Ontology-Based Services: Usage Scenarios and Test Ontologies

Deliverable TONES-D14

all

[1] Free University of Bozen-Bolzano, [2] Università di Roma "La Sapienza",
[3] The University of Manchester, [4] Technische Universität Dresden,
[5] Technische Universität Hamburg-Harburg

Project:	FP6-7603 – Thinking ONtolgiES (TONES)
Workpackage:	WP7– Experimentation And Testing of the Common Logical Framework
Lead Participant:	Hamburg University of Technology
Reviewer:	Bernardo Cuenca Grau
Document Type:	Deliverable
Classification:	Public
Distribution:	TONES Consortium
Status:	Request for Contributions
Document file:	D14.pdf
Version:	0.0
Date:	13.04.2007
Number of pages:	40

Abstract

In this deliverable we define a number of usage scenarios for tasks that have to be solved during the whole life-cycle of ontologies. The detailed description of these tasks relevant for, e.g., ontology design, maintenance, access, usage, interoperation were already provided in previous deliverables. Now, we investigate the combination of various elementary tasks for the above-mentioned processes, and we analyse the interplay of corresponding reasoning services. We integrate and apply task-specific reasoning techniques to a set of test ontologies. The selection of test ontologies and their specific purposes is presented in this report.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	2007	First draft
v.1.1	2007	Final version

Contents

1 Introduction

In previous deliverables of the TONES project, we defined several reasoning-based methods and techniques for performing tasks relevant for ontology life-cyle stages such as design, maintenance, access, and interoperation. Techniques build a set of inference services defined w.r.t. the logical framework introduced in TONES. For showing the significance of the proposed framework, in this report we investigate the application of techniques as part of larger usage scenarios. Usage scenarios are seen as exemplified combinations of techniques in order to realize tasks identified in previous deliverables. A number of test ontologies is selected for investigating the applicability of reasoning services for realizing ontology-oriented tasks. We first describe the main ideas of offline and online usage scenarios in general, and then present a detailed view on each usage scenario based on the application of methods and techniques investigated in TONES.

For the sake of completeness, we start with an overview on reasoning-based techniques identified for tasks developed in previous work packages (no priority is induced by the order of appearance).

1. Computing Subsumption Hierarchies

   In order to understand the reasoning technique of computing subsumption hierarchy, we need to recap the notion of subsumption of concepts. Concept subsumption is one of the classical inference problems in description logic; the problem is to decide whether, given two concept terms C and D, C is subsumed by D relative to the ontology $\mathcal{O}$, written $C \sqsubseteq_{\mathcal{O}} D$. Semantically, this means that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{O}$.

   Computing the subsumption hierarchy of an ontology $\mathcal{O}$, also known as classifying the ontology $\mathcal{O}$, is a reasoning technique that is used to re-arrange all the concept names in $\mathcal{O}$ according to the subsumption relationships. To be more precise, it identifies the ordering $\sqsubseteq_{\mathcal{O}}$ restricted to concept names that occur in the ontology. If the ontology $\mathcal{O}$ has n concept names, then with a naive approach $\mathcal{O}$ can be classified by carrying out n^2 subsumption tests, i.e. one for each pair of concept names. However, one usually tries to avoid as many subsumption tests as possible, as such tests are very expensive. In the literature, there have been approaches to optimize classification by reusing subsumption results previously computed and by using explicitly told information in the ontology.

2. Consistency Checking

   The consistency problem is defined as follows. A concept C is called consistent w.r.t. a TBox $\mathcal{T}$ if there exists a model of C that is also a model of $\mathcal{T}$. An ABox $\mathcal{A}$ is consistent w.r.t. a TBox $\mathcal{T}$ if $\mathcal{A}$ has model $\mathcal{I}$ which is also a model of $\mathcal{T}$. A knowledge base ($\mathcal{T}$, $\mathcal{A}$) is called consistent if there exists a model for $\mathcal{A}$ which is also a model for $\mathcal{T}$.

   For the definitions above, corresponding decision problems are defined as usual. In order to solve these problems, practical description logic systems implement algorithms as so-called inference services. Besides a basic reasoning service to compute the subsumption relationship between every pair of concept names mentioned in a TBox as discussed above, another important inference service for practical knowledge representation is to check whether a certain concept name occurring in a TBox is inconsistent. Usually, inconsistent concept names are the consequence of modeling errors. Checking the consistency of all concept names mentioned in a TBox without computing the parents and children is called a TBox coherence check.

3. Ontology Extraction

The ontology extraction task consists in an automatic procedure to extract a conceptual view from a relational database. This provides a bootstrap ontology which faithfully describes the relational source to which the procedure has been applied. Subsequently, the generated ontology can be enriched and maintained by means of techniques and tools described in previous TONES deliverables.

The procedure for extracting a conceptual view from a relational database is based on heuristics which analyze the actual constraints in the database. The wrapping of relational data sources is provided by means of an extracted ontology where each term (classes and relations) is associated with a view over the original data. To represent the extracted ontology, instead of a graphical notation as often used in the database community, we employ an ontology language thus providing a precise semantics to the extracted schema. Our extraction procedure relies on information from the database schema, (i.e., key and foreign key structure, restrictions on attributes and dependencies between relations), and automatically extracts all the relevant semantics if an input relational schema was designed using a well-founded methodology. The resulting ontology can be used to access the data source by means of query rewriting using the defined views.

4. Module Extraction

   Suppose that the user wants to extract a fragment $\mathcal{Q}_1$ of an ontology $\mathcal{Q}$ that satisfies some given criteria w.r.t. an input signature $\mathbf{S}$. For example, one may want $\mathcal{Q}_1$ to entail all the consequences that hold in $\mathcal{Q}$ w.r.t. the input signature. In practice, we have exploited the notion of locality for extracting modules. Given the signature $\mathbf{S}$ to be reused and the input ontology $\mathcal{Q}$, our algorithms check if each of the axioms in $\mathcal{Q}$ satisfies our locality condition w.r.t. the input signature $\mathbf{S}$. In case an axiom $\alpha \in \mathcal{P}$ does not satisfy these conditions, the algorithm includes α in the module and expands the signature $\mathbf{S}$ to $\mathbf{S} \cup \mathsf{Sig}(\alpha)$. The remaining axioms are tested for locality w.r.t. the expanded signature. The process is repeated until a fixpoint is reached. This technique is described in detail in [22].

5. Error Repair

   Suppose that an ontology contains a set of unsatisfiable concept names. Current tools do not typically provide support for the user to repair these errors. In practice one can use a set of heuristics for guiding the repair process. In particular, it is useful to filter out and subsequently rank the axioms in the ontology according to a set of criteria. The filtering out is done according to whether the axiom under consideration participates in an inconsistency; this can be determined, for example, using axiom pinpointing. The ranking is done according to the impact of removing the axiom from the ontology; for example, it is reasonable to assume that removing the axiom invalidates as few subsumption relationships as possible.

6. Computation of Commonalities

   In addition to standard inference services and retrieval inference services discussed above, recently another set of inference problems has been defined, decision problems have been shown to be decidable, and practical inference algorithms as well as system implementations have been developed (cf., [46]). Depending on the ontology language, a solution for the problems mentioned below need not necessarily exist. Furthermore, algorithms for the problems are known only for non-expressive description logics in most cases.

   Least-common subsumers (LCS) [5, 6, 8] represents the commonalities between a set of concepts. For a given set of concepts C_1,...,C_n their least-common subsumer E is defined

as follows:

(a) $C_i \sqsubseteq E$ for all i = 1,...,n and

(b) If E' is a concept satisfying $C_i \sqsubseteq E'$ for all i = 1,...,n then $E \sqsubseteq E'$.

Most-specific concept (MSC) [11]: Given a finite set of assertions in *ABox* $\mathcal{A}$ of the form $C(a)$ or $r(a, b)$, where C is a concept and r is a role name, the concept E represents the most-specific concept of the individual a in *ABox* $\mathcal{A}$ if it satisfies:

(a) $\mathcal{A} \models E(a)$, and

(b) If E' is a concept satisfying $\mathcal{A} \models E'(a)$ then $E \sqsubseteq E'$.

7. Matching

   In its simplest form, a concept matching problem consists of a concept description C and a *concept pattern* D, i.e., a concept description with *concept variables*. Matching D against C means finding a substitution of variables in D by concept descriptions such that C is equivalent to the instantiated concept pattern D.

   It has been first introduced in [39] for the application of generating explanations. Intuitively, if a concept can be matched against a pattern P, then their syntax trees share the "upper part", i.e., where P is fully specified, while deviations may occur at leaves labeled with variables. Hence, the set of all concepts that can be matched against P contains infinitely many concepts structurally similar to P to some extent. In this sense, matching P against several concepts and returning those which can be matched, can be seen as a search with the fully specified part of P as search criterion.

8. Axiom Pinpointing

   Axiom pinpointing is, in its simplest meaning, finding out which axioms are responsible for a problem that has occurred in an ontology. Suppose the user has noticed that there is a problem in his ontology, and he wants to find out the source of this problem. In this setting, axiom pinpointing can be useful in finding the source of the problem, explaining it in an understandable way, and making suggestions in order to solve the problem. It find usage in the task of error management, which was mentioned in D13 [1].

9. Approximation

   Intuitively, the technique for computing concept approximation provides means for "translation" from an expressive DL to a typically less expressive DL. The technique to compute an actual approximation from a concept description is based on structural algorithms first described in [14, 13]. These works devise computation algorithms for obtaining $\mathcal{ALEN}$-concept descriptions from $\mathcal{ALCN}$-concept descriptions.

   Approximation is useful in any setting where concept description from an expressive DL have to be "simplified" to a user DL with only as little information loss as possible. Furthermore, concept approximation can help to provide inference services that have only computation methods for rather inexpressive DLs for concept descriptions in more expressive DLs. The method here is to first approximate the concept descriptions in a DL for which the inference service is available and then apply the inference.

10. Model Generation and Model Checking

The *model generation problem* is a problem of finding interpretations $\mathcal{I}$ which satisfy all axioms of $\mathcal{T}$ and $\mathcal{A}$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is a ABox of an ontology O. In case of a *model checking problem* the goal is to prove whether a given interpretation $\mathcal{I}$ is a model of O. Current tableau algorithms are not well applicable as model generation procedures since they only return (a description of) a so-called single canonical model. Instead, model finders are able to enumerate all models systematically. This can indeed be useful for ontology design tasks. In order to support the ontology development process in an incremental way, well-known model-generation tools can be adopted accordingly and provide major benefits for human ontology designers.

The ontology designer is often not interested in just testing the satisfiability of an ontology by checking whether one single model exists, but possibly wants to inspect a number of generated models instead. This way, unintended models might be identified. Using model finding service, the ontology engineer can adjust the ontology by examining automatically generated relational model structures.

11. Abduction

    Abduction, or finding the "best" explanation, is a method of reasoning employed in areas in which one chooses which hypothesis would best explain the relevant evidence. In other words, abduction is a reasoning process that starts from a set of assertions α and derives their most likely explanations in terms of another set of assertions such that α is entailed w.r.t. the background knowledge.

    Formally described, the *abduction* inference service aims to construct a set of (minimal) explanations Δ for a given set of assertions Γ such that Δ is consistent w.r.t. to the ontology $(\mathcal{T}, \mathcal{A})$ and satisfies following conditions [23]:

    (a) $\mathcal{T} \cup \mathcal{A} \cup \Delta \models \Gamma$ and

    (b) If Δ' is an ABox satisfying $\mathcal{T} \cup \mathcal{A} \cup \Delta' \models \Gamma$, then $\mathcal{T} \cup \mathcal{A} \cup \Delta' \models \Delta$ (Minimality)

    (c) $\Delta \not\models \Gamma$ (Relevance)

12. Ontology Integration and Conservative Extension

    The notion of a conservative extension has revealed as central for formalizing ontology integration tasks. Intuitively, an ontology $\mathcal{P} \cup \mathcal{Q}$ is a conservative extension of $\mathcal{Q}$ for a signature $\mathbf{S}$ if and only if every logical consequence α of $\mathcal{P} \cup \mathcal{Q}$ constructed using only symbols from $\mathbf{S}$ is already a consequence of $\mathcal{Q}$. In practice, deciding conservative extensions is hard or even undecidable for reasonably expressive description logics. No practical algorithms are currently known. In practice, one may look for sufficient conditions for conservative extensions—that is, if $\mathcal{P}, \mathcal{Q}$ satisfy our conditions then we can guarantee that $\mathcal{P} \cup \mathcal{Q}$ is a conservative extension of $\mathcal{Q}$; the converse, however, does not necessarily hold. These techniques can be implemented efficiently. In particular, in TONES we have introduced and used the notion of *locality* of an ontology.

13. Query Answering w.r.t. ABoxes

    Query answering w.r.t. an ontology is in general a deductive process of finding domain objects that satisfy the query in all possible worlds constrained by the ontology. We consider ontology-based information access in the scenarios where data is already available in an ABox and an ABox as source of information is located. There are a number of ABox query languages with different semantics and expressivities. In the literature (e.g.

[33, 25, 47]), two different semantics for these kinds of queries are discussed. In *standard* conjunctive queries, variables in the head and in query atoms in the body are bound to (possibly anonymous) domain objects. In so-called *grounded* conjunctive queries, variables are bound to named domain objects (object constants). In grounded conjunctive queries the standard semantics can only be obtained for so-called tree-shaped queries by using existential restrictions in query atoms.

DL-Lite is a family of DLs specifically tailored to deal with large amounts of data and reasoning with queries. While the expressive power of the DLs in the *DL-Lite* family is carefully selected to achieve a low complexity of reasoning problems, such DLs are expressive enough to capture many notions of both ontologies, and conceptual modeling formalisms used in databases and software engineering (i.e., ER and UML class diagrams). We consider query answering for the class of unions of conjunctive queries (UCQs), which is the most expressive class of queries for which decidability of query answering has been proved in DLs [19, 41]. Notably, standard query answering of UCQs over a *DL-Lite* knowledge base $\mathcal{K}$ can be solved by means of evaluation of suitable first-order logic queries over the ABox of $\mathcal{K}$ considered as a flat relational database [17, 18]. This allows for using well established Relational Data Base Management System (RDBMS) technology for query answering in *DL-Lite*.

14. TBox Query Answering

    In practical DL systems retrieval of the "structural" information from the ontology is an essential part of the query language, e.g., retrieval of the concept names mentioned in a knowledge base, retrieval of the set of roles, retrieval of the concept parents and children (defined analogously to the role parents and children) and so on. In TBox queries, variables are bound to objects of the taxonomy (concepts, roles). In this sense, the taxonomy of the TBox is seen as a virtual ABox, which can be used in queries. Query answering w.r.t. taxonomy ABoxes has been supported for several years now in mature description logic inference systems. For instance, one can retrieve individuals that are instances of a certain concept C but can be proven to be non-instances of subconcepts of C that are subsumed by another concept D, say.

15. Access to External Data Sources

    Accessing external data sources is the problem of accessing databases that are independent from the ontology, and that are related to the ontology through suitable mappings. As external sources we consider relational databases. As ontologies, we consider, for instance, *DL-Lite* knowledge bases. Since relational databases store only values (not objects), objects that are instances of the concepts in the knowledge base need to be constructed from such values. In other words, in order to specify an effective mapping between external data sources and the ontology, we have to deal with the so-called "impedance mismatch" problem.

16. Ontology Interoperation

    Ontology interoperation concerns multiple independent ontologies that are operating with each other. We consider ontologies expressed in, for instance, *DL-Lite*, and analyze different forms of interoperation: (i) Ontology-to-data storage, where ontologies are used to design and access a data storage (the ontology and the underlying database are tightly related, but the impedance mismatch problem mentioned above is of concern); (ii) Ontology-based data integration, i.e., access to external sources, already discussed above; (iii)

Ontology-to-ontology, where information flows through a mapping from a source ontology to a target one; (iv) Network of ontologies, i.e., peer-to-peer ontology-based interoperation, where no constraints on the topology of the network and on the information flow are posed.

17. ABox Updates

    Ontologies can be changed during the development phase as well as when they are already in use. At design time, these changes mostly concern the intensional part, the TBox, whereas usage-time changes usually affect the extensional part, the ABox. Recent investigations on DL-based ontology updates pursue two different directions. While some approaches study so-called syntactic updates and incremental reasoning ([31, 30, 28]), others make an attempt to define a formal update semantics based on model theory rather than information told at the syntactic level ([24, 38, 43, 42, 29]).

18. Query Formulation Support

    The key idea underlying the support for query formulation is to access data sources by means of an ontology describing the sources in a formal way. The purpose of the system is to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data.

    To this end the system relies on automated reasoning techniques in order to exploit the implicit information conveyed by the ontology. In fact, the intelligence of the interface in proposing the user-relevant refinement of the query being composed is driven by an ontology describing the domain of the data in the information system.

    The system uses a verbalization technique to present the conjunctive query as a natural language expression closer to the user understanding. This is performed in an automatic way by using meta-information associated with the ontology terms. The verbalization of single ontology terms must be provided in advance by the ontology engineers.

    The final purpose of the tool is to generate a conjunctive query ready to be executed by some evaluation engine associated to the information system.

19. Query Subsumption Checking

    Query subsumption checking is very significant for ontology design and maintenance, and for online optimizations. Specifically, query subsumption checking is the problem of determining whether a query q_1 expressed over an ontology is contained in a query q_2 expressed over the same ontology, i.e., establishing whether in all possible models of the ontology, the result set of q_1 is a subset of the result set of q_2.

2 General Usage Scenarios

We assume that in the whole life-cycle of an ontology or versions of an ontology, there are different phases. Distinguishing between offline and online phases, i.e., phases in which the ontology is designed or maintained and phases in which the ontology is used for problem solving tasks, we anticipate corresponding usage scenarios for the techniques described in the previous section. We first start with an integrated view on specific scenarios (offline and online) for giving an overview about the interplay of various techniques for realizing various tasks or subtasks. Both, offline and online usage scenarios introduced here are presented in more detail in Section 3. It should be noted that for some tasks, such as e.g. ontology extraction, the associated techniques mentioned in the literature might have the same name. It should be clear from context whether we refer to the tasks or the techniques, however.

2.1 Offline Usage Scenarios

Offline scenarios are defined w.r.t. the viewpoint of "ontologies under development in the construction workshop". In general, a well-structured ontology can only be achieved through a principled and systematic design process. To guarantee a high-quality structure of an ontology throughout the whole life-cycle, it is necessary to carry out design and maintenance tasks in a structured and systematic way (see Figure 1). We focus on ontology design in this section but emphasize that we view maintenance as life-long design of ontologies.

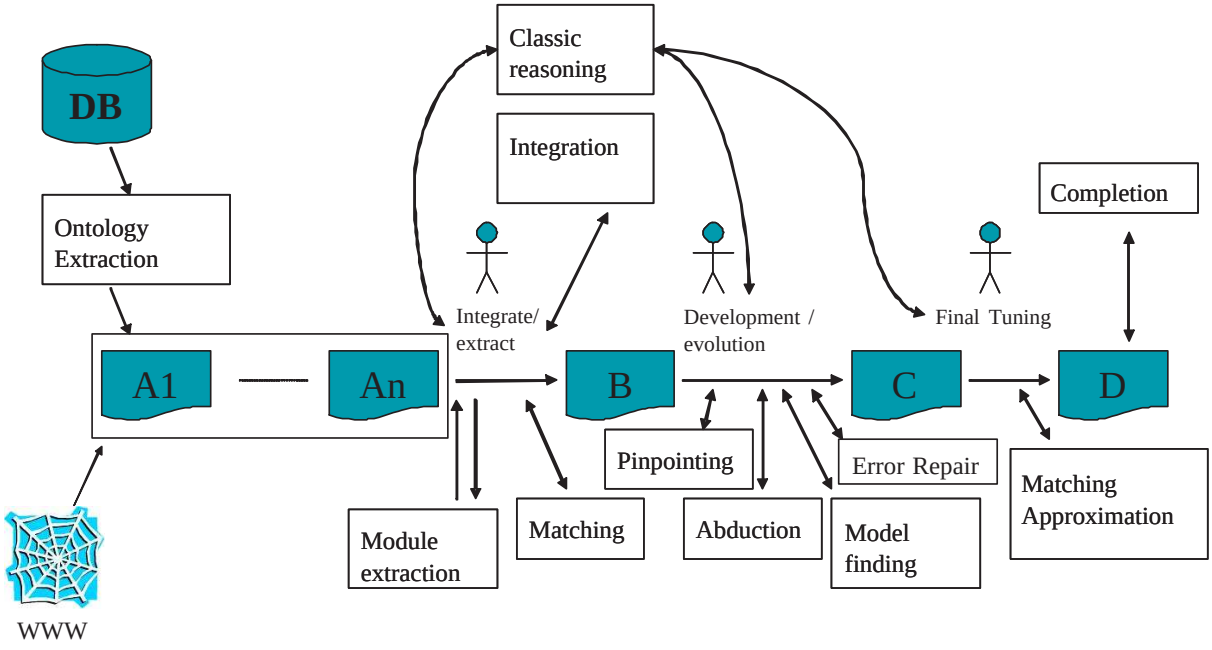

Figure 1: General usage scenario for ontology design

Ontology design can be performed using different sources:

1. Existing ontologies
2. Existing database schemas and the database instances
3. Additional sources from the Web

4. Building ontologies from scratch

For using existing database schemas we define usage scenarios for *ontology extraction*. The idea is to obtain an ontology (TBox) as well as a set of DB-mappings describing transformations from terms of the ontology to the given database schema by applying some transformation algorithms and vice versa.

The first step of the design process yields a set of ontology versions A_1 to A_n. These ontologies may also contain ABox information (e.g., generated from the data in DB). The next step is to obtain an ontology B from A_n by applying techniques such as, for instance, *module extraction* and *matching*. Both scenarios might correlate with the *integration* usage scenario and might also require some user interaction.

Ontology B is the starting point for the actual development/evolution phase. Hereby we do apply several reasoning techniques and usage scenarios (e.g., *axiom pinpointing*, *error repair*, *abduction* and *model finding*) to obtain an ontology C that is quite close to the expected requirements. The result of the design process is ontology D. It is derived from C by doing some fine-tuning that mostly involves the *ontology completion* usage scenario. The user also will be supported in that part of ontology design by the non-standard inference services like *matching* and *approximation*.

2.2 Online Usage Scenarios

Online usage scenarios are oriented towards the view of "ontologies at work". For online usage scenarios, runtimes for computational processing behind the techniques are even more important than for offline scenarios.

TBox and ABox Access For the tasks in ontology-based access, processing and usage we assume to have two distinct kinds of input. We do either have a complete ontology (TBox and ABox) or a TBox together with a database covering the intensional part of the ontology (see Figure 2).

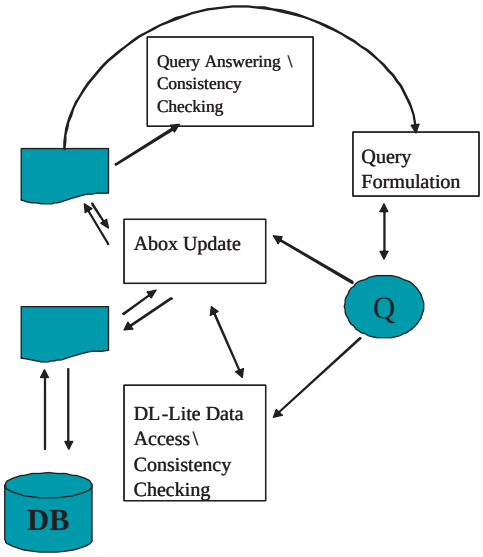

Figure 2: General Usage Scenario for Ontology Access

When we have an ontology consisting of TBox and ABox, *ABox query answering* and *consistency checking* can be applied. For defining queries, the usage scenario *query formulation* will be of importance. For both cases we can define the usage scenario of *updating ABoxes*. Furthermore, we can find additional usage scenarios depending on the ontology representation formalism. For some reasons, users may be also interested in obtaining information about the taxonomical structure of the ontology. In these cases, the usage scenario of *TBox query answering* is applicable. Combined TBox and ABox queries are also possible.

In case we have a TBox together with a database for the extensional part of the ontology, ontology-based data-access in the context of relational databases can be performed (for instance, with ontologies based on DL-Lite. In this usage scenario standard inference services like *consistency checking* can be also involved.

Ontology Interoperation Ontology interoperation concerns multiple independent ontologies that are in operation with each other. Ontology interoperation tasks do not require changes in the single ontologies, but mappings have to be specified among the knowledge contained in each of them. On the basis of these mappings each ontology can take advantage of the knowledge in the others to give value-added services to its clients. The study on ontology interoperation is recently started within the Work Package 6 of the TONES project. Among various interoperation tasks the focus has been initially posed on forms of extensional query answering, intensional query answering, updating of extensional knowledge, in the four detailed usage scenarios mentioned in the next sections: *ontology-to-data-storage*, (ontology-based data access), *ontology-to-data-sources* (ontology-based data integration), *ontology-to-ontology* (unidirectional flow of information), and *network of ontologies* (peer-to-peer). More details on tasks and techniques for ontology interoperation will be provided in the next deliverables D16 and D24.

3 Detailed Usage Scenarios

In the following sections, we describe particular usage scenarios that refine certain parts of general usage scenarios presented in Section 2. For defining a single usage scenario, we give a general description of the usage scenario as well as mention how it relates to ontology-based tasks identified in previous project deliverables. Moreover, for some of the scenarios we specify which kind of input (e.g., well established ontology, database, query etc.) and which kind of output (e.g., new ontology, query and so on) we expect for this usage scenario. If applicable, we discuss also what are success indicators for testing the scenarios (e.g., performance data like run time and memory, size of the answer, correctness of results, usability, feasibility of query answering, efficiency etc.) We also mention candidate ontologies for testing the usage scenario. An overview of test ontologies we are using can be found at the end of this deliverable.

3.1 Usage Scenarios for Ontology Design and Maintenance

3.1.1 TBox and ABox Extraction from a Database

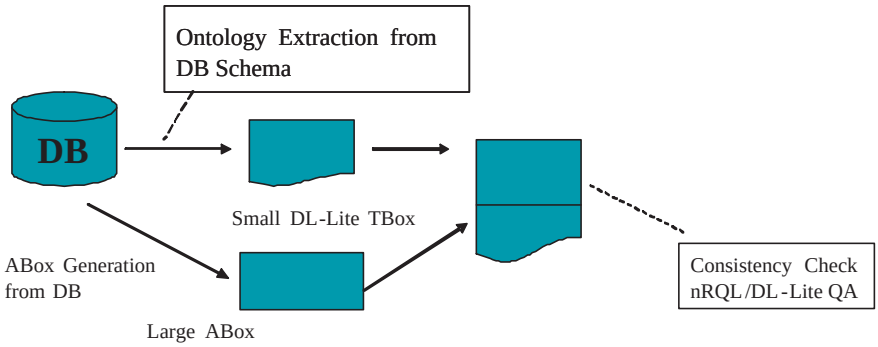

Figure 3: Usage Scenario for Ontology Extraction from DB

This usage scenario can be summarized as follows.

1. We start from a relational database (with integrity constraints) and apply techniques studied by FUB to extract a DL-Lite TBox (whose size comparable to the size of the database schema, which typically is small). The corresponding ABox is extracted from the data in the DB.

2. The knowledge base consisting of the ABox and TBox is refined by applying standard and non-standard reasoning algorithms for ontology design and evolution (see general scenario depicted in Figure 1).

3. On the resulting knowledge base query answering is performed.

The input for the usage scenario is a relational database. Both schema and data are considered. The schema might, possibly be defined with integrity constraints. The output of the process described above is a DL-Lite TBox, whose size is presumably not huge, if compared with the size of the database instance, and an ABox, representing the data stored in the original database. All reasoning services described in the general scenario of Figure 1 can be applied on the output ontology. In particular, query management techniques will be applied in this scenario. Such techniques are very significant for ontology design, since we can make use of queries to construct

views that can be exploited for the task of view-based ontology design and maintenance. More precisely, query satisfiability, disjointness, and subsumption checking will be considered.

3.1.2 TBox and DB-Mapping Extraction from a Database

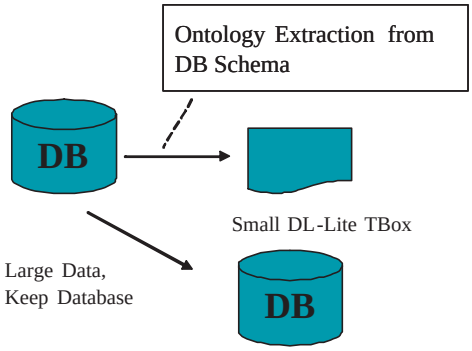

Figure 4: Usage Scenario for TBox and DB-Mapping Extraction

1. We start from a relational database (with integrity constraints) and apply techniques studied by FUB to extract a *DL-Lite* TBox. Based on the resulting TBox, mappings to the original database are synthesized.
2. The TBox is refined by applying standard and non-standard reasoning algorithms for ontology design and evolution (see general scenario). The mappings are adapted to take into account TBox evolution.
3. The extracted TBox and mappings may be used to check quality of data stored in the database by checking the satisfaction of properties expressed over the TBox. The properties to be checked are those that represent relevant domain knowledge.
4. On the resulting TBox that is mapped to the database (situated ontology), query answering is performed.

3.1.3 Information Extraction via Abduction

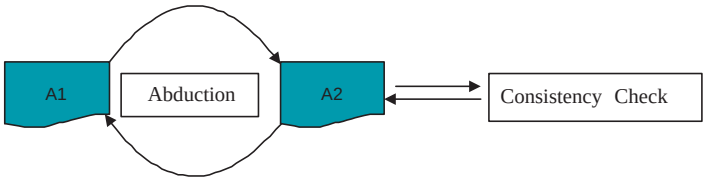

Figure 5: Usage Scenario for Information Extraction via Abduction

In the Semantic Web context, information extraction is the process of analyzing existing data and transforming a given information source into an ABox. We assume that an information source contains media data such as still images, audio and video files, or natural language text. Now suppose, the user wants to somehow summarize the content of the given data in an abstract, high-level way. In this case, information extraction is required. A formalization of information extraction can be given in an abductive context.

In order to construct a high-level interpretation, the ABox part of the ontology is extended with some new assertions describing individuals and their relations. These descriptions are derived by media interpretation processes using the ontology (we assume that the ontology axioms are denoted in a set Σ).

We consider the scenario of the interpretation process using still images as input. The output is a symbolic description of the media content represented as an ABox. This ABox is the result of an abduction process (s. Figure 5).

We illustrate this usage scenario using an example from the athletics domain (Figure 6. Assuming that it is possible to detect a horizontal bar bar_1, and a human $human_1$ by image analysis processes, the output of the analysis phase is represented as an ABox Γ. Assertions for the individuals and (some of) their relations detected by analysing Figure 6 are shown in Figure 7.

Figure 6: Image displaying a high jump or a pole vault event.

$$\begin{array}{rcl} bar_1 & : & Horizontal\_Bar \\ human_1 & : & Human \\ (bar_1, human_1) & : & near \end{array}$$

Figure 7: ABox Γ representing the result of the image analysis phase.

In order to continue the interpretation example, we assume that the ontology contains the axioms shown in Figure 8 (the ABox of the ontology is assumed to be empty). For some purposes, description logics are not expressive enough. Thus, some additional mechanism is required without jeopardizing decidability. In order to capture constraints among aggregate parts, we assume that the ontology is extended with DL-safe rules (rules that are applied to ABox individuals only. In Figure 9 a set of rules for the athletics example is specified. Note that the spatial constraints *touches* and *near* for the parts of a $Pole\_Vault$ event (or a $High\_Jump$ event) are not imposed by the TBox in Figure 8. Thus, rules are used to represent additional knowledge. Since spatial relations depend on the specific "subphases" of the events, corresponding clauses are included on the right-hand sides of the rules. For instance, a jumper as part of a $High\_Jump$ is near the bar if the image shows a $High\_Jump$ in the jump phase.

In the following we assume that rules such as those shown in Figure 9 are part of the TBox Σ. In order to provide a high-level interpretation, i.e. to provide a description of the image content in the form of high-level aggregates, we assume that spatial relations between certain objects detected by low-level analysis processes are not arbitrary.

In order to construct an interpretation for the image in Figure 6, an explanation is computed why it is the case that a human is near a horizontal bar. Such explanations are considered the results of image interpretation processes. As mentioned above, the idea is to use the abduction inference service for deriving these kinds of (minimal) explanations (in the sense of interpretations). Minimal explanations can be extended appropriately in order to match expectations and task context.

We start with the computation of a minimal explanation in the athletics scenario. For this purpose, we slightly modify the abduction equation by taking into consideration that initially the ABox does not need to be empty. Thus, we divide Γ (see Figure 7) into a part Γ_2 that the agent

$$
\begin{array}{rcl}
Athlete & \equiv & Human \sqcap \exists hasProfession.Sport \\
Jumper & \sqsubseteq & Athlete \\
Pole & \sqsubseteq & SportEquipment \\
Horizontal\_Bar & \sqsubseteq & SportEquipment \\
Foam\_Mat & \sqsubseteq & SportEquipment \\
Jumping\_Event & \sqsubseteq & Event \sqcap \\
 & & \exists hasParticapant.Jumper \sqcap \\
 & & \exists_{\leq 1} hasParticapant.Jumper \\
Pole\_Vault & \sqsubseteq & Jumping\_Event \sqcap \\
 & & \exists hasPart.Pole \sqcap \\
 & & \exists hasPart.Horizontal\_Bar \\
 & & \exists hasPart.Foam\_Mat \\
High\_Jump & \sqsubseteq & Jumping\_Event \sqcap \\
 & & \exists hasPart.Horizontal\_Bar \\
 & & \exists hasPart.Foam\_Mat \\
PV\_InStartPhase & \sqsubseteq & \top \\
PV\_InEndStartPhase & \sqsubseteq & \top \\
HJ\_InJumpPhase & \sqsubseteq & \top
\end{array}
$$

Figure 8: A tiny example TBox Σ for the athletics domain.

$$
\begin{array}{rcl}
touches(Y,Z) & \leftarrow & Pole\_Vault(X), \\
 & & PV\_InStartPhase(X), \\
 & & hasParticapant(X,Y), Jumper(Y), \\
 & & hasPart(X,Z), Pole(Z). \\
near(Y,Z) & \leftarrow & Pole\_Vault(X), \\
 & & PV\_InEndStartPhase(X), \\
 & & hasPart(X,Y), Horizontal\_Bar(Y), \\
 & & hasParticapant(X,Z), Jumper(Z). \\
near(Y,Z) & \leftarrow & High\_Jump(X), \\
 & & HJ\_InJumpPhase(X), \\
 & & hasPart(X,Y), Horizontal\_Bar(Y), \\
 & & hasParticapant(X,Z), Jumper(Z).
\end{array}
$$

Figure 9: Additional restrictions for $Pole\_Vault$ and $High\_Jump$ in the form of rules.

would like to have explained, and a part Γ_1 that the interpretation agent takes for granted. In our case Γ_2 is $\{(bar_1, human_1) : near\}$ and Γ_1 is $\{human_1 : Human, \quad bar_1 : Horizontal\_Bar\}$.

Coming back to the abduction problem specified above, we need solution(s) for the equation $\Sigma \cup \Delta \cup \Gamma_1 \models \Gamma_2$. In other words, given the background ontology Σ from Figures 8 and 9, the query

$$Q_1 := \{() \mid near(bar_1, human_1)\}$$

derived from Γ_2 should return *true*.

Obviously, this is not the case if Δ is empty. In order to see how an appropriate Δ could be derived, let us have a look at the rules in Figure 9. In particular, let us focus on the rules for $Pole\_Vault$ first. If we apply the rules to the query in a backward chaining way (i.e. from left to right) and unify corresponding terms we get variable bindings for Y and Z. The "unbound" variable X of the corresponding rules is instantiated with fresh individuals (e.g. pv_1). It is easy

to see that two possible solutions for the abduction equation can be derived. For this example the output of the interpretation process are two interpretation ABoxes representing two possible interpretations of the same image (see Figures 10 and 11).

$human_1$	:	$Human$
bar_1	:	$Horizontal\_Bar$
$(bar_1, human_1)$	:	$near$
hj_1	:	$High\_jump$
hj_1	:	$HJ\_InJumpPhase$
$human_1$	:	$Jumper$
$(hj_1, human_1)$	:	$hasParticapant$
(hj_1, bar_1)	:	$hasPart$

Figure 10: ABox representing the first result of the abduction process.

$human_1$	:	$Human$
bar_1	:	$Horizontal\_Bar$
$(bar_1, human_1)$	:	$near$
pv_1	:	$Pole\_Vault$
pv_1	:	$PV\_InEndStartPhase$
$human_1$	:	$Jumper$
$(pv_1, human_1)$	:	$hasParticapant$
(pv_1, bar_1)	:	$hasPart$

Figure 11: ABox representing the second result of the abduction process.

Note that due to the involvement of $human_1$ in the pole vault event in Figure 10, $human_1$ is now seen as an instance of $Jumper$, and, due to the TBox, also as an *Athlete*. Thus, information from high-level events also influences information that is available about the related parts. With queries for $Jumper$s the corresponding media objects would not have been found otherwise. Thus, recognizing high-level events is of utmost importance in information retrieval systems (and pure content-based retrieval does not help).

The example discussed here covers the interpretation of still images. It is necessary, however, to keep in mind that each media object might consist of multiple modalities, each of which will be the basis of modality-specific interpretation results (ABoxes). In order to provide for an integrated representation of the interpretation of media objects as a whole, these modality-specific interpretation results must be appropriately integrated.

3.1.4 Module Extraction / Integration

In this section, we focus on the use of modularity to support the *partial reuse* of ontologies. In particular, we consider the scenario in which we are developing an ontology $\mathcal{P}$ and want to reuse a set $\mathbf{S}$ of symbols from a "foreign" ontology $\mathcal{Q}$.

For exposition, suppose that an ontology engineer wants to build an ontology about research projects. The ontology defines different types of projects according to the research topics they focus on. The ontology engineer in charge of the projects' ontology may use terms such as Cystic_Fibrosis and Genetic_Disorder in his descriptions of medical research projects. In order to improve the precision of the ontology, he may want to add more details about the meaning of these terms; for reasons of cost and accuracy he would prefer to do this by reusing information from a medical ontology. The ontology engineer is supposed to be an expert on research projects; he may be unfamiliar, however, with most of the topics the projects cover and, in particular, with the terms Cystic_Fibrosis and Genetic_Disorder reused from the medical ontology $\mathcal{Q}$. Since the designer of the projects' ontology is not an expert in medicine and relies on the designers on $\mathcal{Q}$, it is to be expected that the meaning of the reused symbols is completely specified in $\mathcal{Q}$; that is, the fact that these symbols are used in the projects ontology $\mathcal{P}$ should not imply that their original meaning in $\mathcal{Q}$ changes. Therefore, it is reasonable to expect in our scenario that $\mathcal{P} \cup \mathcal{Q}$ yields the same consequences concerning the reused symbols as $\mathcal{Q}$ alone does.

- *Input to the usage scenario*: An ontology $\mathcal{P}$ being developed, an external ontology $\mathcal{Q}$ and the set $\mathbf{S}$ of symbols that $\mathcal{P}$ reuses from $\mathcal{Q}$.

- *Expected Output*: An ontology $\mathcal{P} \cup \mathcal{Q}$.
- *Success indicator*: $\mathcal{P} \cup \mathcal{Q}$ entails the same consequences over the signature of $\mathcal{Q}$ as $\mathcal{Q}$ does.

Moreover, in realistic application scenarios, it is often not reasonable to assume the foreign ontology $\mathcal{Q}$ to be "fixed"; that is, $\mathcal{Q}$ may evolve beyond the control of the modelers of $\mathcal{P}$. The modelers of $\mathcal{P}$ may not be willing (or authorized) to modify $\mathcal{Q}$ or, most importantly, they may decide at a later time to reuse the symbols Cystic_Fibrosis and Genetic_Disorder from a medical ontology other than $\mathcal{Q}$. Therefore, for application scenarios in which the external ontology $\mathcal{Q}$ may change, it is reasonable to "abstract" from the particular $\mathcal{Q}$ under consideration. In particular, the fact that the axioms in $\mathcal{P}$ do not change the meaning of the external symbols should be independent of the particular meaning of these symbols.

- *Input to the usage scenario*: An ontology $\mathcal{P}$ being developed and a signature $\mathbf{S}$.
- *Expected Output*: An ontology $\mathcal{P} \cup \mathcal{Q}$, where $\mathcal{Q}$ can be any external ontology such that $\mathcal{P}, \mathcal{Q}$ only share symbols from $\mathbf{S}$.
- *Success indicator*: $\mathcal{P} \cup \mathcal{Q}$ entails the same consequences over the signature of $\mathcal{Q}$ as $\mathcal{Q}$ does.

Even if $\mathcal{P}$ "safely" reuses a set of symbols from an ontology $\mathcal{Q}$, it may still be the case that $\mathcal{Q}$ is a large ontology. In particular, in our example, it is typically the case that medical ontologies are very large, and importing the whole ontology would make the consequences of the additional information costly to compute and difficult for our ontology engineers (who are not medical experts) to understand. In practice, therefore, we need to extract a module $\mathcal{Q}_1$ of $\mathcal{Q}$ that includes just the relevant information. Ideally, this module should be *as small as possible* while still *guaranteeing* to capture the meaning of the terms used; that is, when answering arbitrary queries against the research projects ontology, importing the module $\mathcal{Q}_1$ would give us *exactly the same answers* as if the whole medical terminology ontology $\mathcal{Q}$ had been imported; that is, $\mathcal{P} \cup \mathcal{Q}_1$ should yield the same logical consequences in the signature of $\mathcal{P}$ as $\mathcal{P} \cup \mathcal{Q}$ does. In this case, importing the module instead of the whole ontology will have no observable effect on the projects. Furthermore, the fact that $\mathcal{Q}_1$ is a module in $\mathcal{Q}$ should be independent from the particular $\mathcal{Q}$ under consideration: if $\mathcal{Q}_1$ "behaves" in the same way as $\mathcal{Q}$ for a given ontology $\mathcal{P}$ but not for a different ontology $\mathcal{P}'$, then $\mathcal{Q}_1$ should not be a module in $\mathcal{Q}$.

- *Input to the usage scenario*: An ontology $\mathcal{Q}$ and a signature $\mathbf{S}$.
- *Expected Output*: A subset $\mathcal{Q}_1 \subseteq \mathcal{Q}$.
- *Success indicator*: $\mathcal{P} \cup \mathcal{Q}_1$ entails the same consequences over the signature of $\mathcal{P}$ as $\mathcal{P} \cup \mathcal{Q}$ for every external ontology $\mathcal{P}$ that imports $\mathcal{Q}_1$ such that all the symbols that $\mathcal{P}$ and $\mathcal{Q}$ share are in $\mathbf{S}$.

3.1.5 Modularization

Suppose that $\mathcal{Q}$ is a well-established ontology, possibly very large. Suppose that an ontology engineer is interested in a particular symbol A (set of symbols $\mathbf{S}$) in the signature $\mathsf{Sig}(\mathcal{Q})$ of $\mathcal{Q}$. The ontology engineer wants to filter relevant axioms in the ontology $\mathcal{Q}$ according to a collection of criteria.

For example, given a concept name $A \in \mathsf{Sig}(\mathcal{Q})$, the ontology engineer may be interested in retrieving a set of axioms $\mathcal{Q}_1 \subseteq \mathcal{Q}$ sufficient to entail all the super-classes of A in $\mathcal{Q}$; that is, if $\mathcal{Q}$ entails $A \sqsubseteq B$, then $\mathcal{Q}_1$ entails $A \sqsubseteq B$. Analogously, given $A \in \mathsf{Sig}(\mathcal{Q})$, the ontology engineer may be interested only in the sub-concepts of A in $\mathcal{Q}$.

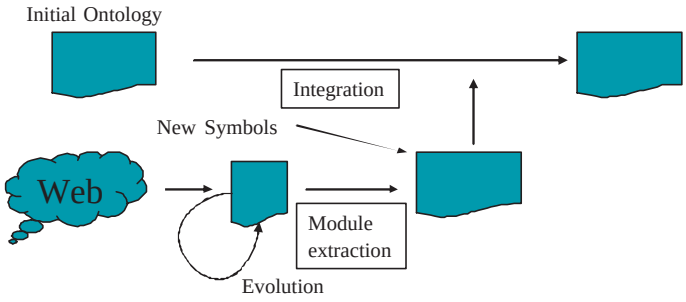

Figure 12: Usage Scenario for Module Extraction / Integration

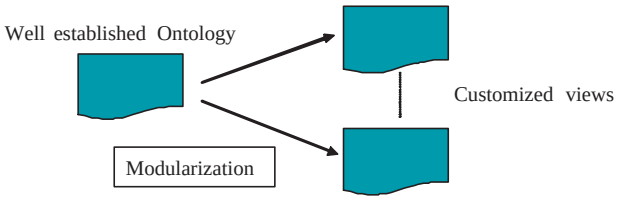

Figure 13: Usage Scenario for Modularization

- *Input to the usage scenario*: An ontology $\mathcal{Q}$, a signature $\mathbf{S}$ and a collection of criteria (such as a collection of entailment relations).
- *Expected Output*: A subset $\mathcal{Q}_1 \subseteq \mathcal{Q}$.
- *Success indicator*: $\mathcal{Q}_1 \subseteq \mathcal{Q}$ meets the input criteria.

3.1.6 Ontology Matching / Integration

In its simplest form, concept matching is the problem of finding a substitution of *variables* in a *concept pattern* D by concept descriptions such that a concept description C is equivalent to the instantiated concept pattern D. It was first considered in [39] for the application of generating explanations. In [7], it was shown that matching can also be used to find similarities in ontologies. More precisely, it can be used to identify multiple concept descriptions that match the same pattern. In a similar way, in [12] use of matching to support the integration of ontologies was presented. In [15], concept matching was extensively investigated.

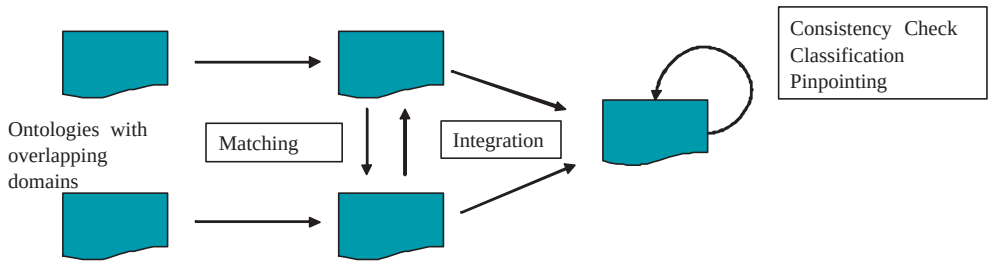

Figure 14: Usage Scenario for Matching / Integration

Consider a scenario, in which several ontologies from the same application domain are to be merged into a new ontology that captures the meaning of the original terms, and their inter-

relationships. This scenario can be motivated by different applications. It can be the case that these ontologies have been developed independently by multiple research groups, and an ontology that merges these ontologies is desired. It can also be the case that there may be many examples of heterogeneous information sources, such as databases or semi-structured data on the internet, for which a uniform, combined ontology is desired. In such cases, it is possible that different ontologies contain similarly defined, yet different concepts for the same notion in the application domain. This would lead to redundancy in the resulting ontology, and spoil the clarity, which is an important quality criteria for ontologies. To some extent, standard reasoning services can be used to detect such cases. For instance, concepts coincide in the subsumption hierarchy if they are equivalent. In cases where this is not possible, inter-ontology concept matching can be used to detect similar concept definitions in different ontologies, and to avoid redundancy in the resulting ontology.

In [15], a setting where matching can be used to avoid redundancy was described. Consider a case where two ontologies independently intend to introduce the same concept but define it differently. For instance, let

$$\mathcal{T}_1 := \{\mathsf{Parent} \equiv \mathsf{Person} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Person}\}$$

and let

$$\mathcal{T}_2 := \{\mathsf{FatherOrMother} \equiv \mathsf{Human} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Human}\}.$$

Clearly, both definitions intend to represent the same notion. If we want to merge these two ontologies, the resulting ontology will redundantly contain two definitions for the notion of parent. In such cases, redundancies can be detected by using matching. In the above example, both ontologies can be queried for concepts matching the concept pattern $X \sqcap \exists \mathsf{has\text{-}child}.X$ to detect that $\mathcal{T}_1$ and $\mathcal{T}_2$ contain a similar concept definition for the same notion.

As mentioned above, in this usage scenario the non-standard reasoning service concept matching is used, which was mentioned in Section 3.4 of [26], and in Section 3.6 of [37]. The input for this usage scenario is a concept pattern, and two acyclic $\mathcal{ALE}$ TBoxes. The expected output is the set of concept descriptions that match the given pattern for each of the input ontologies. The relevant success indicators of our usage scenario are run-time, maximum amount of memory used, and the complexity of the concept pattern as well as the TBox. The candidate test ontology is a pruned version of DICE [21].

3.1.7 Model Generation

Suppose the ontology designer would like not just to check the consistency of the developed ontology but also see all possible models by inspecting their graphical presentation. This way, unwanted models can be discovered and the ontology designer can interactively adjust the ontology if required. Well-known model generation tools support the process of systematical enumeration of all models quite well. Current tableau algorithms return just (a description of) a so-called single canonical model [4]. However, for checking the satisfiability of ontologies and computing the taxonomy, tableau algorithms have been proven to be very effective. Thus, the idea is to benefit from a synergy between DL systems and model-generating tools by providing a new reasoning service, a model finding service. Invoking this service, usage scenarios such as analyzing concept subsumption and concept equivalence by model inspection become possible. The applicability of finite model finders (by the example of Alloy Analyzer) for ontology design tasks was studied in [1]. For applying model-checking techniques offered by the Alloy system, translation rules from DL ($\mathcal{ALC}$ and some constructs of $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$) into Alloy have been proposed.

- *Input for the usage scenario*: An ontology (TBox), a counterexample for concept subsumption (or concept equivalence), model finding tool (e.g., Alloy), configuration parameters for the model finding tool (e.g., problem size).
- *Expected Output*: Model descriptions, model visualizations.

3.1.8 Authoring/Refining Concept Descriptions

As an ontology is essentially a pile of concept descriptions, ontology design and maintenance repeatedly involve authoring new concept descriptions and refining existing ones. The addition of new concept descriptions, as well as refinement of old concept descriptions, might very well lead to changes in the structure of the ontology under development. These changes could be local or global: A local change happens if the concept under consideration is classified to a new position in the concept hierarchy or if the concept turns to be (un)satisfiable. A global change happens if much – if not all – of the ontology structure is altered. An obvious example of a global change made by concept authoring is that the new concept description is globally inconsistent and thus makes the whole ontology inconsistent as well. This is an extreme scenario where reasoning techniques for consistency checking have to be applied. In this section, we are more concerned with the scenario where local changes to the ontology structure are made.

It is readily shown in the previous deliverables that authoring and refining concept descriptions straightforwardly relate to the ontology-based task *computing subsumption hierarchy*. To briefly recap, subsumption hierarchy is a directed acyclic graph (aka. a multiple tree) of concepts occurring in the ontology. Concepts placed in a higher position are said to be subsumers (more general concepts) of those placed in a lower position in the subsumption hierarchy. The computation of the subsumption hierarchy out of an ontology is sometimes known as the *ontology classification*, which is one of the most classic ontology-based tasks.

Automatically Generated Ontology as Input The ontology developers might have obtained a preliminary version of their ontology in some way from existing data. A few possibilities include to extract an ontology from a database schema (see Section 3.1.1), to structure text-based resources from the Web, and to migrate and enrich an existing terminology or classification system. The last approach is ubiquitous in the domains in which classification data is well studied and readily available, for instance, the biomedical domain. This preliminary ontology may be considered the input to this usage scenario. Such an ontology is usually assumed to be of low quality both in terms of structure and semantics. Since the ontology is likely to be automatically generated by machine, it is usually sorely in need of expert refinement. The resulting structure might not be perfect, nor might the subsumption hierarchy be ontologically correct according to the domain of interest.

3.1.9 Support for Reuse

Using of Commonalities In the main usage scenario for computation of commonalities, the user starts from an ontology that she wants to extend. Since the user is an expert in her application domain, but not in the field of logic, the user needs support in formulating the concept descriptions in the used DL that capture the intended notions. Suppose, the user starts from an ontology, classifies it, but finds that a particular part of the concept hierarchy has too many sibling concepts and is not capturing all notions from the application domain. In this case the user would like to select a collection of somewhat closely related (sibling) concepts from the ontology and introduce a new concept that subsumes the selected concepts. By this

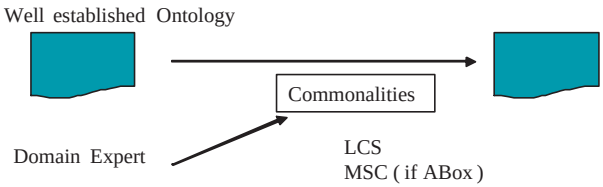

Figure 15: Usage Scenario for Reusing of Commonalities

procedure she would introduce an intermediate level in the concept hierarchy. Now, in order to support the definition of such a new concept one can employ the computation of commonalities for the selected concepts to obtain a candidate concept description. The concept description obtained by computing the commonalities is then inspected by the user and, perhaps, edited by her, assigned a new concept name and then added to the ontology. Or, if the returned concept description does not complement the existing ontology nicely, the user picks a new collection of concepts from the ontology. The main inference employed to realize this usage scenario is the computation of *least common subsumers* (lcs).

The main usage scenario has two variants. One is described by the *bottom-up* approach introduced in [6], where the TBox is extended by picking related and somewhat similar individuals from the ABox to form a new concept in the TBox. To this end, each of the selected individuals is first generalized into a concept description by computing their *most specific concept* (msc) and then all of these concept descriptions are generalized by computing their lcs. In the second variant of the base usage scenario described above the user starts from a *background knowledge base* that she wants to extend and adapt to her application by a *user knowledge base*. Typically the *user knowledge base* is written in a less expressive DL that the *background knowledge base*. Now, in this setting the concept description may only use concept constructors present in the user DL, but can refer to names that are defined in the background knowledge base. However, we will concentrate on the base usage scenario here. To sum up, computing common subsumers finds its usage in the tasks generating concept descriptions, bottom-up construction of ontologies, structuring the ontology, and ontology customization, which were previously mentioned in [37].

The main usage scenario is based on the inference service of computing the lcs. It is important to note that for DLs supporting concept disjunction, the computation of the lcs is trivial and not of much use in our scenario, since the lcs of a collection of such concept descriptions is simply their disjunction. So, in applying the lcs, the user does not learn new things about the commonalities of the selected concept descriptions. A way to remedy this to some extend is to first compute the *concept approximation* (see also Section 3.1.9 of this Deliverable) and then apply the lcs, see [14].

In the setting where a background knowledge base is extended by a user knowledge base appropriate techniques were introduced in [3]. Here the user builds a user ontology in the user DL by extending a background ontology $\mathcal{T}$ written in an expressive DL and by using the names from the signature of the background ontology. Thus, if the user DL offers all concept constructors from $\mathcal{ALE}$ and the background DL is $\mathcal{ALC}$, the user DL is called $\mathcal{ALE}(\mathcal{T})$, since it may use names defined in $\mathcal{T}$. Results on the existence of the lcs in this setting can be found in [45]. In more detail, the $\mathcal{ALE}(\mathcal{T})$-lcs w.r.t. unfoldable $\mathcal{ALC}$-TBoxes exists, while for $\mathcal{ALC}$-TBoxes with GCIs or even just $\mathcal{ALC}$-TBoxes containing cyclic definitions the lcs does not need to exist. However, the relaxed notion of a *good common subsumer*, that does return a common subsumer of the input concept description, but not necessarily the least one, might still be

helpful to solve the ontology tasks addressed above.

In our usage scenarios we expect an unfoldable $\mathcal{ALEN}$-TBox and collections of concept names that describe related concepts and to each of which the computation of commonalities should be applied. Alternatively, for the setting of customizing a background ontology by a user ontology, we expect an (unfoldable) $\mathcal{ALC}$ background TBox and an $\mathcal{ALE}(\mathcal{T})$ user TBox and, again, collections of concept names from the combined TBox.

We expect that the computation of commonalities yields a new concept description that captures the common characteristics of the input concept descriptions. Success Indicators for testing the computation of commonalities is the run-time needed, the memory needed during computation. Regarding the inference we should take into account the size of the TBox and the size of the output. Since we need unfoldable ontologies, we propose (a pruned version of) the DICE ontology, see [21].

Concept Simplification For concept simplification the main usage scenario is to support users who are domain experts, but no experts in knowledge representation. However, since the user is not a DL expert, she employs a less expressive DL and needs support, for example through the bottom-up approach when building this user-specific extension of an existing ontology. There are several reasons for the user to employ a restricted DL in this setting:

1. a restricted DL may be easier to comprehend and use for a non-expert
2. it may allow for a more intuitive graphical or frame-like user interface [9, 36, 10]
3. to use the bottom-up approach, the lcs must exist and make sense, i.e., only DLs that do not offer concept disjunction are of interest
4. it must be possible to compute it with reasonable effort

However, the existing ontology that the user wants to inspect or wants to extend is written in a more expressive DL, which might be hard to understand for a user with little expertise in logic. Now, in order to display the concept descriptions from the ontology in a more comprehensive way to the user, the initial concept descriptions should be "transformed" into a concept description in the user DL. This should not be done simply by pruning the concept description, i.e., removing sub-concept descriptions that contain constructors present in the ontology DL, but not present in the user DL. Instead one would need to compute a "close" (w.r.t. subsumption) concept description in the user DL of the initial concept description and display this description to the user. In our case the obtained concept description in the user DL generalizes the initial concept description from the ontology only as much as necessary. The inference service to compute such a concept description is called *concept approximation*.

Since the concept description returned by approximation can grow very large, it is necessary to display the approximated concept description in a succinct way. This leads us to an extension of the above usage scenario. In the extended scenario a *minimal rewriting* of the approximation concept description is computed. A minimal rewriting is a concept description equivalent to the original concept description of smaller, more precisely of minimal size. These kind of rewritings (re-)introduce concept names from the TBox for sub-concept descriptions in the approximation concept and thus are more succinct.

The basic usage scenario for concept simplification as well as the extended one illustrate the ontology task of *concept inspection* mentioned in [37]. In case where the initial ontology is a ready-made one, perhaps obtained by an ontology provider, the above usage scenario in addition covers the ontology task of *ontology customization* also mentioned in [37]. In this setting the

user employs the user DL not only for inspection, but adds new concepts to the ontology to adapt it to her application.

In addition to the mentioned usage scenarios the two mentioned inferences are important to extend other so-called *non-standard inferences* such as the computation of least common subsumers (see Section 3.1.9) to more expressive DLs – at least by obtaining an approximative solution by first approximating the in put concept description and then applying the desired non-standard inference.

The two inference services, computing concept approximation and computing a minimal rewriting, are so far mainly investigated for unfoldable TBoxes. Approximation was first investigated in [14] where an algorithm for computing $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions was devised. This method was extended to number restrictions in [13].

The approach of computing non-standard inferences w.r.t. a background ontology was introduced in [3]. Here the user builds a user ontology in the user DL by extending a background ontology $\mathcal{T}$ written in an expressive DL and by using the names from the signature of the background ontology. Thus, if the user DL offers all concept constructors from $\mathcal{ALE}$ and the background DL is $\mathcal{ALC}$, the user DL is called $\mathcal{ALE}(\mathcal{T})$, since it may use names defined in $\mathcal{T}$. Results on the existence of the lcs in this setting can be applied to approximation, see [45]. In more detail, the $\mathcal{ALE}(\mathcal{T})$-approximation w.r.t. unfoldable $\mathcal{ALC}$-TBoxes exists, while for $\mathcal{ALC}$-TBoxes with GCIs or even just $\mathcal{ALC}$-TBoxes containing cyclic definitions the approximation does not need to exist. However, similar to the case of the lcs, a relaxed notion of concept approximation, that does not return the least generalization of the input concept in the target DL, but a more general one, might still be helpful to solve the ontology tasks addressed above.

Regarding minimal rewriting we can realize the extended usage scenario by the results from [2]. Here minimal rewriting is investigated for the DL $\mathcal{ALE}$ w.r.t. unfoldable TBoxes. Intuitively, the algorithm proceeds in two phases. In the first phase redundant concept names from the TBox are conjoined to sub-concept descriptions of the input concept description. In the second phase redundant $\mathcal{ALE}$- (sub-)concept descriptions are eliminated. Thus the resulting concept description uses $\mathcal{ALE}$-concept constructors and names from $\mathcal{T}$. Unsurprisingly, the algorithm devised in [2] for $\mathcal{ALE}$ directly transfers to the case for $\mathcal{ALE}(\mathcal{T})$.

For our usage scenario we expect either an unfoldable $\mathcal{ALCN}$-TBox (or a TBox of lesser expressivity) to apply concept approximation to the concept descriptions. Alternatively, we can start from an $\mathcal{ALC}$ background TBox and a user TBox of lesser expressivity to apply concept approximation in the customization setting. In case we chose $\mathcal{ALE}$ (or $\mathcal{ALE}(\mathcal{T})$) as the user DL in each of the two scenarios, we can apply minimal concept rewriting to the approximated concept descriptions.

The concept simplification scenario can be tested with the different inputs described above. For this usage scenario it is essential to measure run-time, size of the concept descriptions obtained by concept approximation and by minimal rewriting. To obtain meaningful measurements in the end, the size of the TBox and the unfolded input concept description should also be measured.

Any unfoldable $\mathcal{ALCN}$ TBox can be used to test concept simplification by approximation. For example, a pruned version of the medical DICE ontology (see [21]) can be used for this purpose.

3.1.10 Ontology Completion

DLs are employed in various application domains, such as natural language processing, configuration, databases, and bio-medical ontologies, but their most notable success so far is due to

the fact that DLs provide the logical underpinning of OWL, the standard ontology language for the semantic web [32]. As a consequence of this standardization, several ontology editors support OWL [36, 40, 35], and ontologies written in OWL are employed in more and more applications. As the size of these ontologies grows, tools that support improving their quality become more important. The tools available until now use DL reasoning to detect inconsistencies and to infer consequences, i.e., implicit knowledge that can be deduced from the explicitly represented knowledge. There are also promising approaches that allow to pinpoint the reasons for inconsistencies and for certain consequences, and that help the ontology engineer to resolve inconsistencies and to remove unwanted consequences [44, 34] (see also Section 7 of [1]). These approaches address the quality dimension of *soundness* of an ontology, both within itself (consistency) and w.r.t. the intended application domain (no unwanted consequences). Here, we are concerned with a different quality dimension: *completeness*. *Ontology completion* is a formally well-founded technique that supports the ontology engineer in checking whether an ontology contains all the relevant information about the application domain, and to extend the ontology appropriately if this is not the case.

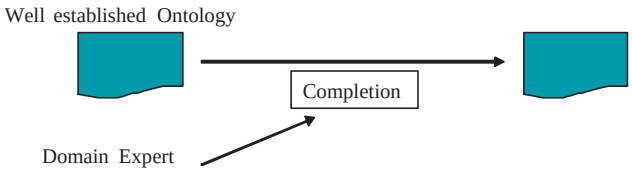

Figure 16: Usage Scenario for Ontology Completion

Consider the following scenario: Suppose the user has a well established ontology from an application domain, and she wants to find out whether the ontology contains all the relevant information about the domain, i.e. she is looking for answers to the following questions:

- Are all the relevant constraints that hold between concepts in the domain captured by the TBox?
- Are all the relevant individuals existing in the domain represented in the ABox?

As an example, consider the OWL ontology for human protein phosphatases that has been described and used in [48]. This ontology was developed based on information from peer-reviewed publications. The human protein phosphatase family has been well characterized experimentally, and detailed knowledge about different classes of such proteins is available. This knowledge is represented in the terminological part of the ontology. Moreover, a large set of human phosphatases has been identified and documented by expert biologists. These are described as individuals in the assertional part of the ontology. One can now ask whether the information about protein phosphatases contained in this ontology is complete. Are all the relationships that hold among the introduced classes of phosphatases captured by the constraints in the TBox, or are there relationships that hold in the domain, but do not follow from the TBox? Are all possible kinds of human protein phosphatases represented by individuals in the ABox, or are there phosphatases that have not yet been included in the ontology or even not yet been identified?

Such questions cannot be answered by an automated tool alone. Clearly, to check whether a given relationship between concepts—which does not follow from the TBox—holds in the domain, one needs to ask a domain expert, and the same is true for questions regarding the

existence of individuals not described in the ABox. The role of the automated tool is to ensure that the expert is asked as few questions as possible; in particular, she should not be asked trivial questions, i.e., questions that could actually be answered based on the represented knowledge. In the above example, answering a non-trivial question regarding human protein phosphatases may require the biologist to study the relevant literature, query existing protein databases, or even to carry out new experiments. Thus, the expert may be prompted to acquire new biological knowledge.

In this usage scenario, the following reasoning services are used: computing the subsumption hierarchy, and instance checking. Whenever the domain expert confirms a question to be true, the relevant GCI is added to the TBox, and the subsumption hierarchy is recomputed. In order to compute the right handsides of the questions, we need to do some instance checks (see Section 10 of [1]).

As input, the scenario expects a well established ontology written in a DL that supports conjunction and negation, and has a TBox formalism that allows for GCIs. Also, an expert from the application domain is required to answer the questions asked. The expected output for the scenario is the input ontology enriched with new subsumption relationships and new instances that are acquired from the expert as answer to the questions asked. The success indicators relevant to this usage scenario are, the number of questions asked to the expert, the understandability of the questions, the CPU time used, and the maximum amount of memory used. By understandability of the question we mean whether the question contains redundancies that make it difficult to be understood by the domain expert, and whether the question can be shortened.

3.2 Usage Scenarios for Ontology Access, Processing and Usage

3.2.1 ABox Query Answering

Suppose the user wants to access some information from a given data source using a certain background ontology. For this, the user poses a query that describes information he is looking for in terms of the underlying ontology. This way, queries can be formulated in the vocabulary closed to the high-level vocabulary of the user. This kind of information retrieval w.r.t. to an ontology is one of main topics in [16].

In many practical application systems based on DLs, powerful ABox query answering is one of the main requirements. The main usage scenario is to answer a user's query to the extensional part of a knowledge base. The base usage scenario is given as follows: As input we have a conjunctive query. The output of the query is a set of assignments for free variables in the query. The success indicators for query answering are similar to those of other usage scenarios: e.g., CPU time for answering the query and amount of memory used. To rate the efficiency w.r.t. scalability one can take into account size and complexity of the source ABox. The only constraint on applicable test ontologies is that they have an extensional part (ABox).

Let us consider an example using grounded conjunctive queries. Suppose that we have the following small TBox and ABox:

$$
\begin{array}{lcl}
Teacher & \equiv & Person \sqcap \exists teaches.Course \\
Student & \equiv & Person \sqcap \exists takes.Course \\
 & & \\
John & : & Person \\
Richard & : & Person \sqcap Famous \\
math & : & Course \\
biology & : & Course \\
teaches(Richard, math) & & \\
takes(John, math) & & \\
takes(John, biology) & &
\end{array}
$$

An example query to determine all the courses that the student John takes which are taught by a famous teacher:

$$Q := \{(x) \mid Student(John), takes(John, x), Course(x), teaches(y, x), Teacher(y), Famous(y)\}$$

The result for the query Q is $\{(math)\}$.

ABox query answering as shown above can be used, for instance, to implement information retrieval systems based on high-level interpretations of media objects. Interpretations are seen as annotations of media objects (metadata) and can be practically represented in RDF or OWL format. In our view, annotations describe "real-world" objects and events. It is not the goal to merely "classify" images and attach keywords but to construct a high-level interpretation of the content of a media object.

A set of media objects with annotations attached to each media object can be made available via a web server with standard application server technology. We assume that the web server provides a query interface (for instance, using the XML-based DIG 1.2 or OWL-QL query language).

$$
\begin{array}{rcl}
human_1 & : & Human \\
bar_1 & : & Horizontal\_Bar \\
(bar_1, human_1) & : & near \\
pv_1 & : & Pole\_Vault \\
pv_1 & : & PV\_InEndStartPhase \\
human_1 & : & Jumper \\
(pv_1, human_1) & : & hasParticapant \\
(pv_1, bar_1) & : & hasPart \\
(human_1, url_1) & : & hasURL \\
(bar_1, url_2) & : & hasURL \\
(url_1) & : & =\texttt{"http://www.img.de/image-1.jpg"} \\
(url_2) & : & =\texttt{"http://www.img.de/image-1.jpg\#(200,400)/(300/500)"} \\
 & \ldots &
\end{array}
$$

Figure 17: An example ABox representing the annotation of the image in Figure 6. The predicate $=_{string}$ stands for a one-place predicate $p(x)$ which is true for $x = string$.

Using the example from Figure 6 we sketch how media interpretations are used to implement a media retrieval system. Figure 17 illustrates the main ideas about annotations for media objects using ABoxes (we omit the TBox for brevity).

A query which might be posed to an information system is shown in Figure 18. As a result, the inference system returns the tuple $(bar_1, human_1)$, and in order to show the image (and highlight the area with the bar), the associated URL names can be retrieved (see also Figure 18). The form $value(x)$ returns a unique binding for a variable (in this case a string) if it exists, and $\emptyset$ otherwise. In case of $URLQuery_1$ the answer is $(url_1, \texttt{"http://www.img.de/image-1.jpg"})$. The result of $URLQuery_2$ is defined analogously. The URLs can be used to actually retrieve the image data. Subsequent queries w.r.t. the annotation individuals bar_1 and $human_1$ are certainly possible. We do not discuss details here, however. In summary, it should be clear now, how annotations with metadata are used in an ontology-based information retrieval system.

It is easy to see that annotations such as the ones shown in Figure 17 can be set up such that the URLs are tied to the ABox individuals comprising the high-level descriptions. In particular, one can easily imagine a situation in which there exist multiple interpretations of an image, which results in multiple annotations being associated with an image. In addition, it is obvious that a repository of media objects together with their annotations (metadata) gives rise to one or more ABoxes that are managed by the ontology-based information system. Not so obvious is how metadata can be automatically derived since manual annotation is too costly in almost all practical scenarios. The derivation of metadata representing high-level interpretation of media content has been already discussed in Section 3.1.3.

$$
\begin{aligned}
ImageQuery_1 &:= \{(X,Y) \mid \quad Horizontal\_Bar(X), Human(Y), near(X,Y)\} \\
URLQuery_1 &:= \{(X, value(X)) \mid \quad hasURL(bar_1, X)\} \\
URLQuery_2 &:= \{(X, value(X)) \mid \quad hasURL(human_1, X)\}
\end{aligned}
$$

Figure 18: Query for "a human with a bar" and subsequent queries for retrieving the URLs w.r.t. the result for $ImageQuery_1$.

3.2.2 TBox Query Answering

Well-known query languages for description logics (e.g., nRQL, OWL-QL etc) allow for formulating so-called "structural queries" that ask for certain aspects of the taxonomy of the knowledge base. The most APIs offer a set of predefined queries for asking about sub/superclass relationships of the TBox. However, these functions are typically restricted to known terms in the knowledge base. Recent investigations on practical DL systems go in the direction to provide a powerful TBox query language that offers the same flexibility as ABox query languages do. TBox querying is supported in Racer 1.9 for several years now. Therefore, we use the Racer syntax in this subsection for specifying TBox-related query examples.

Intuitively, whereas the ABox queries bind variables against ABox individuals, the variables in a TBox query are bound against taxonomy nodes. Suppose the user wants to retrieve all concepts equivalent to some concept C. For this query, e.g., the following construct in nRQL can be used (C must be an atomic concept):

```
(tbox-retrieve (?x) (?x C))
```

From the technical point of view, in nRQL a special relational structure of the taxonomy (a so-called taxonomy ABox) is constructed, which can be then queried. Under this perspective, a TBox query is just an ordinary ABox query which is evaluated on the (virtual) taxonomy ABox. As mentioned above, currently only atomic concepts are allowed in a TBox query. However, recent investigations consider also the efficient implementation of TBox queries with complex concepts involved such as the following example shows:

```
(tbox-retrieve (?y) (and (?x ?y has-child) (?x (some R top)))
```

Here, we are looking for children of the concept `(some R top)`. The problem is that in order to answer this query, the TBox must be classified first, namely only for this one query.

Finally, let us present an example which demonstrates how to combine TBox queries and ABox queries. Sometimes, the user wants to retrieve only *direct instances* of a concept / OWL class. An individual is called a direct instance of a concept / OWL class if there is no subconcept / subclass of which the individual is also an instance. Let D be a subconcept of C defined as follows:

```
(define-concept C (some r top))
(define-concept D (and C E))
```

We can verify that D is indeed a child concept of C, using the following TBox query:

```
(tbox-retrieve (?x) (C ?x has-child))
```

Let us create two individuals so that i and j are instances of C; moreover, j is also an instances of D:

```
(related i j r)
(related j k r)
(instance j E)

(retrieve (?x) (?x C))
> (((?x j)) ((?x i)))

(retrieve (?x) (?x D))
> (((?x j)))
```

The previous queries demonstrated that both i as well as j are instances of C. However, only i is a direct instance of C. We can retrieve direct instances of C as follows:

```
(retrieve1 (?x C)
     (((:lambda (x)
          (if (some (lambda (subclass)
                      (retrieve () `(,x ,subclass)))
                    (flatten
                      (tbox-retrieve1 `(C ?subclass has-child)
                        `(((:lambda (subclass) subclass) ?subclass)))))
              :reject
           `(?x ,x)))
        ?x)))
> (((?x i)))
```

As we see from the example above, a programming language (in this case MiniLisp expression language of nRQL) is needed to specify arbitrary custom retrieval (filter) predicates. With MiniLisp, e.g., a user can write a simple (termination safe) "program" which is executed on the RacerPro server. MiniLisp programs give the flexibility to specify the format of the query answers, write query answers to a file, start subqueries, etc.

The base usage scenario for TBox query answering is similar to ABox query answering: As input we have a TBox (or mixed TBox and ABox) query. The output of the query is a set of assignments for free variables in the query. The success indicators are, e.g., CPU time and memory usage.

3.3 Usage Scenarios for Ontology Interoperation

Ontology interoperation concerns both an ontology that interoperates with a data storage (which can be tightly coupled with the ontology or external to the ontology), and multiple independent ontologies that are operating with each other. We analyze different usage scenarios of interoperation, which are described in the following.

3.3.1 Ontology-To-Data-Storage

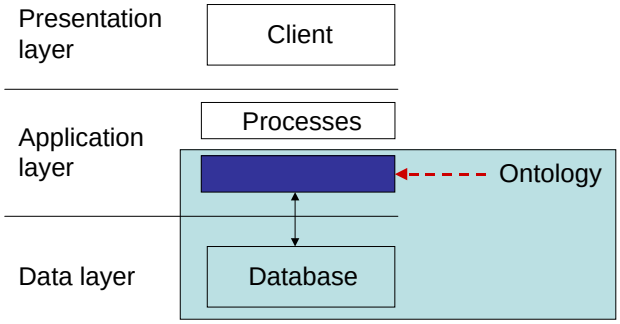

Figure 19: Ontology-To-Data-Storage architecture

In this scenario, an ontology is used to design and access a data storage. We consider Relational Database Management Systems (RDBMSs) as data storage engines. A database managed by the DBMS can be also accessed independently of the ontology. However, the ontology and the database are tightly related, indeed, the database can be changed to accommodate the ontology. The main problems to be faced in this scenario are the impedance mismatch between objects instances of the ontology and values stored in the database, and the problem of answering queries posed over the ontology in terms of data stored in the database. The first problem is dealt with by means of suitable mappings that relate the database schema to the ontology. Such mappings allow for generating abstract objects of the ontology and for populating (partially) their attributes. The second problem is dealt with by means of suitable query reformulation techniques that allow to rewrite queries specified in terms of the ontology in queries posed to the database. Reasoning services which depends on data (e.g., *DL-Lite* query answering, consistency check, update at instance level) will be considered in this scenario. The main measures for the success of the applied ontology-based techniques will be soundness and completeness of the techniques, and also the possibility of maintain reasoning tractable. More precisely, since database instances may be very large, computation going beyond SQL (i.e., beyond LOGSPACE in data complexity) is problematic, and a measure of success for us is to have reasoning techniques (e.g., query answering techniques) that are reducible to SQL query evaluation over a DBMS.

3.3.2 Ontology-To-Data-Sources

This scenario represents an advanced form of data integration, where we access data sources that are independent from the ontology, and that are related to the ontology through suitable

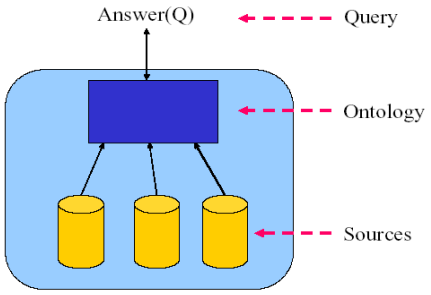

Figure 20: Ontology-To-Data-Sources architecture

mappings. As external sources we consider relational databases. As ontologies, we consider *DL-Lite* TBoxes. Differently from the scenario of ontology-to-data storage, sources cannot change for accommodate the ontology. However, also in this case, the impedance mismatch problem and the query answering problem described in the scenario above are of concern. Also, the problem of reformulating updates/erasure over the ontology to updates over the sources is particularly challenging. Again, maintaining reasoning in LOGSPACE, i.e., SQL reducible, is a measure of success for us in this scenario.

3.3.3 Ontology-To-Ontology

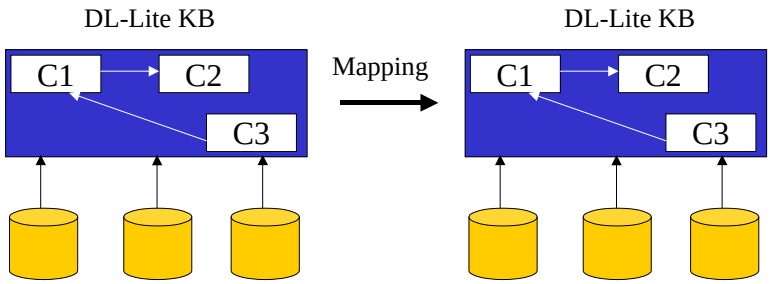

Figure 21: Ontology-to-Ontology

We consider two different (possibly situated) ontologies with unidirectional mapping, i.e., information flows only from one ontology to the other. We call them source ontology and target ontology. We consider exchanging of instances according to the mapping specification. This task consists in establishing which ABox assertions need to be materialized in the target ABox on the basis of the knowledge provided by the source ontology and the mapping. We then consider query answering over the target ontology. After the exchange of the data, this task can be performed locally, i.e., as the source ontology were not connected to the target one. We also consider the task of virtual, i.e., on-demand, information integration, and perform again query answering over the target ontology. In this case, the source ontology cannot be disregarded during query answering, and the problem arises of establishing what needs to be asked to the source ontology to provide the correct and complete answer to the original query. This scenario can be easily generalized to the scenario in which a set of ontologies is hierarchically connected.

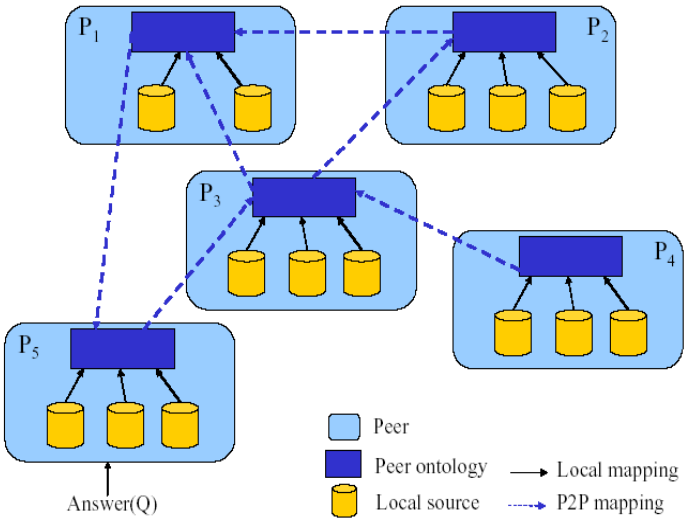

Figure 22: Ontology-to-Ontology

3.3.4 Network Of Ontologies

In this scenario, ontologies are freely connected in a network of peer-ontologies (peer-to-peer scenario). A peer ontology is a stand-alone or situated ontology that is part of system formed by other peer ontologies. Interaction between the peer ontologies is modeled by explicit peer mappings. No constraints are posed on the topology of the mappings, i.e., cycles in the mappings are allowed. Besides reasoning services considered in the ontology-to-ontology scenario, also epistemic reasoning over multiple ontologies is considered in this scenario. This allows us to suitable model single peers as autonomous agents that exchange information in a peer-to-peer network.

4 Test Ontologies

In this section, ontologies for testing usage scenarios discussed in the previous sections are listed. These ontologies were selected from sources outside of our project. For every ontology, we provide a short description and mention at least some of the following features:

- DL fragment
- Number of concepts, roles
- Number of TBox axioms and type of the TBox (cyclic, with GCIs etc.)
- Number of ABox concept and roles assertions
- Number of objects
- Concrete domain predicates: owl-style or constraint system
- Format of ontology (OWL, DAML, DIG, KRSS etc.)

Ontology	Domain	TBox	ABox
DICE	Medical	Yes	No
SNOMED	Medical	Yes	No
LUBM	University	Yes	Yes
UOB	University	Yes	Yes
VICODI	History	Yes	Yes
SEMINTEC	Financial Services	Yes	Yes
FMA	Anatomy	Yes	No
NotGalen	Medical	Yes	No
NCI	Medical	Yes	No
TAMBIS	Bioinformatics	Yes	No
BIOPAX	Biological pathways	Yes	Yes

Table 1: Test Ontologies

The DICE Ontology The DICE (Diagnosis for intensive care evaluation) ontology is a medical ontology that focuses on the modeling of reasons to admission of intensive care [21]. The knowledge base consists only of a TBox. This ontology is particular interesting for reasoning tasks that are mainly investigated for unfoldable ontologies, since it is neither cyclic nor does it contain GCIs, see [20]. The DICE ontology uses the concept constructors: conjunction, existential restrictions, value restrictions, qualified number restrictions and disjointness statements for concept names. This roughly corresponds to $\mathcal{ALEQ}$. It contains 2490 concept definitions (1134 of which are primitive definitions) and about 30 roles. The DICE ontology is available in KRSS and DIG format.

The SNOMED Ontology The SNOMED ontology is a medical ontology with expressivity of EL+ (EL augmented with complex role inclusions) and consisting of 379,691 concepts and 52 roles. It contains 38,719 concept definitions, 340,972 primitive concept definitions, and 12 (complex) role inclusions. It's an acyclic TBox without GCIs. No ABox concepts or role assertions are provided. The ontology format is an extension of KRSS.

The LUBM Ontology The so-called Lehigh University Benchmark (LUBM), available under `http://swat.cse.lehigh.edu/projects/lubm/`, was developed to facilitate the evaluation and comparison of OWL semantic web repositories with different reasoning and storage capabilities. The LUBM consists of an OWL ontology for modeling universities; i.e., the ontology contains concepts for persons, students, professors, publications, courses etc. as well as appropriate relationships for such a universe of discourse. The ontology does not use disjunctions or number restrictions, but it uses existential quantifiers, so it is in Horn-$\mathcal{ALCHI}$ fragment It currently defines 43 classes and 32 properties (including 25 object properties and 7 datatype properties). A benchmark generator can be used to generate synthetic extensional data corresponding to this ontology; i.e., a set of descriptions for specific departments, professors, students, courses etc. The dataset containing OWL files for 50 universities have over 6,800,000 triples in total. Currently, LUBM is de facto standard for benchmarking reasoners over huge ontologies. Although the TBox is simple, much effort has been invested in the definition of an ABox generator that produces instance descriptions that statistically match real-world scenarios.

The UOB Ontology The University Ontology Benchmark is extended form of the Lehigh University Benchmark (LUBM). The ontology is available under `http://www.alphaworks.ibm.com/tech/semanticstk`. Similar to LUBM, it is proposed for benchmarking reasoner systems over ontologies with huge data repositories. Moreover, it extends LUBM with some additional TBox axioms in order to use the full expressivity of OWL Lite and OWL DL.

The VICODI Ontology The VICODI ontology is a manually created ontology about european history. The TBox consists of role and concept inclusion axioms, and domain and range specifications; it does not contain disjunctions, existential quantification, or number restrictions. However, the ABox is relatively large, with many interconnected instances. Currently, the ontology has around 170 concepts, 15 properties and more than 15,000 individuals. The ontology can be downloaded under `http://www.vicodi.org`. The ontology is available in OWL format.

The SEMINTEC Ontology The Semintec ontology was originally created by the Semintec project at the University of Poznan (`http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`). This ontology is based on a real-world TBox and models financial services. Like VICODI, this ontology is relatively simple. The TBox contains 222 axioms of simple structure that do not use existential quantifiers or disjunctions. However, functionality assertions requiring equality reasoning and disjointness are using.

The Foundational Model of Anatomy (FMA) Ontology The Foundational Model of Anatomy ontology (FMA) is an evolving ontology concerned with the representation of human anatomy. Its ontological framework can be applied and extended to other species.

The Foundational Model of Anatomy ontology (FMA) is now open source and available for general use (`http://sig.biostr.washington.edu/projects/fm/AboutFM.html`).

The Foundational Model of Anatomy ontology has four interrelated components:

1. The *Anatomy Taxonomy* classifies anatomical entities according to the characteristics they share and by which they can be distinguished from one another.

2. The *Anatomical Structural Abstraction* specifies the part-whole and spatial relationships that exist between the entities represented in the Anatomy Taxonomy;

3. The *Anatomical Transformation Abstraction* specifies the morphological transformation of the entities represented in the Anatomy Taxonomy and during prenatal development and the postnatal life cycle;

4. The *Metaknowledge* ontology specifies the principles, rules and definitions according to which classes and relationships in the other three components of FMA are represented.

The FMA ontology contains approximately 75,000 classes and over 120,000 terms. FMA has been created using Protege-3.0. The structure of the ontology is simple and can be represented using $\mathcal{EL}$.

The GALEN Ontology GALEN is a large medical ontology, represented in the Description Logic $\mathcal{SHIF}$, designed for supporting clinical information systems. GALEN's original goal was to describe all and only what is medically sensible. The practical goal, however, has been to design an ontology that is sufficiently expressive to capture the concepts commonly used by clinicians and classify them correctly.

The full version of GALEN contains 23139 concept names and 950 role names. The ontology does not contain concrete roles or individuals. The ontology can be downloaded from `http://www.co-ode.org/ontologies/galen`.

For testing purposes, we will consider a simplified version of GALEN which contains 2749 concept names, 413 role names and no concrete roles or individuals. This simplified version is represented in the description logic $\mathcal{SHF}$ and thus, as opposed to the full version, does not contain inverse roles. It is available online at `http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl`.

The National Cancer Institute Ontology (NCI) The NCI ontology is a huge, carefully designed, ontology dealing with the biomedical domain. NCI has become a reference terminology covering areas of basic and clinical science, built with the goal of facilitating translational research in cancer. It represents knowledge about a wide range of domains, such as diseases, drugs, anatomy, genes, gene products, techniques, and biological processes, among others, all with a cancer-centric focus in content, and originally designed to support coding activities across the National Cancer Institute.

The OWL version of the NCI ontology we are considering can be downloaded at `http://www.mindswap.org/2003/CancerOntology/`. This version is represented in the $\mathcal{EL}$ description logic; it contains 27652 concept names, 70 roles and no concrete roles or individuals. The ontology is simple in structure and contains only definitions.

The TAMBIS Ontology The TAMBIS (Transparent Access to Multiple Bioinformatics Information Sources) OWL ontology is a biological science ontology developed by the TAMBIS2 project. The goal of the project was to use an ontology to enable biologists to ask questions over multiple external databases using a common query interface. The TAMBIS ontology describes a wide range of bioinformatics tasks and resources, and has a central role within the TAMBIS system.

The version of the ontology we are considering can be downloaded from the following URI: `http://www.mindswap.org/ontologies/tambis-full.owl`.

The ontology is represented in the description logic $\mathcal{SHIN}$. It contains 395 concept names, 100 role names and no concrete roles or individuals.

The BIOPAX Ontology The goal of the BioPAX group is to develop a common exchange format for biological pathways data. This concerns exchanging chemical compound information, followed by metabolic pathway representation. Different levels of the BIOPAX ontology can be found at `http://www.biopax.org/`. The BIOPAX ontology is defined in OWL.

References

[1] F. Baader, R. Bernardi, D. Calvanese, A. Calì, B. Cuenca Grau, M. Garcia, G. De Giacomo, A. Kaplunova, O. Kutz, D. Lembo, M. Lenzerini, L. Lubyte, C. Lutz, M. Milicic, R. Möller, B. Parsia, R. Rosati, U. Sattler, B. Sertkaya, S. Tessaris, C. Thorne, and A.-Y. Turhan. D13: Techniques for Ontology Design and Maintenance. Project deliverable, TONES, 2007. http://www.tonesproject.org.

[2] F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[3] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In J.J. Alferes and J.A. Leite, editors, *Proc. of the 9th Eur. Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 400–412, Lisbon, Portugal, 2004. Springer.

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

[5] Franz Baader and Ralf Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proc. of the 22nd German Annual Conf. on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1998.

[6] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 96–101, 1999.

[7] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 34–38, 1997.

[8] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Applied Logic*, 2007. To be published.

[9] P.G. Baker, C.A. Goble, S. Bechhofer, N.W. Paton, R. Stevens, and A. Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999.

[10] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, volume 2174 of *Lecture Notes in Artificial Intelligence*, pages 396–408, Vienna, Sep 2001. Springer. OilEd download page `http://oiled.man.ac.uk`.

[11] Alexander Borgida and Ralf Küsters. What's not in a name? Initial explorations of a structural approach to integrating large concept knowledge-bases. Technical Report DCS-TR-391, Rutgers University, 1999.

[12] Alexander Borgida and Ralf Küsters. What's not in a name: Some properties of a purely structural approach to integrating large DL knowledge bases. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, pages 65–78. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-33/`, 2000.

[13] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In I. Horrocks and S. Tessaris, editors, *Proc. of the 2002 Description Logic Workshop (DL 2002)*, number 53. RWTH Aachen, April 2002. See `http://CEUR-WS.org/Vol-53/`.

[14] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2002)*, San Francisco, CA, 2002. Morgan Kaufmann Publishers.

[15] Sebastian Brandt. *Standard and Non-standard reasoning in Description Logics*. Ph.D. dissertation, Institute for Theoretical Computer Science, TU Dresden, Germany, 2006.

[16] D. Calvanese, E. Franconi, B. Glimm, B. Cuenca Grau, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, R. Möller, R. Rosati, U. Sattler, S. Tessaris, and A.-Y. Turhan. D10: Tasks for Ontology Access, Processing, and Usage. Project deliverable, TONES, 2006. http://www.tonesproject.org.

[17] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[18] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, 2006.

[19] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[20] R. Cornet and A. Abu-Hanna. Using non-primitive concept definitions for improving dl-based knowledge bases. In V. Haarslev and R. Möller, editors, *Proc. of the 2004 Description Logic Workshop (DL 2004)*, number 104, 2004. See `http://CEUR-WS.org/Vol-104/`.

[21] Ronald Cornet and Ameen Abu-Hanna. Using description logics for managing medical terminologies. In Pedro Barahona Michel Dojat, Elpida Keravnou, editor, *Artificial Intelligence in Medicine: 9th Conference on Artificial Intelligence, in Medicine in Europe (AIME 2003)*, LNCS, pages 61 – 70. Springer, 2003.

[22] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. of the 16th International World Wide Web Conference (WWW-2007)*. ACM, 2007.

[23] C. Elsenbroich, O. Kutz, and U. Sattler. A case for abductive reasoning over ontologies. In *Proc. OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11*, 2006.

[24] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On the Update of Description Logic Ontologies at the Instance Level. In *Proceedings of the 21th National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, AAAI'06*. AAAI Press, 2006.

[25] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*. AAAI Press, 2007.

[26] B. Cuenca Grau, C. Lutz, U. Sattler, and A.-Y. Turhan. D11: Tasks for Ontology Integration and Merging. Project deliverable, TONES, 2006. http://www.tonesproject.org.

[27] V. Haarslev and R. Möller. RACER System Description. In *Int. Joint Conference on Automated Reasoning*, 2001.

[28] Volker Haarslev and Ralf Möller. Incremental Query Answering for Implementing Document Retrieval Services. In *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7*, pages 85–94, 2003.

[29] Peter Haase and Ljiljana Stojanovic. Consistent Evolution of OWL Ontologies. In Asuncin Gmez-Prez and Jrme Euzenat, editors, *Proceedings of the Second European Semantic Web Conference, Heraklion, Greece, 2005*, volume 3532 of *Lecture Notes in Computer Science*, pages 182–197. Springer, may 2005.

[30] Christian Halaschek-Wiener, Bijan Parsia, and Evren Sirin. Description Logic Reasoning with Syntactic Updates. In *Proceedings of the 5th Int. Conference on Ontologies, Databases, and Applications of Semantics, ODBase'06*, 2006. Submitted.

[31] Christian Halaschek-Wiener, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Description Logic Reasoning for Dynamic ABoxes. In *Proceedings of the Int. Workshop on Description Logics, DL'06*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[32] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[33] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stefan Tobies. How to decide query containment under constraints using a description logic. In *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[34] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in OWL ontologies. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications. Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184. Springer-Verlag, 2006.

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

[36] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer-Verlag, 2004.

[37] Domenico Lembo, Carsten Lutz, and Boontawee Suntisrivaraporn. D05: Tasks for Ontology Design and Maintenance. Project deliverable, TONES, 2006. http://www.tonesproject.org.

[38] Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating Description Logic ABoxes. In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Proceedings*

of the Tenth Int. Conference on Principles of Knowledge Representation and Reasoning, KR'06, pages 46–56. AAAI Press, 2006.

[39] Deborah L. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Department of Computer Science, Rutgers University, October 1996. Also available as Rutgers Technical Report Number LCSR-TR-277.

[40] Daniel Oberle, Raphael Volz, Steffen Staab, and Boris Motik. An extensible ontology software environment. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 299–320. Springer-Verlag, 2004.

[41] Maria Magdalena Ortiz de la Fuente, Diego Calvanese, and Thomas Eiter. Data complexity of answering unions of conjunctive queries in $\mathcal{SHIQ}$. Technical report, Faculty of Computer Science, Free University of Bozen-Bolzano, March 2006. Available at `http://www.inf.unibz.it/~calvanese/papers/orti-calv-eite-TR-2006-03.pdf`.

[42] Bijan Parsia, Christian Halaschek-Wiener, and Evren Sirin. Towards Incremental Reasoning Through Updates in OWL-DL. Available as `http://www.mindswap.org/papers/2006/incclass.pdf`, 2006.

[43] Mathieu Roger, Ana Simonet, and Michel Simonet. Toward Updates in Description Logics. In *Proceedings of the Int. Workshop on Description Logics, DL'02*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2002.

[44] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies (extended abstract). In *Proc. of the 15th Belgium-Netherlands Conf. on Artificial Intelligence*, 2003.

[45] A.-Y. Turhan. *On the computation of common subsumers in Description Logics*. PhD thesis, TU Dresden, forth coming.

[46] Anni-Yasmin Turhan and Christian Kissig. SONIC—Non-standard inferences go OILED. In *Proc. of the 2nd Int. Joint Conf. on Automated Reasoning (IJCAR 2004)*, volume 3097 of *Lecture Notes in Computer Science*, pages 321–325. Springer, 2004.

[47] M. Wessel and R. Möller. A flexible dl-based architecture for deductive information systems. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proc. IJCAR-06 Workshop on Empirically Successful Computerized Reasoning (ESCoR)*, pages 92–111, 2006.

[48] Katy Wolstencroft, Andy Brass, Ian Horrocks, Phillip W. Lord, Ulrike Sattler, Daniele Turi, and Robert Stevens. A little semantic web goes a long way in biology. In *Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 786–800. Springer-Verlag, 2005.