, then the assertional effects of $\mathcal{T}$ on $\mathbf{KB}$ represent knowledge about individuals which is gained. If $\mathcal{T}$ is removed from $\mathbf{KB}_T$, then the assertional effects of $\mathcal{T}$ on $\mathbf{KB}$ represent the knowledge about individuals which is lost.

5.2.1 Deciding the Existence of Assertional Effects

Since the set $\mathcal{C}(\Sigma, \mathcal{DL})$ is in general infinite for DLs equally or more expressive than $\mathcal{ALC}$, I consider at first the decidability of the general question whether a particular TBox $\mathcal{T}$ affects a particular knowledge base $\mathbf{KB}$. In order to do so, I first define the notion of *reachability* of a concept C from an individual x in a knowledge base $\mathbf{KB}$. Let R^- denotes an inverse role such that $(R^-)^{\mathcal{I}} = \{(y,x)|(x,y) \in R^{\mathcal{I}}\}$, and let $N_R^- = \{R|R \in N_R \text{ or } R^- \in N_R\}$ in description logics which include inverse roles.

Definition 3. (Path) $w_0 R_1 w_1 \ldots R_n w_n$ *is called a path in an interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *if and only if for* $i = 1 \ldots n$ *it holds that* $w_i \in \Delta^{\mathcal{I}}$*, and* $(w_{i-1}, w_i) \in R_i^{\mathcal{I}}$ *and* $R_i \in N_R^-$.

Definition 4. (Reachability) *A concept* C *is reachable from* $x \in N_I$ *w.r.t.* $\mathbf{KB}$ *iff there is a model* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *of* $\mathbf{KB}$ *in which* $x^{\mathcal{I}} = w_0$*, and a path* $w_0 R_1 w_1 \ldots R_n w_n$ *exists in* $\mathcal{I}$ *such that* $w_n \in C^{\mathcal{I}}$.

In other words, C is reachable from x in $\mathbf{KB}$ iff either $C(x)$ or $\exists R_1 \ldots R_n.C(x)$ for $n > 0$ is satisfiable w.r.t. $\mathbf{KB}$.
The definition of reachability is motivated by the fact that it can be shown that reachability is equivalent to the existence of assertional effects. First, this is shown under the condition that the DL in question is decidable under a tableaux decision procedure. Later, I will show a small generalisation.

Theorem 1 *In description logics weaker or equivalent to* $\mathcal{SROIQ}$*,* $\mathcal{T} = \{\top \sqsubseteq C_T\}$ *affects* $\mathbf{KB}$ *iff an individual* x *exists in* $\mathbf{KB}$ *such that* $\neg C_T$ *is reachable from* x *in* $\mathbf{KB}$.

The proof for Theorem 1 is based on the fact that for $\mathcal{SROIQ}$ a tableaux decision procedure exists. For exact descriptions of tableaux algorithms I refer the reader to [58] where a tableaux decision procedure for $\mathcal{SHOIQ}$, a language that encompasses $\mathcal{SHOIN}$, is described and to [57] for a tableaux decision procedure for $\mathcal{SROIQ}$. Nonetheless, some basic notions related to tableaux decision procedures are reviewed before delving into the proof for Theorem 1.

A *completion graph* for a knowledge base $\mathbf{KB}$ formulated in the vocabulary Σ and the description logic $\mathcal{DL}$ is a labelled directed graph $G = (V, E, \mathcal{L}, \not\doteq)$ where each node $x \in V$ is labelled with a set $\mathcal{L}(x) \subseteq \mathcal{C}(\Sigma, \mathcal{DL})$ and each edge $(x, y) \in E$ is labelled with a set of role names such that $\mathcal{L}(x, y) \subseteq N_R^-$. The symmetric binary relation $\not\doteq$ between the nodes of G is used to store inequalities between nodes. A set of *completion rules* are used to manipulate the underlying completion graph(s).
A completion graph contains a *clash* iff a label $\mathcal{L}(x)$ contains either $\bot$ or both A and $\neg A$. A completion graph which does not contain a clash is called *open*, while a a completion graph which contains a clash is called *closed*. A completion graph to which no more completion rules apply is called *complete*.
More specifically I make use of the following connections between consistency, completion graphs and models, which hold whenever a tableaux algorithm has been shown to be a decision procedure for a DL language.

- **If a knowledge base is consistent, then an open and complete completion graph can be constructed.** This is the basis of the completeness property of tableaux decision algorithms.
- **Every open and complete completion graph can be translated into a model.** This is the basis of the correctness property of tableaux decision algorithms. The relevant part of this translation is the following: If a completion graph $G = (V, E, \mathcal{L}, \not\doteq)$ is translated into a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, then every node $x \in V$ corresponds to at least one node $w \in \Delta^{\mathcal{I}}$ such that for all $C \in \mathcal{C}(\Sigma, \mathcal{DL})$, $C \in \mathcal{L}(x)$ if and only if $w \in C^{\mathcal{I}}$. For every edge $(x, y) \in E$, there are at least two nodes $v, w \in \Delta^{\mathcal{I}}$ corresponding to x and y such that for all $R \in N_R^-$, $R \in \mathcal{L}(x, y)$ if and only if $(v, w) \in R^{\mathcal{I}}$.
- **All nodes in a completion graph for a knowledge base $\mathbf{KB}$ are either an individual or connected to an individual iff $\mathbf{KB}$ contains at least one individual.** This property follows from the procedure of tableaux-based algorithms, which start with a set of initial nodes consisting of individual nodes (individuals or nominals) and create only new nodes which are connected to an existing node. If $\mathbf{KB}$ does not contain any individual however, tableaux-based algorithms typically start with an "invented" initial single node labelled with C_T given that $\mathcal{T} = \top \sqsubseteq C_T$. Remember that the existence of individuals was assumed as a precondition.

In the following I say that a concept C can be consistently added to the label of a node $w \in V$ of a completion graph $G = (V, E, \mathcal{L}, \not\doteq)$ iff G can be completed into an open and complete graph using the completion rules after C is added to the label of w.

Proof. Let $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$ and $\mathbf{KB}_T = (\mathcal{T}_0 \cup \mathcal{T}, \mathcal{R}, \mathcal{A})$, and $\mathcal{T} = \{\top \sqsubseteq C_T\}$.
$\Leftarrow$ **If $\neg C_T$ is reachable from an individual x in $\mathbf{KB}$, $\mathcal{T}$ affects x in $\mathbf{KB}$**

If $\neg C_T$ is reachable from an individual x, there is an $n \in \mathbb{N}_0$ such that if $n = 0$ then $\neg C_T(x)$ or if $n > 0$ then $(\exists R_1 \ldots R_n.\neg C_T)(x)$ is satisfiable w.r.t. $\mathbf{KB}$. If $n = 0$ this means that $\mathbf{KB} \not\models C_T(x)$ or if $n > 0$ this means that $\mathbf{KB} \not\models (\forall R_1 \ldots R_n.C_T)(x)$. On the other hand, $\top \sqsubseteq C_T \models \top \sqsubseteq \forall R_1 \ldots R_n.C_T$ is trivially true, and therefore $\mathbf{KB}_T \models C_T(x)$ and $\mathbf{KB}_T \models (\forall R_1 \ldots R_n.C_T)(x)$.
$\Rightarrow$ **If $\mathcal{T}$ affects $\mathbf{KB}$, then an individual x exists such that $\neg C_T$ is reachable from x in $\mathbf{KB}$**
Proof by contradiction, i.e. it is assumed that an assertional effect exists but that $\neg C_T$ is not reachable from x in $\mathbf{KB}$.
Let $D(x)$ be one of the possibly many assertional effects of $\mathcal{T}$ on $\mathbf{KB}$.
Since $\neg D(x)$ is consistent w.r.t. $\mathbf{KB}$, an open and complete completion graph $G = (V, E, \mathcal{L}, \dot{\neq})$ for $\mathbf{KB}$ can be constructed such that $\neg D \in \mathcal{L}(x)$.
$\neg D(x)$ is inconsistent with the extended knowledge base $\mathbf{KB}_T$. Therefore the following procedure, extending the open and complete graph G leads to only closed completion graphs: Add C_T to the label of a node in V. Follow the completion rules, and ensure that nodes newly created in the process are also labelled with C_T. Repeat for all nodes in G until for one node w_C adding C_T to $\mathcal{L}(w_C)$ leads to only closed completion graphs.
Then however, $\neg C_T$ can be consistently added to $\mathcal{L}(w_C)$.
Since all nodes in a completion graph either are an individual node or connected to one, there is then an individual $y \in V$ from which a path to w_C can be constructed. Call this path $yR_1w_1 \ldots R_nw_C$. If $n = 0$, then $y = w_C$.
G can be translated into a model $\mathcal{I}$ such that $w_C \in (\neg C_T)^{\mathcal{I}}$, and $(y, w_1) \in R_1^{\mathcal{I}}$, $(w_1, w_2) \in R_2^{\mathcal{I}}, \ldots, (w_{n-1}, w_C) \in R_n^{\mathcal{I}}$. Then, $\neg C_T$ is reachable in $\mathbf{KB}$ from y.

As by-product from the equivalence between assertional effects and reachability the following corollary can be derived.

Corollary 1 *If $\mathcal{T} = \{\top \sqsubseteq C_T\}$ affects $\mathbf{KB}$, then an assertional effect $C(x)$ exists such that $C \doteq \forall R_1 \ldots R_n.C_T$ and $R_i \in \Sigma, i = 1 \ldots n$. If $n = 0$, this corresponds to $C = C_T$. n is bounded by the maximal number of nodes in completion graphs for the corresponding description logic.*

Then, the following theorem about decidability follows immediately:

Corollary 2 *The existence of assertional effects of $\mathcal{T}$ on $\mathbf{KB}$ can be decided in all logics decidable under tableaux algorithms.*

Some interesting observations follow from the these results: First, in order to express such assertional effects, DLs which contain at least $\mathcal{ALC}$, i.e. which include negation over complex concepts and qualified universal/existential quantification, are required. Second, if an assertional effect exists, then not all assertional effects are necessarily of the form $C(x)$ with $C \doteq \forall R_1 \ldots R_n.C_T$. As a simple example consider extending the knowledge base $\mathbf{KB} = \{R(a, b)\}$ with the TBox $\mathcal{T} = \{\top \sqsubseteq \forall R.A\}$. In this case, the effect $(\forall R.A)(a)$ will be found if looking for assertional effects of the above-mentioned pattern, but $A(b)$ will be missed. Third, although bounded, n can be quite high: In $\mathcal{SHOIQ}$ already, n is bounded double-exponentially with the size of the

closure of $\mathcal{T}_0$ and $\mathcal{A}$ (the smallest set containing all subconcepts of $\mathcal{T}_0$ and $\mathcal{A}$ that is also closed under negation) and the number of roles and inverses occurring in the input knowledge base $\mathbf{KB} = (\mathcal{T}_0, \mathcal{A})$ [58][1].

5.2.2 Generalisation to DLs with the Connected Model Property

In description logics which additionally provide role union and the reflexive-transitive closure (Kleene operator *), "C is reachable from x in $\mathbf{KB}$" can also be expressed as $\exists(\bigsqcup_{R_i \in N_R^-} R_i)^*.C(x)$. This led to the question of whether Theorem 1, which states equivalence between assertional effects and reachability, can be generalised to require a more general property of the underlying description logic than being decidable under a tableaux decision procedure. Indeed, it can be shown that only the *connected model property* is required:

Definition 5. (Connected model) *A model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is connected if and only if for every $w \in \Delta^{\mathcal{I}}$ there is an element $x \in N_I$, $x^{\mathcal{I}} = w_o$ such that there is a path $w_0 R_1 w_1 \dots R_n w$ in $\mathcal{I}$.*

A logic is said to have the *connected model property* if every satisfiable concept or consistent knowledge base has a connected model. Since tree and forest models are connected models, all logics which enjoy the tree (forest) model property, also have the connected model property.

Theorem 2 *In description logics with the connected model property $\mathcal{T} = \{\top \sqsubseteq C_T\}$ affects $\mathbf{KB}$ iff an individual x exists in $\mathbf{KB}$ such that $\neg C_T$ is reachable from x in $\mathbf{KB}$.*

Proof. Let $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$ and $\mathbf{KB}_T = (\mathcal{T}_0 \cup \mathcal{T}, \mathcal{R}, \mathcal{A})$, and $\mathcal{T} = \{\top \sqsubseteq C_T\}$.
$\Leftarrow$ If $\neg C_T$ is reachable from an individual x in $\mathbf{KB}$, $\mathcal{T}$ affects x in $\mathbf{KB}$
This direction is the same as in the proof for the tableaux-based Theorem 1.
$\Rightarrow$ If $\mathcal{T}$ affects $\mathbf{KB}$, then an individual x exists such that $\neg C_T$ is reachable from x in $\mathbf{KB}$
Proof by contradiction, i.e. it is assumed that an assertional effect exists but that $\neg C_T$ is not reachable from x in $\mathbf{KB}$.
Let $D(x)$ be one of the possibly many assertional effects of $\mathcal{T}$ on $\mathbf{KB}$.
Then there is a connected model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\neg D(x)$ w.r.t.
$\mathcal{I}$ is not a model however of the extended knowledge base $\mathbf{KB}_T$, since $\neg D(x)$ is inconsistent w.r.t. $\mathbf{KB}_T$.
Therefore, there is an element $w \in \Delta^{\mathcal{I}}$ such that $w \notin (C_T)^{\mathcal{I}}$. Otherwise, $\Delta^{\mathcal{I}} = (C_T)^{\mathcal{I}}$ and $\mathcal{I}$ would also be a model of $\mathbf{KB}_T$.
Since $w \notin (C_T)^{\mathcal{I}}$, it holds that $w \in (\neg C_T)^{\mathcal{I}}$.
Because of the connected model property, there is then an individual $y \in N_I$ and $y^{\mathcal{I}} = w_0$ such that there is a path $w_0 R_1 w_1 \dots R_n w_C$. If $n = 0$ this means that $w_0 = w_C$.
Then, $\neg C_T$ is reachable from the individual y, which contradicts the assumption that $\neg C_T$ is not reachable from any individual in the knowledge base.

[1] In [58], only the size of the closure of $\mathcal{T}$ is mentioned, however this is because they note in the beginning that they incorporate the ABox into the TBox by expressing every $C(x)$ as $x \sqsubseteq C$ and every $R(x, y)$ as $x \sqsubseteq \exists R.y$.

Therefore, the problem of deciding whether $\mathcal{T}$ affects $\mathbf{KB}$ can be posed as consistency checks of the form $\exists(\bigsqcup_{R_i \in N_R^-} R_i)^*.\neg C_T(x)$ for all $x \in N_I$ in DLs which have the connected model property and the required concept and role constructors. Such a reformulation has the advantage of posing the original problem as a standard reasoning problem. Naturally, this reformulation only makes sense if the resulting logic is also decidable. A logic for which all these requirements hold is for instance $\mathcal{ALCQIb}^+_{reg}$, which has been shown to be decidable [95]. Unfortunately no reasoners exist to date for this DL to the best of our knowledge. Note also, that specifically $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$, the DLs underlying OWL 1 DL and OWL 2 DL, are not contained within $\mathcal{ALCQIb}^+_{reg}$ since the latter misses nominals.

5.3 Related work

Both conceptually and technically, conservative extensions in description logics are a closely related topic. In short, deciding conservativity for two TBoxes $\mathcal{T}_0$ and $\mathcal{T}$ means finding out whether $\mathcal{T}_0 \cup \mathcal{T}$ entails any inclusion axioms expressible in a given vocabulary and a given DL that are not entailed by $\mathcal{T}_0$ alone [44, 76]. It can easily be shown that non-conservativity of $\mathcal{T}_0 \cup \mathcal{T}$ with respect to $\mathcal{T}_0$ is a precondition for the existence of TBox effects of $\mathcal{T}$ on $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$: By inventing an ABox $\mathcal{A} = \{\top(x)\}$, any inclusion axiom entailed by $\mathcal{T}_0 \cup \mathcal{T}$ but not by $\mathcal{T}_0$ alone produces an effect on x.

However, non-conservativity is not a guarantee for the existence of effects. To illustrate the latter, consider the following knowledge base $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$:

$$\begin{aligned} &A \sqsubseteq \forall R.A \\ &A \sqsubseteq \exists R.A \\ &A(x) \end{aligned} \tag{5.6}$$

which is extended with $\mathcal{T}$:

$$\top \sqsubseteq A \tag{5.7}$$

Obviously, $\mathcal{T}_0 \cup \mathcal{T}$ is not a conservative extension of $\mathcal{T}_0$ w.r.t. $\Sigma = \{A, R, x\}$ and the description logic $\mathcal{ALC}$. There are however, no effects on the individual x, since $\mathbf{KB}$ already entails all types that can be constructed from the vocabulary $\{A, R\}$ for x in $\mathcal{ALC}$. Complexity results for deciding conservativity therefore give a *lower bound* on the complexity of deciding effects according to the deductive definition.

Depending on the choice of vocabulary, conservativity can be reduced to subsumption if the full vocabulary of $\mathbf{KB}$ is considered [44]. Interestingly, the problem becomes harder if the vocabulary under consideration is a subset of the vocabulary used by $\mathbf{KB}$. Then, conservativity is decidable up to $\mathcal{ALCQI}$ [76].

Using these results from conservativity now opens up the possibility to extend the notion of assertional TBox effects as considered so far. Remember that assertional TBox effects on a knowledge base have been defined only for the case where Σ is the vocabulary in which $\mathbf{KB}_T$ is formulated (Definition 5.2). If a contributor is interested in effects in terms of a smaller vocabulary $\Sigma' \subset \Sigma$, the following procedure can be taken

to circumvent this small restriction: Given is the knowledge base $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}, \mathcal{A})$ which shall be extended with the TBox $\mathcal{T}$. Let $\mathbf{KB}_T = (\mathcal{T}_0 \cup \mathcal{T}, \mathcal{R}, \mathcal{A})$ and let Σ be the vocabulary in which $\mathbf{KB}_T$ is formulated. Furthermore we assume that $\Sigma' \subset \Sigma$ is the vocabulary and $\mathcal{DL}$ the description logic in which the assertional effects on $\mathbf{KB}$ shall be formulated. Then, in a first step it must be decided whether $\mathcal{T}_0 \cup \mathcal{T}$ is a conservative extension of $\mathcal{T}_0$ w.r.t. Σ' and $\mathcal{DL}$ can be decided. If $\mathcal{T}_0 \cup \mathcal{T}$ is not a conservative extension of $\mathcal{T}_0$ w.r.t. Σ' and $\mathcal{DL}$, then a witness concept C'_T such that $\neg C'_T$ is satisfiable w.r.t. $\mathcal{T}_0$ but unsatisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{T}$ exists. A decision procedure for conservativity such as e.g. in [76] outputs such a witness concept[2]. Given this C'_T, the question of whether and which assertional effects of $\mathcal{T}$ on $\mathbf{KB}$ exist can be reformulated to the question whether $\top \sqsubseteq C'_T$ affects $\mathbf{KB}$, under the condition that $\mathcal{DL}$ contains at least $\mathcal{ALC}$.

Conceptually, I stress the difference in underlying motivation between conservative extensions and assertional TBox effects on a knowledge base. Comparing TBoxes for differences is a general approach to support the frequent task of extending or refining an ontology. The rationale behind focusing on effects in terms of instance data is directed towards ontology edits in a specific ontology application scenario, namely where the ontology describes data in a knowledge base. In this scenario it is important that ontology and data are well aligned with each other in order to maintain conceptual consistency. Second, expressing effects of terminological axioms (general truths in a domain) in terms of concrete facts illustrates them in an easily understandable way[3]. This gives users an opportunity to double-check on whether the effectuated changes were actually "meant this way".

In continuance of the idea underlying conservativity, Kontchakov et al [66] study the differences between DL-Lite TBoxes. Especially, the authors study query-differences over arbitrary ABoxes. By query-difference the authors understand the different query answers that must be given to a query on a particular ABox for the two OWL DL-Lite TBoxes. In their work, the authors always assume that only the TBoxes are given, and the task is to decide whether there exists a query q and an ABox $\mathcal{A}$ such that there is a difference in result when querying $\mathcal{A}$ with q. It turns out, that the set of query-differences is either empty (the two TBoxes do not differ in terms of queries given a specific vocabulary at all) or infinite (infinitely many ABoxes exist after all). Naturally it would be interesting to consider effects in terms of queries over a specific ABox, which I have not done so far. I speculate that the results for such a problem formulation will be similar than the comparison of deductively defined effects with conservative extension, namely that it is at least as hard as deciding the existence of query-differences, and not immediately clear how the existence of query-differences then can be decided for a given ABox.

On a more general note, ontology editors like Swoop [126] and Protégé [102] display *inferred types* for all individuals and concepts in the loaded knowledge base. Typically, such inferred types involve only primitive concepts. Additionally, the dynamic

[2] Note that this particular decision procedure would actually output the negation of C'_T.

[3] Compare also [75], in which the creation of concept definitions from exemplary individuals is being discussed for exactly the same reason. It often seems to be helpful to think in concrete terms when formulating abstractions.

aspect of the ontology edits is not considered in that inferred types are shown for a complete knowledge base (static feedback) and no relation is automatically made to the most recent activities.

5.4 Discussion

5.4.1 Informative Effects

Consider again the knowledge base excerpt given above (Equations 5.1- 5.4). Assume further that the knowledge base also describes famous historical persons, and contains facts such as "Sophocles is a Greek" (5.8)[4]. Extending this knowledge base with the statement that every island is located only in water-areas as above (Equation 5.5) led to undesirable results as discussed already in Section 5.1 above. The example is now modified to consider effects when the following two axioms are added to the knowledge base. First, a disjointness axioms stating that nothing can be both an area of land and an area of water (5.9). Second, an existential restriction which expresses that each island must be located in some water-area (5.10).

$$Greek(sophocles) \tag{5.8}$$
$$LandArea \sqcap WaterArea \sqsubseteq \bot \tag{5.9}$$
$$Island \sqsubseteq \exists locatedIn.WaterArea \tag{5.10}$$

It can be seen that the definition of assertional effects captures knowledge gained such as

$$\neg WaterArea(crete) \tag{5.11}$$
$$\neg LandArea(mediterranean) \tag{5.12}$$
$$(\exists locatedIn.WaterArea)(crete) \tag{5.13}$$
$$(\neg WaterArea \sqcup \neg LandArea)(sophocles) \tag{5.14}$$
$$(\neg Island \sqcup \exists locatedIn.WaterArea)(sophocles) \tag{5.15}$$

This short list demonstrates that some assertional effects are more informative than others and this quality does not directly depend on whether complex concepts are involved or not. Presumably good heuristics for discerning more informative from less informative effects can be found, as for instance thresholding the length of effects. However, such considerations are out of scope for this work as I am primarily concerned fundamentally with the formal definition of assertional effects and providing the computational means to find assertional effects.

5.4.2 Exemplary Effects

A critical issue concerning assertional in the envisioned scenario concerns the quantity of data to be dealt with. If a knowledge base contains data about a million song-titles

[4] Omitting the discussion for the moment whether, since Sophocles is dead, he is or was Greek.

and a new axiom stating that "every song has an author" is added, it is clearly not desirable to see a million effects of the form "The song XY has an author". Instead, assertional effects should be expressed using *exemplary* individuals only. A lead into that direction could be given by techniques such as ABox summarization [35]. The authors exploit the observation that similar individuals are related in similar ways to other individuals. For instance, songs have titles, belong to albums and have maybe been in some charts for a given period of time. Songs do not however have parents or children as do human persons. Thus, individuals about which *similar assertions* exist in a knowledge base can be grouped together. The main issues which need to be studied when applying this ABox summarization to the computation of exemplary assertional effects are that (i) ABox summarization has been defined for $\mathcal{SHIN}$ only in [35] and (ii) the summary ABox does not preserve consistency, i.e. it is possible that the summary ABox is inconsistent while the original ABox is consistent w.r.t. a knowledge base. Apart from these theoretical issues however, the benefit clearly lies not only in enabling an improved presentation to the user but also in increasing the computational performance (though not the computational complexity class).

5.4.3 Extending the Definitions of Effects

The definition of assertional TBox effects on a knowledge base (Definition 2) can be extended into a variety of directions. One such extension, namely the possibility to restrict the vocabulary under consideration to a subset of the vocabulary Σ in which $\mathbf{KB}_T$ is formulated, has already been discussed in Section 5.3 in relation with conservative extensions. Another obvious but much more simple extension is to consider also role assertions $R(x, y)$ as assertional effects. This is trivial in DL languages in which the set of roles which can occur in role assertions is finite, as is the case in both $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$ and thus also in OWL 1 and OWL 2. Then, for every $R \in N_R^{ext}$ and every pair (x, y), $x, y \in N_I$, it simply needs to be checked whether $\mathbf{KB} \not\models R(x, y)$ and $\mathbf{KB}_T \models R(x, y)$. In $\mathcal{SHOIN}$, $N_R^{ext} = N_R^- = \{R | R \in N_R \text{ or } R^- \in N_R\}$. For $\mathcal{SROIQ}$, N_R^{ext} must be extended to be closed under negation. I note that unless nominals occur in $\mathcal{T}$, terminological axioms can not cause the gain or loss of role assertion axioms. Furthermore, also equality or inequality assertions between individuals could be considered as effects. Such effects may occur when $\mathcal{T}$ contains nominals or number restrictions. Finally, the notion of assertional effects could also be extended to consider assertional effects of the addition or removal of role axioms.

6

Ontology Evaluation in MoKi

It has already been noted that evaluation is a vital activity in ontology engineering, and also that tool support for evaluation activities is poor. Typically, ontology engineering environments focus in their design on preventing modelling errors rather than on finding modelling errors. While the first is necessary, and indeed a number of modelling mistakes can be prevented by good tool design, mistakes will nonetheless be made. The goal of the validation modules in MoKi is essentially to embed support for ontology evaluation directly within an ontology engineering environment. Following the requirements discussed for evaluation methods during ontology engineering, the validation modules in MoKi aim to give feedback on how the models can be improved rather than rating the quality of the models on a numerical scale. An additional goal was to trigger model refinement already during modelling.

Considerations of how these goals can be realised in MoKi led to the following categorisation of ontology evaluation functionalities within a computational ontology engineering environment: Evaluation functionalities that give feedback on the models directly during modelling on the one hand and functionalities that serve to review the models in a separate evaluation activity on the other hand can be differentiated by looking at the "place" where they appear to users in the user interface. The first are displayed locally, when a model element is described. The latter are displayed at a separate place and consider the whole model and thus are called "global". Second, one can differentiate between functionalities that take a user's recent actions into account (dynamic) and those that consider the model as it is (static). Together this leads to a categorisation of ontology evaluation functionalities into the categories "global and static", "global and dynamic", "local and static" and "local and dynamic". This categorisation is depicted in Fig. 6.1, where also the four validation modules available in MoKi (ontology questionnaire, assertional effects, models checklist, quality indicator) are filled in at their place.

The ontology questionnaire and the models checklist are global (and static) evaluation functionalities, that can be used in a separate evaluation step. While the models checklist can be used to validate both informal and formal aspects of the domain model contained within MoKi, the ontology questionnaire uses reasoning over the formal representation and can be used to validate the formal model. The quality indicator is a local and static evaluation functionality that visualises on element pages the same character-

istics that are given for the complete model in the models checklist. The assertional effects finally give local and dynamic feedback to MoKi users depending on their most recent activities.

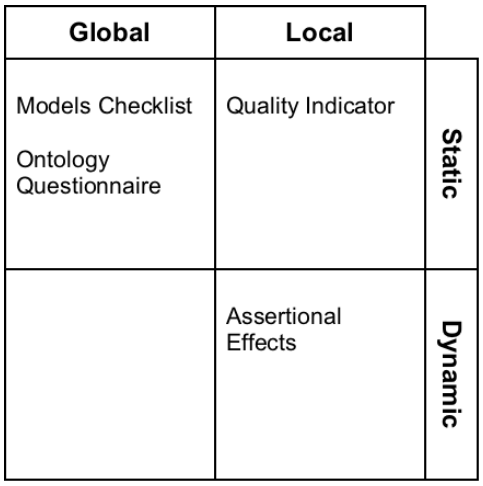


Fig. 6.1. Within a computational ontology engineering environment, ontology evaluation functionalities can be categorised according to where they are displayed in the user interface (global vs. local) and whether they react to a user's activities or consider the model as is (dynamic vs. static). The first correlates with whether evaluation takes place as a separate activity (global) or directly during modelling (local). The above figure shows the categorisation of the MoKi validation modules into these categories.

6.1 The MoKi Validation Modules

The MokiValidation plugin, which is itself a module of MoKi (see Chap. 3), contains four validation modules: The ontology questionnaire, assertional effects, the models checklist and the quality indicator. The ontology questionnaire displays inferences from the domain model contained within MoKi. It supports the process of reviewing inferences for ontology evaluation purposes that has been described in Chap. 4 above, and corresponds closely to the prototypical implementation described there. The assertional effects module displays assertional effects (see Chap. 5 above) after modifications to a knowledge base, provided any exist. Both the ontology questionnaire and the assertional effects module support ontology evaluation based on the formal interpretation given to the domain model within MoKi, and use reasoning services to do so. These two modules put the insights gained from the theoretical investigations covered in Chaps. 4 and 5 above to practical use. In addition, MoKi contains two validation modules that can be considered as implementation of (heuristic) modelling guidelines, the models checklist and the quality indicator. Given very general guidelines on good practices in modelling, such as e.g. model elements should be well documented, the models checklist lists elements that do not comply to modelling guidelines, and the quality indicator

visualises how well a given element complies with modelling guidelines. The latter two validation modules therefore complement the approach to support ontology evaluation by providing inferences, on which this thesis focuses, with a heuristic approach to ontology evaluation.
This chapter describes the functionality and rationale of each validation module. General implementation principles for the MoKi validation modules are described in Chap. 6.2 below, and implementation details are then described module-by-module in Chap. 6.3.

6.1.1 Ontology Questionnaire

The ontology questionnaire displays inferred statements, i.e. statements that logically follow from the knowledge explicitly contained within MoKi. It gives MoKi users an insight into knowledge that is implicitly entailed by the domain model contained within MoKi, and the possibility to review the domain ontology without having to go through the set of explicitly modelled statements. The ontology questionnaire is accessible via the MoKi menu ("Inferences - Do You Agree?", see also Fig. 3.4) or via its special page address ("Special:MV_OntologyQuestionnaire").

Functionality and Rationale

Inferences are limited to statements referring to concept expressions that occur in the explicitly stated ontology, as is usual in comparable tools. The ontology questionnaire within MoKi (Fig. 6.2) corresponds quite closely to the description in Chap. 4, minus one and plus three features. First, the ontology questionnaire within MoKi includes inferred assertional and role axioms in addition to inferred terminological axioms. Second, it divides inferred statements into "Most Striking Inferences" and "Inferences derived from the Concept and Property Hierarchy". The latter category displays all inferences that follow from the transitivity of the subsumption hierarchy for concepts and properties. The first category then comprises, by elimination, all other inferences. Assertional axioms, in the current implementation, always show up in the first category. The idea behind this separation of inferences into two categories is to give MoKi users the possibility on the one hand to limit review of inferences to those that are derived by more complex mechanisms than subsumption, while on the other hand still enabling them to go execute a full review and thus to also check the correctness of the hierarchies. This also partly addresses the wish of users, according to the study described in Chap. 4, to have a prioritised list of inferences which should be checked.

> **Example:** $Article \sqsubseteq Publication$ and $Publication \sqsubseteq Document$ together entail that $Article \sqsubseteq Document$. This inference follows from the transitivity of the subsumption relation, and is therefore displayed in the section "Inferences derived from the Concept and Property Hierarchy".

Third, the ontology questionnaire within MoKi displays side-effects of deleting explicitly made statements. So when a MoKi user decides to delete some explitly made statements in order to remove a selected, undesired, inference, (s)he will be informed about inferences that will be lost in addition to the selected inference. Fourth, the ontology

questionnaire in MoKi does not preselect statements for deletion, given an inference. Originally, such a preselection was made based on minimal hitting sets[1].

Functionality

Explanations for an inference can be viewed by clicking on "Why?" at the end of line for the corresponding inference (see Fig. 6.2). An inference can have one or more explanations (see Fig. 6.3), where a single explanation corresponds to a justification as defined in Def. 1, i.e. it is a minimal set of statements that entail the selected inference. If the reviewing MoKi user, the reviewer, disagrees with an inference, (s)he has the possibility to remove it directly from within the ontology questionnaire by deleting explicitly made statements from the domain model. Since multiple explanations are independent from each other, and each suffices to entail the inference it explains, the reviewer must select at least one statement from each explanation in order to remove the inference (s)he disagrees with. Before the selected statements are actually removed from the domain model contained within MoKi, the reviewer is asked for confirmation (Fig. 6.4). This confirmation window is intended less as an annoying "Are you sure?"-dialog but as a double-check regarding side-effects of deleting one or more statements. Obviously, if at least one statement from each explanation was selected, the inference the reviewer disagreed with will not appear anymore after deleting the selected statements. However, this deletion may also lead to the loss of other inferences. The confirmation window lists the inferences that will be lost alongside, and gives the reviewer the possibility to cancel and select other statements for deletion. A final success page informs the reviewer of the actual deletion (Fig. 6.5).

Summary

The ontology questionnaire is used for evaluating the domain model contained within MoKi in a separate evaluation step, and not directly during modelling and thus belongs to the category of global (and static) ontology evaluation functionalities. It reasons over the formal representation of the domain model in description logics, and thus it becomes more and more relevant and also interesting to use the ontology questionnaire as the formalisation of the domain model within MoKi proceeds[2].

6.1.2 Assertional Effects

Assertional effects, as defined in Chap. 5 in Def. 2, are entailed concept assertions that illustrate the consequences of ontology editing activities (addition or removal of terminological axioms to/from an ontology). Terminological axioms can be added and removed MoKi by creating, editing, deleting or restoring concept pages. The correspondence between MoKi's user interface elements on concept pages and terminological axioms is listed in Table 3.2. Assertional effects completely ignore process or any other models contained within MoKi.

[1] A minimal hitting set is a minimal set of explicit statements such that if all statements in it are deleted, the selected inference can not be inferred anymore.

[2] Assuming that the goal of MoKi is met, in that its users perform both informal and formal modelling within MoKi, and based on the hypothesis that the creation of an ontology follows an evolution from more "sketchy" and informal to more axiomatised and formal.

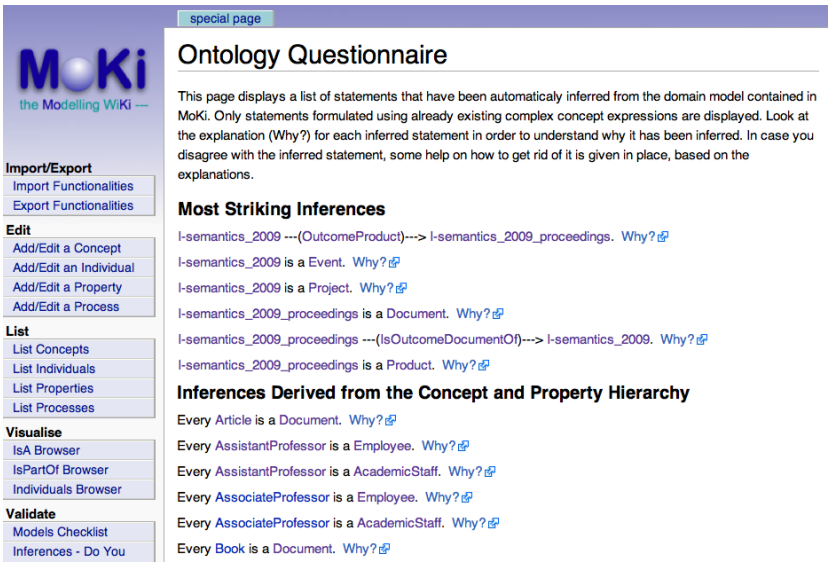


Fig. 6.2. The ontology questionnaire in MoKi. Inferences are limited to inferred statements that refer to explicitly mentioned concept expressions, and are divided into "Most Striking Inferences" and "Inferences Derived from the Concept and Property Hierarchy". For each inference, its explanation(s) can be accessed by clicking on "Why?" at the end of the line.

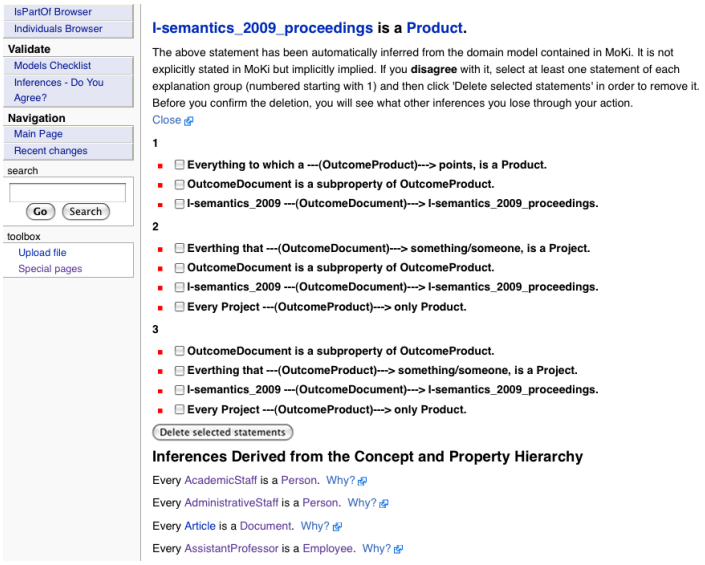
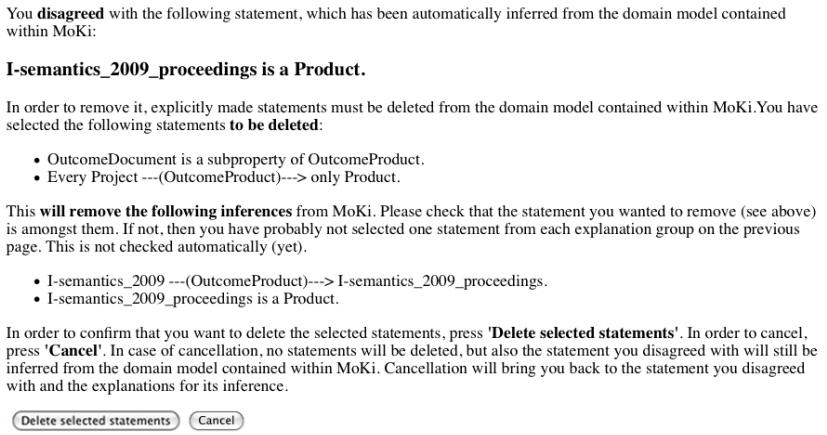


Fig. 6.3. Multiple explanations for an inference are displayed in groups and numbered starting with "1". If the MoKi user disagrees with the inference, (s)he can select statements from the explanations (at least one from each group) and click on "Delete selected statements" in order to delete the explicitly selected statements from the domain model and thus get rid of the inference.


You **disagreed** with the following statement, which has been automatically inferred from the domain model contained within MoKi:

I-semantics_2009_proceedings is a Product.

In order to remove it, explicitly made statements must be deleted from the domain model contained within MoKi.You have selected the following statements **to be deleted**:

- OutcomeDocument is a subproperty of OutcomeProduct.
- Every Project ---(OutcomeProduct)---> only Product.

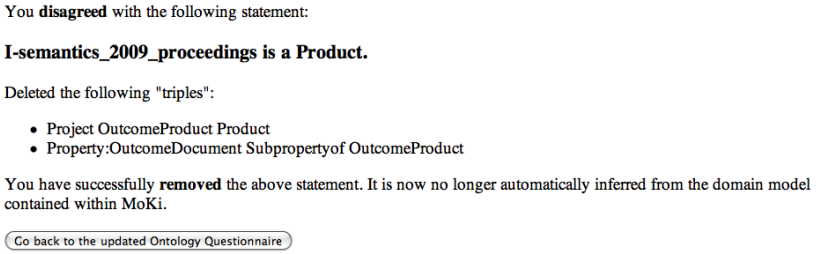
This **will remove the following inferences** from MoKi. Please check that the statement you wanted to remove (see above) is amongst them. If not, then you have probably not selected one statement from each explanation group on the previous page. This is not checked automatically (yet).

- I-semantics_2009 ---(OutcomeProduct)---> I-semantics_2009_proceedings.
- I-semantics_2009_proceedings is a Product.

In order to confirm that you want to delete the selected statements, press '**Delete selected statements**'. In order to cancel, press '**Cancel**'. In case of cancellation, no statements will be deleted, but also the statement you disagreed with will still be inferred from the domain model contained within MoKi. Cancellation will bring you back to the statement you disagreed with and the explanations for its inference.

Delete selected statements Cancel

Fig. 6.4. The confirmation page displayed to a MoKi user who has selected several statements to be deleted from the domain model in order to remove a single inference. However, the deletion of statements can also lead to the removal of other inferences. Thus, this confirmation window also serves as an information about the side-effects of deletion that the MoKi user may not be aware of otherwise.


You **disagreed** with the following statement:

I-semantics_2009_proceedings is a Product.

Deleted the following "triples":

- Project OutcomeProduct Product
- Property:OutcomeDocument Subpropertyof OutcomeProduct

You have successfully **removed** the above statement. It is now no longer automatically inferred from the domain model contained within MoKi.

Go back to the updated Ontology Questionnaire

Fig. 6.5. After the selected statements have been deleted successfully from within the ontology questionnaire, the above success page is shown to the MoKi user. It displays the deleted statements as well as the inference the MoKi user disagreed with.

Functionality and Rationale

Assertional effects can occur if logical changes to the MoKi content are made. Changes to for instance verbal descriptions can not lead to assertional effects. In case a logical change is detected, an approximation of the decision procedure for assertional effects is executed. The approximation consists of not searching up to the theoretically required path length but giving up earlier (see below for some more details). Therefore, while assertional effects that are displayed are always correct, it may be that assertional effects exist when none are displayed. This is in order to avoid searching unnecessarily long for a negative decision. The underlying rationale is that assertional effects are intended only as a support for the MoKi users and thus reactivity of MoKi takes precedence over completeness of results.

If editing a concept page leads to assertional effects, these are displayed on the concept page directly after saving the page (Fig. 6.6). The assertional effects are only displayed directly after the ontology edit however, and are not displayed when the MoKi user

navigates back to the same concept page after other activities. The same happens when new concept pages are created, so that directly after saving a newly created concept page, assertional effects will be displayed. Also the deletion of a concept page can lead to assertional effects. These are not displayed however, primarily because it is not clear where to display them in a meaningful way[3]. Similarly, assertional effects that are caused by the restoration of a concept page are not displayed. On a meta-level it is clear that the assertional effects from restoring a concept page are the same as when newly creating this concept page. However, there are two usability issues in this case. First, it is not clear whether a representation of assertional effects in case of a page restoration are meaningful, since restoration could be viewed as mostly a maintenance activity in contrast to a modelling activity. Second, it is unclear where the effects should be presented[4].

In addition to considering concept assertions as assertional effects, as has been extensively discussed in Chap. 5.2, in MoKi also role assertions involving primitive roles are considered as assertional effects, and assertional effects caused by the addition or removal of role axioms are computed as well. The effective definition of assertional effects in MoKi is therefore Def. 6, given below in the description of implementation details in Chap. 6.2. Now, also creating, editing, deleting and restoring property pages can lead to assertional effects and therefore assertional effects may also appear on property pages (Fig. 6.7). The underlying mechanisms are completely analogous to those described for concept pages as above. Therefore, although deleting and restoring property pages can lead to assertional effects, these are not displayed in MoKi.

Extending the Definition of Assertional Effects

The deductive definition of assertional effects as given in Chap. 5 has been extended in two directions for implementation within MoKi. Note that the following extensions are at various points targeted specifically towards the expressivity currently supported by MoKi.

The first extension concerns the inclusion of inferred role assertions as assertional effects. OWL 2 (and the description logic language $\mathcal{SROIQ}$ it is based upon) includes positive as well as negative role assertions. Checking for gained or lost role assertions in this case does not pose a theoretical problem, since the set of individual names N_I, as well as the set of role names N_R are finite. Within MoKi, only positive role assertions are considered as assertional effects, which is a reflection on MoKi's current expressivity however since neither negative role assertions, nor role disjointness axioms can be expressed in MoKi as it is..

The second extension concerns the inclusion of added or removed role axioms as causes

[3] After deleting a wiki page, MediaWiki provides a deletion message that basically tells the wiki user that a page been deleted. It is possible that in future versions of MoKi, assertional effects that result from the deletion of a concept will be displayed on this page.

[4] As with deletion, MediaWiki provides a restoration message after a successful restoration has taken place. The assertional effects caused by such a restoration could be placed either on this page, or at the concept page that has been restored the next time it is visited. The latter seems to provide usability problems, since the display of assertional effects would then be "more than one click away" from the action that caused them.

of assertional effects. This is a bit more tricky from a theoretical point of view. A quick analysis of role axioms available $\mathcal{SROIQ}$ (see Table 2.2) reveals however, that role axioms by themselves can lead only to role assertions (involving both primitive and negated primitive roles) and (in)equality assertions (e.g. in the case of functional and inverse functional property axioms). (In)Equality assertions are not considered as assertional effects within MoKi, but this may be included at a later stage of development. Only role disjointness axioms could lead to negative role assertions, but role disjointness can currently not be expressed in MoKi. The extension of assertional effects to consider also updates of the role box is therefore well covered within MoKi by the consideration of positive role assertions as assertional effects.
Together, these two extensions lead to the following definition for assertional effects as implemented in MoKi:

Definition 6. (Assertional effects) *Let* $\mathbf{KB} = (\mathcal{T}_0, \mathcal{R}_0, \mathcal{A})$ *and* $\mathbf{KB}_T = (\mathcal{T}_0 \cup \mathcal{T}, \mathcal{R}_0 \cup \mathcal{R}, \mathcal{A})$. *Let* $\Sigma = (N_C, N_R, N_I)$ *be the vocabulary in which* $\mathbf{KB}_T$ *is formulated and let* $\mathcal{DL}$ *be the DL in which the assertional effects shall be formulated.*

- $\alpha \in \{C(x), R(y,z)\}$ *such that* $C \in \mathcal{C}(\Sigma, \mathcal{DL})$, $R \in N_R$ *and* $x, y, z \in N_I$ *is an assertional effect of* $\mathcal{T}$ *on* $\mathbf{KB}$ *iff* $\mathbf{KB} \not\models \alpha$ *and* $\mathbf{KB}_T \models \alpha$.
- $\mathcal{T} \cup \mathcal{R}$ *affects an individual* $x \in N_I$ *in* $\mathbf{KB}$ *iff an assertional effect* $C(x)$ *of* $\mathcal{T}$ *on* $\mathbf{KB}$ *exists, or*
- $\mathcal{T} \cup \mathcal{R}$ *affects the individuals* $y, z \in N_I$ *in* $\mathbf{KB}$ *iff an assertional effect* $R(y,z)$ *of* $\mathcal{T} \cup \mathcal{R}$ *on* $\mathbf{KB}$ *exists.*
- $\mathcal{T} \cup \mathcal{R}$ *affects a knowledge base* $\mathbf{KB}$ *iff an assertional effect* α *of* $\mathcal{T} \cup \mathcal{R}$ *on* $\mathbf{KB}$ *exists.*

The Practical Decision Procedure

The practical decision procedure of detecting assertional effects as implemented for MoKi consists first of an approximation of assertional effects, which limits the search for assertional effects of the form $(\forall R_1 \ldots R_n.C_T)(x)$ to a rather small n. Instead of a double-exponential dependency of n on the size of the original knowledge base $\mathbf{KB}$, the approximate decision procedure only searches up to n linear with the size of the original knowledge base $\mathbf{KB}$. Thus, while all assertional effects that are detected are correct, it is not guaranteed that assertional effects will be found if they exist. Second, the practical decision procedure tries to shorten the assertional effects if possible. given an assertional effect of the form $C(x)$, this is done by searching subconcepts (meaning subformulae) of C and determining whether these constitute an assertional effect on x.

> **Example:** If an assertional effect of the form $(\neg C \sqcup D)(x)$ is found, and $D(x)$ is also an assertional effect, then the longer effect is thrown away and only the shorter effect is displayed.

On the other hand, the practical decision procedure also searches directly for assertional effects of the form $C(x)$, where C is a primitive or negated primitive concept.

Finally, a small hitch in the current MoKi implementation concerns the treatment of domain and range restrictions. From a logical point of view, these should be treated as

terminological axioms as concerns assertional effects[5]. They are however not treated as such, and thus are not included in the transformed TBox $\mathcal{T} = \{\top \sqsubseteq C_T\}$ which is necessary to search for assertional effects exhaustively. In practice, given MoKi's current expressivity, this does not present a problem, since MoKi only allows primitive concepts to restrict a property's domain and range, and assertional effects involving primitive and negated primitive concepts are searched for explicitly in the current implementation. As soon as MoKi's expressivity is increased to support also domain and range restrictions involving complex concepts however, this will need to be remedied.

Summary

Assertional effects give dynamic feedback to the MoKi users in that they directly react on a user's most recent activities. They are also directly visible during modelling and thus can trigger immediate revisions. Assertional effects do not point to modelling errors but instead make visible the implicit implications of ontology edits, and illustrate them as concrete examples using individuals from the knowledge base.

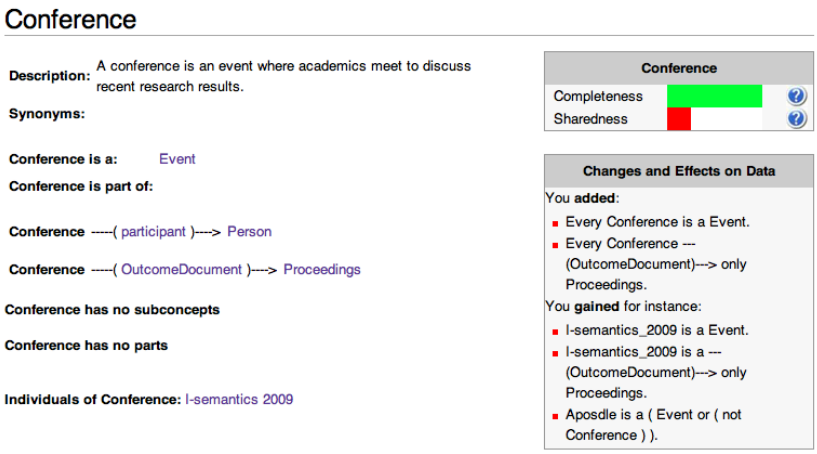


Fig. 6.6. After editing a concept page, assertional effects on individuals that are described within MoKi are displayed in the box titled "Changes and Effects on Data". Both the explicitly made changes (below "You added" in the screenshot) as well as the assertional effects caused by the changes (below "You gained for instance" in the screenshot) are displayed.

[5] As a reminder, a domain restriction corresponds to an axiom of the form $\exists R.\top \sqsubseteq C$ and a range restriction corresponds to an axiom of the form $\exists R^{-}.\top \sqsubseteq C$, see also Chap. 2.2.

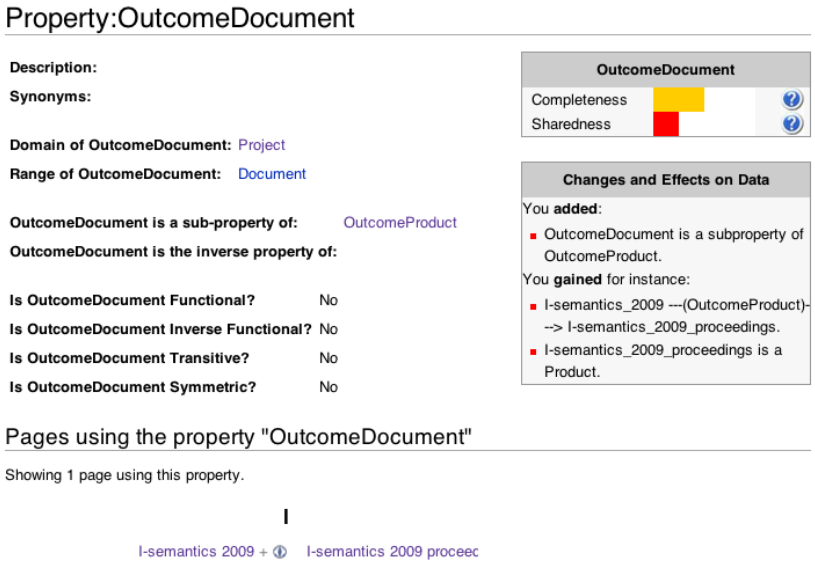


Fig. 6.7. After editing a property page, assertional effects on individuals that are described within MoKi are displayed in the box titled "Changes and Effects on Data". Both the explicitly made changes (below "You added" in the screenshot) as well as the assertional effects caused by the changes (below "You gained for instance" in the screenshot) are displayed.

6.1.3 Models Checklist

The models checklist (Fig. 6.8) contains a set of checks, and for each check the elements that fit are listed. The checks concern characteristics that typically, but not always, point to oversights and modelling mistakes. Technically, the single checks are pre-defined queries asked about MoKi's content. The models checklist is therefore a heuristic tool for ontology evaluation. It can be thought of as a set of modelling guidelines that have been implemented as automatic checks for elements that do not comply with the modelling guidelines.

> **Example:** A typical modelling guideline is to verbally describe model elements and document design decisions. While it is impossible with the current state-of-the-art to automatically determine how good a description really is, it is possible to automatically check for model elements that are not documented at all.

The models checklist is accessible via the MoKi menu via the entry "Models Checklist" (Fig. 3.4) or via its page address ("Models_Checklist").

Functionality and Rationale

MoKi lists all kinds of model elements (concepts, individuals, properties, processes) without verbal descriptions. Documenting model elements by describing them verbally is considered good style since this makes clear in the language of domain experts, namely in natural language, what the element is about. For all kinds of elements, MoKi

lists elements to which only one person has contributed (called non-shared elements). This pays tribute to the definition of an ontology as a shared conceptualisation, as well as to the well-known four-eyes principle, namely that having at least a second person review some engineering artefact usually increases quality of the artefact[6]. Individuals are excepted in the default configuration because they are seen not as designed elements but as elements used for explanation, testing or automatically provided by some source in general. This viewpoint may not hold for all possible MoKi setups, but the default models checklist can be easily edited to include the list again for individuals as well, using the corresponding special page given in Table 6.2. Additionally, MoKi implements some checks that are specific to the different kinds of elements. For concepts, MoKi additionally checks for orphaned concepts and for un-instantiated concepts. The first are concepts which are not part of a subsumption or meronymic hierarchy. This is considered a potential modelling error since mostly a concept in a domain ontology should be grouped together with sibling concepts beneath a more general concept or be itself used to group together more specific concepts. However, it is also conceivable that the ontology is per design non-hierarchical.

> **Example:** A simple ontology about the food chain between animals could contain only the information about which animal eats which other animal, without representing information about the hierarchy of species and subspecies between animals.

Not instantiating concepts is not really a modelling mistake. However, there are two reasons why this check is included nevertheless. In the case of a general ontology, individual data can be used as examples for concepts and thus provide additional documentation, or test logical axioms that describe concepts. If MoKi is used to hold a knowledge base, uninstantiated concepts are redundant with respect to the data. If it is indeed irrelevant in a particular setting whether concepts are instantiated or not, the check can easily be removed from the models checklist. For individuals, MoKi additionally gives a list of individuals which have no asserted type. This is typically an oversight. For properties, MoKi additionally checks whether domains and ranges are defined. While it is not strictly a modelling error to leave out domain and/or range restrictions, both these restrictions make an ontology logically more precise. It can be expected that as knowledge becomes more and more formalised, such restrictions should be added.

Summary

The goal of the models checklist is to make it easy for users to review the models in MoKi based on modelling guidelines. However, these are only guidelines and not every element that is pointed out by the models checklist is wrongly modelled. The models checklist contains checks both on informal and formal aspects of the models contained in MoKi. Informal aspects that are checked for are the existence of verbal description and the sharedness. Formal aspects are checked via the check for orphaned concepts,

[6] Although there is a small hitch: The MoKi takes notice only of people editing pages. So if a second person were to review a page and totally agree with everything on it, MoKi would not realise that a second person had been involved and still regard this page as “non-shared”.

uninstantiated concepts, individuals without types and properties without a defined domain or range. Such characteristics were categorised as usability-related features and structure-related features in Chap. 2.4. The models checklist represents a separate modelling step, reachable from the phases of informal and formal modelling, since it is a wiki page on which no modelling can directly take place.

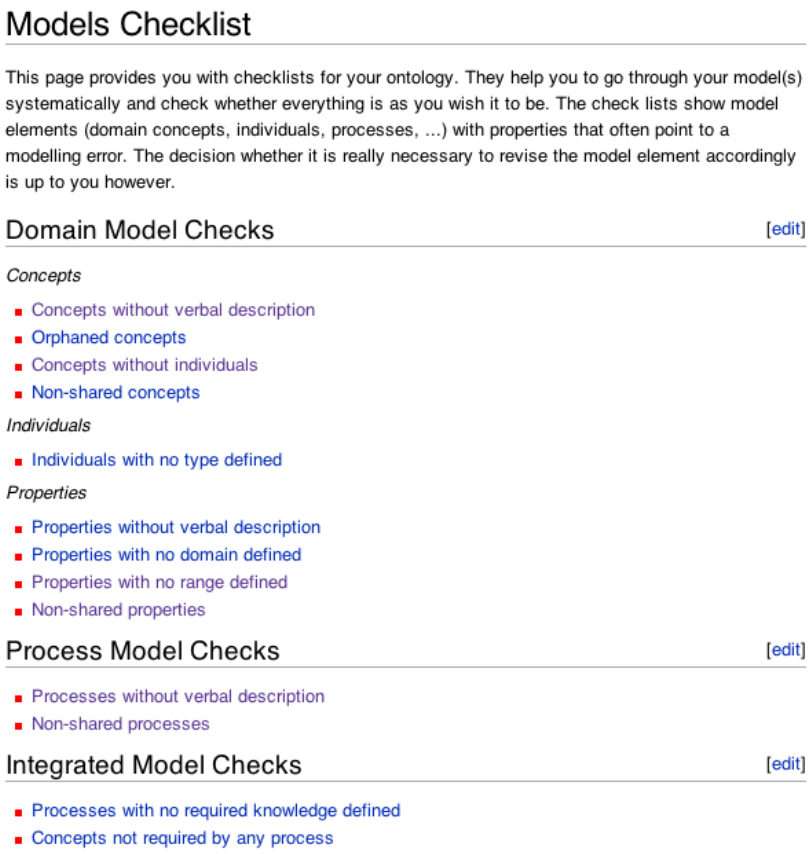


Fig. 6.8. The models checklist contains a set of checks, and for each check the elements that fit are listed. The checks concern characteristics that typically, but not always, point to oversights and modelling mistakes.

6.1.4 Quality Indicator

The quality indicator (Fig. 6.10) visualises the completeness and sharedness of single model elements and appears on wiki pages that represent model elements. Both completeness and sharedness are visualised as bars that grow from left to right in four discrete steps, and take on different colours (red, orange, yellow, green) according to value. Whenever a model element shows one of the characteristics listed below it collects points, and the quality indicator bar grows with the achieved points. The points given for each characteristic follow rules-of-thumb, and in particular also the division

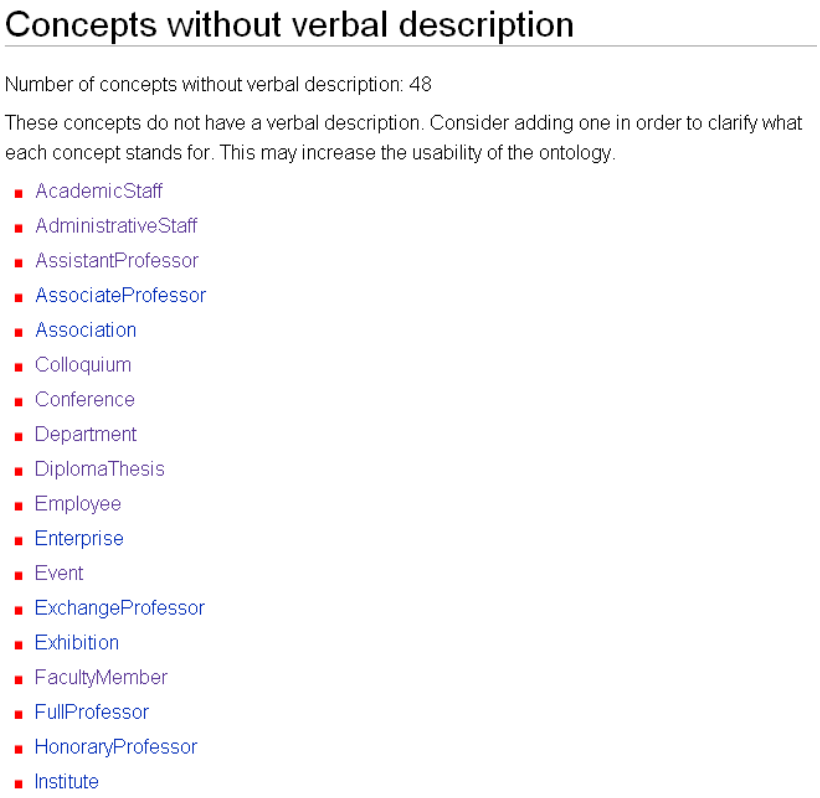


Fig. 6.9. Models checklist: List of concepts without a verbal description.

of the values into four partitions (red, orange, yellow, green) is based on common-sense only.

Functionality

Completeness captures how much information (verbal, structural) about the element is available. It takes into account usability information like the existence of a verbal description, the existence of synonyms and structural information like the embedding into a hierarchy for concepts and properties, the existence of relations to other concepts, the existence of domain and range restrictions for properties and the existence of a defined type and of relations to other individuals for individuals. Sharedness captures how many people have contributed to the description of the element. As with the models checklist above, sharedness is not rated for individuals.
Sharedness in particular makes sense only in settings where multiple persons are expected to contribute to the MoKi content, and the value partitioning must be configured accordingly to the number of expected participants.

Example: Within the APOSDLE project, MoKi was used to model six domains (an earlier version of MoKi however). In each case there was only a single person who was responsible for entering information into MoKi. In this

case, the sharedness would not have been indicative of the conceptual sharedness of the model elements.

Example: MoKi has also been used in a lecture about knowledge management, where multiple groups of around 50 students filled a single MoKi with content. In this case, sharedness makes sense, and the value partition can be configured for instance that at least 5 persons must have contributed to a page before it reaches the green, i.e. full, bar.

Summary

Completeness and sharedness are heuristic measures insofar as a low value does not necessarily mean that the element is modelled incorrectly and a high value does not guarantee that the element is is modelled correctly. The quality indicator is effectively a local, compact visualisation of compliance of a model element to the corresponding modelling guidelines. The characteristics visualised by the quality indicator are similar to the characteristics available in the models checklist, and thus stem both from literature (usability-related and structure-related features) as well as from application in several real-world ontology engineering settings where they have been used to iteratively revise models [97]. The concrete number of points given for each characteristic as well as the value partition have not been evaluated in practice.

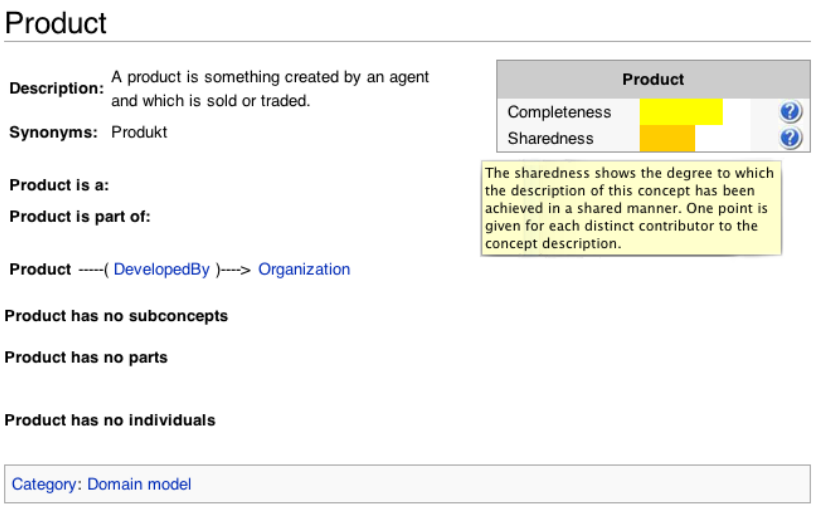


Fig. 6.10. The above figure shows the quality indicator on a concept page (yellow and orange bars). The yellow box below the quality indicator is additional information about the quality indicator that is available to MoKi users on pointing their mouse over the question marks to the right of the quality indicator.

6.2 Implementation Principles

The overall design of MoKi has been covered in Chap. 3.3 above. This chapter goes into details on the extension mechanism used to implement the MoKi validation modules, and describes additions to MoKi's basic design that were necessary in order to implement the MoKi validation modules, namely a closer integration of PHP and Java and a second data storage alongside MediaWiki's native database.

6.2.1 Extending MediaWiki Through Special Pages and Hooks

MediaWiki provides different possibilities for extensions. Two, namely special pages and hooks, will be described in more detail below since these were used to implement the MoKi validation modules. Other kinds of extensions are tag extensions (new tags of the form "<example_tag></example_tag>" can be defined), magic words (new reserved words that trigger functions can be defined), parser functions (a kind of magic words that accept parameters) and skins (changing the appearance of a wiki). Details on different kinds of MediaWiki extensions and how to write them can be found at [82].

Loading the MoKi *Validation Modules*

The MoKi extension code resides in the `extensions` subdirectory of the MediaWiki installation directory, as is typical for MediaWiki extensions. In order to have MediaWiki load the MoKi code at startup, the following line is included in the `LocalSettings.php` file in the MediaWiki installation directory:

```
require_once("extensions/MoKi/MoKi_Settings.php");
```

The file `MoKi_Settings.php` in the `extensions/MoKi` subdirectory itself loads other modules of MoKi, amongst them the MoKi validation modules. They are implemented as yet another substructure that resides in the `extensions/MoKi/modules/MoKiValidation` directory, so that a single file `MV_Settings.php` is responsible for loading and configuring all and only MoKi validation modules. The following line is therefore included in the `MoKi_Settings.php` file:

```
require_once('modules/MokiValidation/MV_Settings.php');
```

Hooks

A hook in MediaWiki is an event, like for instance that a page is opened for editing. MediaWiki itself contains a series of event handlers that are triggered at each hook. A hook extension is a function (an event handler) that gets triggered at a given event in addition to all other event handlers (event handlers by MediaWiki plus event handlers by other installed extensions). In order to make MediaWiki execute a hook extension at a given hook, the extension must be registered with MediaWiki. A list of available hooks as well as documentation on how to write a hook extension can be found at [55].

Example: In order to have the function `mvAE_display`, which is responsible for displaying assertional effects if they exist, called always just before MediaWiki displays a wiki page, it is registered with MediaWiki as event handler for the `OutputPageBeforeHTML` hook:

```
$wgHooks['OutputPageBeforeHTML'][] = 'mvAE_display';
```

Special Pages

While normal wiki pages can be created, edited and deleted manually by wiki users through the user interface of a MediaWiki installation, special pages are created automatically and can not be edited. Typically, special pages perform some computation like finding pages that are not linked to from anywhere and display the result. Special pages have the prefix `Special:` in front of their page name.

Example: One special page provided by MediaWiki itself is for instance the "Special:Specialpages" page, which displays all special pages registered within a MediaWiki installation.

Special pages are implemented as PHP classes that extend the predefined `SpecialPage` class. The `SpecialPage` class predefines a constructor and an `execute()` method. The latter is called and executed when the special page is loaded in a browser. Typically, the `execute()` method performs some computation and then outputs wiki or HTML code. A special page extension must also register with MediaWiki.

Example: The special page "Special:MV_OntologyQuestionnaire" is registered as follows with MediaWiki in the `MV_OntologyQuestionaire.php` file:

```
# Let MediaWiki know about your new special page.
$wgSpecialPages['MV_OntologyQuestionnaire'] =
    'MV_OntologyQuestionnaire';
# Tell MediaWiki to load the extension body.
$wgAutoloadClasses['MV_OntologyQuestionnaire'] =
    $dir . 'MV_OntologyQuestionnaire_body.php';
```

Example: The main code of the special page "Special:MV_OntologyQuestionnaire" resides in the `MV_OntologyQuestionaire_body.php` file, with the following class skeleton:

```
<?php
class MV_OntologyQuestionnaire extends SpecialPage {
    function __construct() {
        parent::__construct('MV_OntologyQuestionnaire' );
        [...]
    }

    function execute( $par ) {
        global $wgOut;
```

```
        [...]
        # Write text in wiki syntax to the output
        $wgOut->addWikiText( wfMsg('mv-oq-intro') );
    }
}
```

6.2.2 PHP and Java revisited

Two MoKi validation modules, the ontology questionnaire and assertional effects, are basically reasoning services on OWL knowledge bases. While there are libraries in PHP that support RDF, there are none that support OWL and neither are there reasoners in PHP. On the other hand, in Java there are at least two open source libraries for OWL (Jena, OwlApi) and an open source reasoner (Pellet). It was therefore decided to implement the backend functionality of the ontology questionnaire and of assertional effects in Java.
Also some core functionalities of MoKi rely on a Java backend, namely the import and export functionalities and the term extraction functionality. However, they are not very user-interactive and not used throughout modelling. For this reason, the simple communication between PHP and Java through calling Java methods in a commandline and passing results from Java to PHP back through files in a temporary directory were sufficient for the import, export and term extraction functionalities[7]. By contrast, the ontology questionnaire requires more user interaction: Explanations for inferences must be given and simple revisions (deletion of statements) of the model from within the ontology questionnaire must be possible. Assertional effects are not interactive but frequently called upon, since essentially after every ontology edit it must be verified whether assertional effects exist or not. Communication between PHP and Java via commandline and files is clearly too unwieldy for such purposes.

It was decided to use the PHP/Java Bridge [15] to achieve a more direct communication between PHP and Java code. The PHP/Java Bridge allows directly calling Java methods from within PHP code and vice versa. For the MoKi validation modules, only the first is necessary. The PHP/Java Bridge is started from within MoKi as an HttpServer at a local port (usually at 8087), and a dedicated MoKi page[8] lets users see whether the server is up and running as it should be, stop and restart the PHP/Java Bridge.

> **Example:** In order to instantiate the Java class `OntologyQuestionnaire`, the base class in Java that handles the backend functionality of the ontology questionnaire and to call the member method `getInferredAxioms`, the following PHP code is executed:

```
$oq = new java("at.tugraz.kmi.oq.OntologyQuestionnaire",
```

[7] In version 1.2 of MoKi, this communication is still kept. In version 1.5Beta, since the ontology questionnaire and assertional effects require a different communication between PHP and Java, also the other functionalities were re-written.

[8] "$MoKi/extensions/MoKi/modules/MoKi_Ext_Utils/BridgeControl.php", where "$MoKi" points to the base URL of the MoKi installation.

```
    $this->owl_filename);
$list_entailed_axioms = $oq->getInferredAxioms();
```

6.2.3 Data Storage in MoKi

MediaWiki can store its content in either a MySQL or a Postgres database. Semantic MediaWiki (SMW) has initially supported storage only in MySQL databases, but since recently also supports the usage of Postgres. Since MoKi itself relies on a working SMW installation, MoKi has been developed on machines with a MySQL database only and has not been tested for Postgres at the backend. While SMW adds database tables to the MediaWiki database, MoKi fully stores its content within MediaWiki and SMW tables.

The Need for a Synchronised OWL Representation of MoKi *Content*

The interpretation of MoKi content as description logic axioms (as given in Tables 3.2-3.4) is done by most MoKi functionalities based directly on the database content. An explicit translation between database content and OWL occurred originally only during import and export. The first translates an OWL ontology into appropriate database content according to the MediaWiki and SMW database schemata, while the second translated database content of a MoKi installation into an OWL ontology. Essentially, the MoKi export functionality first calls SMW's built-in RDF export functionality, and then translates the created RDF represenation into an OWL representation of the MoKi content. The difference between the RDF and the OWL representation lies less in the formalism than in the interpretation given to content. The SMW's built-in RDF export functionality interprets MoKi content as content of a normal SMW installation, while the translation from RDF to OWL is a MoKi functionality that interprets MoKi content according to Tables 3.2-3.4.

> **Example:** Assume that MoKi contains a relation expressed as "Conference —(participant)—> Person" in MoKi's user interface (as e.g. visible on Fig. 6.6). In RDF as directly exported from SMW, this would be represented as the triple `Conference participant Person.` The MoKi export functionality would translate this to the following representation in OWL however: $Conference \sqsubseteq \forall participant.Person.$

The models checklist, the quality indicator, and all core MoKi functionality described in Chap. 3 above, can work directly on the database content, since the interpretation of MoKi content as description logic axioms is sufficiently easy for their purposes. The ontology questionnaire and assertional effects need full-blown OWL 2 reasoning over the MoKi content however, and thus any reasoner would need an OWL (or another description logic) representation of MoKi content. Furthermore, the assertional effects rely on information about the dynamics of the logical content, i.e. the addition and deletion of axioms.

In a first prototypical implementation of the ontology questionnaire and the assertional effects within MoKi, the available ontology export functionality was used to create an OWL representation of the MoKi content whenever needed. The problem

with this solution was its low performance, due to the fact that whatever happened, the whole MoKi content had to be exported into RDF and then translated to OWL. This would have been acceptable for the ontology questionnaire, since it could be expected that the ontology questionnaire would not be called very frequently. For the assertional effects on the other hand, this is not at all acceptable since they have to be computed every essentially every time a concept or property page is edited. Furthermore, assertional effects would have required a complete export of the MoKi content twice at each concept and property page edit since the assertional effects work on the difference between a knowledge base before and after an ontology edit.
This led to the decision to keep a constantly synchronised OWL representation of the MoKi content. The goal was to increase the average performance of MoKi while still enabling functionalities that rely on an OWL representation of the MoKi content.

The MoKi *OWL knowledge base*

The OWL representation of the domain model contained within MoKi is called the MoKi OWL knowledge base. It contains all information about the current domain model contained within MoKi, but does not contain information about processes or possibly other content of MoKi. Concerning the domain model, the MoKi OWL knowledge base most notably does not contain information contained within "Free Notes" on model element pages, nor information about the revision history of single pages, users who edited pages nor does it contain the information about the talk pages that relate to model element pages.
In the current implementation, the MoKi OWL knowledge base is stored in a single OWL file in the `images/OntologyExport` subdirectory of the MediaWiki installation directory. Changes to the MoKi content can not be made directly through manipulating the MoKi OWL knowledge base but must be made through the MoKi user interface.

When To Update the MoKi *OWL Knowledge Base*

An analysis of the MoKi user interface shows, that changes to the MoKi content that should be reflected in the MoKi OWL knowledge base can be made through the following activities in the following places:

- Creating, editing, deleting or restoring concept pages
- Creating, editing, deleting or restoring property pages
- Creating, editing, deleting or restoring individual pages
- Modifications in the IsA Browser
- Modifications in the IsPartOfBrowser
- Ontology import functionality
- Data cleaner functionality

Incremental vs. Batch Update of the MoKi *OWL Knowledge Base*

Updating the MoKi OWL knowledge base can be done either incrementally or as a whole. In the first cases, only a small amount of changes (the addition or deletion of

axioms) are registered and applied to the MoKi OWL knowledge base. In the second case, whenever a change occurs, the complete MoKi content that is relevant for the MoKi OWL knowledge base is exported into RDF and translated to OWL. The redundancy is clear: When a single concept, property or individual page is edited, changes occur only in a very small part of the OWL ontology that represents the MoKi content, but still the complete content is exported and translated.
Further analysis of the list of activities that necessitate an update of the MoKi OWL knowledge base shows that the first three activities (creating, editing, deleting or restoring model element pages) are most appropriate for incremental update of the MoKi OWL knowledge base. The last two, ontology import and data cleaner functionality, perform batch update operations on the MoKi content. Embedding incremental updates into these functionalities would slow them down. Modifications in the IsA/IsPartOfBrowser may be single updates or multiple updates, in the end depending on how many changes the user makes within the tree visualisation. It has therefore been decided to incrementally update the MoKi knowledge base when concept, property or individual pages are created, edited, deleted or restored and to suspend incremental update in the other cases. Note that with restoration, the actual implementation deviates from this plan in that updating the MoKi knowledge base after a page restoration is implemented as suspending the incremental update (see further below for a more detailed discussion of this case).
Practically, suspension is achieved by deleting the MoKi OWL knowledge base and lazily exporting it again only when needed, i.e. when assertional effects or the ontology questionnaire require the MoKi OWL knowledge base again.

Implementation of the Incremental Update Functionality

Essentially, the update functionality views the domain model contained within MoKi as an OWL knowledge base, which in turn can be seen as a set of axioms. This is in line with the definition of a knowledge base as $\mathbf{KB} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ as is typical in description logics. In addition, an OWL knowledge base also contains annotation axioms which contain meta-information about elements without logical meaning, such as e.g. verbal descriptions of elements. Then, the core update functionality only needs to deal with atomic ontology edits, the addition or removal of axioms to/from the MoKi OWL knowledge base. The challenge here is to actually detect axioms which have been removed or added in a non-atomic edit activity such as for instance creating, editing, deleting or restoring a model element page. A model element page in this discussion always means either a concept, a property or an individual page.
The technical basis of the incremental OWL update is again a mixture of the SMW built-in RDF export and the translation from the RDF representation into OWL. In contrast to the existing ontology export functionality, only the currently edited model element page is exported into RDF and translated to OWL.

I first discuss the incremental update functionality conceptually, and start with how it works when a single model element page is edited. Based on this, the incremental update functionality can easily be extended to support also creating, deleting or restoring a model element page.
The OWL representation of the model element page after the page has been edited

gives only a static view. In order to detect which axioms have been added or removed, it must be compared with some prior version. A direct comparison with the complete MoKi OWL knowledge base, which represents the state of the domain model before the page edit, would be possible[9] but tedious. Therefore it has been decided, to create the OWL representation of the edited model element page before saving the changes as well as after. This gives, after a model element page `A` has been edited, two OWL representations $\mathbf{KB}'_1$ (stored in a file called `MV_A_orig.owl`) and $\mathbf{KB}'_2$ (stored in a file called `MV_A.owl`), where the comma $'$ denotes that both are only a small excerpt of the complete MoKi OWL knowledge base $\mathbf{KB}$. The incremental update functionality then compares $\mathbf{KB}'_1$ and $\mathbf{KB}'_2$ by a simple set comparison of the axioms in $\mathbf{KB}'_1$ and $\mathbf{KB}'_2$ and determines the sets of added and deleted axioms. The set of added axioms is then added to, and the set of deleted axioms is deleted from, the complete MoKi OWL knowledge base $\mathbf{KB}$.
In case a page is created, the comparison is easy since $\mathbf{KB}'_1$ is empty. In case a model element page `A` is deleted, simply all references to the model element A are deleted from the MoKi OWL knowledge base $\mathbf{KB}$.

The above conceptual design lends itself clearly to be implemented as a series of hook extensions. The code for all these hooks is contained within the file `MV_AssertionalEffects.php` in the `MoKiValidation` directory.
The state of a model element page before changes are saved is retrieved within an `ArticleSave` hook called `mvAE_saveOriginal`. This function outputs, for a concept/property page with name `A`, the file `MV_A_orig.owl`. The state of a model element after changes have been saved is retrieved within an `ArticleSaveComplete` hook called `mvAE_saveNew`. This function outputs, for a concept/property page with name `A`, the file `MV_A.owl`. Also the creation of a page is treated by the `mvAE_saveNew` function . The function `mvAE_saveNew` takes care of computing the difference between $\mathbf{KB}'_1$ and $\mathbf{KB}'_2$ and updates the MoKi OWL knowledge base accordingly. The difference between $\mathbf{KB}'_1$ and $\mathbf{KB}'_2$ is computed by an instance of the Java class `KnowledgeBasePair`, whose only job is to compare two knowledge bases and return comparison values and changes in terms of sets of axioms that must be added or removed in order to transform one knowledge base into the other. Updating the MoKi OWL knowledge base is also done in Java through static methods of a utility class called `Utilities`. The deletion of a page is noticed within an `ArticleDelete` hook called `mvAE_delete`. This function creates the OWL representation of the model element page as it was before deletion. The restoration of a page is noticed within an `ArticleUndelete` hook called `mvAE_undelete`. This function however does not create the OWL representation of the page after restoration, but follows the suspension of incremental update as implemented for the IsA and IsPartOf browsers and the batch update functionalities of ontology import and data cleaning. The main reason for this is that the restoration functionality in MoKi is not

[9] A direct comparison with the complete MoKi OWL knowledge base in order to detect changes made by editing a single element page `A` is possible since editing a single element page can only lead to the addition or removal of axioms where `A` is the subject of the axiom, e.g. $A \sqsubseteq B$. This is inherent in the design of the concept, property and individual page templates.

fully mature since it is not the inverse of a page deletion[10] and is thus in general recommended to be used only with care. Then, since restoration is not expected to be carried out very often, the suspension is not expected to cause a significant loss in performance.

6.3 Implementation of the MoKi Validation Modules

6.3.1 Ontology Questionnaire

The ontology questionnaire is accessible via the MoKi menu entry "Inferences - Do You Agree" or via its special page address "Special:MV_OntologyQuestionnaire". It is implemented as a PHP class that extends the MediaWiki class `SpecialPage` and resides in the file `MV_OntologyQuestionnaire_body.php`. Subsequently, I will abbreviate this class as `OQ_PHP`. The ontology questionnaire uses the MoKi OWL knowledge base as source of information about the domain model contained within MoKi.
Its `execute()` function first checks whether the MoKi OWL knowledge base is available, and if this is not the case, freshly exports the complete MoKi content which is relevant into RDF and translates it to OWL. It then goes on to load the MoKi OWL knowledge base into a Java instance of the class `MoKiOntologyQuestionnaire` (subsequently abbreviated as `OQ_Java`). This Java class is implemented as a quasi-singleton, such that for each OWL file on any filesystem, there is at most one current `OQ_Java` instance. Using the singleton pattern allows the `OQ_PHP` to post intermediate results, such as selecting statements for deletion, confirming or cancelling deletion, back to itself while remembering all previous actions in the Java instance of `OQ_Java` as long as this Java instance is not cleared.

> **Example:** The `OQ_Java` instance remembers the selection of axioms that are to be deleted and computes side-effects of this deletion without actually performing it. In a next step, `OQ_PHP` only needs to know whether deletion has been confirmed or cancelled, without having to remember the actually selected statements since this information is persisted in the singleton `OQ_Java` instance.

The `OQ_PHP` changes its behaviour depending on the requested URL (parameteres that affect its behaviour are the fragment and the query) and on the content of the `$_POST` variable, which is an associative array storing the variables the PHP `OQ_PHP` has received via the an HTTP POST action. The basic actions that `OQ_PHP` performs are the following:

- Display the list of axioms that can be inferred from the MoKi OWL knowledge base
- Jump to a specific inferred statement in the list

[10] The restore functionality only restores the information that was stored within MediaWiki on the concept page that is being restored. A concept can be referred to from other pages however, and the delete functionality deletes all such references but without persisting these additional deletions. They can not be restored therefore. It is possible that in future versions of MoKi, this will be remedied.

– Display the explanation for a selected inferred statement in the list
– Tentatively delete statements selected from explanations, display side-effects, and ask for confirmation.
– Actually delete the selected statements and display a success page.
– Cancel deletion of selected statements and return to the explanation from where the deletion process was started.

Let "$MoKi" denote the base URL of a MoKi installation. Then, if the simple URL "$MoKi/index.php/Special:MV_OntologyQuestionnaire" is requested,
`OQ_PHP` displays the two lists of inferred statements, one caused by the hierarchy of the concepts and properties, and all others (Fig. 6.2).
`OQ_PHP` indexes inferred axioms with a letter $j \in \{A, B\}$ that denotes in which of the two lists an inferences is displayed, and a number i that refers to the position of the axiom within the corresponding list.

> **Example:** The seventh axiom from top of the axioms derived from the concept or property hierarchy has the index "7B".

The fragment is used to denote the position of the display in this list, using the above described indices.

> **Example:** `OQ_PHP` displays the axiom "7B" on top of the page when the URL "$MoKi/index.php/Special:MV_OntologyQuestionnaire#7B" is requested

This anchor is used for internal purposes mostly, so that in case of cancellation of any action, the `OQ_PHP` is able to jump back to exactly the same view as before the cancellation.
If the URL contains a query of the form "explain=ij", the explanation for the indicated axiom is displayed (Fig. 6.3).

> **Example:** `OQ_PHP` displays the explanation for the axiom "7B" when the URL "$MoKi/index.php/Special:MV_OntologyQuestionnaire?explain=7B" is requested.

If `$_POST` contains a value for the key `'selected_axioms'`, `OQ_PHP` asks the `OQ_Java` to create a dedicated Java `OQAxiomRemover` instance which can mark the selected axioms for deletion and can compute side-effects. These are then displayed by `OQ_PHP` (Fig. 6.4).
If `$_POST` contains a value for the key `'remove_btn'`, `OQ_PHP` asks the dedicated `OQAxiomRemover` instance to actually delete the axioms from the MoKi OWL knowledge base and calls a dedicated PHP class called `MV_OWL2DBMapper` to delete the selected axioms in MoKi's database. After deletion, the `OQ_Java` is cleared which forces a fresh loading of the MoKi OWL knowledge base and of the `OQAxiomRemover` instance. A success page is displayed (Fig. 6.5).
If `$_POST` contains a value for the key `'cancel_btn'`, `OQ_PHP` clears the `OQ_Java` which again enforce a fresh loading of the MoKi OWL knowledge base and of the `OQAxiomRemover` instance, and returns to the display of the explanation from which the tentative deletion has been started (Fig. 6.3).

A more in-depth view at an intermediate level of technicality on the ontology questionnaire's implementation within MoKi is given in App. B.1 in a functional view, a logical view (class description and class diagram) and a process view (user actions and triggered method calls, and a sequence diagram).

6.3.2 Assertional Effects

A dedicated `OutputPageBeforeHTML` hook called `mvAE_display` implements the assertional effects functionality within MoKi. The code is located within the file `MV_AssertionalEffectsHook.php`. The assertional effects functionality uses the MoKi OWL knowledge base as source of information about the domain model contained within MoKi.
Assertional effects can be displayed on concept and property pages after a logical change has been made to the MoKi content, i.e. terminological or role axioms have been added or deleted. In any case, even if no assertional effects exist, the logical changes are displayed. It is first necessary to detect logical differences between the domain model before and after an ontology edit. To this purpose, the assertional effects functionality within MoKi relies on the incremental OWL update functionality to create, for a concept/page with name `A`, the two files `MV_A_orig.owl` and `MV_A.owl`. In case only the latter file exists, the page has been newly created. The two OWL files conceptually represent the knowledge bases $\mathbf{KB}'_1$ and $\mathbf{KB}'_2$, which are both small excerpts of the complete MoKi OWL knowledge base $\mathbf{KB}$. The difference between $\mathbf{KB}'_1$ and $\mathbf{KB}'_2$ is computed by an instance of the Java class `KnowledgeBasePair`. If logical differences exist, an instance of the Java class `TeaTypes` is created, which takes the complete MoKi OWL knowledge base $\mathbf{KB}$ plus the changes (added and deleted terminological and role axioms) as input and from there on computes assertional effects as described above.
If assertional effects are found, they are displayed by inserting HTML source code directly between the HTML source code of the quality indicator and that of the rest of the concept page.

A more in-depth view at an intermediate level of technicality on the implementation of assertional effects within MoKi is given in App. B.2 in a functional view, a logical view (class description and class diagram) and a process view (backend actions when a concept or property page is edited or newly created, and an activity diagram).

6.3.3 Models Checklist

The models checklist is a normal wiki page that links to a number of MediaWiki special pages, each presenting the results of a single check to the MoKi user. It is typically created during the MoKi installation process. Alternatively, a `setup_moki.xml` file is available in each MoKi distribution that can be input via the special page "Special:SemanticPageImport". As a third alternative, the same page could be created manually as a simple wiki page within a MoKi installation.
Since the models checklist page is a normal wiki page, it can be modified for different settings by simply editing it. Links can be removed and added (choosing from all

available checks) in usual wiki syntax. The checks concerning the domain model are summarised in Tables 6.1-6.3. Additional checks for processes and for model integration are available in the current MoKi implementation but not described here.

Subsequently, I describe in some more detail the implementation of the check for concepts without verbal descriptions. The other checks are implemented very much in analogy. All checks query the database representation of the models contained within MoKi, and do not use the MoKi OWL knowledge base.
The special page "Special:Concepts_without_verbal_description" is implemented as a PHP class which extends the predefined MediaWiki class `SpecialPage` and resides, together with all other special pages that check concepts for certain characteristics, in a file called `MV_ConceptChecks_body.php`.
Its `execute()` function calls a more generic, globally available, function that is able to retrieve all model elements of a specific kind (i.e. concepts, properties, individuals or processes) without a verbal description. This function in turn first uses a built-in function of the SemanticForms extension that retrieves all pages of a given category, in this case to retrieve all concept pages. It then iterates through the result list, and checks for each concept page with name `ConceptName` whether a triple of the form `ConceptName Description DescriptionText.` exists. Concept pages for which this triple does not exist are returned as having no verbal description. The `execute()` function then outputs a header, a count of concepts without verbal description, a help text, and a list of returned concepts, formatted as a HTML bullet point list (see Fig. 6.9).

6.3.4 Quality Indicator

The quality indicator is implemented as a `OutputPageBeforeHTML` hook called `mvQualityIndicatorHook`. An `OutputPageBeforeHTML` hook is triggered just before a wiki page is displayed as HTML in the browser.
The function `mvQualityIndicatorHook` computes the points for completeness and sharedness for the element whose page is being displayed and prepends the HTML source code that visualises the bars to the HTML body. Similar as above, I describe only the quality indicator for concepts in detail. The implementation for properties, individuals, and processes is very much in analogy, with only a slight difference in characteristics that are computed. A complete list of characteristics for each kind of model elements is given below in Tables 6.4- 6.7. Similar to the models checklist, the quality indicator uses the database representation of the models contained within MoKi, and does not use the MoKi OWL knowledge base.

On a given concept page with name `ConceptName`, the quality indicator computes the the points for completeness as follows: If `ConceptName` has a verbal description, 3 points are given. If at least one synonym is defined, 1 point is given. If `ConceptName` is part of a hierarchy (i.e. if `ConceptName` has superconcepts, subconcepts, has any parts or is part of anything else), 2 points are given. If `ConceptName` is related to other concepts via user-defined properties, 1 point is given for each relation. In order to compute these characteristics, the quality indicator accesses the same, globally available, functions that are also accessed by the models checklist, the only

Checklist for concepts	
	Concepts without verbal description
Lists:	Concepts without a verbal description.
Rationale:	Verbal descriptions for model elements increase the usability of the ontology.
Special Page:	Special:Concepts_without_verbal_description
	Orphaned concepts
Lists:	Concepts which are not part of any hierarchy, i.e. have no super or subconcepts, have no parts, and are not part of anything.
Rationale:	Probably redundant concepts. Typically, concepts are organised hierarchically in an ontology.
Special Page:	Special:Orphaned_Concepts
	Concepts without individuals
Lists:	Concepts which are not instantiated by any individual.
Rationale:	This is not necessarily a modelling mistake, except if it is explicitly intended that every concept shall be instantiated. Instance data can be used to document concepts as examples and to test logical axioms concerning concepts.
Special Page:	Special:Concepts_Without_Individuals
	Non-shared concepts
Lists:	Concepts which have been edited only by a single person.
Rationale:	It is good practice to model an ontology collaboratively. At least, a second person should review the ontology.
Special Page:	Special:Non_shared_Concepts

Table 6.1. Automatically generated checklist for concepts. The checklist lists concepts with characteristics that point to potential modelling mistakes.

Checklist for individuals	
	Individuals belonging to no concept
Lists:	Individuals for who no type is asserted, i.e. whose type is `owl:Thing`.
Rationale:	Most surely a modelling mistake by oversight. Typically, at least one type of an individual is known.
Special Page:	Special:Individuals_With_No_Type_Defined
	Non-shared individuals
Lists:	Individuals which have been edited only by a single person.
Rationale:	Whether sharedness is a relevant characteristic for individuals depends on the kind of knowledge base. In knowledge bases where individuals are subject of discussion, such as in encyclopaedias, sharedness is relevant also for individuals. Per default this check is not shown in the models checklist.
Special Page:	Special:Non_shared_Individuals

Table 6.2. Automatically generated checklist for individuals. The checklist lists individuals with characteristics that point to potential modelling mistakes.

Checklist for properties	
	Properties without verbal description
Lists:	Properties without verbal description.
Rationale:	Verbal descriptions for model elements increase the usability of the ontology.
Special Page:	Special:Properties_without_verbal_description
	Properties with no domain defined
Lists:	Properties for which no domain is defined.
Rationale:	It is not a modelling mistake by itself to not define the domain of a property. However, defining a property's domain increases the logical precision of the ontology and increases the benefits of reasoning over the ontology.
Special Page:	Special:Properties_with_no_domain
	Properties with no range defined
Lists:	Properties for which no range is defined.
Rationale:	It is not a modelling mistake by itself to not define the range of a property. However, defining a property's range increases the logical precision of the ontology and increases the benefits of reasoning over the ontology.
Special Page:	Special:Properties_with_no_range
	Non-shared properties
Lists:	Properties which have been edited only by a single person.
Rationale:	It is good practice to model an ontology collaboratively. At least, a second person should review the ontology.
Special Page:	Special:Non_shared_Properties

Table 6.3. Automatically generated checklist for properties. The checklist lists properties with characteristics that potentially point to modelling mistakes.

difference being that the quality indicator does not need to iterate through a list of concepts but checks only for a single concept.
The collected points are divided into value partitions, i.e. if a concept reaches less than 3 points, it gets a red bar for completeness, if it reaches 3 or 4 points, the concept gets an orange bar. If a concept reaches 5 points, it gets a yellow bar and above that it gets a green bar.

Completeness for concepts	
Points given if . . .	**#Points**
A verbal description for the concept exists	3
At least one synonym exists	1
The concept is part of a hierarchy	2
A user-defined relation to another concept exists	1 for each relation

Table 6.4. The distribution of points given for each characteristic that contributes to the completeness measure of a concept.

Completeness for properties	
Points given if . . .	**#Points**
A verbal description for the property exists	3
At least one synonym exists	1
The property is a subproperty or an inverse of another property, or has a domain or range defined	2

Table 6.5. The distribution of points given for each characteristic that contributes to the completeness measure of a property.

Completeness for individuals	
Points given if . . .	**#Points**
A verbal description for the individual exists	1
At least one synonym exists	1
The individual has a defined type	2
A user-defined relation to another individual exists	1 for each relation

Table 6.6. The distribution of points given for each characteristic that contributes to the completeness measure of an individual. The existence of a verbal description is rated lower than elsewhere, because individuals are concrete real-world entities that in general need less explanation than the more abstract concepts.

Sharedness	
Points given if . . .	**#Points**
Multiple users have contributed to this model element page	1 for each contributor

Table 6.7. The sharedness measure counts the number of active contributors to a model element page. It does not register how many people have viewed the page and agreed with it.

6.4 Discussion

Concerning the contribution to the field of ontology engineering (environments) and ontology evaluation, there are two clear innovations. First, the ontology questionnaire contains an innovation regarding the display of inferences insofar as side-effects of a "repair" activity, such as deleting explicitly made statements in order to remove an undesired inference, are displayed to the user. Additionally, the integration of the ontology questionnaire within MoKi is such, that the MoKi users are pointed towards using it for ontology evaluation purposes. A future change in the implementation could be to move the ontology questionnaire more into the direction of being a "real" questionnaire by providing the users with a possibility to explicitly agree or disagree with single inferred statements. Such a questionnaire could be useful in organisational ontology engineering situations where a formal documentation of ontology evaluation procedures is required.

Second, evaluation is typically seen as an activity separate from each corresponding ontology engineering phase (first, informal modelling, then evaluation of informal model etc.). While MoKi also supports such evaluation activities, it provides functionality to make users aware of potential issues already during modelling. Thus, ontology evaluation in MoKi becomes tightly integrated with modelling itself, moving ontology evalu-

ation into the direction of what is known as “formative evaluation”. To the best of the author’s knowledge, this is an innovative feature which is indeed not present in other comparable ontology engineering environments.

A user evaluation of the MoKi validation functionalities would naturally be of benefit. However, evaluations exist for prior prototypes of the ontology questionnaire and the models checklist (which uses the same characteristics as the quality indicator) [41, 98]. An evaluation of usefulness in a real ontology engineering setting is missing only for the assertional effects functionality, but here the focus was on the theoretical development in the scope of this thesis.

7

Conclusion

The description-logic based formalisms which have established themselves as de-facto standard in the Semantic Web both benefit and suffer from their current, very high, expressivity. On the one hand, complex issues can be described in description logics (although researchers are constantly pushing towards even more expressive languages), while on the other hand human ontology engineers have even now trouble grasping the implications of their explicitly expressed knowledge. This difficulty constitutes a severe impediment to systematic ontology evaluation. Furthermore, in ontology engineering as opposed to software engineering there is no execution environment in which to test an ontology per default, safe the usage of reasoners to check ontologies for logical consistency. This thesis has picked up the state-of-the art in ontology evaluation, which has been at a rather conceptual level, and led it from there to automated support for ontology evaluation.

Summary

To this purpose, first the benefits of including a review of entailed statements into the ontology engineering activity of formal model evaluation have been investigated. An analysis based on a list of typical modelling errors in OWL and experiences from a practical application of the ontology questionnaire revealed the following main points. While missing information cannot be detected through such a review, a variety of modelling errors related to open world reasoning, domain and range constructs and more general logic-related issues can be highlighted. This indicates that the underlying idea is of practical interest for ontology engineers, which is supported by the finding of an experimental study that the amount of invested time is manageable. Furthermore, results of the experimental study indicate that knowledge engineers understand the purpose of such a procedure, and together with the limited time efforts thus are easily motivated to go through such a process. On a different level, the experimental study also gave insight into the complex problem solving approaches taken by ontology engineers, which by large exceeds the support a tool such as the ontology questionnaire can offer based on inference explanations. A deeper research of human creativity and conflict resolution mechanisms involved in this process was unfortunately outside the scope of this thesis. However, I regard this as the most interesting forward-leading research question concerning the investigated approach of reviewing inferred statements

as part of an ontology evaluation procedure. Such research is necessary in order to provide more humanly intelligent automated support for ontology repair. The summarily positive findings have also led to the implementation of the ontology questionnaire within MoKi, which delivers inferences, explanations for inferences, supports the deletion of the cause of an undesired inference and computes side-effects of such a deletion as well.

In a second venture, the delivery of concrete effects of ontology editing activities directly during modelling have been investigated. In collaborative knowledge bases users are expected to contribute not by adding textual or multimedia content as is currently the practice in Web 2.0 like environments, but also facts about individuals or general knowledge e.g. in the form of terminological or role axioms. Contributors to a knowledge base benefit from seeing implications of their ontology editing activities on existing data in three ways. First, assertional effects give concrete examples of the general knowledge (terminological and role axioms) which was removed or added. Second, such effects give contributors the possibility to review their contribution from a viewpoint in addition of the explicitly added, removed or modified statements. Finally, since such effects can be displayed directly during modelling, contributors can immediately assess whether what they modelled is interpreted by a reasoner as intended.
The focus of investigation in this part was on providing a formal definition of what has been termed intuitively "knowledge lost or gained about data". On a theoretical side, I showed that the existence of assertional effects can be decided if the DL language in question is decidable through a tableaux algorithm, or if it has the connected model property contains role disjunction and the Kleene-operator. The first condition holds for $\mathcal{SROIQ}$, the DL language underlying OWL 2, which makes the integration of a reasoning service deciding on the existence of assertional effects and displaying them interesting for OWL 2 modelling tools. Following such considerations, the assertional effects functionality was integrated in MoKi, including several practical extensions in comparison to the exhaustive theoretical discussion on assertional effects. The main open challenge for broad applicability of displaying assertional effects is to find suitable exemplary individuals within a knowledge base on which to illustrate effects.

Both theoretically oriented ventures have led to the implementation of validation functionalities within MoKi. Together with two more validation functionalities in MoKi, the models checklist and the quality indicator, MoKi has thus been extended to broadly support ontology evaluation activities both during modelling and as a separate evaluation activity in the ontology engineering process. This is definitely one aspect that makes of MoKi an outstanding ontology engineering environment compared to other state-of-the art tools.

Self-Assessment

The goal of this thesis was to investigate how reasoning, as something that machines can do well given a logic-based knowledge representation formalism, can be used to support humans in assessing whether what they expressed as logical formulae corresponds to what they intended to say. Some progress in understanding has been made by this thesis in particular in the area of delivering inferences and finding modelling errors through reviewing inferences. Among the achievements are first the confirmation that

ontology engineers do find reviewing inferred statements useful, while occasionally preferring to solve detected problems less technically than can be currently supported for instance through inference explanation. Second, a new reasoning service that computes effects of terminological axioms on data has been proposed to fill the gap between TBox related reasoning, on which there is a lot of research, and data. Third, the achieved results have been integrated in an ontology engineering environment, MoKi, and are thus easily available for a wider public to use. Nonetheless, as has been pointed out above, there are also unresolved issues that would lend themselves well to further research. First, this concerns the question how people resolve modelling errors once they have detected them. As has been pointed out, such resolution may be more complex and indirect than solutions that can be automatically proposed by using ontology debugging and repair services. Second, this concerns the provision of exemplary individuals given a knowledge base with a lot of data, assuming that many individuals in the knowledge base are similar to some extent.

References

[1] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing. http://www.w3.org/TR/rdfa-syntax/, Last visited: 2010-02-18.

[2] Carlos E. Alchourròn, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[3] APOSDLE - Advanced Process Oriented Self-Directed Learning Environment. http://www.aposdle.org. Last visited: 2010-02-18.

[4] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[5] Ask - Semantic search in Semantic MediaWiki. http://semantic-mediawiki.org/wiki/Help:Semantic_search. Last visited: 2010-02-18.

[6] Attempto Controlled English. http://attempto.ifi.uzh.ch. Last visited: 2010-02-18.

[7] Sören Auer, Sebastian Dietzold, and Thomas Riechert. Ontowiki - a tool for social, semantic collaboration. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, 2006.

[8] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[9] Michael Backhaus and Janet Kelso. Bowiki - a collaborative annotation and ontology curation framework. In Natalya Fridman Noy, Harith Alani, Gerd Stumme, Peter Mika, York Sure, and Denny Vrandecic, editors, *CKC*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[10] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, Last visited: 2010-02-18.

[11] David A. Bell, Guilin Qi, and Weiru Liu. Approaches to inconsistency handling in description-logic based ontologies. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *OTM Workshops (2)*, volume 4806 of *Lecture Notes in Computer Science*, pages 1303–1311. Springer, 2007.

[12] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[13] BOWiki. http://bowiki.net/. Last visited: 2010-02-18.

[14] Janez Brank, Marko Grobelnik, and Dunja Mladenic. A survey of ontology evaluation techniques. In *Conference on Data Mining and Warehouses*, 2005.

[15] PHP/Java Bridge. http://php-java-bridge.sourceforge.net/pjb/. Last visited: 2010-02-18.

[16] Michel Buffa and Fabien Gandon. Sweetwiki: semantic web enabled technologies in wiki. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 69–78, New York, NY, USA, 2006. ACM.

[17] Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini. *Ontology Learning From Text: Methods, Evaluation and Applications*, volume 123 of *Frontiers in Artificial Intelligence and Applications*, chapter Ontology Learning from Text: An Overview, pages 1–10. IOS Press, 2005.

[18] Patryk Burek, Robert Hoehndorf, Frank Loebe, Johann Visagie, Heinrich. Herre, and Janet Kelso. A top-level ontology of functions and its application in the open biomedical ontologies. *Bioinformatics*, 22(14), July 2006.

[19] Business Process Modeling Notation. http://www.bpmn.org/. Last visited: 2010-02-18.

[20] Philipp Cimiano and Johanna Völker. Text2Onto - a framework for ontology learning and data-driven change discovery. In Andres Montoyo, Rafael Munoz, and Elisabeth Metais, editors, *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB),*, volume 3513 of *Lecture Notes in Computer Science*, pages 227–238, Alicante, Spain, June 2005. Springer. ISBN: 3-540-26031-5.

[21] Nancy J. Cooke. Varieties of knowledge elicitation techniques. *International Journal of Human-Computer Studies*, 41(6):801–849, 1994.

[22] Oscar Corcho, Mariano Fernandez-Lopez, Asuncion Gomez-Perez, and Oscar Vicente. WebODE: an integrated workbench for ontology representation, reasoning and exchange. In *13th International Conference on Knowledge Engineering an Knowledge Management (EKAW'02)*, volume 2473 of *Lecture Notes on Artificial Intelligence*, pages 138–153. Springer-Verlag, October 2002.

[23] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[24] Dewey Decimal Classification system. http://www.oclc.org/dewey/, last visited: 2010-02-18.

[25] DHTMLx Tree Library. http://www.dhtmlx.com/docs/products/dhtmlxTree/index.shtml. Last visited: 2010-02-18.

[26] dmoz Open Directory Project. http://www.dmoz.org. Last visited: 2010-02-18.

[27] DOLCE: A Descriptive ontology for Linguistic and Cognitive Engineering. http://www.loa-cnr.it/DOLCE.html, last visited: 2010-02-18.

[28] John Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *In Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1998.

[29] Nick Drummond. Outline/existential view. http://code.google.com/p/co-ode-owl-plugins/wiki/OutlineView, Last visited: 2010-02-18.

[30] Fact++. http://code.google.com/p/factplusplus/, last visited: 2010-02-18.

[31] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: a tool for collaborative ontology construction. *Int. J. Hum.-Comput. Stud.*, 46(6):707–727, 1997.

[32] Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* MIT Press, 2003.

[33] Mariano Fernandez, Asuncion Gomez-Perez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40, Stanford, USA, March 1997.

[34] Mariano Fernandez-Lopez, Asuncion Gomez-Perez, Juan Pazos Sierra, and Alejandro Pazos Sierra. Building a chemical ontology using Methontology and the Ontology Design Environment. *Intelligent Systems and Their Applications, IEEE*, 4(1):37–46, Jan/Feb 1999.

[35] Achille Fokoue, Aaron Kershenbaum, Li Ma, Edith Schonberg, and Kavitha Srinivas. The summary abox: Cutting ontologies down to size. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2006.

[36] Semantic Forms. http://www.mediawiki.org/wiki/Extension:Semantic_Forms. Last visited: 2010-02-18.

[37] A. Gangemi. Ontology design patterns for semantic web content. In Yolanda Gil, Enrico Motta, Richard V. Benjamins, and Mark Musen, editors, *The Semantic Web - Proceedings of the Fourth International Semantic Web Conference, ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 262–276. Springer Berlin, 2005.

[38] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling Ontology Evaluation and Validation. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications: 3rd European Semantic Web Conference, ESWC 2006 Budva, Montenegro, June 11-14,*

2006 Proceedings, volume 4011 of *Lecture Notes in Computer Science*, pages 140 – 154. Springer, 2006.

[39] Lars Marius Garshol. Living with topic maps and rdf-s/owl. Technical report, Ontopia.

[40] Chiara Ghidini, Viktoria Pammer, Peter Scheir, Luciano Serafini, and Stefanie Lindstaedt. Aposdle: learn@work with semantic web technology. In *I-SEMANTICS '07, 3rd International Conference on Semantic Technologies*, 2007.

[41] Chiara Ghidini, Marco Rospocher, Barbara Kump, Viktoria Pammer, Andreas Faatz, and Andreas Zinnen. Integrated modelling methodology version 2, APOSDLE Deliverable 1.6, April 2009.

[42] Chiara Ghidini, Marco Rospocher, Luciano Serafini, Barbara Kump, Viktoria Pammer, Andreas Faatz, and Joanna Guss. Integrated Modelling Methodology Version 1, APOSDLE Deliverable 1.3. Deliverable D3.1, Aposdle (www.aposdle.org), Nov 2007.

[43] Chiara Ghidini, Marco Rospocher, Luciano Serafini, Barbara Kump, Viktoria Pammer, Andreas Faatz, Andreas Zinnen, Joanna Guss, and Stefanie Lindstaedt. Collaborative knowledge engineering via Semantic MediaWiki. In *Proceedings of the Third International Conference on Semantic Systems (I-Semantics 2008), Graz, Austria, Sept. 3-5 2008*, pages 134–141, 2008.

[44] Silvio Ghilardi, Carsten Lutz, and Frank Wolter. Did I damage my ontology? A case for conservative extensions in description logics. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of KR2006: the 20th International Conference on Principles of Knowledge Representation and Reasoning, Lake District, UK, June 2–5, 2006*, pages 187–197. AAAI Press, 2006.

[45] Bernhard Gissing and Klaus Tochtermann. *Corporate Web 2.0 Bankd I: Web 2.0 und Unternehmen - Wie passt das zusammen?* Shaker Verlag, 2007.

[46] Asuncion Goméz-Pérez. Some ideas and examples to evaluate ontologies. In *Proceedings of the 11th Conference on Artificial Intelligence for Applications*, pages 229–305. IEEE Computer Society, Feb 1995.

[47] Michael Granitzer. *Konzeption und Entwicklung eines generischen Wissenserschliessungsframeworks*. PhD thesis, Graz University of Technology, 2006.

[48] Michael Grüninger. Designing and evaluating generic ontologies. In *Workshop on Ontological Engineering, European Conference on Artificial Intelligence*, 1996.

[49] Michael Grüninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing: Montreal, Quebec, Canada: 1995, August, 20-25*, pages 6.1–6.10, 1995.

[50] Nicola Guarino. Formal ontology and information systems. In Nicola Guarino, editor, *Formal Ontology and Information Systems, Proceedings of FOIS 1998.*, pages 3–15. IOS Press, 1998.

[51] Nicola Guarino and Pierdaniele Giaretta. Ontologies and Knowledge Bases: Towards a Terminological Clarification. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32, 1995.

[52] Nicola Guarino and Chris Welty. *Handbook on Ontologies*, chapter An Overview of OntoClean, pages 151–172. International Handbooks on Information Systems. Springer, 2004.

[53] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 353–367. Springer, 2005.

[54] Pascal Hitzler, Rudolf Sebastian, and Markus Krötzsch. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, London, 2009.

[55] MediaWiki Hooks. `http://www.mediawiki.org/wiki/Manual:Hooks`. Last visited: 2010-02-18.

[56] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in owl. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2008.

[57] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

[58] Ian Horrocks and Ulrike Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, 2007.

[59] José Kahan and Marja-Ritta Koivunen. Annotea: an open rdf infrastructure for shared web annotations. In *WWW 2001: Proceedings of the 10th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2001. ACM.

[60] Kaarel Kaljurand and Norbert E. Fuchs. Verbalizing OWL in Attempto Controlled English. In *Proceedings of Third International Workshop on OWL: Experiences and Directions, Innsbruck, Austria (6th–7th June 2007)*, volume 258, 2007.

[61] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon J B Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 267–280, Berlin, Heidelberg, November 2007. Springer Verlag.

[62] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A web ontology editing browser. *Elsevier Journal of Web Semantics*, 4(2):144–153, 2006.

[63] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005.

[64] KiWi - Knowledge in a Wiki. `http://www.kiwi-project.eu/`. Last visited: 2010-02-18.

[65] Werner Klieber, Vedran Sabol, Markus Muhr, Roman Kern, Georg Öttl, and Michael Granitzer. Knolwedge discovery useing the knowminer framework. In *IADIS International Conference Information Systems*, pages 307–314, 2009.

[66] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Can you tell the difference between dl-lite ontologies? In *KR 2008: Eleventh International Conference on Principles of Knowledge Representation and Reasoning*, 2008.

[67] Markus Krötzsch, Denny Vrandecic, Max Völkel, Heiko Haller, and Rudi Studer. Semantic wikipedia. *Journal of Web Semantics*, 5:251–261, Sept. 2007.

[68] Tobias Kuhn. AceWiki: A Natural and Expressive Semantic Wiki. In *Proceedings of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*. CEUR Workshop Proceedings, 2008.

[69] Tobias Kuhn. Acewiki: Collaborative ontology management in controlled natural language. In *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008)*, Tenerife, Spain, June 2 2008. CEUR Workshop Proceedings.

[70] Leonid Libkin and Cristina Sirangelo. Open and closed world assumptions in data exchange. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[71] Stefanie Lindstaedt, Tobias Ley, and Harald Mayer. Aposdle - new ways to work, learn and collaborate. In *Proccedings of the 4th Conference on Professional Knowledge Management WM2007*, pages 227–234, Potsdam, Germany, March 28-30 2007. GITO-Verlag Berlin.

[72] Stefanie Lindstaedt, Peter Scheir, Robert Lokaiczyk, Barbara Kump, Günter Beham, and Viktoria Pammer. Knowledge services for work-integrated learning. In *Proceedings of the European Conference on Technology Enhanced Learning (ECTEL) 2008 Maastricht, The Netherlands, September 16-19*, pages 234–244, 2008.

[73] Linked Data - Connect Distributed Data Across the Web. http://linkeddata.org/, Last visited: 2010-02-18.

[74] Paul Benjamin Lowry, Aaron Curtis, and Michelle René Lowry. Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice. *Journal of Business Communication*, 41(1):66–99, 2004.

[75] Carsten Lutz, Franz Baader, Enrico Franconi, Domenico Lembo, Ralf Möller, Ricardo Rosati, Ulrike Sattler, Boontawee Suntisrivaraporn, and Sergio Tessaris. Reasoning support for ontology design. In Bernardo Cuenca Grau, Pascal Hitzler, Connor Shankey, and Evan Wallace, editors, *OWL: Experiences and Directions 2006*, 2006.

[76] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative extensions in expressive description logics. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*. AAAI Press, 2007.

[77] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[78] Frank Manola and Eric Miller. RDF Primer. http://www.w3.org/TR/rdf-primer/, Last visited: 2010-02-18.

[79] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, Alessandro Oltramari, and Luc Schneider. The wonderweb library of foundational ontologies. Deliverable D17, National Research Council, Institute of Cognitive Sciences and Technology, May 2003.

[80] Deborah L. McGuinness. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.*, chapter Ch. 5: Ontologies Come Of Age, pages 171–194. MIT Press, 2003.

[81] MediaWiki. http://www.mediawiki.org. Last visited: 2010-02-18.

[82] MediaWiki Extensions. http://www.mediawiki.org/wiki/Manual:Extensions. Last visited: 2010-02-18.

[83] MoKi - The MOdelling WiKI. http://moki.fbk.eu. Last visited: 2010-02-18.

[84] Graham Moore. Rdf and topicmaps: An exercise in convergence. In *XML Europe 2001*, Berlin, Germany, 2001.

[85] myOntology. http://www.myontology.org/. Last visited: 2010-02-18.

[86] Allen Newell. The knowledge level (presidential address). *AI Magazine*, 2(2):1–20, 33, 1980.

[87] Ian Niles and Adam Pease. Towards a standard upper ontology. In Chris Welty and Barry Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, October 2001.

[88] Natalya F. Noy, Abhita Chugh, and Harith Alani. The ckc challenge: Exploring tools for collaborative knowledge construction. *Intelligent Systems, IEEE*, 23(1):64–68, January-February 2008.

[89] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, March 2001.

[90] Natasha Noy and Alan Rector. Defining N-ary Relations on the Semantic Web, http://www.w3.org/TR/swbp-n-aryRelations/. Last visited: 2010-02-18.

[91] Onto Wiki. http://ontowiki.net/Projects/OntoWiki. Last visited: 2010-02-18.

[92] Ontology Design Patterns. http://ontologydesignpatterns.org. Last visited: 2010-02-18.

[93] Ontology Questionnaire. http://services.know-center.tugraz.at:8080/InteractiveOntologyQuestionnaire. Last visited: 2010-02-18.

[94] Tim O'Reilly. What is web 2.0: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.htm. Last visited: 2010-02-18.

[95] Magdalena Ortiz. An automata-based algorithm for description logics around $\mathcal{SRIQ}$. In *Proceedings of the Fourth Latin American Workshop on Logic/Languages, Algorithms and Non-Monotonic Reasoning 2008*

(LANMR'08), volume 408, Puebla,Mexico, October 2008. CEUR Workshop Proceedings.

[96] OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax. http://www.w3.org/tr/owl2-overview/. Last visited: 2010-02-18.

[97] Viktoria Pammer, Barbara Kump, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Stefanie Lindstaedt. Revision support for modeling tasks, topics and skills. In *Proceedings of I-SEMANTICS 2009, 5th International Conference on Semantic Systems, Graz, Austria, September 2-4*, pages 501–508, Graz, Austria, September 2-4 2009.

[98] Viktoria Pammer and Stefanie Lindstaedt. Ontology evaluation through assessment of inferred statements: Study of a prototypical implementation of an ontology questionnaire for owl dl ontologies. In Dimitris Karagiannis and Zhi Jinpeng, editors, *Knowledge Science, Engineering and Management, Third International Conference, KSEM 2009*, number 5914 in Lecture Notes in Artificial Intelligence, pages 394–405, Vienna, Austria, November 25-27 2009. Springer.

[99] Pellet - The Open Source OWL DL Reasoner. `http://clarkparsia.com/pellet/`. Last visited: 2010-02-18.

[100] Steve Pepper, Fabio Vitali, Lars Marius Garshol, Nicola Gessa, and Valentina Presutti. A Survey of RDF/Topic Maps Interoperability Proposals. `http://www.w3.org/TR/rdftm-survey/`, Last visited: 2010-02-18.

[101] Helena Sofia Pinto and João P. Martins. Ontologies: How can they be built? *Knowledge and Information Systems*, 6(4):441–464, July 2004.

[102] Protégé. `http://protege.stanford.edu/`. Last visited: 2010-02-18.

[103] Protégé - Explanation Workbench. `http://owl.cs.manchester.ac.uk/explanation/`, Last visited: 2010-02-18.

[104] Guilin Qi and Fangkai Yang. A survey of revision approaches in description logics. In *Proceedings of the 21st International Workshop on Description Logics (DL'08)*, volume 353. CEUR, MAY 2008.

[105] Racer. `http://www.racer-systems.com/`, last visited: 2010-02-18.

[106] RaDON - Repair and Diagnosis in Ontology Networks. `http://www.neon-toolkit.org/wiki/RaDON`, Last visited: 2010-02-18.

[107] RDFa Primer - Bridging the Human and Data Webs. `http://www.w3.org/TR/xhtml-rdfa-primer/`. Last visited: 2010-02-18.

[108] Alan Rector. Representing specified values in owl: "value partitions" and "value sets",. `http://www.w3.org/TR/swbp-specified-values/`, Last visited: 2010-02-18, May 17 2005.

[109] Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *EKAW*, volume 3257 of *Lecture Notes in Computer Science*. Springer, 2004.

[110] Marco Rospocher, Chiara Ghidini, Viktoria Pammer, Luciano Serafini, and Stefanie Lindstaedt. MoKi: The Modelling Wiki. In *Proceedings of the Forth Semantic Wiki Workshop (SemWiki 2009), co-located with 6th European Semantic*

Web Conference (ESWC 2009), volume 464 of *CEUR Workshop Proceedings*, pages 113–128, 2009.

[111] Catherine Roussey, Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of owl ontology antipatterns. In *K-CAP '09: Proceedings of the fifth international conference on Knowledge capture*, pages 205–206, New York, NY, USA, 2009. ACM.

[112] Stuart Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Pearson Education International, 2003.

[113] Werner Schachner and Klaus Tochtermann. *Corporate Web 2.0 Band II: Web 2.0 und Unternehmen - das passt zusammen!* Shaker Verlag, 2008.

[114] Sebastian Schaffert. Ikewiki: A semantic wiki for collaborative knowledge management. In *1st International Workshop on Semantic Technologies in Collaborative Applications STICA 06*, Manchester, UK, June 2006.

[115] Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic wikis. *IEEE Software*, 25(4):8–11, 2008.

[116] Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, and Stephanie Stroka. Kiwi - a platform for semantic social software. In *Proceedings of the Forth Semantic Wiki Workshop (SemWiki 2009), co-located with 6th European Semantic Web Conference (ESWC 2009)*, volume 464, pages 171–185, Crete, Greece, June 1 2009. CEUR Workshop Proceedings.

[117] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[118] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39(3):317–349, 2007.

[119] Katharina Siorpaes and Martin Hepp. myontology: The marriage of ontology engineering and collective intelligence. In *Bridging the Gep between Semantic Web and Web 2.0 (SemNet 2007), co-located with the 4th European Semantic Web Conference (ESWC 2007)*, pages 127–138, Innsbruck, Austria, June 7 2007. Springer.

[120] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[121] Peter Spyns, Robert Meersman, and Mustafa Jarrar. Data modelling versus ontology engineering. *SIGMOD Rec.*, 31(4):12–17, 2002.

[122] Rudi Studer, Anupriya Ankolekar, Pascal Hitzler, and York Sure. A semantic future for ai. *IEEE Intelligent Systems*, 21(4):8–9, Juli 2006.

[123] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge Engineering: Principles and Methods. *IEEE Transactions on Data Knowledge Engineering*, 25(1-2):161–197, 1998.

[124] Suggested Upper Merged Ontology (SUMO). http://www.ontologyportal.org/. Last visited: 2010-02-18.

[125] SweetWiki - Semantic WEb Enabled Technology Wiki. http://sweetwiki.inria.fr/sweetwiki/. Last visited: 2010-02-18.

[126] Swoop. http://code.google.com/p/swoop/. Last visited: 2010-02-18.

[127] Martin Tanler. Unit testing for ontologies. Bachelor Thesis, Digital Enterprise Research Institute, University of Innsbruck.

[128] Adolfo Lozano Tello and Asunción Gómez-Pérez. Ontometric: A method to choose the appropriate ontology. *Journal of Database Management*, 15(2):1–18, 2004.

[129] Christoph Tempich, Elena Simperl, H. Sofia Pinto, and Rudi Studer. Argumentation-based ontology engineering. *IEEE Intelligent Systems*, 22(6), November 2007. to appear.

[130] The NeOn Toolkit. http://www.neon-toolkit.org/. Last visited: 2010-02-18.

[131] The OWL API. http://owlapi.sourceforge.net/. Last visited: 2010-02-18.

[132] Stefan Herbert Tiran. Ursprung von Inferenzen finden und bei Bedarf löschen. Unveröffentlichte Bakkalaureatsarbeit, Technische Universität Graz, 2008.

[133] Klaus Tochtermann. Das Future Internet. *ispa News*, 4:13–15, 2009.

[134] Topic Maps. http://www.topicmaps.org. Last visited: 2010-02-18.

[135] Tanja Tudorache, Natalia Noy, Samson Tu, and Mark A. Musen. Supporting collaborative ontology development in protege. In *Seventh International Semantic Web Conference*, Karlsruhe, Germany, 2008. Springer.

[136] Unified Modeling Language. http://www.uml.org/. Last visited: 2010-02-18.

[137] Mike Uschold and Michael Grüninger. Ontologies: Principles, methods, applications. In *Knowledge Engineering Review*, volume 11, pages 93–155. 1996.

[138] Mike Uschold and Martin King. Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conduction with IJCAI-95*, Montreal, Canada, 1995.

[139] Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Aeon - an approach to the automatic evaluation of ontologies. *Journal of Applied Ontology*, 3(1-2):41–62, 2008.

[140] Denny Vrandecic and Aldo Gangemi. Unit tests for ontologies. In Mustafa Jarrar, Claude Ostyn, Werner Ceusters, and Andreas Persidis, editors, *Proceedings of the 1st International Workshop on Ontology content and evaluation in Enterprise*, LNCS, Montpellier, France, OCT 2006. Springer.

[141] Denny Vrandecic, H. Sofia Pinto, York Sure, and Christoph Tempich. The diligent knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, Oktober 2005.

[142] Wikipedia. http://www.wikipedia.org. Last visited: 2010-02-18.

[143] XHTML 1.0 The Extensible HyperText Markup Language. http://www.w3.org/TR/xhtml1/. Last visited: 2010-02-18.

A

Ontology Questionnaire Manual

This manual was distributed in December 2008 to the ontology engineers who used the ontology questionnaire in the scope of the APOSDLE project [3, 71]. It describes the usage of the then-current version of the Ontology Questionnaire [93].

1 Ontology Questionnaire

These questionnaires are meant to propose to the Knowledge Experts statements and questions that are extracted from the models contained in the MoKi and aim to verify if the Knowledge Experts (KE) agree with those statements. If not, this obviously triggers a request for some manual verification and revision of parts of the models contained in the MoKi. The questionnaire concerns only the domain model.
APs should use it on-line once the first revision (manual and automatic checks) is completed.

The ontology questionnaire is made for the purpose of letting a Knowledge Expert verify the "knowledge" that can be inferred from an ontology and remove it in case it was not intended. The rationale behind this is, that neither the knowledge expert nor the knowledge engineer explicitly state wrong things. Nevertheless, they might encode their knowledge in the ontology in such a way that they do not agree with everything that can be inferred from it. This can be due either to not well knowing the used formalism (OWL DL) or to having a large and complex domain ontology. After seeing the inferred statements, the KE or knowledge engineer might disagree with an inferred statement and wish to remove it. This is not directly possible because it is inferred and not stated. The ontology questionnaire finds the reason for an inferred statement, and lets the user remove the reason for the inference.

In the following I use the terms "axiom" and "statement" interchangeably.

1.1 Conceptual walk through the ontology questionnaire

The ontology questionnaire uses a reasoner on an (OWL DL) ontology to infer statements.

Example: "ANOVA subClassOf Test" is a statement. It states that the concept "ANOVA" is a subclass of the concept "Test". Other ways of expressing this could be: "Everything which is an ANOVA is also a Test" (in nearly natural language) or " $ANOVA \subseteq Test$ (in a formal language).

It then shows the list of inferences to the knowledge expert. The knowledge expert should read through these statements carefully. In case of disagreeal, the knowledge engineer can get the reason why this statement was inferred.

Example: "ANOVA subClassOf Test" was inferred because of the statements "ANOVA subClassOf Parametric_Test" and "Parametric_Test subClassOf Test". If the KE disagrees, either of the two statements must be removed. Then, the offending statement "ANOVA subClassOf Test" will not be inferred anymore from the ontology.

Aposdle - Specific

One important point for the usage of the Questionnaire in APOSDLE is, that although the changed ontology can in principle be saved directly, for APOSDLE you must note the axioms you deleted and delete them manually in the MoKi. How this can best be done is also described in detail below – we expect that to be quite fast and easy however.

1.2 Step-by-Step through the Questionnaire

1.2.1 Start the questionnaire

Figure 1

Click on the link “Click here to start the interactive ontology questionnaire”.

1.2.2 Upload your domain ontology on

Upload the domain ontology for which you want to verify the inferences.

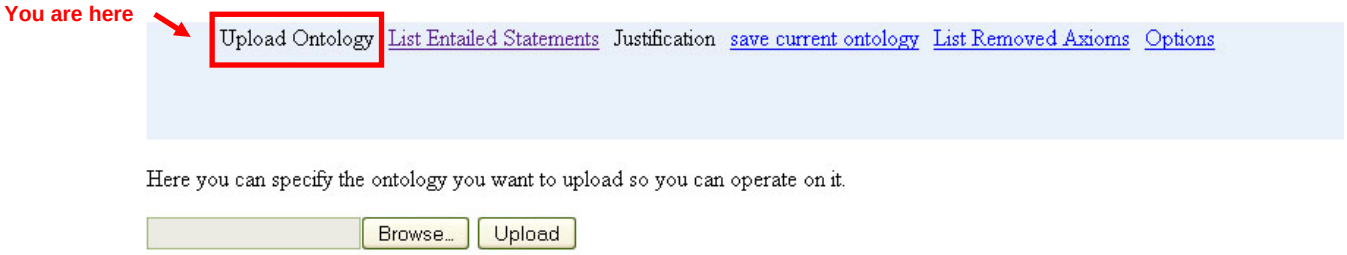


Figure 2

Click on “Browse” to open a file dialogue and browse for your ontology-file. Click on “Upload” to upload it.

Aposdle-Specific

If XXX is a prefix like “EADS”, “ISN” and so on for your company, this file is called “XXXdomain-ontology.owl”. If you do not know where you can find it, ask your coach for it.

1.2.3 Navigation

The header of the page shows the following entries:

- Upload Ontology
- List Entailed Statements
- Justification
- Save current ontology
- List Removed Axioms
- Options

These are the different views of the ontology. At every point in time, the views that are open to you are displayed as links. Click on them to go there.
A plain-text-view is either closed to you or you are currently seeing it.

1.2.4 List inferred statements

You will automatically be transferred to the “List Entailed Statements”-View.

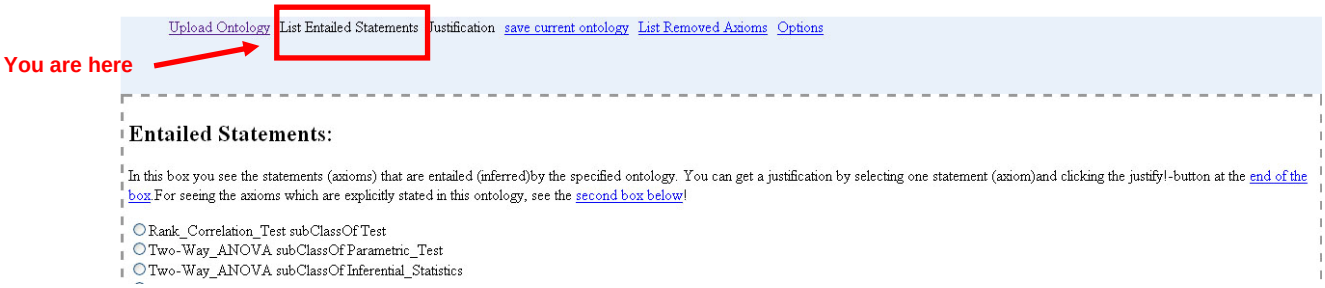


Figure 3

On the displayed page, you see two boxes: One with the title “Entailed Statements” and one with the title “Axioms” (see Figure 4). We call the first the “Entailed Statements” – box and the second the “Explicit Statements” – box.
The first box shows the statements which are inferred from the uploaded ontology. If you open the *.owl-file with a text editor you would not find these statements written there.
The second box shows the statements which were explicitly given in the MoKi.

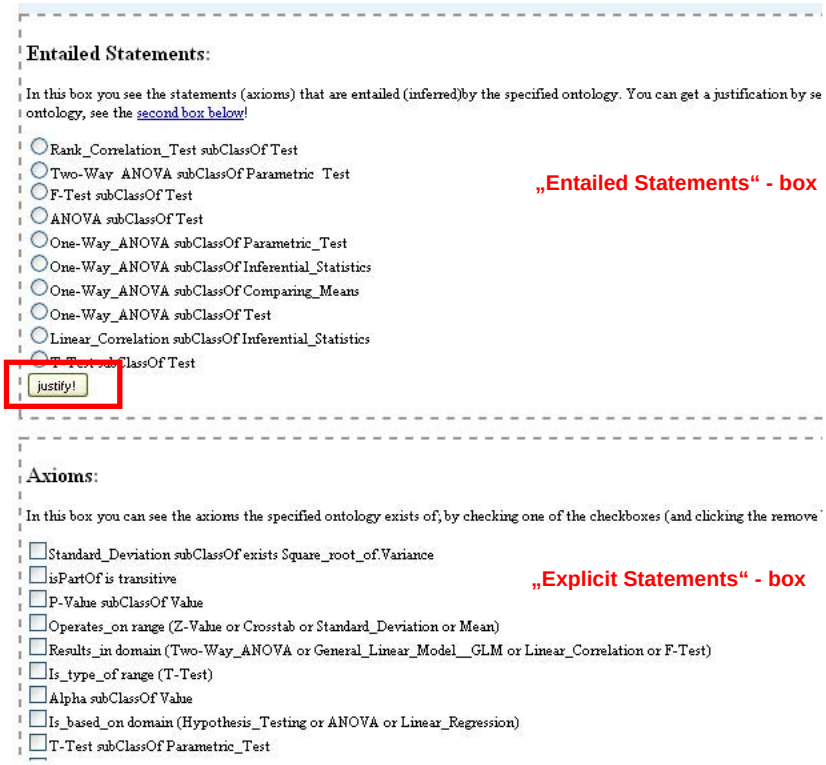


Figure 4

1.2.5 Find out the reason for an inferred statement and optionally delete it

If you want to know why a statement has been inferred, select the corresponding radio button and click the button “Justify” at the bottom of the “Entailed Statements” – box (see Figure 4).

You will be taken to the Justification – View (see Figure 5).

In the first line you will see for which statement you are shown the reason. In the rose box you find one or more groups of statements. Each **group represents one reason** for the selected axiom.

- You can now simply **go back** to the list of entailed statements, or to another view.
- If you want to **delete the selected axiom** from the ontology you have two choices. You can not directly delete an inferred axiom, because it is not explicitly stated in the ontology. You can only remove the reasons why this axiom was inferred.
  - In the blue box there is a suggestion which axiom to remove. In order to accept this choice, click on the button “Delete Minimum Hitting Set”.

In the rose box you find one or more groups of statements. As each group represents one reason for the inferred axiom, you must remove one line from each group. You can do nothing wrong: the radio buttons ensure that you have selected one from each group. Click on the button “Remove” to remove all selected axioms.

Note that deleting axioms in the ontology does not change in any way your local ontology file! All changes are made on the server on a temporary model!

Note that after removing the reason(s) for the selected axiom you can go back to the “List Entailed Statements” – View and if you check, you should not find it in the list anymore.

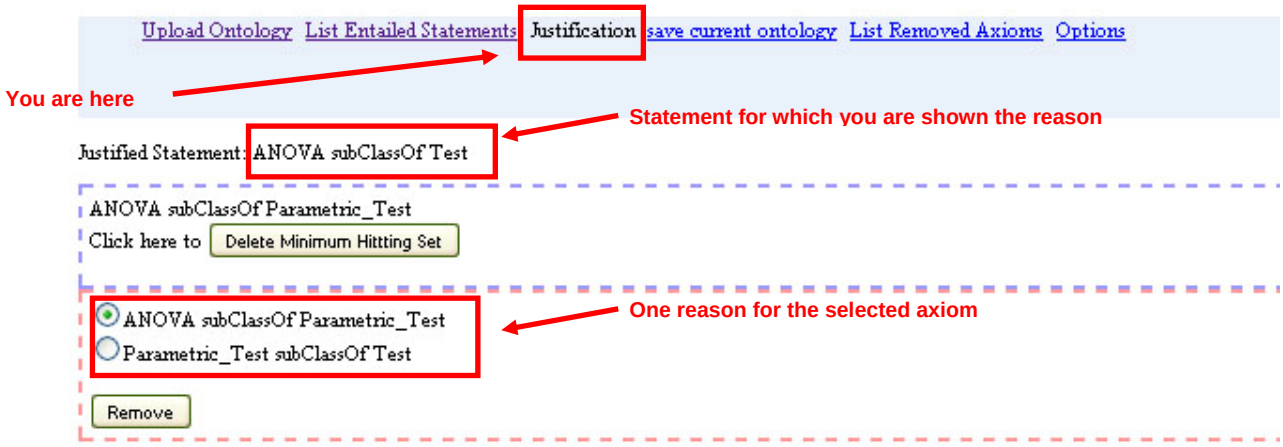


Figure 5

1.2.6 Undo and check which axioms you have already removed

Go to the “List Removed Axioms” – View.

You see a list of axioms / statements which you have removed from the ontology since you uploaded it to the questionnaire. By checking the checkbox in front of one or more axioms and then clicking “Reinsert!” you can add them again to the ontology, thus undoing your changes.

Aposdle – Specific

When you are finished with the questionnaire, i.e. when you have reviewed all inferred statements and are ready to assert the changes, go the the “List Removed Axioms” – View. For each axiom that is listed there: If it says “A subClassOf B”, then go to the concept page of the concept “A” in the MoKi. In the line “Is A” you should see the concept “B”. Edit the concept description and remove the concept “B”.
Please, for evaluation purposes, copy and paste the list of removed axioms into an email and send it to Viktoria Pammer (vpammer@know-center.at).

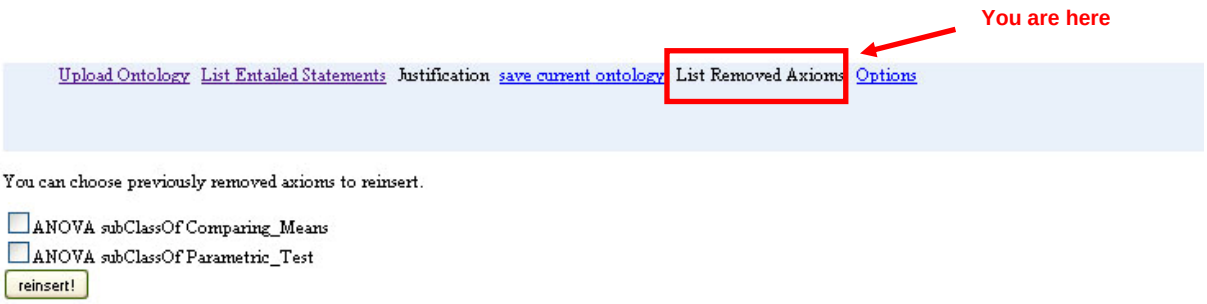


Figure 6

1.3 Additional features

1.3.1 Delete explicitly given statements

In the “Explicit statements” – box (see Figure 4) you see statements that were explicitly given in the ontology. If you decide you do not want to state this after all, you can simply check the checkbox corresponding to the statement and click on the “Delete” Button at the bottom of the box.

1.3.2 Save current ontology

In case you want to save the changed ontology to your local system, to to the “Save current ontology”-View. Depending on the browser you use, you will either be prompted directly to save the file or you will see a lot of text (RDF/XML) in the browser window. In this case, click on File and “Save As…” to save the ontology.

Note that the ontology questionnaire does not store labels, comments or similar things!

1.3.3 Options

In the “Options”-View you can (un)check the option “Use symbolic rendering engine”. After changing the selection you must click “Submit”.

If this checkbox ix checked, the statements will be shown as $ANOVA \subseteq Test$. If it is unchecked, statements will be shown as “Anova subClassOf Test”.

1.4 Known issues and bugs

The ontology questionnaire does not deal with imported ontologies. So if an ontology contains imports, the reasoning is done only over the statements within the uploaded file.

The ontology questionnaire does not store labels, comments or similar things.

The ontology questionnaire relies on Pellet to do the reasoning. If Pellet cannot deal with an ontology, the questionnaire cannot either. In case uploading an ontology takes too long, try loading the ontology into Protégé 4 and classifying it with Pellet. If this works, you have discovered a bug in the ontology questionnaire. If Pellet in Protégé 4 also fails, then this ontology can simply not be dealt with.

B

Software Architecture of the MoKi Validation Modules

This appendix documents the software architecture of the two MoKi validation modules ontology questionnaire and assertional effects functionality at an intermediate level of technicality. Some simplifications have occasionally been made where this was assumed to increase intelligibility.
The modules models checklist and quality indicator have been deemed simple enough to be sufficiently described by Chaps. 6.1.3 and 6.1.4.

B.1 Ontology Questionnaire

B.1.1 Functional View

The ontology questionnaire displays statements that are inferred from the MoKi OWL knowledge base but not explicitly stated within the latter. For each inferred statement, the ontology questionnaire delivers a list of explanations (each itself a list of logical statements) on request. In case statements from within the explanations are selected, the ontology questionnaire computes a list of inferences that would be lost if the selected axioms were to be removed. In case removal is confirmed, the ontology questionnaire deletes the selected statements both from MoKi's database and from the MoKi OWL knowledge base.
These actions provided by the ontology questionnaire, also given above in Chap. 6.1.1[1], are linked in the list below to user actions that trigger them:

- Load page: Display the list of axioms that can be inferred from the MoKi OWL knowledge base
- Get explanations for an inference: Display the explanation for a selected inferred statement in the list
- Select statements for deletion: Tentatively delete statements selected from explanations, display side-effects, and ask for confirmation.
- Confirm deletion: Actually delete the selected statements and display a success page.

[1] The action "Jump to a specific inferred statement in the list" is not given here, since this is implemented as a simple HTML anchor and additionally it is not a core feature.

– Cancel deletion: Cancel deletion of selected statements and return to the explanation from where the deletion process was started.

B.1.2 Logical View

Figure B.1 shows the classes involved in providing the ontology questionnaire's functionality within MoKi, their most important methods and members. The responsibilities of all classes are explained below.
Furthermore, the Java package `at.tugraz.kmi.verb` is used to render OWL axioms. This package and its members are not depicted and not described here.

MV_OntologyQuestionnaire

The special page "Special:MV_OntologyQuestionnaire" is the user interface entrypoint to the ontology questionnaire. The PHP class `MV_OntologyQuestionnaire` implements the predefined MediaWiki class `SpecialPage` and is responsible for dealing with all user interactions. Complex computations are delegated to the Java backend in the `at.tugraz.kmi.oq` package and, in order to remove selected statements from MoKi's database, to the PHP class `MV_OWL2DBMapper`.

MV_OWL2DBMapper

The PHP class `MV_OWL2DBMapper` contains functionality to remove selected OWL statements from MoKi's database. It does so by editing model element pages. It calls upon the Java class `MoKiAxiomCategoriser` to categorise the OWL statements that shall be deleted according to "subject".

MoKiOntologyQuestionnaire

The Java class `MoKiOntologyQuestionnaire` is the main entrypoint for the PHP class `MV_OntologyQuestionnaire` to access complex background functionality that delivers inferred statements given an OWL knowledge base, explanations for inferences and gives access to a dedicated instance of the Java class `OQAxiomRemover`.
It extends a more generally usable Java class `OntologyQuestionnaire` with functionality to keep a singleton `MoKiOntologyQuestionnaire` for a given ontology file in memory, to own a single instance of the Java class `OQAxiomRemover`, and organises OWL axioms according to IDs in order to pass them more easily as parameters of POST operations through the `MV_OntologyQuestionnaire`.

OQAxiomRemover

The Java class `OQAxiomRemover` marks statements (axioms) for removal, computes which inferences would be lost in case these were removed (side-effects), and removes them on request.

MoKiAxiomCategoriser

The Java class `MoKiAxiomCategoriser` owns a list of OWL axioms and returns them categorised according to "subject", i.e. the MoKi page in which such a statement would appear. For instance, the subject of a class assertion axiom $C(a)$ is the MoKi page corresponding to a, since membership in MoKi is defined on individuals' pages.

B.1.3 Process View

Figure B.2 shows the possible user actions within the ontology questionnaire and the calls between different (PHP and Java) classes that are triggered in response.
Calls to members of the `at.tugraz.kmi.verb` package, necessary to render OWL axioms within the ontology questionnaire in a more easily intelligible way, are not depicted and not described here.

Load Page

When a MoKi users loads the "Special:MV_OntologyQuestionnaire" page, the class `MV_OntologyQuestionnaire` retrieves the singleton instance of the class `MoKiOntologyQuestionnaire` responsible for the MoKi OWL knowledge base and clears it (`getMoKiOntologyQuestionnaire()`, `clear()`).
`MV_OntologyQuestionnaire` then asks the `MoKiOntologyQuestionnaire` singleton to deliver inferred statements, both those derived from the concept and property hierarchy and more complex inferences (`getInferredAxioms()`, `getHierarchicalAxioms()`). These statements are rendered and output to the MoKi user (see e.g. Fig. 6.2).

Get Explanations for Inference

When a MoKi user clicks on the "Why?" at the end of line of an inferred statement, (s)he gets one or more explanations for the inference. Technically, a query with the key `explain` and the value ij, where $i \in \{A, B\}$ denotes to which of the two lists of inferred statements the selected statement belongs and $j = 0 \ldots n$ indexes the statements in each list, is created.
Most operations triggered by this user action are the same as for loading the page. Additionally, `MV_OntologyQuestionnaire` asks for explanations for the selected inference from `MoKiOntologyQuestionnaire` and outputs also these (`getExplanationIds()`, see e.g. Fig. 6.3).

Select Statements for Deletion

When a MoKi user selects statements from the given explanations and presses the "Delete selected statements" button, (s)he arrives at a confirmation page that displays side-effects of the deletion and asks for a confirmation before actually deleting the selected statements. Technically, a variable `selected_axioms` which contains the IDs of the statements selected for deletion is posted.
`MV_OntologyQuestionnaire` first retrieves the statements selected for deletion via their IDs and asks `MoKiOntologyQuestionnaire` to give it a dedicated `OQAxiomRemover` (`getOQAxiomRemover()`). Then, it passes the statements selected for deletion to the `OQAxiomRemover` and asks for side-effects in case the deletion actually takes place (`getSideEffectsOfRemoval()`). Finally, `MV_OntologyQuestionnaire` renders the selected inference that shall be removed, the explicit statements selected for deletion, and the side-effects (see e.g. Fig. 6.4).

Confirm Deletion

When the MoKi user confirms the deletion of the selected statements by pressing again a "Delete selected statements" button, the actual deletion takes place and a success page is displayed. `MV_OntologyQuestionnaire` first creates an `MV_OWL2DBMapper` and asks it to delete the selected statements from MoKi's database (`construct()`, `deleteAxioms()`). Then, it asks the `OQAxiomRemover` to delete the selected statements from the MoKi OWL knowledge base (`removeSelectedAxioms()`). Finally, `MV_OntologyQuestionnaire` clears the `OQAxiomRemover` and outputs a success page to the user (`clearOQAxiomRemover()`, see e.g. Fig. 6.5).

Cancel Deletion

The MoKi user can also decide to cancel the deletion of the selected statemets by pressing the "Cancel button". In this case, `MV_OntologyQuestionnaire` clears the `OQAxiomRemover` (to free it for possibly another experimental deletion of statements) and loads `MV_OntologyQuestionnaire` such that the explanations for the selected inference are displayed again.

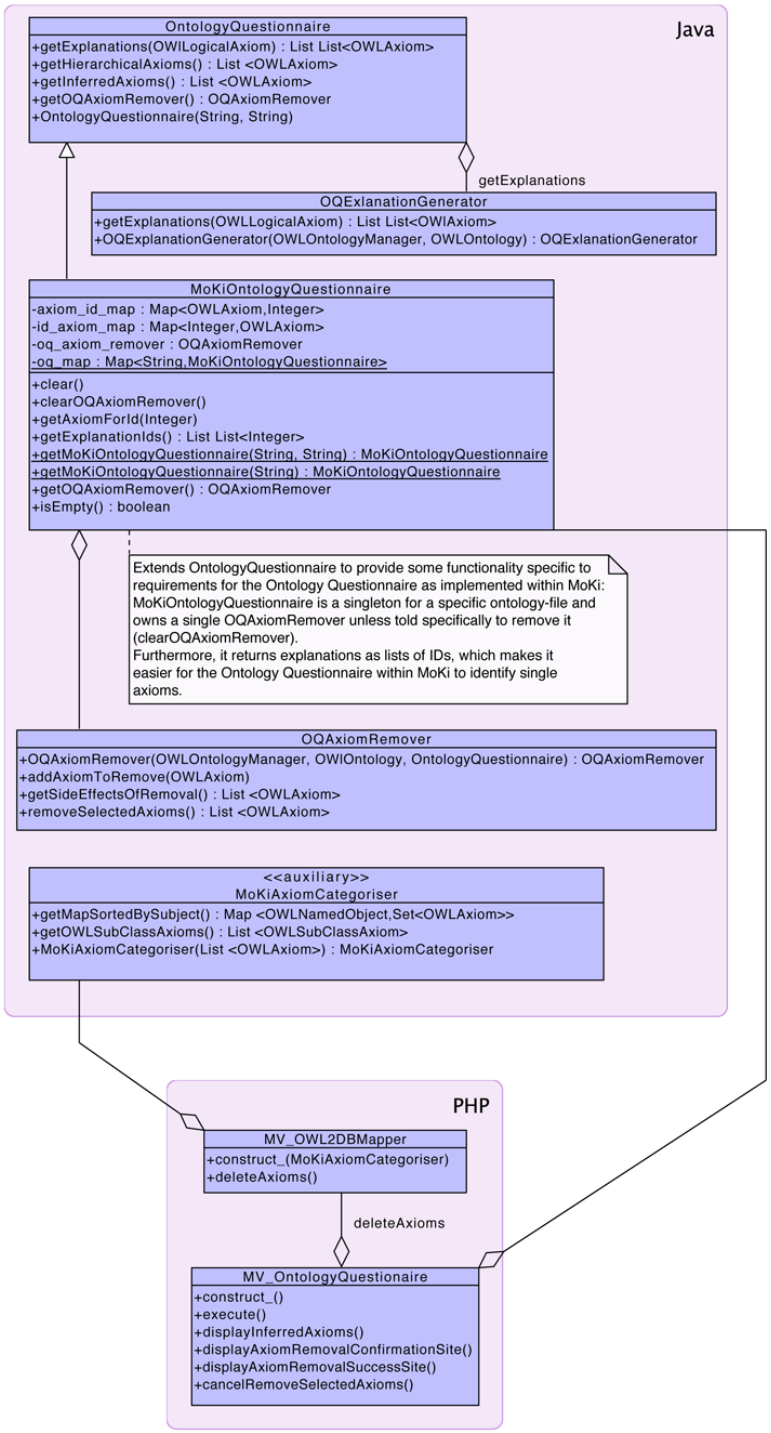


Fig. B.1. Class diagram of the ontology questionnaire within MoKi. Classes in the `at.tugraz.kmi.verb` package are not depicted.

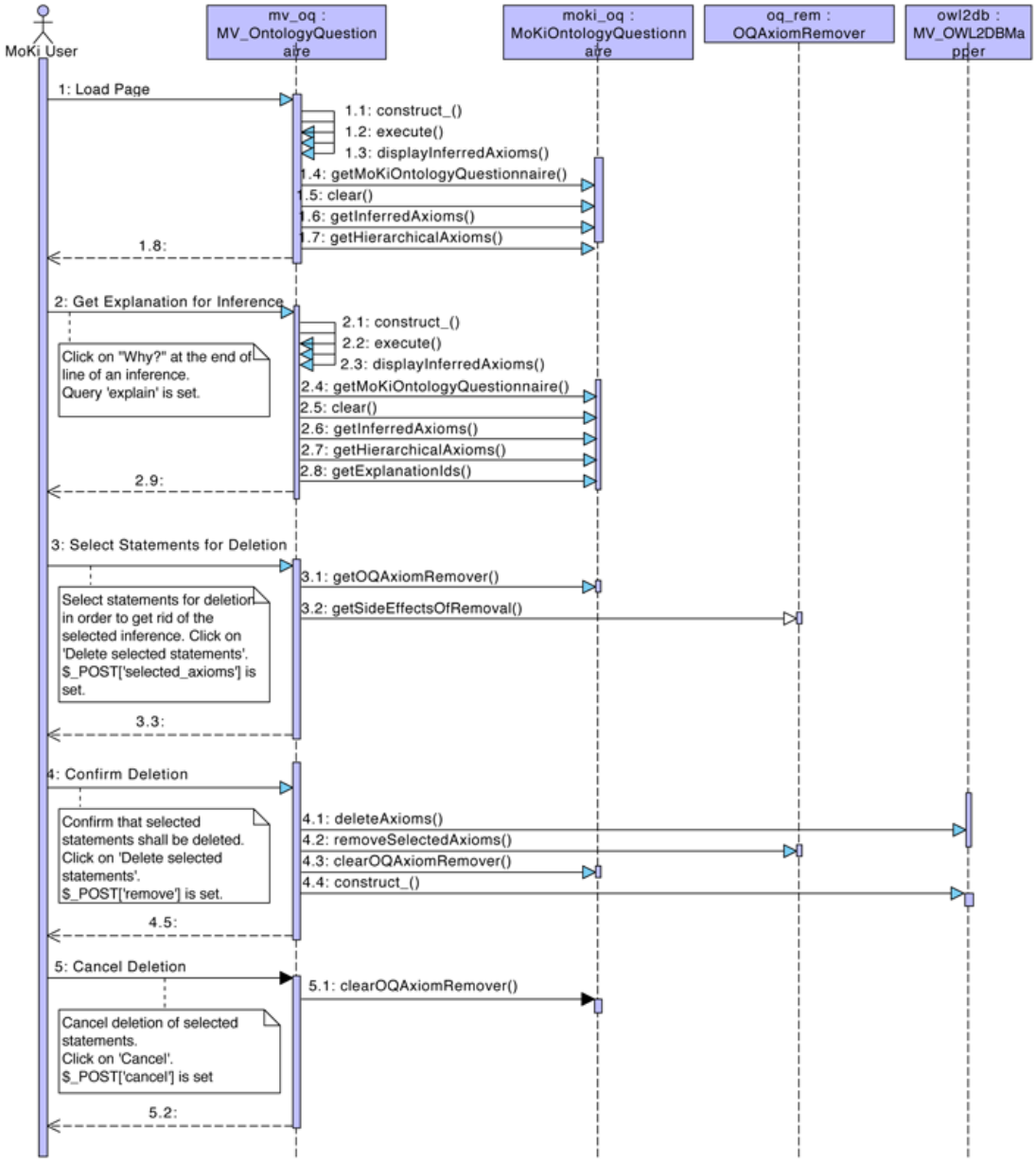


Fig. B.2. Sequence diagram of the ontology questionnaire within MoKi depicting the most relevant calls that are triggered by user interactions. Generally, calls to the members of the `at.tugraz.kmi.verb` package are not depicted.

B.2 Assertional Effects

B.2.1 Functional View

The assertional effects functionality detects and displays assertional effects caused by ontology editing activities such as adding, deleting or editing terminological or role axioms within MoKi. Assertional effects hereby are class or role assertions that have not been inferred from the MoKi's OWL knowledge base before manipulations but are inferred after manipulations (knowledge gained about data) or vice versa (knowledge lost about data). A formal definition of assertional effects as implemented in MoKi is given above in Def. 6.
Assertional effects are depicted on concept and property pages within MoKi, after either creating them or editing existing pages. After deletion or restoration of a whole wiki page, after editing the MoKi content in the visual IsA and IsPartOf browsers, after importing a domain model or batch deleting large parts of the domain model via the data cleaner functionality, no assertional effects are displayed although these may exist from a logical point of view.

B.2.2 Logical View

Figure B.3 shows the classes involved in providing the assertional effects functionality within MoKi, their most important methods and members. The responsibilities of all classes are explained below.
Furthermore, the Java package `at.tugraz.kmi.verb` is used to render OWL axioms. This package and its members are not depicted and not described here.

KnowledgeBasePair

The Java class `KnowledgeBasePair` holds pair of two knowledge bases A and B. It always considers A to be the starting point of the changes and B the result of changes, and when talking about additional or removed axioms, axioms that need to be added to or removed from A in order to arrive at B are meant. An instance of the class `KnowledgeBasePair` can be constructed with two knowledge bases or with a single knowledge base and explicitly a set of additional and a set of removed OWL axioms. `KnowledgeBasePair` delivers functionality and serves as a data container that links A and B and results of their comparisons. For each KnowledgeBasePair, every computation is done only once. Comparisons do not make use of reasoning but compare sets of axioms.
The `KnowledgeBasePair` is used by the assertional effects functionality to detect logical changes in the MoKi OWL knowledge base, as well as by the incremental OWL update functionality to detect any kind of change (also non-logical changes such as for instance changes of verbal descriptions) in a single page and to update the MoKi OWL knowledge base accordingly.

TeaTypes

The Java class `TeaTypes` is responsible for computing assertional effects given an instance of the class `KnowledgeBasePair`. It computes assertional effects separately, once for additional axioms and once for removed axioms. It provides a number of methods for retrieving various kinds of assertional effects, in ascending order of computational complexity: First, it is conceivable that the `KnowledgeBasePair` instance with which `TeaTypes` was initialised holds knowledge bases in which assertions have been explicitly added (`getExplicitAssertionalEffects()`). Second, assertional effects involving primitive concepts and roles can be retrieved (`getAtomicAssertionalEffects`) as well as assertional effects involving negated primitve concepts (`getNegativeAtomicAssertionalEffects()`). Finally, assertional effects can be detected through the decision procedure described in Chap. 5.2.1 (`getComplexAssertionalEffects()`), although `TeaTypes` limits the search depth and thus only performs an approximation.
An aggregation method (`getCollectedAssertionalEffects()`) computes all above-mentioned kinds of effects, does not return explicitly added or removed concept and role assertions, and tries to simplify complex assertional effects.

AssertionalEffectsContainer

The Java class `AssertionalEffectsContainer` holds the result of a computation of `TeaTypes`. It holds the `KnowledgeBasePair` for which assertional effects were computed, it knows which kind of assertional effects were computed (explicit, atomic, negative atomic, complex, collected), and it holds gained or lost assertional axioms.

SortAssertionalEffectsContainer

The Java class `SortAssertionalEffectsContainer` provides sort functionality for instances of the class `AssertionalEffectsContainer`. The current implementation supports sorting by individual name (the first name to occur for role assertions) and by effect size (the size of the involved concept or role).

B.2.3 Process View

Figure B.4 shows the actions in the background after a MoKi user confirmed editing a concept or property page, or created a new concept or property page (the actions are the same in both cases).
Sorting, done by `SortAssertionalEffectsContainer`, and calls to members of the `at.tugraz.kmi.verb` package, necessary to render OWL axioms within the asertional effects functionality in a more easily intelligible way, are not depicted and not described below.

Save OWL Representation Before Edit

The `ArticleSave` hook is triggered after a MoKi user has confirmed the edit of an existing or newly created page, but before changes are persisted in the MoKi database. Thus, at this stage the incremental OWL update functionality exports the OWL representation of the edited page as it was before the edit.

Save OWL Representation After Edit

The `ArticleSaveComplete` hook is triggered after changes in an edited page are persisted in the MoKi database. Thus, at this stage the incremental OWL update functionality exports the OWL representation of the edited page as it is after the edit.

Detect Logical Changes

An instance of the Java class `KnowledgeBasePair` is constructed with the edited page before and after edit as input. It detects logical changes. If none are made, the edited wiki page is output as it is, with no further additions.

Prepare KnowledgeBasePair with MoKi *OWL Knowledge Base and the Detected Additional and Removed Axioms*

A new instance of the Java class `KnowledgeBasePair` is constructed with the MoKi OWL knowledge base and the additional and removed axioms detected in the previous step as input.

Prepare TeaTypes

The latter `KnowledgeBasePair` instance is used to construct an instance of the class `TeaTypes`.

Compute Assertional Effects

`TeaTypes` is used to compute all available kinds of assertional effects (atomic, negative atomic, complex), excluding explicitly added or removed assertions and including assertional effects with simplified complex concepts if possible (`getCollectedAssertionalEffects()`).

Add Logical Changes and Assertional Effects to Output Page

If logical changes were detected, these are added to the output page. If also assertional effects were detected, these are output, too (see e.g. Fig. 6.6 and 6.7).

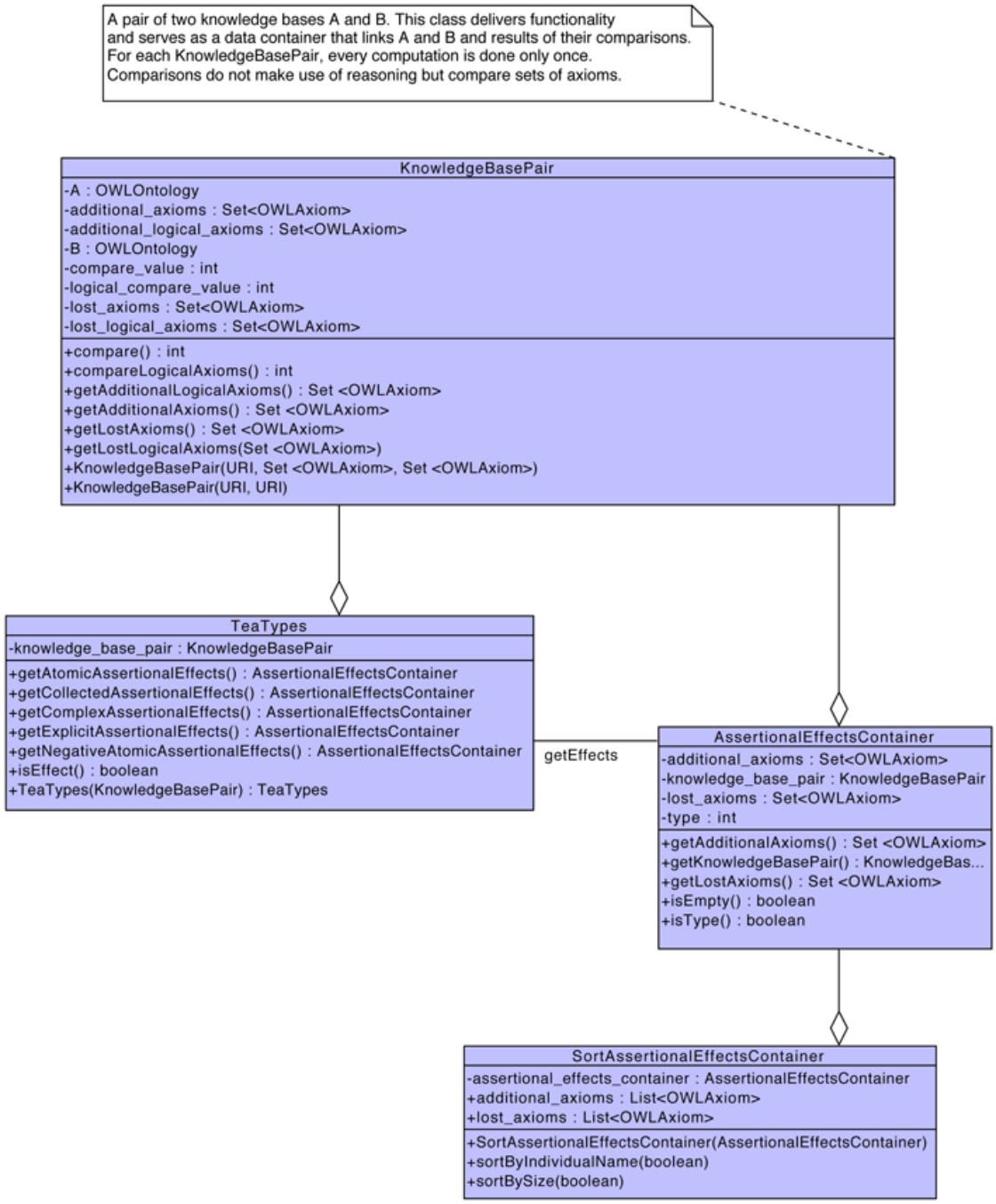


Fig. B.3. Class diagram of the assertional effects functionality within MoKi. Classes in the `at.tugraz.kmi.verb` package are not depicted.

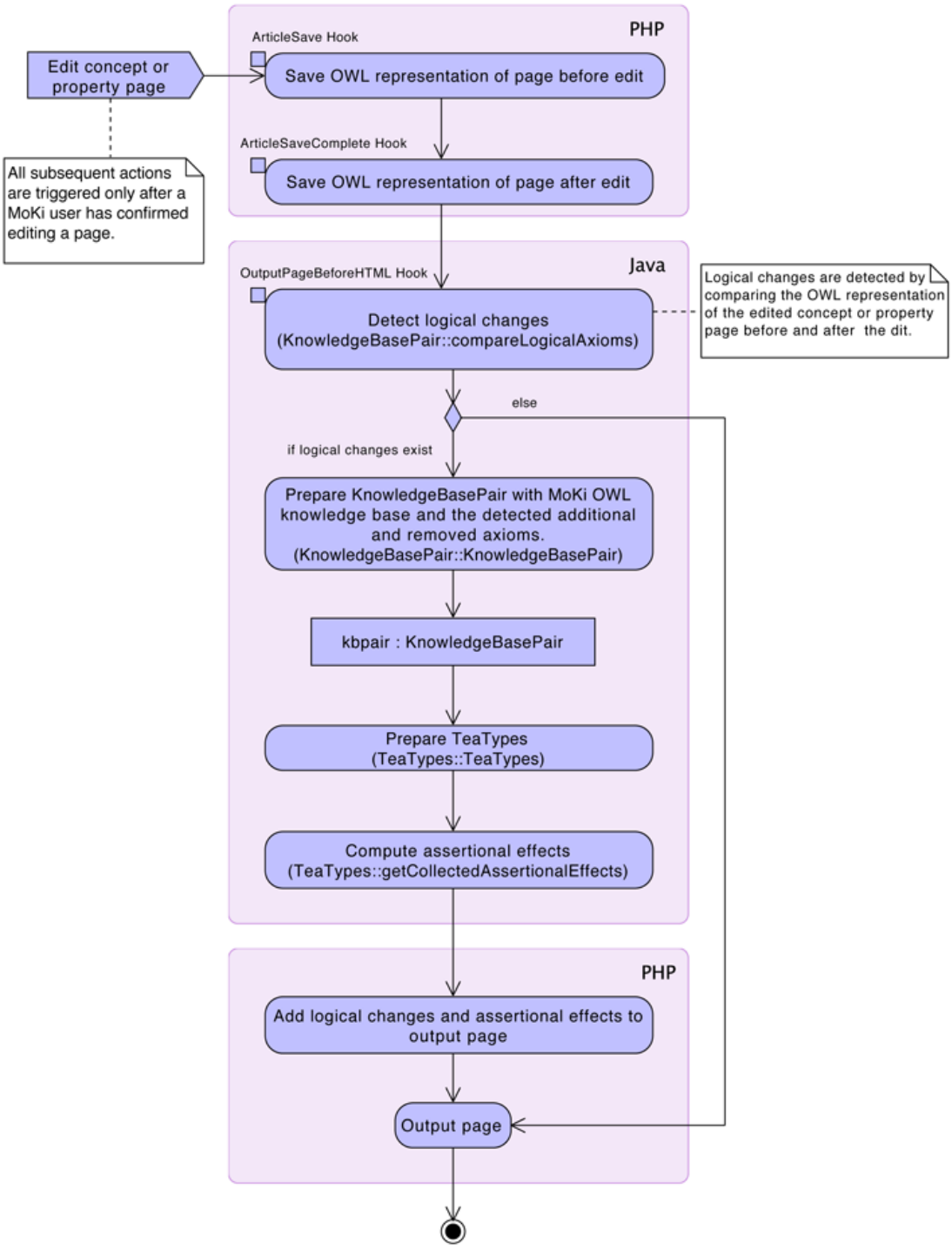


Fig. B.4. Activity diagram of the assertional effects functionality within MoKi. Note that the actions are the same in both cases, whether a concept or property page has been newly created or edited. Sorting and calls to the members of the `at.tugraz.kmi.verb` package are not depicted.