Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl für Automatentheorie

Dresden, den 26.03.2009

Model Exploration to Support Understanding of Ontologies

Diplomarbeit zur Erlangung des akademischen Grades
Diplom-Informatiker

Diplomand:
Johannes Bauer
Geboren 19.09.1982 in Hamburg

Betreuer:
Prof. Dr. rer. nat. Carsten Lutz

Betreuender Hochschullehrer:
Prof. Dr. Ing. Franz Baader

Abstract:

Ontologies play an increasingly important role in areas such as the life and clinical sciences and the Semantic Web. Domain experts, i.e. biologists and clinicians, build and re-use (parts of) ontologies, and make use of reasoners to explicate some of the knowledge captured in an ontology. Understanding this knowledge is a highly complex task for which some tool support has recently become available, yet it has also become clear that more support is needed.

The goal of this work was to investigate whether model exploration, i.e. the interactive generation and presentation of models of an ontology, can be used to help ontology engineers understand the knowledge captured in an ontology. In order to achieve this goal, a Protégé 4 plug-in supporting generation of and interaction with models was implemented. In addition, a pilot study was carried out to evaluate its effectiveness in supporting users in understanding an ontology.

Erklärung: Ich versichere hiermit, dass ich die vorliegende Arbeit selbst verfasst und dazu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle wörtlichen oder sinngemäßen Zitate sind als solche kenntlich gemacht.

Datum Johannes Bauer

Ich danke: Den Betreuern meiner Diplomarbeit, sowie den Mitgliedern der Information Management Group an der University of Manchester, besonders Professor Ulrike Sattler und Bijan Parsia, für den Rat und die Unterstützung, die sie mir während der Bearbeitung meiner Arbeit haben zuteil werden lassen.

Contents

1 Motivation

Knowledge representation (KR) researchers continue to develop ever-more powerful description logic (DL) systems, while practical researchers deal with the challenges involved in making these systems available to users outside the DL community, enabling practitioners in such areas as the life and clinical sciences to build large logical descriptions called *ontologies* of their respective fields of knowledge (e.g. SNOMED CT[1], NCI Thesaurus[2]).

Ontologies are used in a number of applications, including as a way of modeling the domain of interest of computer programs [40]. They can then be used to generate actual program code or drive domain-independent software, two approaches offering, to different degrees, verifiability, re-usability, and stability over change both of the domain and the specific circumstances of the modeled knowledge.

At different stages of an ontology's life-cycle, the people working with it require some sort of understanding it; which parts of it encode what piece of knowledge explicitly or implicitly, how it is meant to be used on its own or in conjunction with other ontologies, and if and where it is broken and how it may be fixed are a number of examples. Tool support is necessary to achieve these kinds of understanding, as ontologies can be as broad and complicated as the fields of knowledge they model, and as it is easy to get lost in the highly complex interactions between the large numbers of logical axioms they comprise.

This need is testified and partially met by the impressive tool support which has recently become available. However, support facilitates use and use spawns demand for more support. Thus, the very increase in popularity of ontology engineering brought about by ontology editors, visualization, Justifications, etc. may be seen as one reason for the demand for even more support.

Looking at anecdotal evidence, we found that some experts of the logical background of ontology engineering liked to use pen and paper to visualize models of ontologies—concrete examples of how those abstract descriptions may be interpreted—to explain or check some of their lesser obvious implications. Think, for example, of an ontology stating that a `Nobleman` is someone who is the son of a `Nobleman`, and a `Commoner` is someone who is not a `Nobleman`. One might be surprised to see that a `Commoner` can have a son who is a `Nobleman`. However, the example depicted in Fig. 1, where that `Nobleman` is at the same time the son of both a `Commoner` and a `Nobleman`, should immediately clear up the situation and make it obvious that the ontology lacks a statement saying that no-one can be son of two men.

From this observation we drew an idea for ontology comprehension support: all the

1. http://www.ihtsdo.org/snomed-ct/
2. http://nciterms.nci.nih.gov/NCIBrowser/Dictionary.do

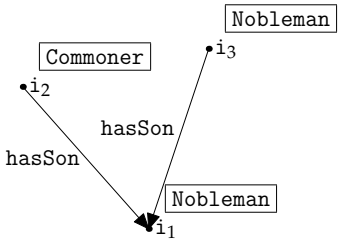


Figure 1: A Clarifying Example

existing prominent ontology inspection tools were aimed at making the raw content of an ontology more accessible by grouping statements, visualizing them, or selecting relevant statements. However, it was still up to the user to apply the logic-based semantics to these statements, which is a difficult task, not least of all because these statements can have very complex interactions sometimes leading to unforeseen and difficult to understand consequences.

We started this project in order to investigate whether a tool could be devised which did something similar to what the above-mentioned experts did: present its users with models of concepts in ontologies to help understand them.

A given ontology can have many models and they can be huge. The idea being as brand new as it was, we did not have a theory as to which parts of which models a user needed to see in order to learn what. In fact, be believed that in some cases, one would need to see a number of models to learn a particular fact about an ontology—e.g. in cases where each model exhibited part of the property being investigated. We thus extended our idea to the notion of *model exploration*: having a tool generate and visualize some model, let the user explore this model, specify changes, see whether these changes were consistent with the ontology, and if so, present a new model incorporating them.

This work is a first step in investigating the usefulness of such an approach. After giving a broad overview of the technical background, ontology usage, and related work in Sections 2, 3, and 4, respectively, it introduces model exploration in greater depth and how models may be generated in Sections 6 and 5. After that, we report in Section 7 on the development of a prototype implementing model exploration as a plug-in for the ontology editor Protégé 4, before describing, in Section 8, a pilot study that was designed and carried out to test the usefulness of that prototype, and concluding with an evaluation and the points that lend themselves for more work on the subject in Section 9.

2 Preliminaries

2.1 Ontologies

In computer science, an *ontology* is a statement of a logical theory that describes the organization and nature of the environment (also: *universe of discourse*) of a knowledge-based agent in terms of definitions of concepts [15, 16, 2].

In recent years there has been much interest in building large such ontologies, like SNOMED CT[3] and the NCI Thesaurus[4]. Amongst other applications, such explicit descriptions of the universe of discourse of an agent can drive agent software ([40]. see Section 3).

2.2 Description Logics

Although in general something does not have to be written in any particular logic to be called an ontology by the above definition, a family of logics, called description logics (DLs) have been developed precisely for knowledge representation. Historically, they succeeded other, non-logical formalisms, like Frames and Semantic Networks, and were conceived for the well-defined semantics earlier knowledge representation mechanisms lacked [36].

From the DL point of view, the organization of a world is described by the classes of and relations between objects in it: *concepts*, *roles*, and *individuals* or *instances* in DL terminology. An axiom written in a DL is a statement that may hold in some worlds and not in others. As part of a set of axioms—an ontology—it can also be seen as a characterization of some aspect of those worlds that adhere to the ontology engineer's view of the organization and nature of the domain of discourse being modeled. These worlds are called an axiom's or ontology's *models*.

A DL consists of a syntax, which specifies what axioms (and sets of axioms) are legal in that DL, and semantics which declare how these axioms are to be interpreted, i.e. what are the models of an axiom. Axioms are typically constructed from a DL-specific set of *constructors*, as well as arbitrary *concept names*, *role names* and *individual names* from the infinite sets N_C, N_R, and N_I, respectively. Finite sets of axioms defining concepts and roles are called *TBoxes* and *RBoxes*, while finite sets of axioms talking of individuals are called *ABoxes*.

Differences in syntax and semantics of DLs reflect in the things that can be expressed in them as well as in the mechanisms needed for reasoning and their complexity. What is a feasible reasoning task in one DL may be intractable in another and even undecidable in a third one. Depending on the expressivity and performance constraints of an application, there is a variety of DLs to choose from.

3. http://www.ihtsdo.org/snomed-ct/
4. http://nciterms.nci.nih.gov/NCIBrowser/Dictionary.do

Although DLs differ greatly in the constructors available in them, there exists a kernel of common constructors with a universal syntax. This syntax will, for brevity, not be introduced here. Instead, the reader is referred to [1]. Also, syntax and semantics for a simple DL, $\mathcal{ALC}$, are given in Section 5.2.1

2.2.1 Semantics—Interpretations and Models

Earlier, we said that an ontology describes the organization and nature of an agent's environment. The notion of a concrete environment, potentially one out of many sharing some such organization, is captured by the term *interpretation* in description logics. Interpretations are structures $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$, the *domain*, is a non-empty set of individuals, and $\cdot^{\mathcal{I}}$ a function mapping every concept name or concept description to a subset of the domain, called its set of *instances*, every individual name to an individual, and every role name to a binary relation over the domain.

What is a model of an axiom in a DL is typically defined inductively over its structure, starting with the models of a concept $\mathtt{C}$, which are just those interpretations which do not map $\mathtt{C}$ to the empty set. The models of a set of axioms are defined as the common models of the individual axioms. A set of axioms—like a TBox or an ABox—is called *consistent* if it has at least one model. An axiom A is called an *entailment* of an ontology $\mathcal{O}$, denoted $\mathcal{O} \models A$, if all models of $\mathcal{O}$ are also models of A.

Basic examples of reasoning tasks for DLs are *satisfiability* and *subsumption testing*. Satisfiability testing a concept with respect to an ontology means to check whether there is a model of the ontology which is also a model of that concept. Subsumption testing is the task of testing whether, for two concepts $\mathtt{C}_1$ and $\mathtt{C}_2$ and every model $\mathcal{I}$ of an ontology $\mathcal{O}$, it is true that ${\mathtt{C}_1}^{\mathcal{I}} \subseteq {\mathtt{C}_2}^{\mathcal{I}}$, formally denoted $\mathcal{O} \models \mathtt{C}_1 \sqsubseteq \mathtt{C}_2$.

Interpretations as Graphs: sometimes, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is thought of as edge- and vertex-labeled graph $G_{\mathcal{I}} = (V, E, \mathcal{L}_V, \mathcal{L}_E)$ where:

$$
\begin{aligned}
V &= \Delta^{\mathcal{I}} \\
E &= \{(\mathtt{i}, \mathtt{i}') \mid \mathtt{i}, \mathtt{i}' \in \Delta^{\mathcal{I}}, \exists \mathtt{r} \in N_R, (\mathtt{i}, \mathtt{i}') \in \mathtt{r}^{\mathcal{I}}\} \\
\mathcal{L}_V &: V \to 2^{N_C}, \mathcal{L}_V(\mathtt{i}) = \{\mathtt{A} \in N_C \mid \mathtt{i} \in \mathtt{A}^{\mathcal{I}}\} \\
\mathcal{L}_E &: E \to 2^{N_R}, \mathcal{L}_E(\mathtt{i}, \mathtt{i}') = \{\mathtt{r} \in N_R \mid (\mathtt{i}, \mathtt{i}') \in \mathtt{r}^{\mathcal{I}}\}
\end{aligned}
$$

We say that $\mathtt{i}$ is an $\mathtt{r}$-predecessor of $\mathtt{i}'$ and $\mathtt{i}'$ an $\mathtt{r}$-successor of $\mathtt{i}$, if $(\mathtt{i}, \mathtt{i}') \in E$ and $\mathtt{r} \in \mathcal{L}_E(\mathtt{i}, \mathtt{i}')$. Any node from which some individual $\mathtt{i}$ is reachable in the graph is called its *ancestor*, any node reachable from $\mathtt{i}$ is referred to as its *descendant*.

Since an interpretation $\mathcal{I}$ and the corresponding graph $G_{\mathcal{I}}$ are intertranslatable, we will use graphs to specify interpretations and speak of interpretations in graph terms. We will also use a simple graphical notation for interpretations and refer to graph diagrams to discuss interpretations. Fig. 2, for example, depicts a model of the ABox

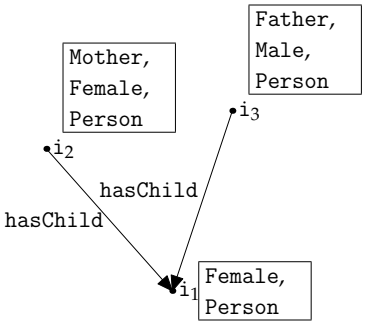


Figure 2: Interpretation as a Graph

$$\mathcal{A} = \left\{ \begin{array}{lll} \texttt{Female}(\texttt{i}_2), & \texttt{Male}(\texttt{i}_3), & \texttt{Female}(\texttt{i}_1), \\ \texttt{Person}(\texttt{i}_2), & \texttt{Person}(\texttt{i}_3), & \texttt{Person}(\texttt{i}_1), \\ \texttt{Mother}(\texttt{i}_2), & \texttt{Father}(\texttt{i}_3), & \\ \texttt{hasChild}(\texttt{i}_2, \texttt{i}_1), & \texttt{hasChild}(\texttt{i}_3, \texttt{i}_1) & \end{array} \right\}$$

2.3 OWL

Much effort has been put into the development of OWL, the "Web Ontology Language", and tool support around it [7, 19]. As its name suggests, it was designed to be a language for representation of ontologies for the Semantic Web [23]. Apart from having DL-style semantics and containing two sub-languages which are DLs in the sense that they are logics in which entailment is decidable, OWL DL and OWL Lite, OWL can be used to encode information about its domain that is not generally encoded in a DL, like meta-information about entities, called annotations. OWL DL and OWL Lite are similar to well-known DLs, $\mathcal{SHOIN}(\mathbf{D})$ ($\mathcal{SROIQ}(\mathbf{D})$ for OWL 2) and $\mathcal{SHIF}(\mathbf{D})$. The interested reader is referred to [23] for more details.

Since we built our model exploration prototype, SuperModel, as a plugin for the OWL ontology editor Protégé 4, OWL and the underlying DLs will play an important part in this work.

2.4 Justifications

Based on *minimal unsatisfiability preserving subsets (MUPS)* [45], *Justifications* [30] have been developed and made available in ontology editors such as Swoop and Protégé 4.

For an ontology $\mathcal{O}$ and an entailment A of $\mathcal{O}$, a subset $\mathcal{J} \subseteq \mathcal{O}$ is a Justification of A if $\mathcal{J} \models A$ and there is no proper subset $\mathcal{J}' \subset \mathcal{J}$ such that $\mathcal{J}' \models A$.

Justifications can be used to investigate the reasons for an unexpected and possibly unwanted entailment in an ontology. Justification generation and presentation facilities have proven a highly successful feature of ontology editing environments [18].

Justifications appear in this work as a related technique for ontology comprehension, and we use it in the user evaluation of our prototype (see Section 8).

3 Ontology Engineering & Usage

Different people may work with an ontology and it may be worked with in different ways: a group of ontology engineers may start building an ontology for their specific application. They may choose to assign the modeling of some aspects to members of their team and to import parts of existing ontologies for others. Members may join or leave the team. At some point, application developers will start building systems using it. In the process, they may find that it is only partially adequate and needs to be fixed; or the circumstances change and after a while the modeling needs to be updated.

This section deals with the ways in which ontologies are worked with, and why and where ontology engineers and users need support for their work.

3.1 Using Ontologies

There is a variety of ways to use an ontology. First of all, the well-defined semantics of the underlying DLs allow for them to standardize the meaning of a set of terms in some area of interest. There have been, e.g. in medical informatics [42], efforts to create ontology-based *controlled vocabularies*: dictionaries, if you will, defining technical terms in relation to each other in an unambiguous, logical fashion. Such a controlled vocabulary can be used to coordinate the use of technical terms in writing or in computer programs.

Ontologies can be used as an abstraction layer above data storage: especially in the area of databases, one can specify data models, i.e. legal states of databases, using an ontology, and formulate queries in the same language as the ontology.

Some of the benefits of this approach are the potential use of one language for model specification and queries, independence of the actual data source, and the existence of both a large body of theoretical knowledge and highly optimized implementations of various reasoning services for ontologies [8].

These reasoning services can be used e.g. to check the consistency of data models and queries, i.e. to test whether there can be a data base with that model and whether, given some data model, a query can ever yield any results, respectively. It can also be used to discover subsumption between data base entities or relationships, which can then be made explicit in the data model.

Puleston et al. describe in [40] a third instance of ontologies in use. They postulate a continuum from *direct* over *hybrid* to *indirect approaches* to driving software applications using ontologies. The two extremes of this continuum differ in the generality of the actual program code; in the direct approach, the ontologies are used to model the domain of interest and through this the program itself; actual, concrete code can then be generated from the ontology (and refined by a programmer). In indirect approaches, ontologies

drive domain-independent software which itself is only concerned with as general issues as for example the user interface and access to a reasoner, which in turn retrieves the actual knowledge from the ontology.

3.2 Making of Ontologies—Authoring and Editing

In the simplest case, an ontology is just put together by adding axioms until the intended knowledge is captured. This can be done by an individual or a team of *ontology engineers*. Between ontology engineers, two roles can be distinguished: *domain experts*, who contribute the knowledge, and *DL experts*, whose job it is to encode it in the ontology language of choice.

Another approach is the partially or fully automatic generation of ontologies from sources as, for example, natural or controlled language text [33, 29], or by abstraction from concrete data generated by domain experts while working in their field [9].

In order to save time and effort, one may re-use parts of already existing ontologies instead of building one from scratch [28]. Importing a third-party ontology dealing with something one would otherwise have to model anew promises all the benefits of modular re-use anywhere in software engineering: faster development, interoperability, and use of external domain knowledge, testing and ongoing development.

Obviously, after or during the development of an ontology, the knowledge, the world, and the requirements may change, in which case an ontology may have to be changed to reflect that as well. Also, it might turn out that some things that are said explicitly or implicitly in an ontology do not faithfully represent our understanding of the world. The problem then needs to be located and fixed.

3.3 Understanding Ontologies

Tasks involved in both editing and using ontologies usually require some sort of understanding it. The following examples illustrate this point and why understanding is not always easy.

What is the domain of discourse: while, of course, it is easy to extract all concrete terms used in an ontology, this does not necessarily suffice to see what exactly is its scope. The names it uses for concepts and relations are really just names and do not have to have anything to do with what information is available about them in the ontology.

Also, some concepts and relations can be very abstract in name or function, and/or need not have a counterpart in the world as we understand it (see e.g. [44]).

What does it say about X: when interested in some concept or role name, one potentially has to do quite a bit of browsing and work hard not to lose track only to find out what an ontology explicitly says about it, as it may be used in a great number of axioms.

When it comes to what the implications of those axioms are, things become even more difficult because reasoning with DL semantics can be quite challenging especially for non-experts of description logics [43]. Seemingly unrelated axioms can interact to produce unexpected effects which may be hard to comprehend.

What information is missing: a common mistake in ontology engineering is to confound one's own intuition about an ontology's content with what is actually there: given concepts `Pizza`, `Meat` and `Vegetable`, and a definition for `VegetarianPizza` as a `Pizza` which `hasTopping` only `Vegetable`, one is naturally inclined to believe that `VegetarianPizzas` cannot have a topping that is `Meat`. However, our knowledge about meat and vegetables is not available to a reasoner, and thus, the concepts `Meat` and `Vegetable` are not distinct unless explicitly defined to be [43].

How some piece of information is expressed: given a complex piece of information encoded in an ontology, it can become important to learn how that information is expressed. One example for where this is important is *debugging* [30, 45], where that information is in fact wrong information and needs to be corrected.

Another example is extending an ontology. Imagine an ontology in which it is true that any `Oak` `hasPart` some `Root`. How to best add that `CherryTrees` also have `Roots` depends on whether `Oaks` have `Roots` because they are `Trees` and `Trees` have `Roots`, or because `Oaks` are `PhotosynthesizingOrganisms` and those are either `OligoCellularOrganisms` or have `Roots`, or because it is stated explicitly that they do.

Whatever the motivation to investigate the way some piece of information is encoded, it can be hard because the axioms involved can be few, scattered all across a huge ontology, and may seem to have nothing to do with each other.

It is easy to see that the above ways of understanding are necessary for the tasks mentioned in 3.1 and 3.2. Tool support is available to facilitate (special cases of) these kinds of understanding. Section 4 lists some of the related approaches.

4 Related Work—Tool Support for Ontology Authors

As we pointed out in Section 1, a number of tools have been developed over the years to help users of ontologies understand them in the broader sense. This section discusses classes and examples of these tools to get a perspective of the landscape in which model exploration exists.

We group these classes into two categories: first, and predominantly, there are *axiom inspection tools*, whose purpose it is to make the axioms of an ontology and their logical implications more accessible, and then there are *model inspection tools*, which generate models for ontologies and one way or another present them to users.

4.1 Axiom Inspection Tools

Ontology editors: in principle, ontologies' axioms can normally be edited in any text editor. However, something like

```
<owl:Class rdf:about="http://www.semanticweb.org/ontologies/ont#C2">
  <rdfs:subClassOf
      rdf:resource="http://www.semanticweb.org/ontologies/ont#C0"/>
</owl:Class>
```

which is the OWL/RDF rendering of the axiom $\texttt{C2} \sqsubseteq \texttt{C0}$, can be quite difficult to read and edit—especially when there are thousands of these. To make this easier, ontology editors like Swoop[5] [31], Protégé 4[6], and OBOEdit[7] not only give axioms a more intuitive—in some cases even a configurable—rendering but also support their users in exploring and manipulating ontologies.

To this end, they often provide task-oriented views on ontologies, as in views showing all axioms concerning concepts, or a particular concept, or all axioms concerning roles, etc. They also typically provide access to reasoners e.g. to classify an ontology to give feedback about its consistency, show inferences like inferred concept subsumptions, and other things.

Finally, ontology editors tend to be environments in which some of the below techniques are integrated one way or another.

Visualizations: sometimes, a visual representation is more suitable than a textual or mathematical one for displaying information. To display the information encoded in ontologies, a number of visualization tools have been developed, which where surveyed by Katifori et al. [32].

The tools treated in [32] differed considerably in the approaches they followed. The authors found six general kinds of visualizations, and note that different approaches have different inherent strengths and drawbacks and some support users in some tasks better than others.

Others: ontology editors and visualizations are the most prominent families of axiom inspection tools. There exist, however, a host of others, like Justifications (see Section 2.4), query tools, ontology metrics analyzers, etc.

5. http://code.google.com/p/swoop/
6. http://www.co-ode.org/downloads/protege-x/
7. http://oboedit.org/

4.2 Model Inspection Tools

Most prominent tools for ontology comprehension belong to the family of axiom inspection tools. Less can be found on model inspection, which is the family our work belongs to. In the following we discuss what we have found on the topic.

An effort to generate and visualize models for OWL ontologies is described in [6], introducing their *music score notation*. This notation is in fact original, but it has the drawback of not being very easy on the screen size it uses for visualization, as the authors note. Also, the authors do not give any account on why they believe their notation is intuitive or of any experimental evidence they have gathered to that effect. Finally, the music score notation only applies to finite models. While it should be possible to extend it to show blocked nodes (see Section 5.2.4), this would probably clutter the display even more.

Tweezers [51] also does something fairly similar to what this work aims for, although with a different intent: in addition to gathering statistics about the reasoning process involved in satisfiability-testing each concept in a given ontology (with the reasoner Pellet), it stores and visualizes the generated completion graphs. Tweezers was developed as a tool for finding those parts of an ontology which make it costly or impossible to reason with, another, very specific way of understanding something about an ontology.

Garcia et al. [14] propose a translation of ontologies from description logics ($\mathcal{ALC}$ and parts of $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$ in their paper) into the Alloy Specification Language and to have models generated and presented by the Alloy Analyzer. Although they do suggest that these models may help explain non-subsumption and finding of non-intended models, they focus on the translation and the performance of the Alloy Analyzer, and do not report any evidence of usefulness for ontology engineers.

For the apparent relevance of its title, Náufel's and Martins' work *"Visualization of Description Logic Models"* [38] needs mentioning here. However, they do not, in fact, visualize any particular models in the sense of our work but so-called *model outlines*, which are visualizations of concept descriptions that the authors hope are more intuitive than regular, textual ones. The follow-up publication [37] does not bear the visualization of models in the title.

None of the above-mentioned approaches allows users to add constraints on the models as described in Section 6.2 or shows anything similar to the status of information we discuss in Section 6.3. Only Tweezers supports infinite models by marking blocked nodes (although without pointing out the blockers).

5 Model Generation

For a model to be explored, it must be generated, which is not exactly a trivial task. However, since model generation is the basis of tableau reasoning in DLs, there is a large body of work dealing with exactly this problem. The more expressive the logic, the more

intricate the model generation algorithms tend to be and the greater the opportunity for all kinds of pitfalls. Reason enough to look into the work done and, in fact, to re-use other people's work—see what existing programs do anyway and how we can use it.

5.1 Model Generation in General

Standard reasoning tasks for DLs are usually handled by reduction to concept satisfiability testing, ie. by proving the existence or non-existence of a model for a concept description—often by trying to generate one, or something similar that proves the existence of a model [4]. It is well-known that, for full FOL formulae, satisfiability testing is undecidable, and thus there can be no complete model generation algorithm for FOL. Most DLs are sub-logics of FOL in which satisfiability is decidable. And indeed, in contrast to the early stages of DL research when decision problems were handled using structural subsumption algorithms [36], recent DL reasoners mostly employ model generation algorithms [50].

Model generation algorithms differ in the kinds of models they generate; in particular, some algorithms produce only finite models and some may produce (finite structures representing) infinite models. As there are cases where the domain modeled by an ontology is inherently finite, it is conceivable that it can be desirable to explore the finite models of an ontology.

Unfortunately, not all DLs possess the finite model property, meaning not every concept which has a model also has a *finite* model. In particular $\mathcal{SHIF}$, $\mathcal{SHOIN}$, and $\mathcal{SROIQ}$, the bases of OWL Lite, OWL DL, and OWL 2 DL, do not have this property as they allow for functionality, inverse roles, and TBox axioms [7, 11].

That said, there are algorithms for certain DLs which can check for finite satisfiability [11]. There are also finite model generation algorithms for general FOL formulae as implemented in e.g. Mace4 [34], but those are necessarily incomplete.

Algorithms for the generation of potentially infinite models are the most general in the sense that they are applicable to the greatest number of DLs. Typically, they are tableau-based and create forest-shaped models (cf. Section 5.2.2).

Since, so far, we are only investigating whether and where model exploration is useful at all, we do not consider any form of generation of finite or non-forest-shaped models in this work. Instead, from now on, we focus on tableau algorithms, complete versions of which exist for a wide range of DLs including $\mathcal{SROIQ}$, and thus OWL 2 DL.

5.2 Tableau Algorithms

The algorithms employed in most DL systems used today are derived from tableau algorithms based on the algorithm first introduced in [46, 50]. This section will give, in Section 5.2.1, an introduction to an algorithm for a comparatively simple DL, $\mathcal{ALC}$, and

then go into a few technicalities like algorithm modifications for advanced DL constructors and optimizations.

As, in the implementation of this work, we used FaCT++, a tableau-based reasoner, these are important to understand the nature of models that are presented for exploration, the information made available to the user, and the data structures that were there to use.

5.2.1 Basic Mechanism—$\mathcal{ALC}$[8]

Although the tableau algorithm for $\mathcal{ALC}$ satisfiability checking can be found elsewhere in the literature, it is given here in order to later be able to explain the changes needed for more expressive DLs in a uniform way.

$\mathcal{ALC}$ is a simple DL whose axioms are built from concept, role, and individual names, the boolean constructors $\sqcup, \sqcap, \neg$, Top ($\top$), Bottom ($\bot$), as well as constructors for universal ($\forall$) and existential ($\exists$) quantification over roles.

For concept names $\mathtt{C}$ and role names $\mathtt{r}$, the grammar for the set $\mathcal{C}_{\mathcal{ALC}}$ of $\mathcal{ALC}$ *concept descriptions* is given by:

$$D ::= \mathtt{C} \mid \top \mid \bot \mid D \sqcup D \mid D \sqcap D \mid \neg D \mid \forall \mathtt{r}.D \mid \exists \mathtt{r}.D$$

Then, for concept descriptions $\mathtt{D}, \mathtt{D}_1, \mathtt{D}_2$, a role name $\mathtt{r}$, and individual names $\mathtt{i}, \mathtt{i}'$, TBox axioms have the form $\mathtt{D}_1 \sqsubseteq \mathtt{D}_2$, whereas $\mathtt{D}(\mathtt{i})$ and $\mathtt{r}(\mathtt{i}, \mathtt{i}')$ are ABox axioms.

For convenience and without loss of generality, we assume that all concept descriptions are in negation normal form (NNF), i.e. negation occurrs only before concept names.[9] Also, for now, we restrict ourselves to *acyclic* TBoxes: a TBox is acyclic if the left-hand sides of all TBox axioms are just concept names and an ordering $\leq$ over the concept names can be found such that, for every axiom $\mathtt{A} \sqsubseteq \mathtt{D}$, for no concept name $\mathtt{D}'$ occurring in $\mathtt{D}$ it is true that $\mathtt{D}' \leq \mathtt{A}$.

A last minor restriction is that no concept name occur on the left-hand side of two axioms in the TBox. Every TBox $\mathcal{T}$ can be turned into an equivalent one satisfying this restriction by replacing every pair of axioms $\mathtt{C} \sqsubseteq \mathtt{D}, \mathtt{C} \sqsubseteq \mathtt{D}'$ in $\mathcal{T}$ by $\mathtt{C} \sqsubseteq \mathtt{D} \sqcap \mathtt{D}'$.

The semantics of $\mathcal{ALC}$ axioms are given by the extension of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for concept descriptions in Fig. 3, and the definition of an axiom's models: $\mathcal{I}$ is a model of $\mathtt{D}_1 \sqsubseteq \mathtt{D}_2$ if ${\mathtt{D}_1}^{\mathcal{I}} \subseteq {\mathtt{D}_2}^{\mathcal{I}}$, of $\mathtt{D}(\mathtt{i})$ if $\mathtt{i}^{\mathcal{I}} \in \mathtt{D}^{\mathcal{I}}$, and of $\mathtt{r}(\mathtt{i}, \mathtt{i}')$ if $(\mathtt{i}^{\mathcal{I}}, \mathtt{i}'^{\mathcal{I}}) \in \mathtt{r}^{\mathcal{I}}$ for concept names $\mathtt{D}$, $\mathtt{D}_1$, $\mathtt{D}_2$, individual names $\mathtt{i}$, $\mathtt{i}'$, and a role name $\mathtt{r}$.

The standard $\mathcal{ALC}$ tableau algorithm checks consistency of ABoxes. It does so by starting with a set $\mathcal{M}$ containing a single ABox $\mathcal{A}_0$, and repeatedly applying the expansion rules given in Fig. 4, which replace ABoxes in $\mathcal{M}$ by one or two new ABoxes. If an ABox contains an *obvious contradiction* consisting of two axioms $\mathtt{D}(\mathtt{i})$ and $\neg\mathtt{D}(\mathtt{i})$ for some concept description $\mathtt{D}$, this is said to be a *clash* and the ABox is called *closed*, otherwise *open*. If none of the rules apply to it, it is called *complete*.

8. This section largely based on [5]
9. Every concept description can be transformed into an equivalent concept description in NNF.

concept description $\mathtt{D}$	**interpretation** $\mathtt{D}^{\mathcal{I}}$
$\mathtt{C}$	$\mathtt{C}^{\mathcal{I}}$
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$\mathtt{D}_1 \sqcup \mathtt{D}_2$	${\mathtt{D}_1}^{\mathcal{I}} \cup {\mathtt{D}_2}^{\mathcal{I}}$
$\mathtt{D}_1 \sqcap \mathtt{D}_2$	${\mathtt{D}_1}^{\mathcal{I}} \cap {\mathtt{D}_2}^{\mathcal{I}}$
$\neg\mathtt{D}_1$	$\Delta^{\mathcal{I}} \setminus {\mathtt{D}_1}^{\mathcal{I}}$
$\forall \mathtt{r}.\mathtt{D}_1$	$\{\mathtt{i} \in \Delta^{\mathcal{I}} \mid (\mathtt{i},\mathtt{i}') \in \mathtt{r}^{\mathcal{I}} \Rightarrow \mathtt{i}' \in {\mathtt{D}_1}^{\mathcal{I}}\}$
$\exists \mathtt{r}.\mathtt{D}_1$	$\{\mathtt{i} \in \Delta^{\mathcal{I}} \mid \exists \mathtt{i}' \in {\mathtt{D}_1}^{\mathcal{I}} : (\mathtt{i},\mathtt{i}') \in \mathtt{r}^{\mathcal{I}}\}$
$\mathtt{D}_1, \mathtt{D}_2$ concept descriptions, $\mathtt{C}$ a concept name, $\mathtt{r}$ a role name	

Figure 3: Interpretation of $\mathcal{ALC}$ concept descriptions

$\rightarrow_{\sqcap}$ -rule :

Condition : $(\mathtt{D}_1 \sqcap \mathtt{D}_2)(\mathtt{i}) \in \mathcal{A} \wedge \{\mathtt{D}_1(\mathtt{i}), \mathtt{D}_2(\mathtt{i})\} \not\subseteq \mathcal{A}$

Action : $\mathcal{A}' = \mathcal{A} \cup \{\mathtt{D}_1(\mathtt{i}), \mathtt{D}_2(\mathtt{i})\}$

$\rightarrow_{\sqcup}$ -rule :

Condition : $(\mathtt{D}_1 \sqcup \mathtt{D}_2)(\mathtt{i}) \in \mathcal{A} \wedge \mathcal{A} \cap \{\mathtt{D}_1(\mathtt{i}), \mathtt{D}_2(\mathtt{i})\} = \emptyset$

Action : $\mathcal{A}' = \mathcal{A} \cup \{\mathtt{D}_1(\mathtt{i})\},\ \mathcal{A}'' = \mathcal{A} \cup \{\mathtt{D}_2(\mathtt{i})\}$

$\rightarrow_{\forall}$ -rule :

Condition : $\exists \mathtt{i}' : \{(\forall \mathtt{r}.\mathtt{D})(\mathtt{i}), \mathtt{r}(\mathtt{i},\mathtt{i}')\} \subseteq \mathcal{A} \wedge \mathtt{D}(\mathtt{i}') \notin \mathcal{A}$

Action : $\mathcal{A}' = \mathcal{A} \cup \{\mathtt{D}(\mathtt{i}')\}$

$\rightarrow_{\exists}$ -rule :

Condition : $(\exists \mathtt{r}.\mathtt{D})(\mathtt{i}) \in \mathcal{A} \wedge \forall \mathtt{i}' : \mathtt{r}(\mathtt{i},\mathtt{i}') \in \mathcal{A} \rightarrow \mathtt{D}(\mathtt{i}') \notin \mathcal{A}$

Action : $\mathcal{A}' = \mathcal{A} \cup \{\mathtt{r}(\mathtt{i},\mathtt{i}_f), \mathtt{D}(\mathtt{i}_f)\}$

$\mathtt{D}, \mathtt{D}_1, \mathtt{D}_2$ concept descriptions, $\mathtt{i}, \mathtt{i}'$ individuals, $\mathtt{i}_f$ a fresh individual

Figure 4: $\mathcal{ALC}$ expansion rules

Let $\mathcal{A}_c$ be an open and complete ABox obtained this way from some $\mathcal{A}_0$. Then, the *canonical model* $\mathcal{I}_c$ of $\mathcal{A}_c$ is:

$$\begin{aligned}
&\forall \mathtt{i} \in N_I : \mathtt{i}^{\mathcal{I}_c} = \mathtt{i} \\
&\forall \mathtt{A} \in N_C : \mathtt{A}^{\mathcal{I}_c} = \{\mathtt{i} \mid \mathtt{A}(\mathtt{i}) \in \mathcal{A}_c\} \\
&\forall \mathtt{r} \in N_R : \mathtt{r}^{\mathcal{I}_c} = \{(\mathtt{i}, \mathtt{i}') \mid \mathtt{r}(\mathtt{i}, \mathtt{i}') \in \mathcal{A}_c\}
\end{aligned}$$

It is not hard to see that $\mathcal{I}_c$ is a finite model for $\mathcal{A}_0$. If, on the other hand, at some point there is no open ABox left in $\mathcal{M}$, $\mathcal{A}_0$ is inconsistent.

A concept $\mathtt{C}$ can be checked for satisfiability by applying the above algorithm to the ABox $\mathcal{A}_\mathtt{C} = \{\mathtt{C}(\mathtt{i})\}$. Since $\mathcal{ALC}$ allows for negation and conjunction, subsumption checking between two concepts $\mathtt{C}_1$ and $\mathtt{C}_2$ can be reduced to satisfiability checking of the concept $\mathtt{C}_1 \sqcap \neg \mathtt{C}_2$. Reasoning with respect to a TBox can be reduced to ABox reasoning without a TBox: concept definitions in the ABox may be *expanded* by iteratively replacing every concept name occurring in the ABox and on the left-hand side of some axiom in the TBox by the corresponding right-hand side.

Proofs for termination, completion, and correctness can be found in [5].

For simplicity, the $\mathcal{ALC}$ tableau algorithm was presented here as an algorithm working on sets of ABoxes. Tableau algorithms for more complex logics, however, need more general data structures that are capable of storing information which cannot be expressed with ABox axioms.

These more general data structures are called *completion graphs* and their particular make-up depends on the algorithms for which they are defined. For $\mathcal{ALC}$, completion graphs can be defined such that there is a one-to-one correspondence between an ABox and an $\mathcal{ALC}$ completion graph:

Let $\mathcal{A}$ be an $\mathcal{ALC}$ ABox. Then, the directed, edge- and vertex-labeled graph

$$G_{\mathcal{ALC}} = (V, E, \mathcal{L}_V, \mathcal{L}_E)$$

is the corresponding completion graph, if V is the set of individual names occurring in $\mathcal{A}$ and

$$\begin{aligned}
E &= \{(\mathtt{i}, \mathtt{i}') \mid \mathtt{i}, \mathtt{i}' \in V, \exists \mathtt{r} \in N_R : \mathtt{r}(\mathtt{i}, \mathtt{i}') \in \mathcal{A}\} \\
\mathcal{L}_V &: V \to 2^{\mathcal{C}_{\mathcal{ALC}}}, \mathcal{L}_V(\mathtt{i}) = \{\mathtt{C} \in \mathcal{C}_{\mathcal{ALC}} \mid \mathtt{C}(\mathtt{i}) \in \mathcal{A}\} \\
\mathcal{L}_E &: E \to 2^{N_R}, \mathcal{L}_R(\mathtt{i}, \mathtt{i}') = \{\mathtt{r} \in N_R \mid \mathtt{r}(\mathtt{i}, \mathtt{i}') \in \mathcal{A}\}
\end{aligned}$$

There is a trivial translation from an $\mathcal{ALC}$ completion graph back to an $\mathcal{ALC}$ ABox and thus a formulation of the $\mathcal{ALC}$ tableau algorithm as an algorithm working on completion graphs is possible, although not quite as convenient.

In particular, a completion graph is open, closed, or complete if the corresponding ABox is, and its canonical model is the canonical model of the corresponding ABox.

5.2.2 Form of Models

Let us have a look at the shape of the models generated by the algorithm presented in Section 5.2.1. Note that the one rule in Fig. 4 adding role axioms to the ABox and thus edges to the completion graph, the $\rightarrow_\exists$-rule, only ever adds edges between an existing and a fresh node to the completion graph. Thus, no cycle involving individuals that were not present from the beginning can exist in an interpretation generated by the algorithm. In particular, interpretations are always tree-shaped if the original completion graph contains only one node to start with. An important special case of this is concept satisfiability checking without an ABox.

When starting the algorithm with a fuller completion graph, the generated interpretations may or may not be tree-shaped; However, it still follows from the above argument that there are no cycles involving new individuals; all cycles in the final interpretation are already present in the initial completion graph. The models generated by this algorithm are said to be *forest-shaped*, meaning that they contain only finitely many cycles. Of course, this is trivial for an algorithm as the one presented above, which generates only finite models. However, as will be discussed in Section 5.2.4, this property are retained and crucial in extensions of above algorithm which may generate (completion graphs describing) infinite models.

5.2.3 Complex Roles

In what follows, a *role* can be either a role name or the *inverse* of a role: in those DLs that allow for inverses, an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ interprets the inverse $\mathtt{R}^-$ of the role $\mathtt{R}$ as the inverse of the relation $\mathtt{R}^\mathcal{I}$:

$$\mathtt{R}^{-\mathcal{I}} = \{(\mathtt{i}', \mathtt{i}) \mid (\mathtt{i}, \mathtt{i}') \in \mathtt{R}^\mathcal{I}\}$$

This is not very important for model exploration, but it is needed to correctly represent the following:

On top of allowing to say things about concepts and about individuals, some expressive description logics include axiom constructors that allow statements about roles.

An important thing one can say about a role is that it is transitive, written $\mathit{Trans}(\mathtt{R})$. A model of an ontology including this axiom must interpret $\mathtt{R}$ as a transitive relation over the domain. The trouble for model exploration is that tableau algorithms usually treat transitivity rather indirectly. Instead of computing all role relationships implied by transitivity in completion graphs, they model their effects. Suppose an ontology contains the following two axioms:

$$\mathit{Trans}(\mathtt{r}),$$
$$\mathtt{C} \sqsubseteq \forall \mathtt{r}.\mathtt{D}$$

Then, it must be true that every $\mathtt{r}$-successor $\mathtt{i}'$ of an individual $\mathtt{i}$ which is a $\mathtt{C}$ must be a $\mathtt{D}$. And so must every $\mathtt{r}$-successor of $\mathtt{i}'$ and so on. This is handled in tableau algorithms for

DLs like $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ by the $\forall_+$-rule [24, 26]. For a transitive role $\mathtt{R}$, a concept $\mathtt{C}$ and a node $\mathtt{i}$ in the completion graph whose label contains the concept $\forall \mathtt{R}.\mathtt{C}$, the $\forall_+$-rule adds $\forall \mathtt{R}.\mathtt{C}$ to the label of every one of $\mathtt{i}$'s $\mathtt{R}$-successors.

Role inclusion axioms are another type of axioms allowing to constrain the interpretation of roles. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model for the role inclusion axiom

$$\mathtt{R} \sqsubseteq \mathtt{S}$$

if $\mathtt{R}^{\mathcal{I}} \subseteq \mathtt{S}^{\mathcal{I}}$ for roles $\mathtt{R}$ and $\mathtt{S}$. For roles $\mathtt{R}_1, \ldots, \mathtt{R}_n$, it is a model of the axiom

$$\mathtt{R}_1 \circ \cdots \circ \mathtt{R}_n \sqsubseteq \mathtt{S}$$

if, for individual names $\mathtt{i}_1, \ldots, \mathtt{i}_{n+1}$, with $(\mathtt{i}_k{}^{\mathcal{I}}, \mathtt{i}_{k+1}{}^{\mathcal{I}}) \in \mathtt{R}_k{}^{\mathcal{I}}$, for $1 \leq k < n$, it is true that also $(\mathtt{i}_1{}^{\mathcal{I}}, \mathtt{i}_{n+1}{}^{\mathcal{I}}) \in \mathtt{S}^{\mathcal{I}}$. Note that transitivity can be expressed by rule inclusion axioms: for a role $\mathtt{R}$, $\mathit{Trans}(\mathtt{R})$ is equivalent to $\mathtt{R} \circ \mathtt{R} \sqsubseteq \mathtt{R}$.

In DLs such as $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ which allow for the latter type of axiom, subject to some restrictions, they are handled using non-deterministic, finite automata (NFA) accepting words over the set of roles [22, 25]. Their construction is induced by the role box $\mathcal{R}$ such that, for roles $\mathtt{R}, \mathtt{R}_1, \mathtt{R}_2, \ldots, \mathtt{R}_n$, the automaton $\mathcal{B}_{\mathtt{R}}$ accepts $w = \mathtt{R}_1\mathtt{R}_2 \ldots \mathtt{R}_n$ if, in every model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{R}$, for individual names $\mathtt{i}_1, \mathtt{i}_2, \ldots, \mathtt{i}_{n+1}$, it is true that $(\mathtt{i}_1, \mathtt{i}_{n+1}) \in \mathtt{R}^{\mathcal{I}}$ if $(\mathtt{i}_k{}^{\mathcal{I}}, \mathtt{i}_{k+1}{}^{\mathcal{I}}) \in \mathtt{R}_k{}^{\mathcal{I}}$, for $1 \leq k \leq n$.

The precise way these automata are built is left out here, as it is not important, at this stage, for model exploration.

What is important is that these automata are not used to add edges to the completion graph. Instead, they again simulate the effect by propagating universal role restrictions: in $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ completion graphs, nodes may be labeled with expressions like $\forall \mathcal{B}_{\mathtt{R}}(q).\mathtt{C}$, where $\mathcal{B}_{\mathtt{R}}$ is an NFA for the role $\mathtt{R}$, q is a state of $\mathcal{B}_{\mathtt{R}}$ and $\mathtt{C}$ is a concept description.

There is a expansion rule, the $\forall_1$-rule, in the tableau algorithms for these logics which adds $\forall \mathcal{B}_{\mathtt{R}}(s)$ to the label of a node $\mathtt{i}$ in the graph if $\mathtt{i}$ is labeled $\forall \mathtt{R}.\mathtt{C}$, where s is the initial state of $\mathcal{B}_{\mathtt{R}}$ $\mathtt{R}$ is a role and $\mathtt{C}$ is a concept description.

Another rule, the $\forall_2$-rule, propagates the automata. If, for roles $\mathtt{R}$ and $\mathtt{P}$, the label of some node $\mathtt{i}$ in a completion graph includes $\mathcal{B}_{\mathtt{R}}(p)$ and $\mathcal{B}_{\mathtt{R}}$ can go from p to q via $\mathtt{P}$, then this rule ensures that all $\mathtt{P}$-successors of $\mathtt{i}$ are labeled with $\mathcal{B}_{\mathtt{R}}(q)$.

Lastly, yet another rule, the $\forall_3$-rule, realizes the effect of universal role restrictions by labeling a node $\mathtt{i}$ with a concept $\mathtt{C}$ if $\forall \mathcal{B}_{\mathtt{R}}(f).\mathtt{C}$ already is in the label of $\mathtt{i}$ and f is an accepting state of $\mathcal{B}_{\mathtt{R}}$.

Fig. 5 illustrates this: the top part shows a role automaton for role $\mathtt{r}$ with respect to the RBox

$$\mathcal{R} = \left\{ \begin{array}{l} \mathtt{r}_1 \circ \mathtt{r}_2 \sqsubseteq \mathtt{r}, \\ \mathit{Trans}(\mathtt{r}) \end{array} \right\}$$

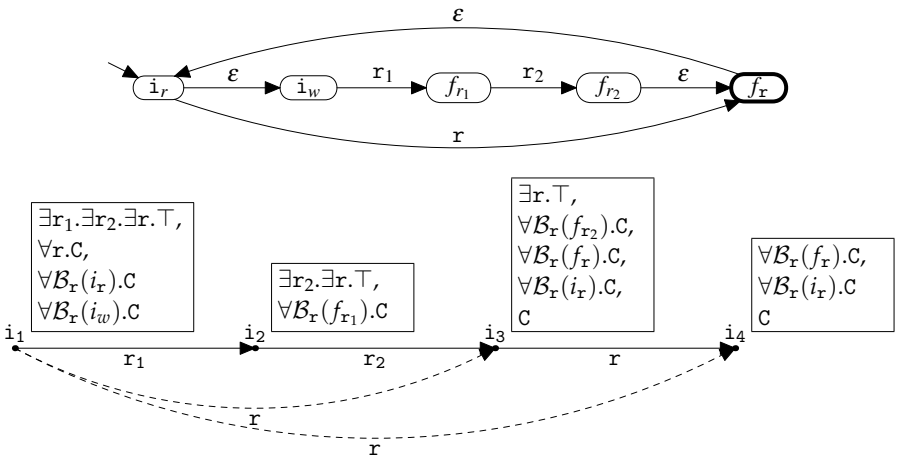


Figure 5: Role Automata

The bottom part of Fig. 5 shows a completion graph for the concept

$$\exists \mathtt{r}_1.\exists \mathtt{r}_2.\exists \mathtt{r}.\top \sqcap \forall \mathtt{r}.\mathtt{C}$$

The solid arrows stand for edges which are actually in the graph, the dashed ones show the relations that are simulated by the automata mechanism. Note that exactly those nodes have C in the label that would be r-successors of $\mathtt{i}_r$ were those edges added to the graph.

Consequences for Model Exploration: in certain expressive DLs, not all role relationships in a model must have an edge as a counter-part in the corresponding completion graph.

5.2.4 Blocking

The restriction to acyclic TBoxes from Section 5.2.1 can be dropped. However, to keep termination of the algorithm, it needs some modification. Consider for example the TBox $\mathcal{T} = \{\mathtt{C} \sqsubseteq \exists \mathtt{r}.\mathtt{C}\}$.

First of all, one obviously runs into an endless loop if one tries to expand $\mathtt{C} \sqsubseteq \exists \mathtt{r}.\mathtt{C}$. This can be handled by expanding lazily, but then the ABox consistency algorithm from Section 5.2.1 will not terminate on the input $\mathcal{M} = \{\mathcal{A}_0 = \{\mathtt{C}(\mathtt{i})\}\}$, as it will keep creating new individuals, making them instances of C and creating r-successors again (see Fig. 6a).

Of course there are models for $\mathcal{A}_0$: an instance of C connected to itself via r would be one (see Fig. 6b). While this works for $\mathcal{ALC}$, other examples in more expressive logics, like the FirstGuard example from [10], rule out such cycles.

Tableau algorithms employ *blocking* as a means to ensure termination in cases like this. The application of *generating rules*, i.e. rules generating role successors, to an individual

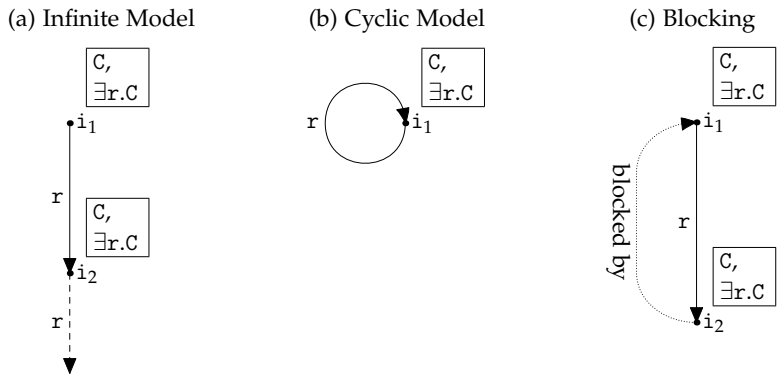


Figure 6: Reasoning with Cyclic TBoxes

$\mathtt{i}$ may be blocked by one of its ancestors $\mathtt{i}'$, if $\mathtt{i}$'s label is contained in the one of $\mathtt{i}'$ (see Fig. 6c). This condition detects cases where the fragment of any model reachable from $\mathtt{i}'$ could be copied to $\mathtt{i}$ to yield a model again. Blocking thus prevents the algorithm from computing the same substructure over and over again.

An algorithm employing the blocking rule builds a finite, graph-shaped structure that, in fact, represents a possibly infinite, forest-shaped model: blocking marks those nodes in the graph where appending the same tree-shaped substructure infinitely often would yield a model.

To make blocking work, a few more refinements to the algorithm are needed. Also, more sophisticated blocking techniques, like *pairwise (anywhere) blocking* [27, 35], exist for even more expressive DLs like $\mathcal{SHIQ}$, but this much suffices to give the basic idea and for the discussion of the data structures found in a tableau reasoner with regard to blocking.

Consequences for Model Exploration: the models generated by tableau algorithms are not necessarily finite—even in cases where finite models exist. Instead, finite completion graphs are created, which stand for potentially infinite forest-like models. In these completion graphs, starting points of infinite tree-like subgraphs, blocked nodes, are identified.

5.2.5 Optimizations

Caching: like blocking, caching tries to prevent multiple computations of isomorphic subgraphs of ABoxes. The goal, however, is different: while blocking is needed to ensure termination, caching tries to save on computation time.

Say, at some point during the run of a tableau algorithm, a clash-free completion graph is computed, in which the label of some node $\mathtt{i}$ is $\mathcal{L}_V(\mathtt{i}) = \{\mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_n\}$. If no

expansion rule is applicable to $\mathtt{i}$ or any of its descendants, the subgraph reachable from $\mathtt{i}$ can be seen as an open and complete completion graph proving the satisfiability of $\mathtt{C}_1 \sqcap \mathtt{C}_2 \sqcap \cdots \sqcap \mathtt{C}_n$.

It is easy to see that, if later a leaf node $\mathtt{i}'$ is generated whose label is $\mathcal{L}_V(\mathtt{i}') = \{\mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_m\}$ where $m \leq n$, then the expense of applying any rules to $\mathtt{i}'$ can be spared as it is clear that they will not lead to a clash, because

$$\mathtt{C}_1 \sqcap \mathtt{C}_2 \sqcap \cdots \sqcap \mathtt{C}_m \sqcap \cdots \sqcap \mathtt{C}_n \sqsubseteq \mathtt{C}_1 \sqcap \mathtt{C}_2 \sqcap \cdots \sqcap \mathtt{C}_m$$

and thus $\mathtt{C}_1 \sqcap \mathtt{C}_2 \sqcap \cdots \sqcap \mathtt{C}_m$ is satisfiable if $\mathtt{C}_1 \sqcap \mathtt{C}_2 \sqcap \cdots \sqcap \mathtt{C}_n$ is [50].

In blocking, the expense of generating a model for $\mathtt{i}$ can be avoided only if $\mathtt{i}$ has an ancestor $\mathtt{i}'$ whose label includes $\mathtt{i}$'s. Cached satisfiability results can be used in more than one run of the tableau algorithm on the same knowledge base; if e.g., first, satisfiability of $\mathtt{C}_1$ is tested and then of $\mathtt{C}_2$, then the second run may profit from results cashed in the earlier satisfiability test.

Consequences for Model Exploration: unless caching is disabled in a reasoner, the completion graphs retrieved from it may be truncated where cached results were used.

Preprocessing: For two concepts $\mathtt{C}$ and $\mathtt{C}'$, the axiom $A = (\mathtt{C} \equiv \mathtt{C}')$ can be used as an abbreviation for the axioms $A' = (\mathtt{C} \sqsubseteq \mathtt{C}')$ and $A'' = (\mathtt{C}' \sqsubseteq \mathtt{C})$: an interpretation $\mathcal{I}$ is a model of A if and only if it is a model of the axioms A' and A''.

Let $\mathtt{C}_1$ and $\mathtt{C}_2$ be concept names and $\mathtt{C}_1 \equiv \mathtt{C}_2$ an axiom in a TBox. $\mathtt{C}_1$ and $\mathtt{C}_2$ are then said to be *synonyms*[10]. If a completion graph contains a node $\mathtt{i}$ labeled both with $\mathtt{C}_1$ and $(\neg\mathtt{C}_2)$, then that will obviously lead to a clash—but only after the algorithm adds the concept $\mathtt{C}_2$ to $\mathtt{i}$'s label, which may be after an arbitrary number of costly computation steps.

That is why practical reasoners choose, from every set of synonyms, one representative and replace all others in the input by that representative [50].

Closely related to pairs of synonyms are *told cycles* within a TBox: sets of axioms

$$\begin{aligned}
\mathtt{C}_1 &\bowtie \mathtt{C}_2 \sqcap \mathtt{D}_1 \\
\mathtt{C}_2 &\bowtie \mathtt{C}_3 \sqcap \mathtt{D}_2 \\
&\vdots \\
\mathtt{C}_{n-1} &\bowtie \mathtt{C}_n \sqcap \mathtt{D}_{n-1} \\
\mathtt{C}_n &\bowtie \mathtt{C}_1 \sqcap \mathtt{D}_n
\end{aligned}$$

where $\mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_n$ are concept names, $D_k, 1 \leq k \leq n$ are concept descriptions and $\bowtie$ either $\equiv$ or $\sqsubseteq$. These cycles can be rewritten equivalently to axioms $\mathtt{C}_1 \equiv \mathtt{C}_k$, for $1 < k \leq n$,

10. And, of course, synonymity is symmetric and transitive.

and $\mathtt{C}_1 \sqsubseteq \mathtt{D}_1 \sqcap \cdots \sqcap \mathtt{D}_n$. After that, all $\mathtt{C}_k, 1 \leq k \leq n$, are synonyms and the reasoner may replace them by $\mathtt{C}_1$ as above [50].

Consequences for Model Exploration: labels of nodes in a completion graph do not always contain all concepts the individuals in the corresponding model must be instances of. Model exploration must complete individual's labels with all concepts synonymous to the ones found in the completion graph.

Dependency Sets: the expansion rules of a tableau algorithm can be divided into *deterministic* and *non-deterministic rules*. A deterministic rule is one whose effect preserves the consistency of a completion graph: after its application, the completion graph can be completed to an open and complete completion graph if and only if it could before the rule's application.

The application of a non-deterministic rule, in turn, presents a guess of the reasoner as to how the completion graph may be completed. Thus, whether or not a sequence of applications of rules to a completion graph during the satisfiability check of a concept $\mathtt{C}$ leads to a clash depends only on the choice of non-deterministic rules and on whether $\mathtt{C}$ is satisfiable.

From the $\mathcal{ALC}$ expansion rules in Fig. 4, the only example of a non-deterministic rule is the $\sqcup$-rule; all others are deterministic.

In general, there is no way to predict which choice of applications of non-deterministic rules is correct, and backtracking is needed. However, there is a way to avoid tracking back over all non-deterministic choices that are unrelated to some clash that is found. This technique, called *backjumping*, works by numbering the non-deterministic decisions made during a run of a reasoner, and storing, with every concept $\mathtt{C}$ in a label of a node $\mathtt{i}$ and role $\mathtt{r}$ in the label of an edge e, a set of numbers $\mathtt{dep}(\mathtt{C}, \mathtt{i})$ or $\mathtt{dep}(\mathtt{r}, e)$, respectively, indicating which non-deterministic decisions $\mathtt{C}$ being present in $\mathtt{i}$'s label or $\mathtt{r}$ being present in the e's label depends on.

Depending on a non-deterministic decision, here, means for a concept or role being in a node's or edge's label that it is there as a direct consequence of that decision or of the application of a deterministic expansion rule which used a concept or role in a node's or edge's label that itself depended on that non-deterministic decision.

In case of a clash, dependency sets can be used to determine the most recent non-deterministic decision leading to that clash and make it possible to track back and undo all changes to the completion graph since that decision was made.

In-depth discussions of backjumping can be found in [21, 20]. While this technique does not affect the models generated by a tableau algorithm, its discussion sheds light on some of the data that can found in the data structures used by a reasoner implementation.

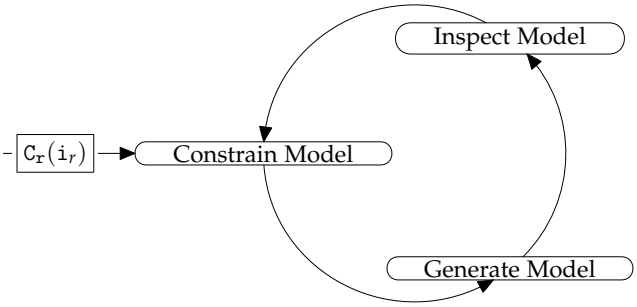


Figure 7: Model Exploration Cycle

Consequences for Model Exploration: model exploration can exploit dependency sets of completion graph node's labels in order to indicate to users which parts of a model have been added purely deterministically and which parts were added after a non-deterministic decision (see Section 7.2).

6 Model Exploration

6.1 The General Idea

The idea behind model exploration is that it may be useful for users to see models of concepts in an ontology to understand their semantics. Unfortunately, there may be very many (even infinitely many) very big (infinitely big) models for a given concept. This is why model exploration includes an interactive component.

What we hope is that seeing *small parts* of *just a few* models suffices in many cases to give users the information they seek. Initially, we show them only one instance in some model of a concept they are interested in, the *root concept*, together with other directly connected individuals, and let them expand it from there. We then let the users successively add constraints and thus specify which models they are interested in (see Fig. 7). Thus, model exploration enables users to explore both the space of and individual models of concepts.

To clarify this point, consider the family ontology depicted in Fig. 8. Suppose a user is interested in the concept `GrandParent`. A model exploration system would first generate a model for `GrandParent`, e.g. the one depicted in Fig. 9. It would initially show the user just two individuals, the root individual i_r and its only successor, its `hasChild`-successor i_1, and their labels (see Fig. 10a).

i_2 would be hidden at first, until the user inquires about i_1's role successors. Then, the excerpt would be expanded and, in this case, the entire model would be visible (see Fig. 10b).

If the user now adds constraints on the model, making i_2 a `Son`, the system would

$$
\begin{aligned}
\texttt{Parent} &\equiv \exists \texttt{hasChild}.\top \\
\texttt{Child} &\equiv \exists \texttt{hasChild}^{-}.\top \\
\texttt{Child} &\sqsubseteq \texttt{Son} \sqcup \texttt{Daughter} \\
\texttt{GrandParent} &\sqsubseteq \exists \texttt{hasChild}.\texttt{Parent} \\
\texttt{Daughter} &\sqsubseteq \texttt{Child} \sqcap \texttt{Female} \\
\texttt{Son} &\sqsubseteq \texttt{Child} \sqcap \texttt{Male} \\
\texttt{Male} &\sqsubseteq \neg \texttt{Female}
\end{aligned}
$$

Figure 8: Simple Family Ontology $\mathcal{O}_F$

$$
\begin{aligned}
\texttt{GrandParent}^{\mathcal{I}_F} &= \{\texttt{i}_\texttt{r}\} & \texttt{Daughter}^{\mathcal{I}_F} &= \{\texttt{i}_1, \texttt{i}_2\} \\
\texttt{Parent}^{\mathcal{I}_F} &= \{\texttt{i}_\texttt{r}, \texttt{i}_1\} & \texttt{Female}^{\mathcal{I}_F} &= \{\texttt{i}_1, \texttt{i}_2\} \\
\texttt{Child}^{\mathcal{I}_F} &= \{\texttt{i}_1, \texttt{i}_2\} & \texttt{hasChild}^{\mathcal{I}_F} &= \{(\texttt{i}_\texttt{r}, \texttt{i}_1), (\texttt{i}_1, \texttt{i}_2)\}
\end{aligned}
$$

Figure 9: Model $\mathcal{I}_F = (\Delta^{\mathcal{I}_F}, \cdot^{\mathcal{I}_F})$ For the Family Ontology

generate a new model, which could be the same as $\mathcal{I}_F$ except for $\texttt{i}_2$ being labeled `Son` and `Male` instead of `Daughter` and `Female`. The system would present the expanded excerpt like in Fig. 10c. Another constraint, asking for $\texttt{i}_1$ having only `Female` children, would obviously be inconsistent and the system would tell the user that.

6.2 Constraints

In order to be able to use standard reasoning algorithms (and existing reasoners), constraints on the models need to be expressed as something a reasoner understands—as axioms. Many constraints can be expressed in the root concept itself. In the example in Section 6.1 the root concept could be first

$$\texttt{GrandParent}$$

then

$$\texttt{GrandParent} \sqcap \forall \texttt{hasChild}.\forall \texttt{hasChild}.\texttt{Son}$$

and finally

$$\texttt{GrandParent} \sqcap \forall \texttt{hasChild}.\forall \texttt{hasChild}.(\texttt{Son} \sqcap \texttt{Female})$$

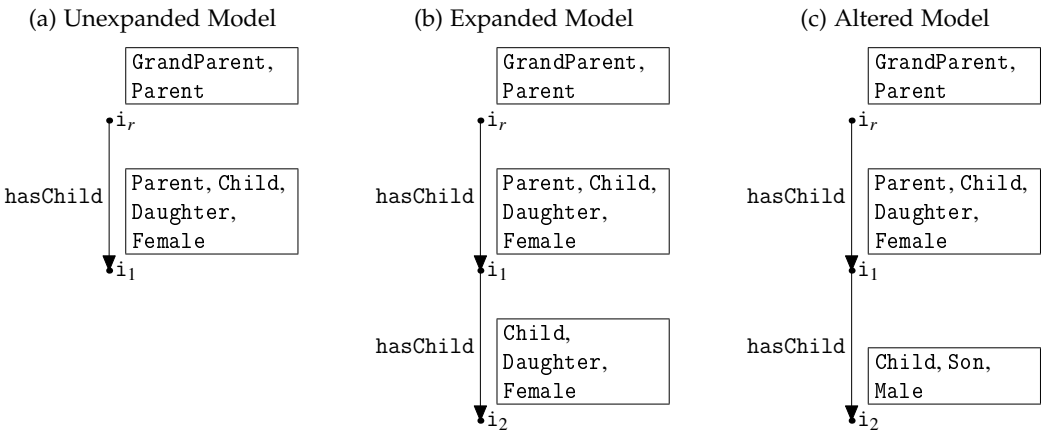


Figure 10: Three Steps in Model Exploration

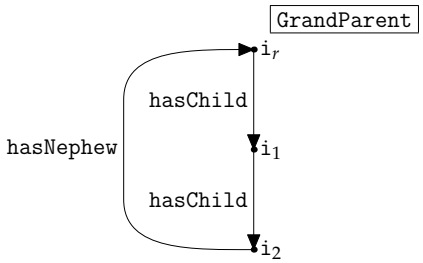


Figure 11: ABox - Specification of a Model

to obtain results like the ones in Fig. 10.

This, however, has its limitations. For example, there is no way to design a root concept which corresponds to an identity constraint for two individuals. Suppose for example, a user wants to know whether there is a model for the family ontology in which a `GrandParent` is her grandchild's nephew (see Fig. 11). This is not expressible as a concept description: one can say that there is a `GrandParent` whose grandchild `has-Nephew` some `GrandParent`, but it is impossible to say that the `GrandParents` are identical. What can be done, though, is to translate all constraints into ABox axioms as follows:

$$\mathcal{A} = \left\{ \begin{array}{l} \texttt{GrandParent}(\texttt{i}_\texttt{r}), \\ \texttt{hasChild}(\texttt{i}_\texttt{r}, \texttt{i}_1), \\ \texttt{hasChild}(\texttt{i}_1, \texttt{i}_2), \\ \texttt{hasNephew}(\texttt{i}_2, \texttt{i}_\texttt{r}) \end{array} \right\}$$

We are now ready to give our approach a more formal definition:

Definition 6.1. Model Exploration: model exploration is an activity cycling through the stages of *model specification*, *model generation*, and *model inspection* (see Fig. 7). In the first stage, the users add axioms to an ABox which serve as constraints for the type of model they want to see next. In model generation, a completion graph for this *constraint ABox* is computed and adorned with additional information. In model inspection, a connected subgraph, the *model excerpt*, of this completion graph is presented graphically to the user and may be expanded (explored) along the edges of the completion graph. The first constraint ABox is obtained from the user-specified *root concept* $\mathtt{C}_r$ by adding the assertion $\mathtt{C}_r(\mathtt{i}_r)$ to an empty ABox. The model excerpt always includes the *root individual* $\mathtt{i}_r$.

6.3 Status of Information

In general, an interpretation need not map an individual name to every individual in its domain. However, for every model of an ontology and a concept, there is a model in which, for every individual, there is at least one name which is mapped to that individual: suppose $\mathcal{I}$ is a model for an ontology $\mathcal{O}$ and a concept $\mathtt{C}$. Then, since the set N_I of individual names is infinite and $\mathcal{O}$ and $\mathtt{C}$ are finite, there are infinitely many individual names not occurring in either $\mathcal{O}$ or $\mathtt{C}$, and thus, an interpretation $\mathcal{I}'$ can be constructed which is identical to $\mathcal{I}$ except that it maps to every individual in $\mathcal{I}$'s domain one of these names exclusively.

$\mathcal{I}'$, then, is also a model for $\mathcal{O}$ and $\mathtt{C}$, since whether or not it satisfies the axioms in $\mathcal{O}$ and is a model of $\mathtt{C}$ depends solely on how it interprets role, concept, and individual names occurring in them which it does identically to $\mathcal{I}$.

To make the presentation of the rest of this section easier, we will assume models in which every individual has a name and visualizations of models which represent individuals by their name. The type of graph diagram we have been using is such a visualization: instead of using unnamed dots as representations for individuals, the individuals in the diagrams carry names like $\mathtt{i}_1, \mathtt{i}_2, \ldots$ by which they can be referred to.

This way we can say that a visualization of a model contains atomic pieces of information regarding the ABox axioms the model satisfies: if in a visualization of a model it can be seen that an individual with name $\mathtt{i}$ is an instance of a concept $\mathtt{C}$ or has an $\mathtt{r}$-successor $\mathtt{i}'$, then that is equivalent to saying that the piece of information visualized is that the model satisfies the axiom $\mathtt{C}(\mathtt{i})$ or $\mathtt{r}(\mathtt{i}, \mathtt{i}')$, respectively.

A model for an ABox must satisfy all of the ABox's axioms, and thus a (partial) visualization of such a model will often contain pieces of information informing the user of axioms being satisfied which are in that ABox. Consider again the model depicted in

Fig. 10c. The ABox for which it was generated is

$$\mathcal{A} = \left\{ \begin{array}{l} \texttt{GrandParent}(\texttt{i}_\texttt{r}), \\ \texttt{hasChild}(\texttt{i}_\texttt{r}, \texttt{i}_1), \\ \texttt{hasChild}(\texttt{i}_1, \texttt{i}_2), \\ \texttt{Son}(\texttt{i}_2) \end{array} \right\}.$$

The visualization of the model in Fig. 10c in fact contains a piece of information for every axiom in $\mathcal{A}$. Naturally, it makes sense to visually distinguish these *asserted* pieces of information from the others.

Another distinction can be made between non-asserted pieces of information stating the satisfaction of those axioms like $\texttt{Parent}(\texttt{i}_\texttt{r})$ and those like $\texttt{Daughter}(\texttt{i}_1)$ by the visualized model. The difference between these axioms is that the former are satisfied by every model of $\mathcal{A}$ and $\mathcal{O}_F$, and the latter only by some. When trying to learn something about the models of an ontology, it is likely to be helpful to know which parts of a model one is looking at are common to all models—we call them *mandatory*—and which ones are not.

We coin the term *mandatory information* as a generalization of the notion of information relating to *implied* axioms. The axiom $\texttt{Parent}(\texttt{i}_\texttt{r})$, for example, is an implication of $\mathcal{O}_F$ and the ABox $\mathcal{A} = \{\texttt{GrandParent}(\texttt{i}_\texttt{r})\}$, whereas $\texttt{hasChild}(\texttt{i}_\texttt{r}, \texttt{i}_1)$ is not. However, in every model $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{O}_F$, ${\texttt{i}_\texttt{r}}^{\mathcal{I}}$ has a $\texttt{hasChild}$-successor. This is a commonality of all models for $\mathcal{A}$ and $\mathcal{O}_F$ which we believe is worth pointing out when visualizing a model. Thus, we would like to call information relating to an axiom $A = \texttt{hasChild}(\texttt{i}_\texttt{r}, \texttt{i}_1)$ in a model for $\mathcal{A}$, $\mathcal{O}_F$, and A mandatory as well.

Now, while asserted pieces of information are asserted independently of each other, pieces of information can only be defined to be mandatory in relation to each other: consider the concept definition

$$\texttt{D} \sqsubseteq \exists \texttt{p}.\texttt{A} \sqcap \exists \texttt{p}.\texttt{B} \sqcap \exists \texttt{p}.\texttt{C} \sqcap \leq 2\,\texttt{p}$$

and the models depicted in Fig. 12, which are the alternatives likely to be created for $\texttt{D}$ by a tableau reasoner.

Since every instance of $\texttt{D}$ has one $\texttt{p}$-successor which is an $\texttt{A}$, one which is a $\texttt{B}$, and one which is a $\texttt{C}$, the following information depicted in Fig. 12a could be said to be mandatory in the sense that similar things are true in every model of $\texttt{D}$

- $\texttt{i}_\texttt{r}$ is a $\texttt{D}$
- $\texttt{i}_\texttt{r}$ has $\texttt{p}$-successors $\texttt{i}_2$ and $\texttt{i}_1$
- $\texttt{i}_2$ is a $\texttt{C}$
- $\texttt{i}_1$ is an $\texttt{A}$ **or** $\texttt{i}_1$ is a $\texttt{B}$

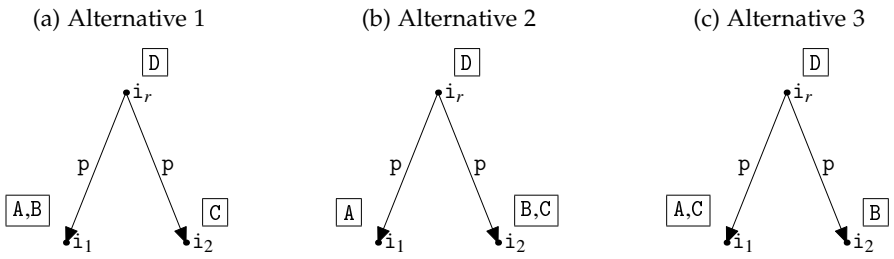


Figure 12: Illustration of Mandatory Information

Stating that $\mathtt{i}_1$ being an `A` **and** being a `B` are both mandatory information would be misleading, because in other models, the `p`-successors of the instance of `D` which are an `A` and a `B`, respectively, are not the same, as can be seen in Figs. 12b and 12c.

In order to be useful for users of model exploration, a definition for information in a visualization being mandatory needs to be simple enough to work with in practice, yet capture a meaningful notion of commonalities of all models of an ontology and a concept. We believe that the following definition strikes a good compromise:

Definition 6.2. Mandatory Superset, Mandatory Subset: A set of ABox axioms S is a *mandatory superset* of an ABox $\mathcal{A}$ wrt. an ontology $\mathcal{O}$, if S includes $\mathcal{A}$ and every model for $\mathcal{A}$ and $\mathcal{O}$ can be transformed into a model for $\mathcal{O}$ and all the axioms in S only by changing its interpretation of individual names occurring in S but not in $\mathcal{A}$ or $\mathcal{O}$.

A set of pieces of information in a visualization of a model for an ABox and an ontology stating the satisfaction of all axioms in a mandatory superset of the ABox wrt. the ontology is called a *mandatory subset* of all the pieces of information in the visualization regarding the satisfaction of ABox axioms by that model.

6.4 Presentation

Section 5 gives an overview of what information can be directly extracted from the data structures found in a reasoner and what else can be inferred more or less straightforwardly. This information can be presented in a number of ways.

Concepts: in a completion graph built in a successful run of a tableau reasoner, nodes are labeled at least with all concept names necessary to define a canonic model. On top of that, some nodes are additionally labeled with complex concept descriptions due to the reasoning process. Whether or not to present the user with these concept descriptions is a difficult question. Consider again the situation depicted in Fig. 10c. In the completion graph, $\mathtt{i}_2$ may not only be labeled with concept names, but also with the concept description $\neg$`Female`, which the tableau algorithm might have inferred from `Son` in $\mathtt{i}_2$'s label. If

users saw this concept in the label, they would not have to make an additional step to see that $\mathtt{i}_1$ cannot have only `Female` children.

The problem with this is that more information may quickly become too much information. Think of a whole hierarchy of subconcepts of `Female`. In the completion graph, the negation of every concept subsumed by `Female` would find its way into the label of $\mathtt{i}_2$, possibly making it harder instead of easier to understand any implications.

Similar points can be made for and against more complex concept descriptions in individuals' labels.

Roles: in Section 5.2.3, we explained that some role relationships are not available immediately from completion graphs generated by tableau reasoners. Here, we will see why this is not only a problem, but may also have its good sides. The reason is that role inclusion axioms and especially transitivity can lead to a huge number of role relationships. While, in general, there is no reason why seeing transitive relations in model exploration would be undesirable, there may be cases where they are just too many.

In our running example, the family ontology, a transitive, symmetric relation `isRelated` could be a super role of all roles like `hasChild`, `hasNephew`, `hasBrother`, etc.[11] A model containing only five individuals which are connected to each other by any of these relations would include 20 `isRelated` relationships, which may well clutter the display.

The fact that they are not present in the completion graph and would need to be added in a post-processing step gives model exploration the opportunity to distinguish these relationships, give them a different visual appearance, and filter them out on user demand.

Blocked Individuals: there are three approaches to dealing with blocking in model generation. The first one is to point out blocked individuals and, optimally, the blockers. This is both comparably easy and correct, as it not only gives users the complete information about the model they inspect, but also some information about the reasoning process, which can be helpful, e.g. for ontology performance considerations as described by Wang and Parsia [51].

The second approach is to pretend blocking did not happen. The moment a user asks for the successors of a blocked individual, the blocking individidual's successors could be cloned to create the illusion of infinite model depth. For users who are not familiar with tableau reasoning, this may be less confusing. On the other hand, it may also be misleading as it can become difficult to distinguish long chains of role relationships from infinite ones. Fig. 13 illustrates this problem: in Fig. 13a, the user has explored the model beyond the blocked individual and can go on exploring forever, while in Fig. 13b the

11. This, of course, would probably only make sense if there are other relationships like `hasAcquaintance` which are not sub-roles of `isRelated`.

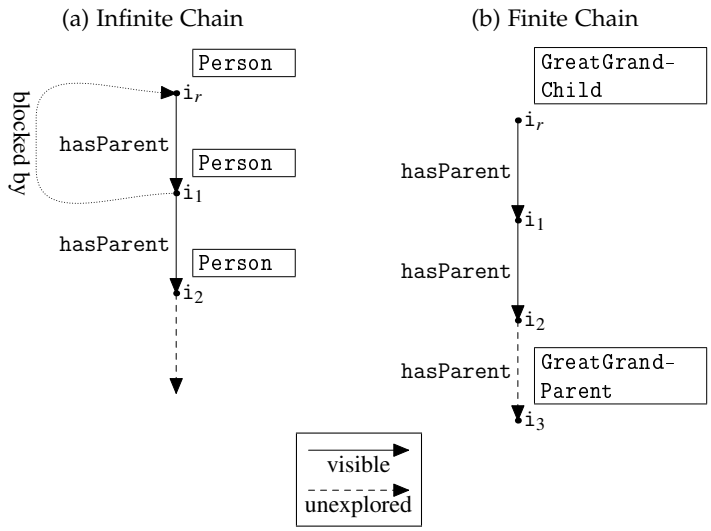


Figure 13: Infinite vs. Finite Model Depth

finite model has not been explored completely. The two cases are hard to distinguish in a visualization.

Lastly, blocked individuals can be pointed out *and* still be unfolded on demand.

Visualization: Section 4.2 discusses the related work on the visualization of description logic models we have found. None of the approaches addresses the different kinds of information (asserted, mandatory, other) and only Tweezers handles blocked nodes.

Without these, visualization of models becomes very similar to the visualization of any kind of ABox-like structure: what is displayed is just a graph of nodes corresponding to individuals, labeled with the concepts they belong to and connected by edges which correspond to role relationships between them. Therefore, since any of the approaches mentioned in Section 4.2 would have to be extended to be used in model exploration, they can be discussed in parallel with other approaches dealing with the more general problem of visualizing such structures.

In general, the design of any visualization of models and ABox-like structures should obey the following design issues drawn from [13]: that there should be a one-to-one translation between the visual language and the DL it is based on such that the semantics of every visual language element can be explained simply and exhaustively, and it relates to established semeiology of the user community and takes the human factors of individual users into account. Other issues are flexibility with regard to the kinds and amount of data that can be displayed, and scalability to large numbers of individuals and role relations between them, meaning that large models can be accommodated on

reasonable screen areas and still be clear. We will indirectly refer to these design issues in the discussion of types of and individual techniques below, and use them in Section 7 to justify and evaluate our own decisions.

The field of techniques used for visualization of models and ABox-like structures can be ordered along a high- vs. low-specialization axis; at one end, there are approaches which are applicable to a great variety of data. At the other, there are those techniques which have been designed specifically for visualizing DL models. We will name examples of techniques from the literature going from low to high specialization, and use them to discuss the benefits and drawbacks found at their respective point in the specialization continuum. Observe that at different levels of specialization, different design issues are met more or less naturally.

At the low-specialization end of displaying models and other ABox-like structures is the representation of individuals as nodes and role relationships as arrows in a simple graph diagram. Tweezers [51] follows this approach. It has the benefit of using a graphical representation which should be familiar and intuitive to everybody and is very flexible with regard to the information it encodes in various node and arrow decorations. It has the drawback of not being as economical in screen size as, for example, the one by Noppens and Liebig described below, and, of course, it needs deciding how to deal with blocking. Also, including the concepts individuals belong to in the graph diagram is somewhat tricky. A simple solution to this is having a separate detail view at the side which shows the concepts an individual belongs to when it is selected.

The particular formalism used in [13] to illustrate the issues concerning the design of visual languages for the visualization of DL knowledge bases is based on established semantic network semeiology. The rationale behind this is to re-use tried and trusted work and make it easy for former users of semantic networks to understand the new formalism. Of course, since its goal is broader than what we need for model exploration, it may give too much room—literally—to representing terminology, and some work would be needed to adapt it to the additional information available in model exploration and, more importantly, make it scale to greater numbers of individuals. This especially applies where model exploration is just one tool in a bigger environment. If one were to choose a visual language for a tool which would, in addition to the actual model, have to provide the user with an interface to the entire background ontology, a semantic network-based approach as suggested there could come into closer consideration.

Noppens and Liebig [39] describe a tool for visualizing "large OWL instance sets". As its focus is on browsing asserted and inferred relationships in large networks of individuals, it does not deal with concept memberships. It supports a step-by-step unfolding of the relations and coerces the diagram of the potentially graph-like structure of the data into a tree, rooted at some individual of interest. Thus, if two individuals $\mathtt{i}_1$ and $\mathtt{i}_2$, both already visible in the display, have a common role successor $\mathtt{i}_3$, then this role successor is displayed twice if the roles are unfolded, and the double-representation is signaled by highlighting all dots in the display standing for the same individual when hovering over

one with the mouse cursor.

Given some individual i and a role r, the browsing tool developed by Noppens and Liebig also groups all of i's r-successors such that the label of the property in the diagram need not be displayed more than once. All of this is done to keep the display as simple as possible and hide all unnecessary information. Being more specific to the problem of accommodating large numbers of individuals and relations between them, this formalism employs more idiosyncratic ways of expressing knowledge in the hope of achieving scalability, but possibly making it harder for newcomers to get used to it.

At the high-specialization end of model displaying solutions are those that are specifically designed for this purpose, as, for example, the music score notation described by Barinskis and Barzdins ([6], see Section 4.2). Such tailor-made solutions have, of course, the benefit of being able to pay attention to all the subtleties of this specific problem and to avoid the trade-offs of more general approaches. However, they require more design and implementation effort than general approaches for which prior work exists. And again, specialization often comes with idiosyncrasy.

6.5 User Interaction

At the core of the concept of model exploration is the cycle of receiving model constraints (ABox axioms) from the user, computing a model, presenting an excerpt and letting the user unfold it, and receiving more constraints (See Fig. 7). This proposed work cycle leaves some room for implementation.

Suppose somebody starts exploring the models for the concept Parent. The first ABox would just be $\mathcal{A} = \{\texttt{Parent}(\texttt{i}_r)\}$. The model for this would most likely have two individuals, $\texttt{i}_r$ and $\texttt{i}_1$, the former labeled with the concept Parent and connected via a hasChild role to the latter.

If, in order to see whether in the ontology individuals can be parents of GrandParents, the user extended the Abox to $\mathcal{A}' = \{\texttt{Parent}(\texttt{i}_r), \texttt{GrandParent}(\texttt{i}_1)\}$, the reasoner could, and probably would, generate a model in which $\texttt{i}_r$ is a Parent and hasChild some individual, say $\texttt{i}_2$, and $\texttt{i}_1$ is a GrandParent with everything that this implies, but totally unconnected to $\texttt{i}_r$, which does not answer the user's question. Correctly, the ABox would have to be extended to $\mathcal{A}'' = \{\texttt{Parent}(\texttt{i}_r), \texttt{hasChild}(\texttt{i}_r, \texttt{i}_1), \texttt{GrandParent}(\texttt{i}_1)\}$.

To generalize, it makes sense in many cases to add concepts and role successors only to individuals which are connected via role assertions to the root individual. An implementation of model exploration can either leave this issue to the user, silently add role relationships to ensure this connection, or ask the user what to do.

Another two minor points are how deeply to expand the model excerpt for the user initially and exactly at what point to generate new models.

Addressing the first question, one could opt to start with just one individual and let the user expand every role relationship they are interested in. Secondly, one could expand the entire model—probably stopping at blocked individuals—and leave only the

unfolding of those relationships to the user that lead into infinity. And thirdly, there is the compromise of expanding the model up to depth one or two and leave the rest to the user.

Generating models can take a long time. Thus, sometimes it would be good not to generate a new model after every change of the ABox. On the other hand, having to ask the model exploration system to generate a model after every change may be confusing; in our user study, we have seen that users sometimes forget that a new model has not yet been generated and make wrong inferences about the ontology because of that. The implementer has the choice of either forcing the generation of a new model, giving the user control of when it happens, or make the behavior configurable.

7 Implementation

This section deals with the prototype implementation of model exploration which was designed in the course of this thesis. It is called SuperModel and it was designed as a plug-in for the OWL ontology editor Protégé 4.

We will first give a brief, high-level overview of SuperModel's user interface (UI) and how users may interact with it. Then, we will explain which pieces of third-party software SuperModel is based upon and how it relates to them. In the end of the section, we will talk about our decisions with regard to the conceptual options and design issues discussed in the previous section.

User Interface and Interaction: SuperModel installs itself as a *Class View Component* in Protégé 4 terminology, i.e. as one of the components which can be added to the configurable user interface of the ontology editor and which offer the user information on the concept level about the currently opened ontology. As a consequence, SuperModel is notified by the editor if, anywhere else in the UI, a concept name is selected. SuperModel can then generate and show a model excerpt for that concept.

When the user first selects a concept name for model exploration in SuperModel, the plug-in initializes its display with two panes (see Fig. 14) the *graph pane* on the left and the *detail pane* on the right.

The graph pane shows a graph-like visualization of the model, with individuals and role relationships represented by nodes and arrows, respectively. To save space and keep the graph simple, two nodes are connected by at most one arrow labeled with the names of all the role relationships connecting them. The arrows are uni- or bidirectional depending on whether there are role relationships going in both directions or not. There is a red arrow going from individuals $\mathtt{i}_1$ to $\mathtt{i}_2$, if $\mathtt{i}_1$ is blocked by $\mathtt{i}_2$.

The detail pane, on the other hand, only ever shows information about a single node: namely the named concepts it belongs to (plus a few more, see below) and role relationships connecting it to other individuals. The individual whose information is shown in

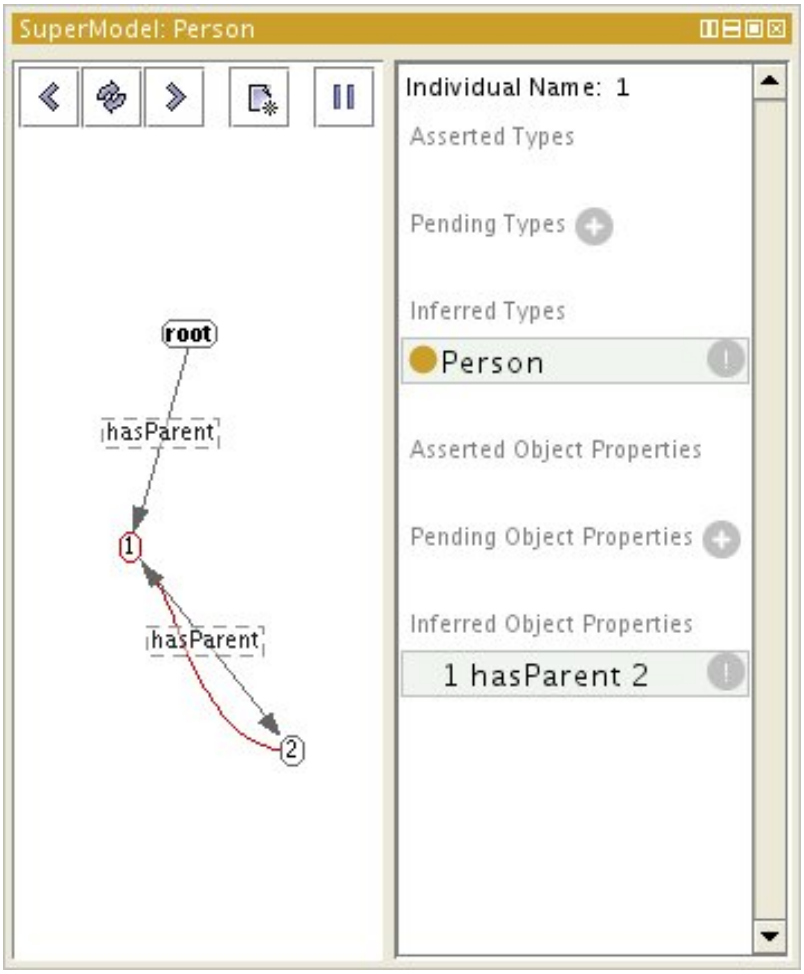


Figure 14: SuperModel's User Interface

this pane can be selected by clicking on a node in the graph pane, which also causes that node to be rendered with a red outline.

Initially, the diagram shows one node, which is generically named `root`, and all its immediate role successors. Their successors, in turn, are not shown. Instead, any node that has role successors which are hidden in the display has a bold outline. If such a node is selected with the mouse cursor, nodes and arrows are added to the display to represent its successors and the role relationships connecting it to them.

In the individual details pane, the user can add concept or role assertions to the currently selected individual as follows. The '+'-Button next to the line saying "Pending Types" calls up a dialog allowing the user to input a concept description. If the user enters a concept `C` and the currently selected individual is `i`, then $\texttt{C}(\texttt{i})$ is added to the constraint ABox. Similarly, the '+'-button next to the 'Pending Object Properties' line lets the user choose a role `r` and a name for a role successor `s`, and adds the role assertion $\texttt{r}(\texttt{i},\texttt{s})$ to the constraint ABox. The user can add any number of these assertions before clicking the 'refresh' button, which initiates the calculation of a new model, which is then displayed.

History buttons, a "back" and a "forward" button allow the user to go back and forth between models they have already seen, starting the exploration of a model anew every time.

7.1 Third-Party Software

7.1.1 Protégé 4 Interface

As mentioned earlier, SuperModel was realized as a plug-in for the ontology editor Protégé 4. This was done for a number of reasons. Firstly it allowed us to use Protégé 4's infrastructure for the user interface, ontology management, and access to reasoners. Secondly, embedding it in a commonly used ontology editor made it possible to see how model exploration could be used together with other, established, mechanisms for ontology understanding and manipulation—it could be observed in the wild, so to say. Last but not least, implementing SuperModel as a plug-in for Protégé 4 increased the probability of it outliving the end of this project by becoming a tool actively used in practice.

Figure 15 shows how SuperModel makes use of the Protégé 4 Java framework: the main UI class, SuperModel is a subclass of the Protégé 4 class AbstractOWLClassViewComponent which allows it to be recognized by Protégé 4 as a plug-in for inspection or manipulation of concepts in an ontology. If an OWL class view component is inserted into the Protégé 4 UI, it is automatically notified via the updateView() method when a concept is selected anywhere else in Protégé 4's UI.

Selecting an individual in the graph pane causes a call to the setRootObject() of the ModelIndividualFrame which manages the details pane of the SuperModel UI. The Protégé 4 framework makes sure that call is propagated via its ModelIndividualFrameList2 to the ModelIndividualTypeFrameSections and ModelIndividualRoleFrameSections. Each frame

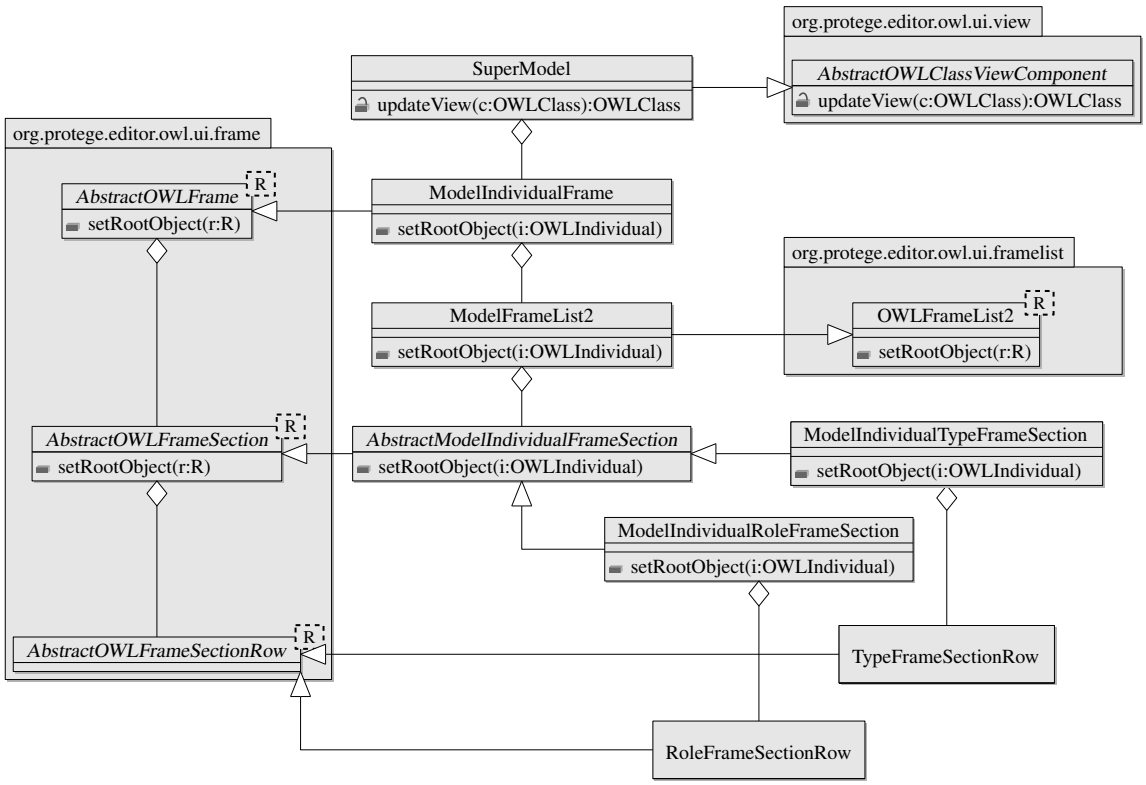


Figure 15: Use of the Protégé 4 Framework in SuperModel

section manages one of the sections for asserted, pending, and inferred types and role relationships seen in the UI and refills itself with ModelIndividualTypeFrameSectionRows and ModelIndividualRoleFrameSectionRows accordingly when its setRootObject() method is called.

7.1.2 OWLAPI Interface

The OWL API[12] is an open-source (LGPL) Java API to be used in programs working with OWL ontologies. It includes classes used to represent, inspect, and manipulate OWL ontologies, handles their serialization and de-serialization into and from various OWL syntaxes, support for reasoners, etc.

Protégé 4 makes heavy use of the OWL API for all of these tasks. It also communicates with its plug-ins using OWL API data structures. For this reason, SuperModel is also based on the OWL API. This also makes SuperModel potentially compatible with other software using the OWL API—using models generated by SuperModel as ontologies in

12. http://owlapi.sourceforge.net/

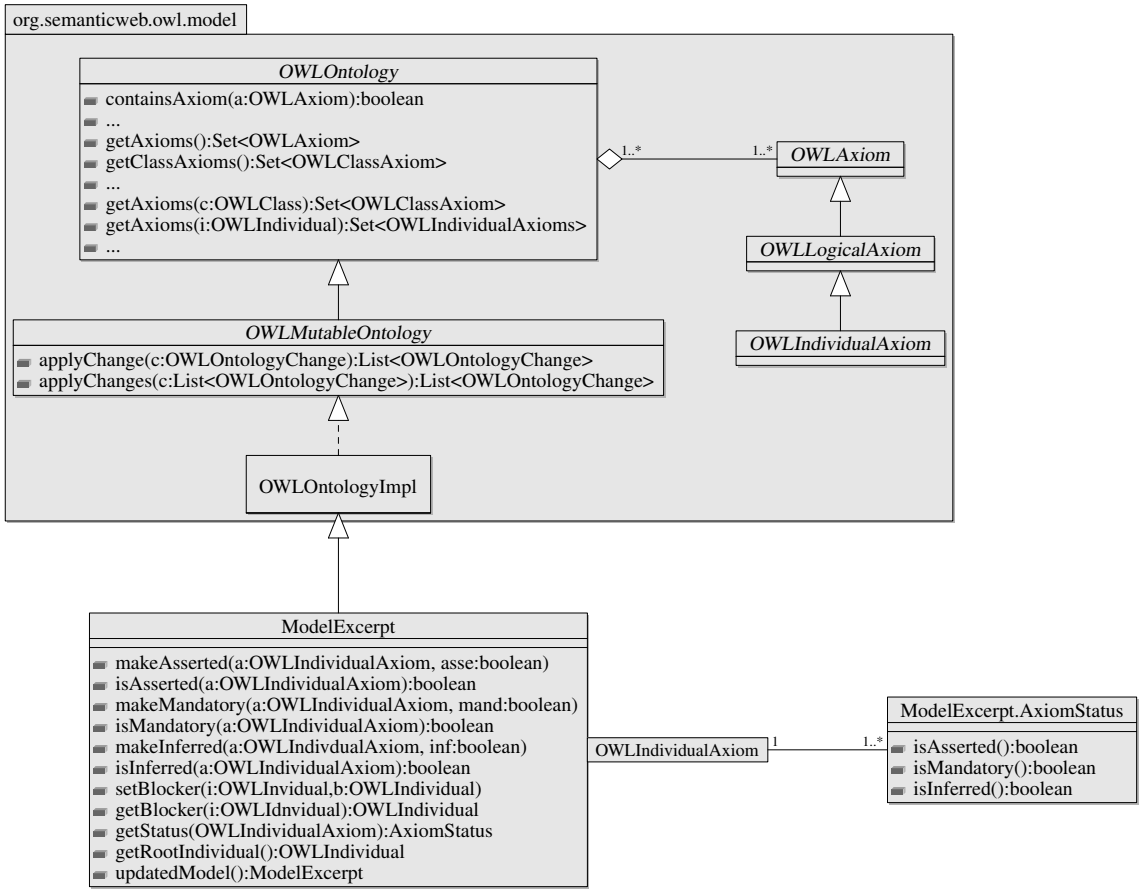


Figure 16: SuperModel and the OWL API

their own right in other applications comes to mind.

OWLOntology is the basic Java interface for the representation of OWL ontologies in the OWL API. Its methods give various types of access to the axioms contained in the ontology, as in asking for all axioms, all axioms of a certain type, axioms containing a certain concept, role, or individual name; and they allow for other, non-logical querying, like querying for annotations and ontology imports. OWLMutableOntology is a sub-interface of OWLOntology which additionally allows for adding and removing information. OWLOntologyImpl is a class implementing the OWLMutableOntology interface (see Fig. 16).

SuperModel's data structure is centered around the ModelExcerpt which is a subclass of OWLOntologyImpl. In addition to the axiom inspection and modification support of its superclass, ModelExcerpt maintains model exploration-related information about the axioms: a given axiom can be asserted, mandatory, and inferred (the latter, if it has been found in a description graph as opposed to added by the user), individuals may be blocked and each of these states can be set and queried for.

7.1.3 Reasoner Interface

The model generation process is initiated from the ModelExcerpt class mentioned earlier. A ModelExcerpt creates a new instance of its class when its updatedModel is called. It copies all its asserted axioms into this new instance and calls its updateModel() method. The new ModelExcerpt instructs an instance of ModelGenerator to retrieve a completion graph and fill in the information it finds in the graph. In the end, the call to updatedModel() on the old ModelExcerpt returns the new excerpt.

So far, no reasoner-specific code is involved. By implementing the interface ModelGenerator, data structures from any reasoner could be used in SuperModel, provided they contain and make accessible the relevant information.

For proximity of the implementer because it is a state-of-the-art $\mathcal{SROIQ}$ tableau reasoner and open-source, we chose FaCT++ [50] as the reasoner to draw our models from.

Unlike other popular reasoners like Pellet [49], Racer [17], and HermiT[13], FaCT++ is written in C++[14] which causes a split in the interface to SuperModel along the Java-C++ interface (JNI). The technicalities resulting from this split would cause a UML diagram to contain too much technical detail to be clear or too little to be correct, which is why it is omitted.

The FPPModelSupplier, our implementation of the ModelGenerator which communicates with FaCT++, basically uses the official Java interface which comes with the FaCT++ distribution to tell the reasoner about the background ontology and the constraint ABox. A single method call, buildModelFor(), has been added to this interface to retrieve the model. This last call first causes the root concept to be classified by FaCT++, and then the completion graph generated during classification to be walked and the information found in it to be copied back to the new ModelExcerpt.

The FaCT++ code itself has been modified in only a few strategic places, mostly changing the visibility of methods to make them accessible to our own C++ class, ModelBuilderCPP, which does the walking and copying. The ModelBuilderCPP is also responsible for filling in information needed to complete the model, like equivalent classes and super roles (see Sections 5.2.5 and 5.2.3).

7.1.4 Prefuse

When we first experimented with exploring models, we started using the prefuse toolkit[15] to generate a simple, graph-like visualization of the structure. The prefuse toolkit is a visualization toolkit which can easily be used for general graph drawing in Java applications.

13. http://www.hermit-reasoner.com/
14. Pellet and Hermit are written in Java, Racer in Common Lisp
15. http://prefuse.org/

During our work on SuperModel, this continued to seem appropriate in terms of flexibility and simplicity, and not too wasteful in terms of display space. The models we explored were almost never too big to be viewed clearly, and we never experienced any trouble explaining its meaning to newcomers.

Thus, for now, the design criteria discussed in Section 6.4 appeared met well enough and we did not feel radically more sophisticated visualization approaches were needed, although in more mature versions we may want to make heavier use of prefuse's customization support to get a clearer display and a more stable graph layout (see Section 8.9.1).

7.2 Conceptual Choices

Status of Information: in order to distinguish asserted information (see Section 6.1), concepts individuals are instances of are shown under "Asserted Types" and their role relationships under "Asserted Roles" if these concept affiliations and role relationships were asserted in the constraint ABox.

We evaluate the dependency sets of labels and role relationships in the completion graph to acquire a mandatory superset of the constraint ABox; if a concept $\mathtt{C}$ is found in the label of some individual named $\mathtt{i}$, and $\mathtt{dep}(\mathtt{C},\mathtt{i})$ is empty, then an axiom $\mathtt{C}(\mathtt{i})$ is added to our mandatory superset. If the dependency set for an edge $(\mathtt{i},\mathtt{i}')$ and a role $\mathtt{r}$ is empty, then the axiom $\mathtt{r}(\mathtt{i},\mathtt{i}')$ is added.

Theorem 7.1. *Given a completion graph $G(V,E,\mathcal{L}_V,\mathcal{L}_E)$ for an ontology $\mathcal{O}$ and a constraint ABox $\mathcal{A}$ constructed by a tableau reasoner implementing the decision procedure for $\mathcal{SROIQ}$ presented in [22], $\mathcal{A} \cup S$ is a mandatory superset of $\mathcal{A}$ if the following is true:*

$$\begin{aligned} S = &\{\mathtt{C}(\bar{\mathtt{i}}) \mid \mathtt{i} \in V, \mathtt{C} \in \mathcal{C} \cap \mathcal{L}_V(\mathtt{i}), \mathtt{dep}(\mathtt{C},\mathtt{i}) = \emptyset\} \\ &\cup \{\mathtt{r}(\bar{\mathtt{i}},\bar{\mathtt{i}}') \mid (\mathtt{i},\mathtt{i}') \in E, \mathtt{r} \in R \cap \mathcal{L}_E(\mathtt{i},\mathtt{i}'), \mathtt{dep}(\mathtt{r},(\mathtt{i},\mathtt{i}')) = \emptyset\} \end{aligned}$$

Where $\mathcal{C}$ is the set of $\mathcal{SROIQ}$ concept descriptions, R the set of roles, and, for $\mathtt{i} \in V$, $\bar{\mathtt{i}}$ is $\mathtt{i}_\mathtt{r}$, if $\mathtt{i}$ was the first node in the graph, $\mathtt{o}$ if $\mathtt{o}$ is the name of a nominal in $\mathcal{L}_V$ and an arbitrary name not occurring in $\mathcal{O}$ or $\mathcal{A}$ otherwise.

Proof. Let $\mathcal{I}$ be a model of $\mathcal{O}$ and $\mathcal{A}$. We have to show that $\mathcal{I}$ can be transformed into a model of $\mathcal{O}$ and all axioms in $\mathcal{A} \cup S$ by changing only its interpretation of individual names not occurring in $\mathcal{A}$ or $\mathcal{O}$. In fact, $\mathcal{I}$ already is a model for every axiom in $\mathcal{A}$, therefore we only need to consider axioms $A \in S$.

Let $A = \mathtt{C}(\bar{\mathtt{i}})$ for some concept description $\mathtt{C}$ and individual name $\bar{\mathtt{i}}$. Then there must be a node $\mathtt{i}$ in the completion graph, $\mathtt{C}$ must be in the label of that node and $\mathtt{dep}(\mathtt{C},\mathtt{i})$ must be empty. It must have been in the graph before any expansion rules were applied, or one of the following deterministic expansion rules must then be responsible for $\mathtt{C}$ being in $\mathtt{i}$'s label:

$\sqsubseteq$*-rule:* $\mathtt{C}$ was added to $\mathcal{L}(\mathtt{i})$ because, for some $\mathtt{D} \in \mathcal{C}$, $\mathtt{D} \sqsubseteq \mathtt{C}$ is in $\mathcal{O}$ and $\mathtt{D}$ already was in the label of $\mathtt{i}$. $\mathsf{dep}(\mathtt{D}, \mathtt{i}) = \emptyset$, since $\mathsf{dep}(\mathtt{C}, \mathtt{i}) = \emptyset$, and thus $\mathtt{D}(\bar{\mathtt{i}})$ must be in S.

Assume $\mathcal{I}$ already satisfies $\mathtt{D}(\bar{\mathtt{i}})$. Then it must also be a model for $\mathtt{C}(\bar{\mathtt{i}})$ since it is a model of $\mathcal{O}$ and thus of $\mathtt{D} \sqsubseteq \mathtt{C}$.

$\sqcap$*-rule:* $(\mathtt{C} \sqcap \mathtt{D})$ was already in the label of $\mathtt{i}$, for some $\mathtt{D} \in \mathcal{C}$, and thus $\mathtt{C}$ was added. Again, $\mathsf{dep}((\mathtt{C} \sqcap \mathtt{D}), \mathtt{i})$ must be empty and $(\mathtt{C} \sqcap \mathtt{D})(\bar{\mathtt{i}}) \in S$

Assume $\mathcal{I}$ is a model of $(\mathtt{C} \sqcap \mathtt{D})(\bar{\mathtt{i}})$. Then $\mathcal{I}$ must also be a model of $\mathtt{C}(\bar{\mathtt{i}})$.

$\exists$*-rule:* there was a node $\mathtt{i}'$ in the completion graph with $\exists \mathtt{R}.\mathtt{C}$ in the label. $\mathtt{i}$ and an edge $(\mathtt{i}', \mathtt{i})$ were added with $\mathtt{R} \in \mathcal{L}_E(\mathtt{i}', \mathtt{i})$ and $\mathtt{C} \in \mathcal{L}_V(\mathtt{i})$. $\mathsf{dep}((\exists \mathtt{R}.\mathtt{C}), \mathtt{i}')$ is empty and thus $(\forall \mathtt{R}.\mathtt{C})(\bar{\mathtt{i}}') \in S$.

Assume $\mathcal{I}$ is a model of $(\exists \mathtt{R}.\mathtt{C})(\bar{\mathtt{i}}')$ but not of $\mathtt{C}(\bar{\mathtt{i}})$. Then $\bar{\mathtt{i}}'^{\mathcal{I}}$ must have an $\mathtt{R}$-successor in $\mathcal{I}$ which is in $\mathtt{C}^{\mathcal{I}}$. Making $\mathcal{I}$ map $\bar{\mathtt{i}}$ to that $\mathtt{R}$-successor makes it a model of $\mathtt{C}(\bar{\mathtt{i}})$, preserving satisfiability of all axioms in S in which $\bar{\mathtt{i}}$ does not occurr.

The only axioms in S which could have originally been satisfied and not be satisfied after $\mathcal{I}$ was altered are axioms which are in S because of a node or edge label that was added after $\mathtt{C}$ was added to $\mathtt{i}$'s label.

$\geq$*-rule:* virtually identical argument as for the $\exists$-rule.

$\forall_3$*-rule:* $\mathtt{C}$ was added to $\mathtt{i}$'s label, because $\forall \mathcal{B}_{\mathtt{R}}(f).\mathtt{C}$ already was in the label of $\mathtt{i}$, where f is the final state of the role automaton for the role $\mathtt{R}$.

Then, because $\mathsf{dep}(\mathtt{C}, \mathtt{i}) = \emptyset$ and by the role automata mechanism, there must have been roles $\mathtt{R}_1, \mathtt{R}_2, \ldots, \mathtt{R}_n$ and a path $e_1, e_2, \ldots, e_n$ in the graph, connecting some node $\mathtt{i}'$ to $\mathtt{i}$, such that $\mathtt{R}_k \in \mathcal{L}_E(e_k)$ and $\mathsf{dep}(\mathtt{R}_k, e_k) = \emptyset$ for $1 \leq k \leq n$, and $\mathtt{R}_1\mathtt{R}_2 \ldots \mathtt{R}_n$ must be in the language accepted by $\mathcal{B}_{\mathtt{R}}$. Also, $\forall \mathtt{R}.\mathtt{C}$ must be in the label of $\mathtt{i}'$ and $\mathsf{dep}((\forall \mathtt{R}.\mathtt{C}), \mathtt{i}') = \emptyset$.

This implies that $(\forall \mathtt{R}.\mathtt{C})(\bar{\mathtt{i}}') \in S$ and $\mathtt{R}_{\mathtt{k}}(\bar{\mathtt{i}}_{\mathtt{k}}, \bar{\mathtt{i}}_{\mathtt{k+1}}) \in S$, for $e_k = (\mathtt{i}_{\mathtt{k}}, \mathtt{i}_{\mathtt{k+1}})$, $1 \leq k \leq n$.

Assume $\mathcal{I}$ is a model for these latter axioms. Since $\mathtt{R}_1\mathtt{R}_2 \ldots \mathtt{R}_n$ is in the language of $\mathcal{B}_{\mathtt{R}}$, it must then also be true that $(\mathtt{i}', \mathtt{i}) \in \mathtt{R}^{\mathcal{I}}$. This in turn implies that, by the semantics of the $\forall$-constructor, it must be a model of the axiom $\mathtt{C}(\bar{\mathtt{i}})$.

If $A = \mathtt{R}(\bar{\mathtt{i}}, \bar{\mathtt{i}}')$, for some role $\mathtt{R}$ and individual names $\bar{\mathtt{i}}$ and $\bar{\mathtt{i}}'$, then there must be an edge e in the completion graph with $\mathtt{R}$ in the label and $\mathsf{dep}(\mathtt{R}, e) = \emptyset$. This can be either because $\mathtt{R}$ was in the label before any expansion rules were applied, or because of the application of the $\exists$-rule or the $\geq$-rule. In the latter case, the transformation of $\mathcal{I}$ described above for the $\exists$-rule makes $\mathcal{I}$ a model for A as well.

We have shown that, for every $A \in S$, $\mathcal{I}$ either is a model of A or can be transformed into one by changing only the mapping of individual names not occurring in $\mathcal{O}$ or $\mathcal{A}$, if A is in S because of a label in the completion graph which was added because of other labels that were already in it, and if $\mathcal{I}$ is a model of those axioms which are in S because of the latter labels.

If, for some $A' \in S$, the original $\mathcal{I}$ is a model, but the transformed interpretation is not, then A' is in S because of a label in the graph that was added after the one responsible for A being in S.

Since the only labels in the completion graph with empty dependency sets which were not added by one of the deterministic rules discussed above were added as translations of ABox axioms in $\mathcal{O}$ or $\mathcal{A}$, and since we assumed that $\mathcal{I}$ is a model for $\mathcal{O}$ and $\mathcal{A}$, we can conclude that, by induction, $\mathcal{I}$ is a model for $\mathcal{O}$ and all axioms in S or can be made one by changing only the interpretation of individual names not occurring in $\mathcal{O}$ or $\mathcal{A}$. □

Theorem 7.1 states that a mandatory superset of a constraint ABox can be constructed from the completion graph for that ABox. As explained in 6.3, however, there is in general more than one mandatory superset. We do not know whether or not the mandatory supersets we generate are the most useful ones for model exploration.

Also, they are not necessarily maximal. Consider the concept definition

$$\mathtt{D} \sqsubseteq (\mathtt{C} \sqcap \mathtt{C}') \sqcup (\mathtt{C} \sqcap \mathtt{C}'')$$

The root individual of the completion graph for $\mathtt{D}$ can be labeled $\{\mathtt{D}, \mathtt{C}, \mathtt{C}'\}$ or $\{\mathtt{D}, \mathtt{C}, \mathtt{C}''\}$, however, $\mathtt{C}$ would have been added after a non-deterministic decision of the reasoner and thus, its dependency set would be non-empty, unnecessarily excluding the axiom $\mathtt{C}(\mathtt{i}_r)$ from our mandatory subset.

If and how mandatory subsets could be extended to include more axioms is left for future work.

Concepts and role relationships in the detail pane of SuperModel are decorated with either a negation sign or an exclamation mark. If the corresponding axioms are in our mandatory subset, they are labeled with an exclamation mark. Concepts standing for axioms not in the mandatory subset are labeled with a negation sign which can be clicked, automatically adding the negated axiom to the constraint ABox; this way, users can test, what models look like in which the individual in question is not labeled with this concept.

An interpretation is completely specified by its interpretation of individual, role, and concept names. Thus, a visualization of a model needs only show individuals, their role relations and the concepts $\mathtt{C} \in N_C$ they belong to.

SuperModel's detail pane shows, in addition to the names of concepts individuals are labeled with, a few concepts of the form $\neg\mathtt{C}$ where $\mathtt{C}$ is a concept name. It does so only in case the dependency set of that concept in the label of the individual in question in the completion graph is empty.

SuperModel does not handle transitive roles or role chains. We left their treatment for future work because of the limited scope of this project and because we did not know whether users would generally benefit from or be overwhelmed by the additional information.

In the previous section, we talked about why it is often desirable that the ABox for which the model is calculated and shown form a connected graph and why one would

let a model exploration system ensure it is connected. SuperModel does follow the policy that all individual names occurring in the constraint ABox have to be connected to the root individual by some chain of axioms in the ABox. When the user adds a constraint for an individual that is not yet connected, it performs a simple breadth-first search in the model excerpt until it finds a set of role relationships connecting it to an individual that is and silently asserts them in the ABox as well.

The rationale for this decision was that we expected the cases where users would actually want unconnected ABox components to be rare, and the model excerpts we would be dealing with to be small, such that, in most cases, the number of role assertions added to the ABox by the system in order to ensure connectedness was going to be small as well.

The reason we implemented SuperModel to unfold the model excerpt only to depth one initially was that we believed it was easier to grasp a model's structure by exploring it than by looking at an already unfolded excerpt; and we thought it best to leave it to the users to unfold only those parts of the excerpt they are interested in.

8 Testing Usefulness

The aim of this work was to test the usefulness of model exploration for the understanding of ontologies by implemeting a model exploration prototype, and testing its usefulness. Shneiderman lists expert reviews, usability testing, surveys, and continuing assessments as viable methods to test usability [48].

Strictly speaking, all of these assess the usability of our prototype's user interface instead of the usefulness of model exploration. However, the best user interface can only be as useful for the understanding of an ontology as the approach behind it. Accordingly, any result suggesting that SuperModel does help also suggests that model exploration in general has its merits. The reason why model exploration was tested in this indirect way is that there exist sound scientific approaches to testing user interfaces, but, to the knowledge of the authors, none for the extremely specific field of testing improvement of understanding of ontologies.

The scope of this work ruled out expert reviews and continuing assessments. The limited publicity of SuperModel—it was just a prototype after all—also limited the number of people having sufficient experience with it to participate in a survey. Usability testing was therefore the method of choice. Incidentally, it was also a welcome opportunity to spread the word and raise awareness of SuperModel's existence.

The procedure for user studies recommended in [12, Part II] which we followed is to collect a number of goals and concerns regarding the piece of software to be tested, and select from this list the ones that are most relevant, promise the greatest information gain, and are most practicable to test.

It is then decided how and where participants are to be recruited such that they are

representative of the intended audience of the software.

After that, experiments are designed to test the goals and concerns, and a rating scheme for the results is devised.

Test materials are prepared, including any necessary legal forms, demographic and other questionnaires, and then the experiment is carried out.

In the end, the test results are evaluated and interpreted according to the rating scheme devised before the experiment.

8.1 Goals and Concerns

Goals, in the sense of [12], are what a piece of software was designed to achieve. They are customarily formulated as statements of success. Concerns are possible difficulties users might have with it, normally expressed as questions about whether or not users have those difficulties. Whether the goals have been accomplished and the difficulties avoided is what a user study tests.

The most general goal underlying our user study derives from this work's hypothesis:

> "Model exploration can help improve the understanding of an ontology or a part of it."

For the purpose of the experiment, this goal clearly needed to be made more specific: there are many things that could be understood about an ontology, some of which are easier to identify and test than others and obviously some aspects of ontologies are more philosophical and their understanding evades all attempts of definition and measurement, like, for example, an ontology's author's intention.

We put together the following short list of concrete, testable goals derived from the above mission statement. Afterwards, we selected from this list those goals that seemed most promising and easiest to test. In Section 3.3, we gave a few examples of ways to understand an ontology. Each of the following specific goals is a special case of helping users understand an ontology in one of these ways.

Goal No.	Goal Statement
1	SuperModel can be used to understand the reason for a given entailment.
2	SuperModel can help find out whether one concept is subsumed by another.
3	SuperModel can help find out, given a concept `C`, what role successors an instance of `C` must/may/may not have and which concepts they must/may/may not be instances of, and the same about their successors.

Table ctd. on next page

Goal No.	Goal Statement
4	SuperModel can be used to understand the reason for a given non-entailment.
5	When using SuperModel, a user can gain knowledge about an ontology circumstantial to the task at hand.

Table 1: Table of Goals

This study is not only to test whether model exploration as described in Section 6 generally allows answering questions about an ontology, but also whether it is a *good* way of answering such questions. In particular, the following concern arises:

> "Is it difficult for users to use model exploration to inquire about information about an ontology?"

Since we were trying to test a general technique through a concrete prototype, and since that prototype was still a bit rough around the edges in spite of the effort that went into designing it, we had to be aware that maybe it was simply too difficult to use. In order not to discard the technique because of difficulties with the implementation, some idea of how much the potential shortcomings of the prototype and especially the prototype's UI influenced the overall utility of the whole thing was needed to relativize the results.

Hence, we had one general concern regarding our prototype:

> "Will the complexity of the user interface of SuperModel be high compared to the actual tasks which it is designed to facilitate?"

Analogously to the general goal above, these concerns needed specification. Some concrete, testable concerns are listed below:

Concern No.	Concern Statement
1	Can users identify the constraints they have to add to get a specific piece of information?
2	Are users confused by the fact that transitive role relationships are not shown in the model?
3	Do users understand the functions of the UI controls?
4	Are users able to add the constraints they want?
5	Do users understand the workflow of SuperModel, i.e. do they understand what set of axioms led to the generation of the model they are looking at?

Table ctd. on next page

Concern No.	Concern Statement
6	Do users understand that a model is generated anew every time they press the "refresh" button? Do they understand that individuals in one model are not related at all to individuals with the same name in the next?
7	Do users spend much time switching between models using the history buttons and then unfolding the graph every time?

Table 2: Table of Concerns

Concerns 1 and 2 relate more to the general concern about model exploration *per se*, and Concerns 3 to 7 more to the one about the user interface. The best we could expect was, of course, a good evaluation for either set of concerns. The worst, on the other hand, would have been good marks for the UI and bad ones for model exploration, which would have suggested that the prototype was only a good implementation of a bad paradigm.

8.2 Choosing Goals and Concerns

In a full-blown evaluation, e.g. for a commercial software product, the number of our goals and concerns would have suggested to carry out more than one study, each focusing on testing only a subset of them. Due to the time and resource constraints on this project, and because we were only testing a prototype to get some feedback about the general direction we were going, we could not do that.

In the end, we decided instead to test only those goals in greater depth we had the greatest hopes to get positive results for. On the other hand, we decided to test all our concerns—albeit less thoroughly. The reason for this was that we wanted to get specific data on how useful model exploration was for specific tasks, and general data on how difficult it was to use SuperModel.

We discarded for testing Goal no. 2 because subsumptions between named classes can be tested by every ontology editor which can access a reasoner and show an inferred subsumption hierarchy. In some cases where one wants to check subsumption between complex concept descriptions, a very well-designed version of SuperModel could maybe be as good and possibly a bit more intuitive than the alternative approach of inserting test classes into the ontology and checking subsumption between them. Since that was speculation and since SuperModel was not that sophisticated yet, we left this for another time.

Testing whether SuperModel helps understand non-entailments, Goal no. 4, is a touchy issue. On the one hand, explaining non-entailments was one thing that motivated this project in the first place. On the other hand, unfortunately, there is no prior work on tools helping understand non-entailments. Thus we lacked both an established methodology to test improvement of understanding, and results from prior attempts with

which to compare SuperModel's performance. However, a generally positive outcome of this study could raise hopes for model exploration being beneficial in this field as well and encourage further investigation.

Initially, we planned on testing only Goal no. 1 in depth, and Goals 3 and 5 only opportunistically along the way. In particular, we planned to have a short first part of the study which was mainly to get the participants used to SuperModel. In this first part we were going to gather some data about Goal no. 3, and see what data we could get for Goal no. 5 in the second part. In our dry-runs, however, we discovered that the first part actually produced encouraging results for Goal no. 3 which made us extend it and gather more detailed data.

We did not expect to be able to design a test which would be able to generate any hard evidence for or against Goal no. 5 being accomplished. However, by simply asking the participants to talk aloud whilst going through the test and observing what they did and said, and by asking them to report on how much they believe they had learned about the ontology in the process of solving the tasks, we hoped some weak indication of whether Goal no. 5 was achieved could be obtained rather cheaply.

8.3 Deciding on and Recruiting Participants

As the audience of ontology authoring tools in general is small in comparison to other products' audiences, and as resources for this study were limited, the choice of subjects was quite simple; they were to be mainly researchers in the field of ontology design and description logics, and they were recruited by sending out e-mails to researchers we knew and who matched the profile: the focus was on people who had some grasp of ontologies, logics, and models, as those are the ones for whom SuperModel had the greatest probability of being useful.

We aimed for a number of subjects between ten and twenty, again mainly determined by the scope of the study. This was more than the recommended number of six to twelve subjects for a full-blown user study [12]: instead of day-long experiments in which every detail of the software is tested—not unusual in user studies of commercial software products—we just wanted to conduct a short study to get a general idea of whether SuperModel was useful and for whom. Thus, we could afford to have shorter experiments and more participants.

In order to be able to differentiate the test results by groups later on, subjects were asked to fill out a pre-test questionnaire (cf. Appendix C).

8.4 Experiments

As explained in Section 8.2, we conducted our experiment in two parts. Originally, we wanted to give the participants the opportunity to familiarize themselves with SuperModel in a warm-up phase before the actual experiment. We wanted to gather some

concern-related data during that phase because we believed some problems could be observed best before the participants had gotten used to them.

In that warm-up phase, we wanted to give the participants a short tutorial on how to use SuperModel to answer a few simple questions about an ontology. During the dry-runs of our study, we discovered that the tutorial went successful enough to gather some subjective data for testing Goal no. 3, which led us to re-dedicate the former warm-up phase as a first experiment.

8.4.1 Task 1

In order to let the participants get a feel for SuperModel's user interface and the way it could be used to learn something about an ontology, the first stage of the experiment started with a short demonstration of how a question about the ontology might be answered using SuperModel, and went on to ask the participants to try to answer similar questions.

Instead of using an already existing ontology, we designed our own toy ontology to make sure the participants had no prior experience with it. The theme of the toy ontology revolved around the computer game NetHack[16]. It included concepts for `Species` (`Human`, `Elf`, `Dwarf`...), `Roles` (professions) (`Knight`, `Archaeologist`, `Barbarian`), `Weapons` (`BroadSword`, `Axe`, `BullWhip`), `SkillLevels` (`Basic`, `Skilled`, `Expert`, `Master`), and others.

It also had two roles: `hasSkillLevel` and `canWield`.

What the ontology modeled was the generation of game characters: the ontology axioms interacted such that a character always consisted of an individual which was an instance of both a subconcept of `Species` and `Role`, as in a `Dwarf Knight` or a `Human Barbarian`. It had to have a `hasSkillLevel` successor and could optionally have a `canWield` successor which was a `Weapon`. Constraints on the species, professions, weapons, and auxiliary concepts made sure that certain weapons could only be wielded by characters with certain species/profession combinations who had a certain skill level.

The first stage of the experiment went as follows: the experimenter briefly explained SuperModel's UI with the help of a printed-out screen shot, and gave a short overview of the toy ontology's asserted concept hierarchy and roles. He then showed the participant how a certain question about the ontology could be answered using SuperModel. The question was:

> "What skill level does a `Barbarian` need to have to be able to wield a `MorningStar`?"

After the demonstration and after any questions regarding the procedure had been answered, participants were asked to answer the following similar questions using SuperModel:

16. http://www.nethack.org

1. "What `Role` (`Archaeologist`, `Knight`, `Barbarian`) does an `Orc` have to be to be able to wield a `BullWhip`?"
2. "By what `Roles` can a `Halberd` be wielded?"
3. "What are the `Weapons` that can be wielded by a `Knight` that is not an `Expert` (i.e. does not have a `SkillLevel` `Expert`)?"

8.4.2 Task 2

As explained above, the second stage of the experiment focused on testing Goal no. 1, i.e. whether it could help understand an entailment. Now, there are other tools out there which do that and the point of adding model exploration to the tool box is to improve on what is already there. Thus, the obligation of testing its usefulness was testing its usefulness in comparison to something already established.

Perhaps the most firmly established tool for understanding entailments are Justifications [30]. Protégé 4 offers Justifications in many places where inferences are shown: for inferred subsumptions, unsatisfiable classes, role relationships between individuals, etc. When we refer to Justifications in this section, we mean Justifications as implemented in Protégé 4.

One could go about comparing SuperModel to Justifications in two ways: first, one could try and see whether it is easier to understand an entailment using SuperModel or Justifications. However, it is was unclear whether SuperModel (or even model exploration in general) could be used efficiently to understand an entailment at all. What we hoped for when designing the study was instead that SuperModel would prove a good auxiliary tool which could improve the performance of other tools, especially Justifications.

As a consequence, what was tested was whether Justifications could be understood more easily if accompanied by SuperModel. One way of doing this would have been giving some participants only Justifications and others Justifications and SuperModel to figure out the reasons for entailments whilst measuring the time needed for understanding and ratio of success in the two conditions. Alternatively, we could have given the same participants SuperModel and Justifications for some entailments and both for others.

The type of quantitative data gained this way, such as the difference in time to understand a Justification with or without the help of SuperModel, could have been an indicator of usefulness. With an optimized user interface and enough samples one could have hoped to get encouraging results even if the total number of solved tasks facilitated by SuperModel had not been convincing on its own.

However, in the preliminary state it was in, it was not actually likely that SuperModel was going to speed up understanding. It was far more probable that the mental context switch between understanding an entailment from a Justification to understanding it from SuperModel and the possibly cumbersome user interface would have slowed down the process.

Because of that and because this procedure would have been more than a bit wasteful with participants' and experimenters' time, a different approach was pursued in this study: every participant was presented one entailment and a Justification at a time and if it was too hard for them to understand, they were additionally given SuperModel. This way, a high number of entailments that were not understood only with Justifications, but, after a reasonably short extra amount of time, with Justifications and SuperModel, was going to be an indicator of the usefulness of model exploration.

8.5 Measuring Usability

8.5.1 Finding Measures for Goals and Concerns

This section develops the measures which needed to be recorded in the experiment from the goals and concerns selected above. The item numbers refer to the entries in Tables 3 and 4, which group them by the tasks in which they are collected.

Goal no. 1, "SuperModel can be used to understand the reason for a given entailment", was going to be achieved if many cases where Justifications could not be understood on their own could be understood using SuperModel (items 2.00, 2.03).

Additional support for this goal was going to be found if the time spent with SuperModel was going to be short compared to the time spent without it on a Justification that was finally understood (items 2.01, 2.02); when participants were confident that they did understand (item 2.04), stated that SuperModel helped understand (2.06); and if participants felt model exploration was in principle a good tool for this task (item 3.02).

Goal no. 3, "SuperModel can help to find out, given a concept `C`, what role successors an instance of `C` must/may/may not have and which concepts they must/may/may not be instances of", is answered positively if participants were able to answer the questions asked in Task 1 in a reasonable amount of time, liked using SuperModel for this task, and would not have preferred using some other tool instead (items 1.01, 1.02, 1.03).

Goal no. 5, "When using SuperModel, a user can gain knowledge about an ontology circumstantial to the task at hand.", was going to be supported by the data if participants reported they gained insight (item 2.12); and expressed the feeling of understanding while using SuperModel (items 1.12, 2.11).

Concern no. 1, "Can users identify the constraints they have to add to find out a specific piece of information?", was going to be answered positively, if they completed Task 1 successfully and did not need much time for it (item 1.01).

Concern no. 2, "Are users confused by the fact that transitive role relationships are not shown in the model?", was answered positively if they said they did (items 1.09, 2.07).

Concern no. 3, "Do users understand the functions of the UI controls?", is not met if wrong buttons are clicked often (items 1.04, 2.08, 1.10, 2.09).

Concern no. 4, "Are users able to add the constraints they want?", is addressed by data items 1.06, 1.11, 2.10.

Concern no. 5, "Do users understand the workflow of SuperModel?", is answered negatively if the experimenter notes many wrong conclusions from the model (data item 1.07).

Concern no. 6, "Do users understand that individuals in one model are unrelated to those in the next?", is answered by item 1.08.

Concern no. 7, "Do users spend much time switching between models and unfolding the graph?", is answered by item 1.05.

What data		**How collected**
1.01	Time to answer questions about the toy ontology	logging
1.02	Whether participants liked using SuperModel for Task 1	questionnaire
1.03	Whether participants would have preferred another tool for Task 1	experimenter
1.04	How often wrong buttons were clicked	experimenter
1.05	How often subjects use the history buttons or select the same two concepts again and again during one task	logger
1.06	Whether specifying a model was started over due to some error	experimenter
1.07	Whether subjects display a misunderstanding of information on the screen when thinking aloud	experimenter
1.08	Whether they try to identify individuals in one model in the next by their arbitrary names	experimenter
1.09	Whether subjects are confused by missing information	survey
1.10	Whether participants express frustration due to wrong choice of button	experimenter
1.11	Whether participants express frustration due to wrong action	experimenter

Table ctd. on next page

What data		How collected
1.12	Whether participants express understanding while using SuperModel ("Now I see!", "Oh, that's interesting.", etc.)	experimenter
1.13	Whether participants found the task realistic	survey

Table 3: Data Collected in Task 1

What data		How collected
2.00	Whether a Justification was understood by itself, or with SuperModel activated	logging
2.01	Time needed until a Justification is given up on or understood w/out SuperModel	logging
2.02	Time needed until a Justification is given up on or understood with SuperModel	logging
2.03	Whether a Justification was understood or given up on	experimenter
2.04	Whether a Justification could be explained convincingly	experimenter
2.05	How confident in their understanding subjects report they are	survey
2.06	Whether subjects report that it was SuperModel that gave them the clues that lead to solutions and insight	survey
2.07	Whether subjects are confused by missing information	survey
2.08	How often wrong buttons are clicked	experimenter
2.09	Whether participants express frustration due to wrong choice of button	experimenter
2.10	Whether participants express frustration due to wrong action	experimenter
2.11	Whether participants express understanding ("Now I see!", "Oh, that's interesting.", etc.)	experimenter
2.12	Insight reported by the subjects	survey

Table 4: Data Collected in Task 2

What data		How collected
3.00	Usability of SuperModel	survey
3.01	Whether participants found SuperModel useful for understanding entailments	survey

Table ctd. on next page

What data		How collected
3.02	Whether participants believed model exploration in general may be useful for understanding entailments	survey
3.03	Whether participants believed model exploration in general may be useful for something else	survey

Table 5: Data Collected After the Test

A few data items did not relate directly to any of the specific goals and concerns. Whether the questions asked in Task 1 seemed realistic to the participants (item 1.13) was part of the post-task questionnaire because the examples were artificial and we wanted to get a hint towards the external validity of that task.

How usable SuperModel seemed (item 3.00) was related to the general, unspecified concern that SuperModel's UI may have been too cumbersome to use to get good results. This was to catch any specific usability issues we might not have foreseen and tested for.

Similarly, items 3.01 and 3.02 were to cross-check internal validity of our experiment by asking generally about how useful SuperModel and model exploration seemed to the participants; different outcomes for the two items were going to give rise to an investigation of our hypothesis that model exploration could be tested by testing SuperModel. An outcome of item 3.01 which was inconsistent with the rest of the study was going to suggest that we had been collecting the wrong data.

Whether participants believed model exploration in general could be useful for something else (item 3.03) was admittedly a stab in the dark: since many of our participants had a lot of experience working with ontologies, we hoped they might come up with possible applications we had not seen ourselves—specific goals for future user studies—which could be investigated in future research.

For similar reasons we gave the participants the opportunity to leave general comments on each one of the questionnaires.

8.5.2 Criteria

Before a user study is conducted, criteria for any quantitative measures, i.e. counts of events, times etc., need to be decided upon; what is going to be counted as a good result for a data item has to be set beforehand.

If a piece of software has been tested before, measures from the last test can be used as a basis for criteria, so improvement between versions can be measured. Alternatively, corresponding measures from competing products can be used for comparison. If neither measures from earlier versions nor from competing products are available, estimations of acceptable outcomes have to serve as criteria [12].

Since SuperModel had not been evaluated before, and because there were no competing implementations of model exploration systems, we had to guess ourselves which times, counts etc. users would normally find acceptable.

1.01, the time to answer a question, obviously depends on the question. Given that the participants were going to be new to using SuperModel, we fixed five minutes as the maximum time we deemed acceptable to answer the simple questions asked in Task 1.

1.04, the number of wrong clicks in Task 1, should, in a software *product* of this size, not exceed five or six per participant.

For the same measure in Task 2, 2.08, this number should be less: more than two wrong UI choices should be rare.

1.05, if users switch back and forth between the same models via the history buttons, or select the same two classes again and again, the effort of getting their bearings and unfold the graph again every time can become tiresome. More than four times should be rare.

2.00, *2.01*, *2.02*, *2.03*, *2.04*, *2.05* are evaluated together: if participants spend some time thinking about an entailment without SuperModel and give up, then see SuperModel and think more than about a quarter of the first interval using Justification and SuperModel, their success may be due to model exploration or simply due to more time.

We thus consider data sets as especially supporting model exploration, if an entailment was understood after seeing SuperModel and if the additional time was less than a fifth of the entire time spent on that entailment.

8.6 Test Materials

8.6.1 Legal Materials

In user studies, as in all experiments, quasi-experiments, surveys, etc. involving human participants, great care must be taken to ensure that the participants are not harmed, physically or mentally. This is most of all an ethical issue, but also a legal one, as the party conducting the study is responsible for the participants. Thus, the preparation for a study must follow the concept of minimal risk [12], which states that

> "the probability and magnitude of harm or discomfort anticipated in the test are not greater, in and of themselves, ordinarily encountered in daily life or during the performance of routine physical or psychological examinations or tests."

We could not see any danger of physical harm in participating in our study. The only (distant) psychological danger we did see was that of participants being put into a situa-

tion where they felt pressure to perform well, or were afraid to make a bad impression on the experimenter, their peers, or their superiors.

To address these ethical and legal concerns, we sent out, to every potential participant, a Participant Information Sheet (cf. Appendix A). This Participant Information Sheet included a statement assuring the personal and test data was to be kept anonymous and explaining that it was not the participant being tested but the software, and a declaration that all of the test was purely voluntary. It also explained the procedure of the test. All of this was done to make sure only those participated in our test who did not see any risk they would not have wanted to take.

Right before the test, after explaining how it was going to be conducted, we asked the participant to fill out a consent form (cf. Appendix B), which stated that they had been informed of the procedure of the test and of their rights.

Although little risk was to be expected of a study like this it is worth mentioning that our precautions were justified by the fact that one potential participant actually backed out of the test after previously agreeing to participate when he had read the Participant Information Sheet.

8.6.2 Questionnaires

Some of the data listed in Section 8.5.1 is gathered in questionnaires filled out by the participants throughout the test. There was a pre-test questionnaire, a post-task questionnaire after Task 1, questionnaires after every round of Task 2, and a post-test questionnaire.

The pre-test questionnaire was concerned with gathering information about the participants' background and skills in order to later be able to distinguish between the different user groups and interpret the results accordingly (cf. Appendix C).

The post-task questionnaire asked the participants about how they experienced the task and how they liked using SuperModel. This information is mainly to do with Goal no. 3 (cf. Appendix D).

The intermittent questionnaires, presented between the problems of Task 2, all took the same form and were presented as part of the test environment program. They asked for information about how they experienced working on the last Justification and how useful SuperModel seemed in understanding it.

The post-test questionnaire was to gather general information, objective, mostly qualitative measures, mainly to cross-check the internal validity of our experiment (cf. Appendix E).

Obviously, the test environment could not only have presented the intermittent questionnaires, but also the pre- and post-test, and post-task questionnaires, relieving the experimenter from the task of having to transfer the data from paper to computer. The reasons they were done on paper are psychological: after entering the test room and having the test procedure, etc. explained to them, it seemed a good idea to have the

participants quietly fill out a paper questionnaire so they could settle down and get comfortable in the environment before being confronted with the test program. The paper surveys after Task 1 and after the experiment were to lead them to step back from the software they were to give their thoughts on and hopefully allow them to think about it more objectively and with a broader perspective.

8.6.3 Justifications

The Justifications given to participants to try to understand—i.e. the mini-ontologies containing all and only the axioms jointly implying a certain entailment—were kindly made available by Matthew Horridge, (co-)author of e.g. [18, 43]. For a study (not yet part of published work) on the presentation of Justifications, he selected entailments from various published ontologies.

From the results of his study, it was possible to rate the difficulty of understanding these entailments. They were put into categories, from 'very easy' over 'easy', 'neither', and 'difficult' to 'impossible'

We selected for our study the 44 entailments that were at least rated 'difficult'. We also added five hand-crafted examples that we expected to be difficult to understand and good to be visualized with SuperModel; these tended to include many role axioms and concepts with big models.

All entailments were obfuscated, meaning all concepts and roles were given generic names like `C1`, `C2`..., and `p1`, `p2`..., so the participants did not recognize them in case they had seen them during their work, and so they could not make inferences from the meaning of concept names and role names in the real world.

8.7 Test Results

8.7.1 Demographics

In this study, a sample of twelve participants was tested. We directed the invitations to the study to people who had some experience with ontologies—authoring and description logic aspects alike. The a posteriori information we can gather from the pre-test questionnaires gives us a more detailed picture: all participants reported they had some experience with at least two ontology editors. Ten out of twelve rated themselves Adepts in the use of at least one ontology editor and eight ticked "Expert" on the questionnaire.

Eight participants reported they were formal logic "Experts", the rest replied they were "Adepts": none of our participants thought they were a "Novice". Similarly, our sample included seven "Experts" and four "Adepts" in the field of model-theoretic semantics of DLs and OWL, the only difference being one formal logic "Expert" rating him/herself a "Novice" in model theory.

Interestingly, there is a moderate, negative correlation of -0.46 between the reported levels of experience with formal logics and ontology editors: all participants of the study

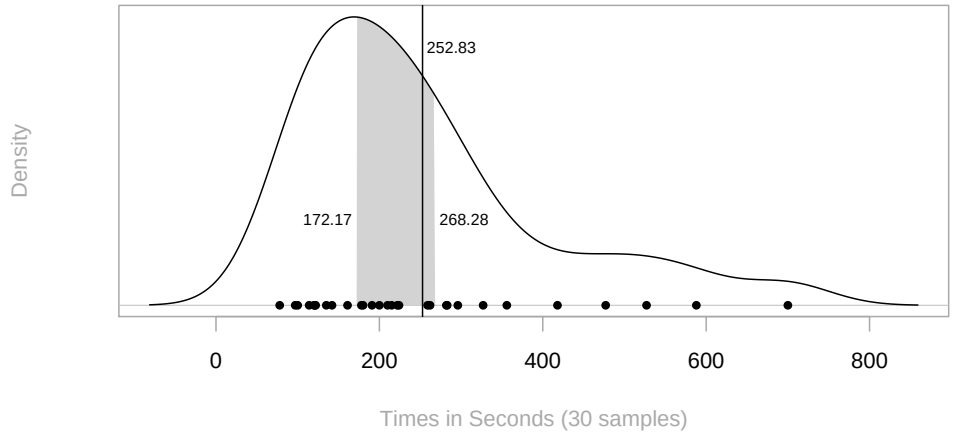


Figure 17: Est. Density and Middle Tercile, and Average of Time needed in Task 1

were self-reported experts of formal logics or some ontology editor, but there was a group of four people who were not experts of formal logics, one group of four who were not expert users of any ontology editor, and a last group covering participants who were both, partitioning the set of participants into three equally-sized subgroups.

8.7.2 Task 1

The first task of the study went fairly successfully: out of the twelve participants, only two could not answer all of the questions presented to them satisfactorily. Only two reported they did not enjoy using SuperModel for the tasks presented to them in this phase of the study. All but one participant found the questions they were asked resembled questions they could ask themselves in a real-life situation; the remaining one found at least "Some of them" realistic.

In almost two-thirds (74.19%) of the cases, participants needed less than five minutes to complete the tasks (data item 1.01). On average, participants needed four minutes and thirteen seconds (252.83 seconds). An estimated probability density function [41] for the distribution of times of all three tasks is visualized in Fig. 17. Its middle tercile lies between about three minutes (172.17 seconds) and four and a half minutes (268.28 seconds). The distribution varied considerably between tasks:

Question No.	Middle Tercile	Average Time	No. of Samples
1	149.85s–274.84s	262.55s	11
2	127.93s–206.60s	220.09s	11
3	222.66s–291.27s	284.50s	8

A minority, albeit a strong minority of five participants, reported they would have preferred answering the questions some other way (item 1.03), specifying either queries in some query language or pen, paper and the pure axioms as their preferred way of answering these questions. Four out of these five classified themselves as "Experts" in formal logics as oppose to four out of the seven who preferred SuperModel. At the same time, three of them did not classify themselves as "Experts" in the use of any ontology editor and two not even as "Adepts"—only one of the other seven only reported he/she was an "Adept" user of two ontology editors. Interestingly, the same group contains the only two subjects that did not complete all tasks in this phase successfully and the two who did not enjoy using SuperModel (item 1.02, cf. Fig. 18).

The counts for wrong buttons, start-overs, confusion of individuals between models, and expression of understanding as well as frustration of any kind (items 1.04, 1.06, 1.08, 1.12, 1.10, 1.11) were negligible. Participants were never confused because information was missing (item 1.09).

There were eight samples out of the thirty in which the history buttons were used (item 1.05) more than the four times deemed acceptable in Section 8.5.2 and eight where the experimenter observed some misunderstanding of the display (item 1.07).

8.7.3 Task 2

In Task 2, 58 samples were collected, i.e. on average, every participant worked on almost five tasks. In the preparation for the study, we hoped that a considerable number of tasks would be difficult enough for participants to fail in the first attempt. This was accomplished in as much as SuperModel was activated 32 times (55.2%, item 2.00).

17 Justifications were given up on and skipped, leaving 15 cases where participants were unable to understand some Justification initially, but could understand it after working on it using SuperModel (item 2.03).

In seven of these latter cases, the time spent with SuperModel was only a quarter of the time previously spent with the Justification alone as was our criterion set in Section 8.5.2 for an especially successful completion of a task in Task 2. In another four cases the Justification was understood after less than half the time spent without SuperModel. Fig. 19 shows a plot of an estimated density function for the ratio between time spent with SuperModel and time spent without it in cases where participants used SuperModel and succeeded in a task.

In the questionnaire following the understanding of a task after activating SuperModel, SuperModel was fully credited with giving the relevant clues only once. How-

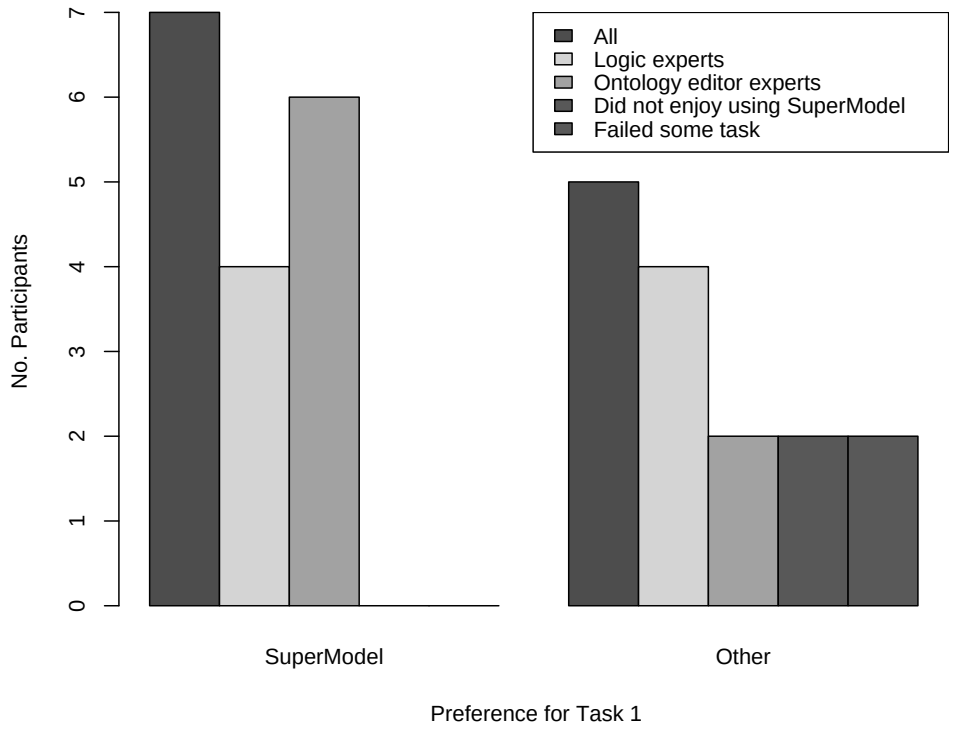


Figure 18: Comparing Participants Who Preferred SuperModel and Those Who Did Not

ever, in ten cases, participants reported it gave them "some of the clues" necessary to understand the entailment (item 2.06, cf. Fig. 20).

Surprisingly, some or all of the credit was given to SuperModel in all of the cases where the participants did succeed with SuperModel but took longer than a quarter of the time they had spent without SuperModel activated. Where they stayed below that bound, SuperModel was given any credit in only three out of seven cases.

Where a Justification could be understood without SuperModel, it took on average 145 seconds (item 2.01). SuperModel was activated on average after 113 seconds. The average time spent with SuperModel activated was 251 seconds—208 seconds in cases where the Justification could eventually be understood and 229 seconds where SuperModel was credited with giving some of the necessary clues (item 2.01).

Again, the counts for wrong clicks, expressions of frustration or understanding were negligible (items 2.08, 2.09, 2.10, 2.11). Also, the participants did not report any additional insight into the ontology (item 2.12). Only in one case was there confusion because of

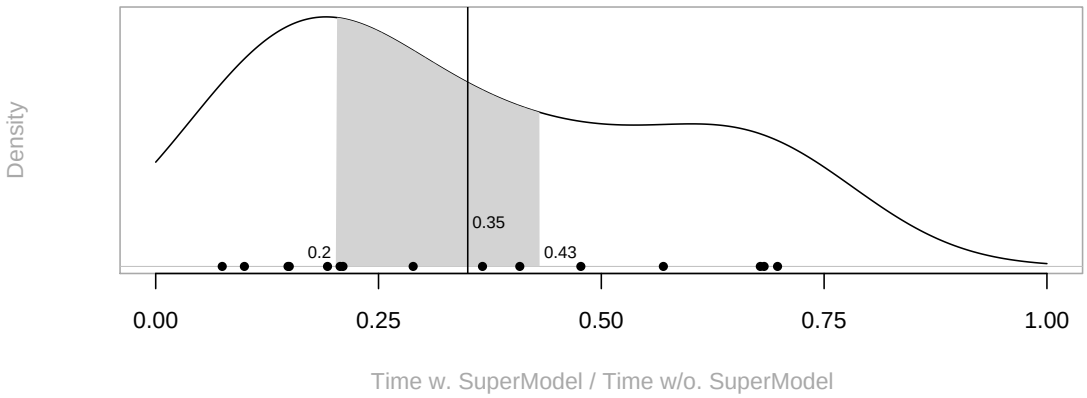


Figure 19: Est. Density and Middle Tercile, and Average

missing information (item 2.07).

8.7.4 Post-Test Questionnaire

Seven participants thought model exploration was definitely useful to understand entailments (item 3.01). Three of these were not sure (item 3.02). None thought either SuperModel or model exploration in general was useless and no-one believed SuperModel was definitely useful but maybe model exploration was not (cf. Fig 21).

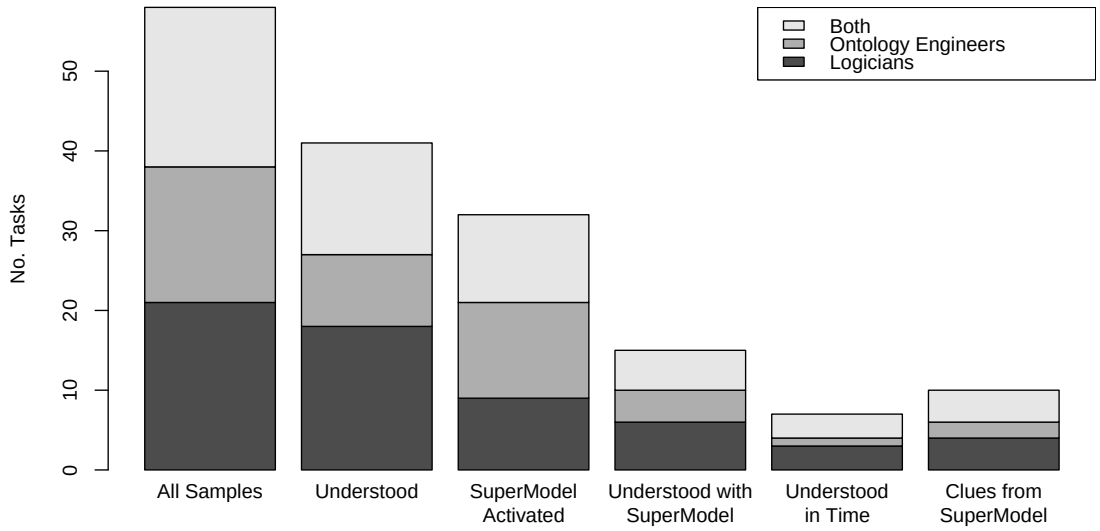
Eight believed model exploration was useful for something else, the rest was not sure.

Two participants thought SuperModel was difficult to use; three found it easy. Because of the scarce data, nothing conclusive can be said about those who found SuperModel difficult to use. Those who found it easy were both self-described experts of formal logics and the model theory of DLs and two of them were experts in using ontology editors, the remaining one was an adept.

8.8 Interpretation

The data for Task 1 of the experiment suggests that the kind of question asked in it can indeed be answered in acceptable time using SuperModel. Together with the demographic data it also suggests that the benefit from SuperModel increases with familiarity with ontology editors and decreases with knowledge about formal logics.

The reason for the former could be that users of ontology editors are used to graphical environments that actively support them in the understanding of ontologies.



All samples, solved samples, those in which SuperModel was activated, those that were understood after SuperModel was activated, those that were understood with SuperModel in acceptable time, and those that were understood with participants reporting they got the relevant clues from SuperModel.

Figure 20: Summary of Task II

The latter may be caused by experts of formal logics having developed their own strategies for extracting information from logical data like ontologies. Especially if they are familiar with the model theory of description logics, they may themselves engage in a mental activity similar to model exploration, but more adjusted to their personal ways of solving problems. For this reason, a model exploration system may not be needed as much for these people and might even be a distraction.

On the other hand, it seems like those who have a high exposure to ontological problems and little experience with their logical background, i.e. people who can appreciate the relevance of these problems and at the same time would presumably find it comparably difficult to envision models by themselves, can profit from model exploration in these tasks.

The fact that the questions asked in this task were attested high relevance for everyday ontology engineering lets us hope that our results for this part of the study carry over to real-life activities in this area.

The fact that, in Task 2, almost half of the cases where participants could not understand a Justification on its own were understood with SuperModel is encouraging.

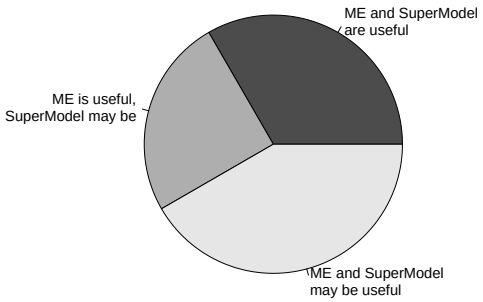


Figure 21: Reported Usefulness of Model Exploration vs. SuperModel

However, in more than half of those cases, they spent considerably more time with SuperModel than we hoped they would, and only in two thirds of the cases they credited SuperModel with giving them at least some of the clues they needed.

Since participants reported SuperModel helped them more in cases where overall it took them longer, we believe that, as we suspected in Section 8.4.2, SuperModel is indeed not a tool which speeds up the understanding of a Justification. Instead, it appears that it makes it possible to understand Justifications which are otherwise very hard or impossible to understand—the ones users have to think about the longest and where they are most likely to be willing to try out a new method of investigating a problem to get help.

The measures on usability of SuperModel's UI suggest that there is room for improvement, but also that the problems are not so grave as to make it difficult to use. This is consistent with the fact that most participants thought its usability was at least 'normal'. We thus believe that a similar study with an improved version of SuperModel would not yield dramatically different results.

We did not find any evidence for our Goal no. 5 being accomplished. The reason for this may have been that the ontologies were very simple and contained little information that could have been learned in addition to what was necessary. Another point may have been that the participants were too busy solving the tasks at hand to take in any peripheral information.

All in all it can be said that SuperModel and thus model exploration appears to be helpful for both applications, although the indication for using SuperModel to answer questions similar to the ones in Task 1 appears to be greater than for using it to facilitate

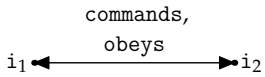


Figure 22: Unclear Command Structure

understanding of Justifications as in Task 2.

8.9 Comments and Suggestions

Our participants and colleagues gave us valuable feedback as answers to open-ended questions in our questionnaires and during informal discussions on model exploration. Some of the most frequent suggestions are discussed below—they indicate starting points for technical improvement of SuperModel and for further development of the notion of model exploration.

8.9.1 SuperModel

The graph-drawing mechanism SuperModel employs uses a force-directed layout of the nodes, which results in a somewhat random display: the node for the root individual is always located at the center of the graph pane initially, but the other nodes are not always in the same places. The simulated forces between nodes results in the graph quivering about and the same model excerpt looking quite different when seen a second time. This is confusing at times and in an improved version of SuperModel, we would like to implement a more deterministic layout mechanism.

The arrows going back and forth between nodes in the graph pane are labeled with the names of all the roles connecting the individuals for which the nodes stand. If the relationships between the individuals go in both directions, it is not apparent from the graph pane which role goes in which direction (see Fig. 22). It would be good to find a way of making this information available in the graph pane without wasting much space.

Individuals occurring in a model which do not occur in any ABox assertion are named automatically with hexadecimal numbers. Some participants suggested to somehow name them by the concepts they are instances of. Of course, these may be too many, and there may be more than one instance of a particular concept in a model. However, it might be possible in some cases to name individuals e.g. by the most specific concept they are instances of and number them where necessary.

8.9.2 Model Exploration

In the study and in discussions with our colleagues, the possibility of showing 'partial models' (*sic*) for unsatisfiable concepts came up repeatedly. The idea was motivated by the fact that SuperModel was very unhelpful for Justifications in which a great number

of the occurring concepts were unsatisfiable. Instead of stating that a concept was unsatisfiable, a model exploration system might display the completion graph as far as the reasoner was able to build it, and indicate the clash.

Another suggestion was specific to the use case in which model exploration and Justifications were used side by side: to understand a Justification, only the axioms in it are necessary. However, as it is, SuperModel uses the entire ontology for its purposes. Instead, the Justification itself, being just a set of axioms, might be used as the background ontology of model exploration.

This could in some cases eliminate unnecessary information from the display of SuperModel, and give rise to another mode of understanding Justifications with model exploration: by removing an axiom from a Justification, its entailment becomes unentailed. Especially in the case of Justifications for unsatisfiability, it could be useful to remove one axiom, explore the models of a previously unsatisfiable concept, and try to see how the removed axiom rules out these models.

9 Conclusion

9.1 Evaluation

In this work, we have introduced model generation, a technique for supporting ontology engineers in their understanding of ontologies. We have explained how structures similar to models can be extracted from tableau reasoners and how the specific reasoning algorithms affect the models and additional information found in these structures. We have discussed the issues and choices involved in implementing model exploration and produced a prototype whose usefulness was assessed in a user study.

The results of this study are encouraging: they suggest that model exploration can indeed help users answer certain questions about an ontology and, to a certain degree, to understand Justifications.

The study has also identified starting points for future work both on our implementation and on model exploration in general.

9.2 Future Work

9.2.1 On the Plug-in

We would like to further develop SuperModel to improve usability, make it more independent of the specific reasoner used, and to enable different styles of use.

In particular, we would like to improve the visualization and the graph interaction and layout mechanisms, and streamline SuperModel for use with Justifications as described in Sections 8.9.1 and 8.9.2, and improve the overall software architecture and implementation.

9.2.2 On Testing Usefulness

Although it did yield valuable results, the study we report on in this work is only a pilot study in scope. To get more reliable and fine-grained results, it would be necessary to conduct a study with longer experiments and more resources (technical and human) which would yield more and more specific data.

However, before conducting a full-blown study, it would be in order to identify more goals to test, design suitable experiments, and conduct one or more additional pilot studies to get a better idea of what are the probable strengths of model exploration and how to confirm them. Especially a means to test improvement of understanding of non-entailments would be interesting.

9.2.3 On Model Exploration

On the theoretical side, we would like to explore the possibilities and options for handling transitive roles and general role inclusion axioms. Also, it would be very interesting to investigate means to give users, for a piece of information they find while exploring a model, an explanation of why the reasoner put this information in the completion graph and, possibly, which other options there were.

There are generally two approaches to those goals: the first one is modifying the reasoner to include more data about the reasoning process in the completion graph. This data could then be used to infer the kinds of information necessary for our goals. The other approach is a postmortem analysis of the graph in which the relevant information is added. An advantage of the latter approach could be greater independence of the reasoner and changes thereof. However, it is unclear to the authors so far, which approach is most practicable, fruitful, etc.

Another thing worth researching is whether the particular models generated by tableau algorithms are good models for model exploration. In some cases it may be useful to generate finite models, although of course this is not in general possible for many DLs.

References

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

[2] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics as Ontology Languages for the Semantic Web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday*, number 2605 in Lecture Notes in Artificial Intelligence, pages 228–248. Springer, 2005.

[3] Franz Baader, Carsten Lutz, and Boris Motik, editors. *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[4] Franz Baader and Werner Nutt. Basic Description Logics. In Baader et al. [1], chapter 2.

[5] Franz Baader and Ulrike Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:2001, 2000.

[6] Martins Barinskis and Guntis Barzdins. Satisfiability Model Visualization Plugin for Deep Consistency Checking of OWL Ontologies. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[7] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn A. Stein. OWL Web Ontology Language Reference. Technical report, W3C.

[8] Alex Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description Logics for Data Bases. In Baader et al. [1], chapter 16.

[9] Sebastian Brandt and Anni-Yasmin Turhan. Using Non-standard Inferences in Description Logics—what does it buy me? In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (KIDLWS'01)*, number 44 in CEUR-WS, Vienna, Austria, September 2001. RWTH Aachen. Proceedings online available from `http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-44/`.

[10] Diego Calvanese. Finite Model Reasoning in Description Logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 282–303. Morgan Kaufmann, 1996.

[11] Diego Calvanese and Giuseppe De Giacomo. Configuration. In Baader et al. [1], chapter 5.

[12] Joseph S. Dumas and Janice C. Redish. *A Practical Guide to Usability Testing*. Intellect Books, Exeter, UK, 1999.

[13] Brian Gaines. Designing Visual Languages for Description Logics. *Journal of Logic, Language and Information*, 2008.

[14] Miguel Garcia, Alissa Kaplunova, and Ralf Möller. Model Generation in Description Logics: What Can We Learn From Software Engineering? Technical report, Institute for Software Systems (STS), Hamburg University of Technology, Germany, 2007. See http://www.sts.tu-harburg.de/tech-reports/papers.html.

[15] Thomas R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[16] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *Int.J.Hum.-Comput.Stud.*, 43(5-6):625–640, 1995.

[17] Volker Haarslev and Ralf Möller. Racer: A Core Inference Engine for the Semantic Web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20*, pages 27–36, 2003.

[18] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and Precise Justifications in OWL. In Sheth et al. [47], pages 323–338.

[19] Matthew Horridge, Dmitry Tsarkov, and Timothy Redmond. Supporting Early Adoption of OWL 1.1 with Protege-OWL and FaCT++. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWLED*, number 216 in CEUR Workshop Proceedings, Athens, Georgia, USA, November10–11 2006. CEUR-WS.org.

[20] Ian Horrocks. Implementation and Optimisation Techniques. In Baader et al. [1], chapter 9.

[21] Ian Horrocks, Ullrich Hustadt, Ulrike Sattler, and Renate Schmidt. Computational Modal Logic. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, chapter 4, pages 181–245. Elsevier, 2006.

[22] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

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

[24] Ian Horrocks and Ulrike Sattler. Decidability of $\mathcal{SHIQ}$ with Complex Role Inclusion Axioms. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 343–348. Morgan Kaufmann, Los Altos, 2003.

[25] Ian Horrocks and Ulrike Sattler. Decidability of $\mathcal{SHIQ}$ with Complex Role Inclusion Axioms. *Artificial Intelligence*, 160(1–2):79–104, December 2004.

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

[27] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In David McAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2000.

[28] Ernesto Jiménez-Ruiz, Bernardo Cuenca-Grau, Ulrike Sattler, Thomas Schneider, and Rafael Berlanga Llavori. Safe and Economic Re-Use of Ontologies: A Logic-Based Methodology and Tool Support. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2008.

[29] Kaarel Kaljurand. ACE View—an ontology and rule editor based on Attempto Controlled English. In *5th OWL Experiences and Directions Workshop (OWLED 2008)*, Karlsruhe, Germany, October 2008.

[30] Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, College Park, MD, USA, 2006. Adviser-James Hendler.

[31] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A Web Ontology Editing Browser. *Journal of Web Semantics*, 4:2005, 2005.

[32] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods—a survey. *ACM Comput. Surv.*, 39(4):10, 2007.

[33] Alexander Maedche and Steffen Staab. Mining Ontologies from Text. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, volume 1937 of *Lecture Notes in Computer Science*, pages 169–189. Springer, 2000.

[34] William McCune. Mace4 Reference Manual and Guide. *CoRR*, cs.SC/0310055, 2003. informal publication.

[35] Boris Motik, Rob Shearer, and Ian Horrocks. A Hypertableau Calculus for $\mathcal{SHIQ}$. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR (http://ceur-ws.org/)*, 2007.

[36] Daniele Nardi and Ronald J. Brachman. An Introduction to Description Logics. In Baader et al. [1], chapter 1.

[37] Fernando Náufel do Amaral. Visualizing the Semantics (Not the Syntax) of Concept Descriptions. In *VI Workshop em Tecnologia da Informação e da Linguagem Humana (TIL 2008), Vila Velha, ES, Brazil*, 2008.

[38] Fernando Náufel do Amaral and Carlos Bazilio Martins. Visualization of Description Logic Models. In Baader et al. [3].

[39] Olaf Noppens and Thorsten Liebig. Interactive Visualization of Large OWL Instance Sets. In *Proc. of the Third Int. Semantic Web User Interaction Workshop (SWUI 2006)*, Athens, GA, USA, November 2006.

[40] Colin Puleston, Bijan Parsia, James Cunningham, and Alan Rector. Integrating Object-Oriented and Ontological Representations: A Case Study in Java and OWL. In Sheth et al. [47], pages 130–145.

[41] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[42] Alan Rector. Medical Informatics. In Baader et al. [1], chapter 13.

[43] Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In *Engineering Knowledge in the Age of the SemanticWeb*, volume 3257 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2004.

[44] Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. All Elephants are Bigger than All Mice. In Baader et al. [3].

[45] Stefan Schlobach and Ronald Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 355–362, Acapulco, Mexico, 2003. Morgan Kaufmann.

[46] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[47] Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors. *The Semantic Web - ISWC*

2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings, volume 5318 of *Lecture Notes in Computer Science*. Springer, 2008.

[48] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

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

[50] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[51] Taowei Wang and Bijan Parsia. Ontology Performance Profiling and Model Examination: First Steps. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon J B Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 589–602, Berlin, Heidelberg, November 2007. Springer Verlag.

APPENDIX

A Participant Information Sheet

University of Manchester, School of Computer Science

Project Title: "Model Exploration To Support Understanding Of Ontologies"

Introduction: This study is to investigate whether interactive generation and exploration of models for ontologies can enhance the understanding of justifications of entailments that hold in them.

What you will be asked to do if you participate: This study consists of two stages, in both of which you will work with a tool, SuperModel, we developed to investigate model exploration.

In stage 1, you will be shown how to answer a simple question about an entailment in an ontology using SuperModel. After that, you will be asked to use the same technique you were shown to answer a different, but similar question.

In stage 2, you will be given justifications of entailments that hold in an ontology. You will be asked to try to understand these justifications and explain them to the experimenter. Some of these entailments are very hard to understand. If it is impossible to understand an entailment only with the given justification, you have the chance to additionally use SuperModel. If the entailment cannot be understood with the justification and SuperModel together, it will simply be skipped. In any case, the software will ask you to fill out a short questionnaire after every task.

The second stage will go on as long as you are comfortable with it. You can stop any time you like, in the first or second stage, during or after a task, without stating your reason and without detriment to yourself.

After the experiment, you will be asked to fill out another short questionnaire.

Will your data be anonymous? Yes. Apart from the pre-, inter-, and post-test questionnaires, the software will gather data about user-induced events happening in the test environment. As this is a test of the software and not of you, none of these data will be linked in any way to your name/identity.

Do you have to take part? No. Participation is completely voluntary. If you agree to participate and then change your mind before or during the test, you can withdraw immediately and, if you wish, your data will be deleted.

B Consent Form

University of Manchester, School of Computer Science

Project Title: "Model Exploration To Support Understanding Of Ontologies"

Participant's Statement: The investigator has explained to me the nature of the research and what I would be asked to do as a volunteer, and has given me my own copy of the volunteer information sheets, which I have read.

I consent to take part as a volunteer and I understand that I am free to withdraw at any time without giving any reason, and without detriment to myself.

Signed, Date, Name (block letters)

Experimenter's Statement: I confirm that I have fully explained the purpose and nature of the investigation and the risks involved

---

Signed, Date, Name (block letters)

C Pre-Test Questionnaire

Please answer the following questions. They help us understand the relationship between the test results and the background and skills of our participants.

Have you used any ontology editors before? Which ones?

Protégé 4:	☐ Expert	☐ Adept	☐ Novice
Swoop:	☐ Expert	☐ Adept	☐ Novice
TopBraid:	☐ Expert	☐ Adept	☐ Novice
others: (please name)			
_____	☐ Expert	☐ Adept	☐ Novice
_____	☐ Expert	☐ Adept	☐ Novice

Please rate your familiarity with formal logics such as First Order Logic, Description Logics (DLs), etc.
Are you an

☐ Expert

☐ Adept

☐ Novice

How familiar are you with entailment justifications/explanations/MUPS/MINA?

☐ Very familiar

☐ Familiar

☐ Unfamiliar

How knowledgeable are you with the model-theoretic semantics of DLs and OWL?
Are you

☐ Very familiar—You can reason with logical models

☐ Familiar with the basics—You know what a logical model is

☐ Not familiar

Please turn.

Have you used SuperModel before?

- ☐ No
- ☐ Yes, an early version
- ☐ Yes, a recent version

Do you have any further comments? They are greatly appreciated.

D Post-Task Questionnaire

Thank you so far! Before going on to the second part of our study, we would like you to answer a few questions about the tasks you were doing.

Did you enjoy using SuperModel for this task?

- ☐ Yes
- ☐ No

Would you have preferred to answer the questions that were given in a different way?

- ☐ Yes (How?) _____
- ☐ No

Could you imagine asking yourself questions like these about a real-life ontology?

- ☐ Yes
- ☐ Some of them
- ☐ No

Should we have asked any other questions that you suspect SuperModel could facilitate answering?

- ☐ Yes (Which ones?) _____
- ☐ I don't know
- ☐ No

Do you have any further comments? They are greatly appreciated.

_____

_____

_____

E Post-Test Questionnaire

Thank you for participating in this user study. There are a few more questions we would like to ask you to complete our data.

Please rate SuperModel's overall usability.

- ☐ Easy
- ☐ Normal
- ☐ Difficult

Do you think SuperModel is useful for understanding entailments?

- ☐ Yes
- ☐ Maybe
- ☐ No

Do you think exploring models for concepts in general is useful for understanding entailments?

- ☐ Yes
- ☐ Maybe
- ☐ No

Do you think exploring models for concepts is useful for something else?

- ☐ Yes
- ☐ Maybe
- ☐ No

If yes, what for?

Please turn.

Do you have any further comments? They are greatly appreciated.

When finished completing this survey, please be sure to ask the experimenter any outstanding questions you might have with regard to this test.