OntoTrack: Combining Browsing and Editing with Reasoning and Explaining for OWL Lite Ontologies

Thorsten Liebig and Olaf Noppens

Dept. of Artificial Intelligence
University of Ulm
D-89069 Ulm
{liebig|noppens}@informatik.uni-ulm.de

Abstract. OntoTrack is a new browsing and editing "in-one-view" ontology authoring tool that combines a hierarchical graphical layout and instant reasoning feedback for (the most rational fraction of) OWL Lite. OntoTrack provides an animated and zoomable view with context sensitive features like click-able miniature branches or selective detail views together with drag-and-drop editing. Each editing step is instantly synchronized with an external reasoner in order to provide appropriate graphical feedback about relevant modeling consequences. The most recent feature of OntoTrack is an on demand textual explanation for subsumption and equivalence between or unsatisfiability of classes. This paper describes the key features of the current implementation and discusses future work as well as some development issues.

1 Introduction

High quality ontologies are crucial for the Semantic Web [1]. Unfortunately, the task of building and maintaining mature ontologies turned out to be difficult even for KR experts. The analysis of three different efforts of formalizing knowledge for a given domain within the Halo project [2] may serve as a prime example here. This evaluation showed, that up to 75 % of the system failures were caused by modeling problems which in most cases were concerned with the act of writing down non-conflicting axioms [3]. In order to avoid those modeling problems the authors suggest to enforce development of convenient and interactive tools for building, maintaining and evaluating ontologies.

However, many ontology authoring tools currently use functionally disjunct interfaces for either editing, browsing, or reasoning with ontologies. Editing interfaces commonly use list-based representations for selection of classes or properties together with predefined forms and pop-up windows for manipulation in an additional display area. From a usability perspective, those interfaces inherently have substantial drawbacks concerning search and navigation speed as well as user orientation and editing efficiency in our opinion [4]. First, the number of visible classes is limited by the screen height requiring scrolling even for middle

S.A. McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp. 244–258, 2004.

sized ontologies. Second, because of the tree centered representation, multiple inheritance needs to be approximated with help of "cloned" classes appearing in the list of descendants of every superclass. Third, accessing or selecting other classes during editing temporally requires additional expand and contract style selection lists for a class hierarchy already on screen.

In order to compensate some of those deficits most editors include additional browsing interfaces based on layout techniques like nested interchangeable views [5], venn diagrams [6], spring embedding [7], or hyperbolic trees [8], [9]. However, the majority of those interfaces do not allow for substantial editing and are designed as view-only plugins in most cases. Furthermore, hyperbolic tree visualization is based on a mapping of euclidian to hyperbolic geometry typically resulting in translocation of nodes after each focus change [10]. In addition, hyperbolic trees have the disadvantage of "fisheye distortion" which makes it difficult to read off-center labels for example. Other visualization techniques like tree maps or venn diagrams heavily exploit nested graphical structurings overlapped with edges, obscuring the greater structure of ontologies in most cases. In our opinion, none of these layout techniques are well suited for typical ontology browsing tasks like comparison of different expansion paths concerning level depth or common ancestors.

OntoTrack implements a novel approach using one integrated view for straightforward browsing, manipulating, and understanding even large OWL Lite schemas (also called TBoxes in Description Logics). Within this view ontologies are layouted as directed root graphs according to their primary structuring relation, the subsumption relationship. OntoTrack provides animated expansion and de-expansion of class descendants, continuous zooming, panning and uses elaborated layout techniques like click-able miniature branches or selective detail views. At the same time OntoTrack allows for quite a number of editing features like mouse-over anchor buttons and graphical selections without switching into a special editing layout. In addition, every single editing step is send to an external reasoner in order to make implicit modeling consequences explicitly available for graphical feedback. OntoTrack is currently enhanced with an on demand component for textual explanation of subsumption and equivalence between or unsatisfiability of classes.

The remainder of this paper is organized as follows. The next section contains a detailed description of the ontology authoring tool OntoTrack. In section 3 we discuss some implementation issues and describe the system architecture as well as work in progress. We will end with an outlook about possible future extensions.

2 OntoTrack: A New Graphical Authoring Tool for OWL Lite$^-$

OntoTrack is a multi-platform Java application combining authoring of ontologies with reasoning about ontologies. As a consequence, running OntoTrack requires to have access to an appropriate reasoning system. Currently,

OntoTrack is adapted to the RACER server [11] via its TCP-based client interface.

The next subsection specifies the OWL language constructs OntoTrack is currently able to handle. For sake of broad usability and a clear and concise graphical representation we restrict this language to a sensible fraction of OWL Lite. This is followed by a more detailed explanation of OntoTracks browsing, editing, reasoning as well as explaining abilities.

2.1 OWL Lite$^-$: A Sensible Fraction of OWL Lite

OntoTrack is designed to be an authoring tool for ontology languages with an expressivity almost comparable to that of OWL Lite. The expressivity of OWL Lite is about that of the DL language $\mathcal{SHIF}(\mathbf{D})$ [12]. OWL Lite is a fraction of OWL DL aiming to provide a useful subset of language features, that are easier to present to naive users and relatively straightforward for tool developers to support [13]. In particular, OWL Lite excludes syntax constructs for unions, complements, and individuals in descriptions or class axioms. In addition, it limits cardinalities to either 0 or 1 and nested descriptions to concept identifiers. However, these restrictions come with relatively little loss in expressive power. With help of indirection and syntactical "tricks" all of OWL DL can be captured in OWL Lite except those descriptions containing either individuals or cardinalities greater than 1 [12]. Atomic negation for example, is easily simulated via complementary cardinality restrictions (e. g. $A \equiv (\geq 1\ r)$ and $negA \equiv (\leq 0\ r)$, where r is a new property). A general concept inclusion (GCI) is expressible by introducing two new classes with one complete and one partial definition for each class (e. g. $C \sqsubseteq D$ together with two complete class descriptions for C and D). A disjunction of two (or more) classes A and B can be simulated by negating the conjunction of the negation of A and B (following the laws of deMorgan and double negation). E. g. the expression $(A \sqcup B)$ is semantically equivalent to $negD$ where $negD \equiv (\leq 0\ q)$ and $D \equiv (negA \sqcap negB) \equiv (\geq 1\ q)$.

Simulating GCIs or disjunction in OWL Lite with help of multiple definitions for a single class identifier is kind of obscure and obviously conflicts with the design goal of OWL Lite. Very likely most of the above will rarely occur in real world OWL Lite ontologies. Instead, users will presumably use OWL DL when it comes to disjunction or GCIs for a certain application domain. Beyond that, there is no intuitive and unambiguous way for rendering classes with multiple, possibly mixed partial and complete, definitions. We therefore decided to restrict the usage of OWL Lite in a way we believe it was originally intended by the language designers itself (called OWL Lite$^-$ in the following). In particular, we restrict classes to be defined only once in a way that could be equally expressed by one single `owl:equivalentClass` or `rdfs:subClassOf` statement over the conjunction of the collection of superclasses and restrictions (i. e. the definition of a class A is $\mathsf{A} \sqsubseteq \mathsf{D}$ or $\mathsf{A} \equiv \mathsf{D}$ where A is an atomic name and has no other definition). In terms of the normative abstract syntax and semantic of OWL Lite [14] this prohibits the use of the *EquivalentClasses* axiom. As a result, GCIs and disjunction are not expressible within OWL Lite$^-$.

Within this context, it is worth mentioning that different analyses of online available ontologies (e. g. [15] or [16]) showed, that most users only exploit a very limited set of the available language constructs. More concrete, only a very small fraction of those ontologies would be outside the scope of the OWL Lite subset processable with OntoTrack.

2.2 Browsing

The primary structuring element of ontologies is the subsumption relationship between classes and properties. Consequently, OntoTrack layouts an ontology as a directed acyclic graph with either classes or properties as nodes and the subsumption relationship as directed edges either in top-down, left-right, bottom-up, or right-left orientation. The drawn hierarchy may vary from the explicitly given subclass statements of the loaded ontology source document. This is due to OntoTracks most important layout principle aiming to visualize the most relevant ontological commitments only. As a consequence, OntoTrack will only show direct subsumption relationships while hiding all those which are redundant due to the transitivity of this relationship. Furthermore, by taking modeling consequences explicitly into account, OntoTrack is sensitive not only to syntactically given but also to semantically implied subsumption relationships.

Figure 1 shows the OntoTrack application window. It solely consists of a menu bar, a search bar, and a combined browsing and manipulation area.

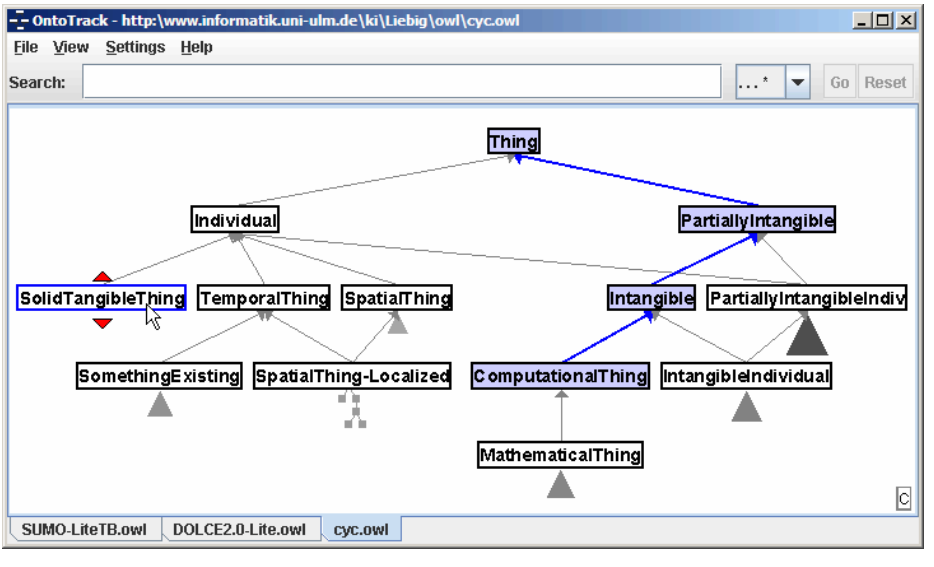

Fig. 1. Upper fraction of the OpenCyc ontology with triangle and miniature subbranches. Anchor buttons for SolidTangibleThing show up because of mouse over action.

OntoTrack can handle multiple ontologies in parallel, which are accessible via corresponding tabs at the bottom of the application window. The ontology displayed in Figure 1 is a fraction of the OpenCyc knowledge base (version 0.7) in top-down orientation. Note, that the subsumption path of the currently selected class (ComputationalThing) up to the OWL root class (Thing) is outlined by a darker node background and thicker edges in order to provide an optimal overview concerning depth and branching.

As an option, not expanded sub-branches of classes are rendered as triangles of varying length, width, and shading, approximating the depth, branching and number of subclasses (see class SomethingExisting in Figure 1 for an example). Depending on the number of descendants, a not expanded sub-branch will alternatively be symbolized as a miniature graph in order to give an overview about the structure of the sub-branch (see SpatialThing-Localized in Figure 1). The icons of miniature graphs are expandable on mouse click and show their class names via mouse-over tool tip. In addition, the whole ontology layout can continously be zoomed or panned simply by right button mouse-down movements.

The direct descendants of a class are expanded resp. de-expanded in an animated fashion on left mouse button click. OntoTracks ontology layout is driven by the expansion direction of the user. The layout strategy aims at "clustering" the descendants of a selected class (locally simulating an ordinary tree representation). This means that an ontology with multiple inheritance may end up with a different layout for exactly the same set of expanded classes – depending on the users expansion order.

OntoTrack always shows all (even implicit) direct subsumption relationships between classes. However, when dealing with classes having multiple ancestors it might be the case that some ancestors are not visible (because they are in not expanded branches). Those ancestors are drawn as clickable thumbnail classes in order to show the semantically correct hierarchy while minimizing the number of expanded classes at the same time. The class names of all thumbnail classes are accessible via tool tips. As an example, Figure 2 shows two thumbnail ancestors of class SpaceRegion appearing after expansion of class SpatialThing. OntoTrack will not expand them by default because of the assumption, that

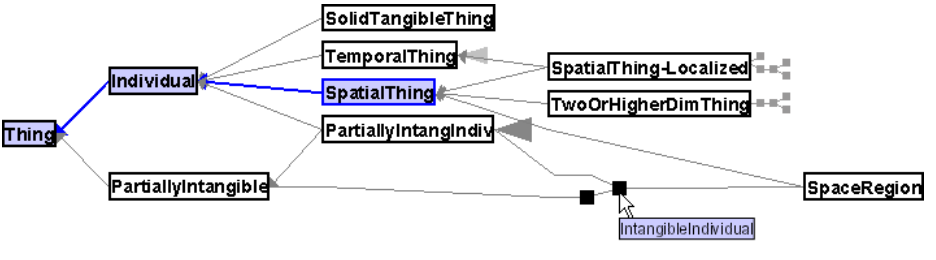

Fig. 2. Ancestor thumbnails of class SpaceRegion after expansion of class SpatialThing.

the user is much more interested in the descendants of SpatialThing (the user actually asked for) instead of their ancestors.

An auxiliary navigation feature of OntoTrack is the radar view, which will appear in the upper left corner of the workspace only if the ontology will exceed the available rendering area at its current scale factor. The radar view consists of a miniaturized graph of the current ontology expansion and an overlayed rectangle representing the current rendering area. The latter can be moved via mouse which will result in a corresponding relocation of the original view. Figure 9 shows a larger ontology together with a radar view.

OntoTrack also implements a string matching search with graphical highlighting based on the principles of the dynamic query approach [17]. Each modification of the search string directly results in an updated highlighting of matching parts of the ontology (expanded or not). Figure 3 shows the sub-string matches

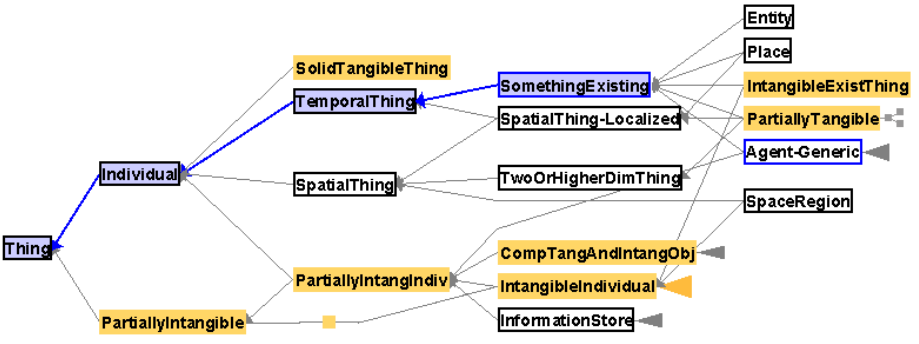

Fig. 3. Match highlighting for sub-string search for "ang" within class names.

for the search string "ang" within class identifiers. Note, that even thumbnails or triangle sub-branches are highlighted. Optionally, the user can fan out the ontology resulting in an expansion of all matching classes via one click.

2.3 Editing

Browsing and editing in OntoTrack is done within one single view. This allows to re-use already available navigation principles for the task of building and manipulating ontology definitions. A new sub- or superclass can easily be added by using OntoTracks anchor mode for example. Here, clickable triangle buttons appear when moving with the mouse pointer over a class node (see class SolidTangibleThing in Figure 1). These anchor buttons allow for specifying an additional superclass or new subclass via mouse click and are sensible with respect to the layout orientation. An existing subclass relationship can be changed via mouse drag-and-drop or deleted by selection and "Del"-key usage as long as this relationship is not implied by other logical axioms of the ontology.

In addition, OntoTrack offers further editing functions while in its "detailed view" mode. The detailed view mode is activated or deactivated for each class separately using the mouse-wheel down- resp. up-wards while being over the class with the mouse pointer (alternatively PgDown/PgUp). When activated, OntoTrack uses an UML-style class diagram showing the list of defined property restrictions for this class in abstract DL syntax in its bottom compartment. OntoTrack currently supports all OWL Lite restrictions: unqualified cardinality restrictions ($\leq$, $\geq$, $=$ with cardinality 0 or 1) as well as existential and universal ($\exists$, $\forall$) quantifications. A restriction can be deleted by clicking on the round (red) button at the right hand side of the corresponding row and a new restriction will be added by clicking on the round (green) button at the bottom of the class box. At present, properties as well as classes within those restrictions are chosen with help of selection lists. In future versions, properties as well as classes may be selectable by mouse click. The example in Figure 4 shows some restrictions for the class TemporalThing.

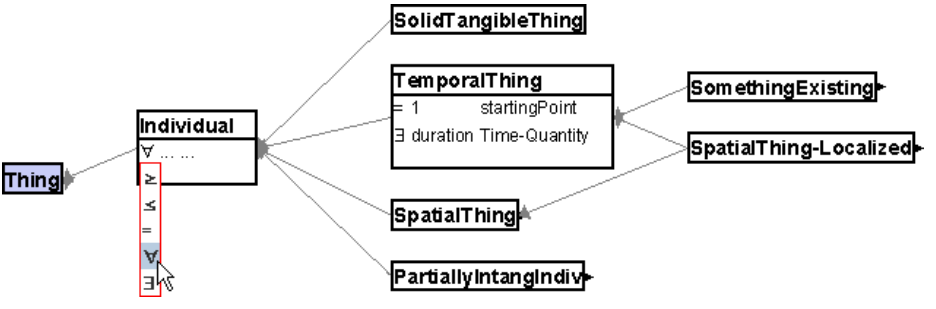

Fig. 4. Detailed view mode for classes Individual and TemporalThing with restrictions resp. specification of an universal quantification.

Semantically, a class is equivalent to (if complete) or subclass of (if partial) the conjunction of the collection of superclasses and restrictions in OntoTrack.

Additional editing features like class deletion or switching between complete and partial definitions are accessible via a right mouse button context menu. In order to visually distinguish between a complete and a partial class definition[1] we adopt the UML notation of *derived types* for complete class definitions. In concrete, a complete class definition is characterized by a slanted line in the upper left corner of the class box (see Figure 8 for examples).

In addition to the class hierarchy view, OntoTrack also provides an analogous graphical representation for properties. Here, different detailed property views allow for manipulation of global domain and range restrictions, global car-

[1] In OWL Lite$^-$ we allow for exactly one definition per class identifier (see subsection 2.1).

dinality constraints as well as logical characteristics of the property with help of selection lists or checkboxes as exemplarily shown in Figure 5.

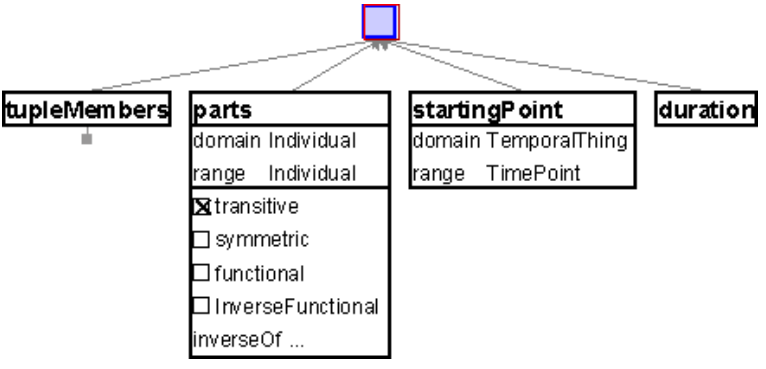

Fig. 5. Property hierarchy with examples of different detailed view modes.

As an extra feature, the property representation can be rendered as a read-only transparent layer onto the class representation and vice versa. The transparency rate of the secondary layer is freely adjustable.

2.4 Reasoning

As mentioned before, OntoTrack is equipped with an interface for interaction with an external OWL reasoner, namely RACER. All changes after each editing step (e. g. list selection, subclass manipulation) are immediately send to RACER. This reasoner will then make all modeling consequences explicitly available. OntoTrack will hand over relevant consequences to the user by providing appropriate graphical feedback. Relevant consequences currently cover subsumption and equivalence between as well as unsatisfiability of classes. For example, adding an existential restriction (minimal or exact cardinality restriction with 1 or existential quantification) on a property with a domain restriction will result in a subsumption relationship between the edited class and the property domain. Those updates are also animated in order not to confuse the user with a new hierarchy layout in one step.

As an example, in Figure 6 we have a contradiction between a restriction in class Individual and a restriction in class TemporalThing. The latter requires exactly one filler for the property duration. The restriction in class Individual however demands for exactly zero fillers. As a consequence, TemporalThing and all subclasses thereof are unsatisfiable and therefore outlined in red.

Note, that the last edited class is not necessarily the (only) one which potentially will become inconsistent. OntoTrack will therefore query its reasoner for conflicting or equivalent definitions in any part of the ontology after each

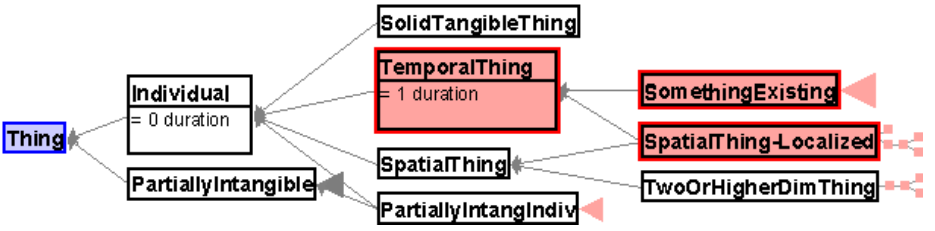

Fig. 6. Contradicting restrictions cause TemporalThing (and at least all its descendants) to be unsatisfiable.

editing step. Concerning the example shown in Figure 6 one may notice that there is at least one unsatisfiable descendant of PartiallyIntangibleIndiv (its subbranch triangle is colored red). This user triggered query strategy also applies for implicit subsumption relationships between classes.

2.5 Explaining

Providing feedback about the logical consequences of recent user changes help users to check for imprecise, redundant or faulty modeling. Important consequences with respect to an ontology authoring tool are subsumption, equivalence and unsatisfiability. Those consequences often depend on logical interrelations of nested definitions, which itself may depend on other definitions. As a result, most consequences are not easily traceable or intuitive to non-experienced users. Current reasoning systems provide no or only limited support for explanation of logical consequences.[2] OntoTracks most recent extention is a prototypical facility for an on demand explanation of subsumption, equivalence and unsatisfiability. This component is based on the work of Borgida et. al. [19] about a sequent calculus for $\mathcal{ALC}$. In [19] the authors sketch an extended DL tableaux algorithm for the generation of sequent proof steps needed for explanation. However, currently available tableaux reasoners potentially capable of handling OWL are either not avaliable as source code or based on highly optimized algorithms not easily extensible for explanation generation. We therefore implemented our own tableaux based explanation generator for OntoTrack.

Currently, this component is able to generate explanations within unfoldable $\mathcal{ALEN}$ ontologies, i. e. ontologies with unique acyclic definitions (negation and disjunction comes implicit due to the refutation strategy of the tableaux proofs). The actual prototype uses a naive tableaux implementation with lazy unfolding. It will generate a quasi natural language explanation compiled of text patterns which correspond to applied rules of the underlying tableaux proof. Note, that this explanation facility does not aim to replace RACER as core reasoning component and will only be activated on user request.

[2] The Classic system is a notable exception here [18].

Explanation for $\mathsf{A} \sqsubseteq \mathsf{B}$:

It holds that A is subsumed by B:

$\mathsf{A} \sqsubseteq \mathsf{B}$

which is equivalent to the unfolded subsumption problem:

$(\exists\, \mathsf{r}\, \mathsf{E}) \sqcap (\forall\, \mathsf{r}\, \mathsf{C}) \sqsubseteq (\exists\, \mathsf{r}\, \mathsf{D})$

In order to check that $(\exists\, \mathsf{r}\, \mathsf{E})$ and $(\forall\, \mathsf{r}\, \mathsf{C})$ is subsumed by $(\exists\, \mathsf{r}\, \mathsf{D})$ we need to check whether the conjunction of E and C is subsumed by D:

$\mathsf{E} \sqcap \mathsf{C} \sqsubseteq \mathsf{D}$

which is equivalent to the unfolded subsumption problem:

$\mathsf{E} \sqcap (\exists\, \mathsf{q}\, \mathsf{E}) \sqsubseteq (\geq 1\, \mathsf{q})$

In order to check that $(\exists\, \mathsf{q}\, \mathsf{E})$ is subsumed by $(\geq 1\, \mathsf{q})$ we need to check whether E is subsumed by Thing:

$\mathsf{E} \sqsubseteq \mathsf{Thing}$

By definition everything is subsumed by Thing.

Fig. 7. Explanation of a subsumption relationship of an example ontology shown in Figure 8.

Figure 7 displays a textual explanation of the subsumption relationship between the two classes A and B which would appear in a separate window.

Figure 8 shows the relevant definitions and their graphical representation in OntoTrack of the appertaining ontology within this small explanation example of Figure 7 (which has been adapted from an example in [19]).

```
ObjectProperty(r)
ObjectProperty(q)
Class(E partial)
Class(A complete
        restriction(r someValuesFrom(E))
        restriction(r allValuesFrom(C)))
Class(C complete
        restriction(q someValuesFrom(E)))
Class(B complete
        restriction(r someValuesFrom(D)))
Class(D complete
        restriction(q minCardinality(1)))
```

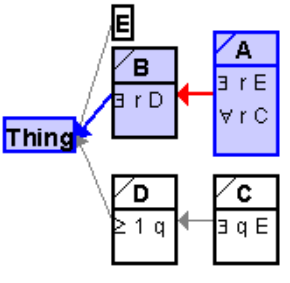

Fig. 8. Example ontology for subsumption explanation in OWL abstract syntax as well as in OntoTrack (relevant definitions are shown in detailed view mode).

3 Implementation Issues and Current Work

A recent analysis of online available ontologies [15] showed, that most ontologies contain more than hundred classes. Upcoming ontologies very likely will consist of several hundreds of classes presumably referencing dozen of other ontologies. Obviously, performance and scalability of ontology authoring tools are key for user acceptance and wide adoption of Semantic Web techniques. We therefore have chosen Piccolo [20] as our graphical library for OntoTrack, which has proven to be sufficiently fast and reliable even for large numbers of graphical objects. Piccolo is a Java2D interface toolkit supporting animation, zooming, multiple cameras, layers, etc. Our implementation of OntoTracks visualization components also adopts the linked tree diagram approach of SpaceTree [10] which itself uses the Piccolo toolkit. SpaceTree makes use of elaborated layout techniques to dynamically zoom and layout tree branches in an animated fashion. Figure 9 shows, that OntoTracks layout technique is even suitable to depict the greater structure of ontologies with more than 60 classes.

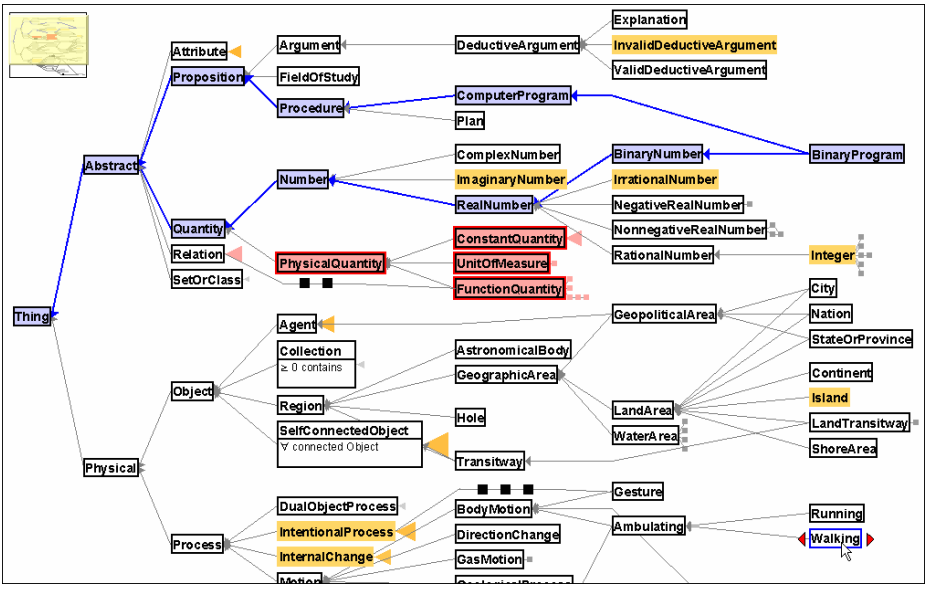

Fig. 9. A modified fraction of the IEEE SUMO ontology [21] with more than 60 classes expanded and radar view in the upper left corner.

For importing and exporting of ontologies we use the Jena 2.1 [22] RDF API and parser functionality. OntoTracks core ontology model is also based on Jena which we enriched with a high-level event notification mechanism needed

for propagating changes from the core model to their corresponding graphical or external reasoner representations resp. vice versa.

The RACER server is used to provide reasoning feedback about logical implied consequences. However, implementing this feedback functionality turned out to become difficult for some reasons. For example, in order to become aware of a new subsumption relationship due to a just added property restriction, OntoTrack needs to query the reasoner about direct superclasses for almost all classes of the ontology in turn.[3] Instead of querying for all possible changes with respect to a specific consequence for each editing step we would like to have an event-triggered publish-subscribe mechanism on reasoner side [23]. Another problem is concerned with incremental reasoning and retraction of definitions. Because of lack of algorithms for appropriately handling incremental additions to a knowledge base [24] complete reclassification after each user interaction is necessary.

A current extension is concerned with the handling of direct and follow-up consequences of user changes. E. g. a user manipulation may result in one unsatisfiable class (not necessarily the one which has been changed) or even many unsatisfiable classes in worst case. Depending on this outcome the editor should inform or even warn the user about the impact of his action in order to identify faulty modeling as early as possible.

OntoTracks prototypical explanation facility is currently implemented in CommonLisp as an external component in parallel to RACER. In order to avoid redundant computation, it is desired to combine reasoning and explaining within one system component in the future. Beyond that, we plan to enhance the explanation presentation with additional mouse-enabled features and different levels of detail information.

Current work also focuses on further editing features. Comments, labels as well as ontology header information are imported as well as exported correctly but are not editable in OntoTrack at the moment. We also work on the integration of further editing and searching functionality like undo operations or regular expression search. Because of user demand we also plan to add a view-only mode for OWL DL ontologies with a detail view showing OWL abstract syntax. Actual graphical extensions cover the optional visualization of entailed disjointness between classes.

4 Summary and Outlook

The broad adoption of Semantic Web technology strongly depends on the availability of convenient and flexible tools for browsing, editing, and evaluation of ontologies. A clear presentation, intuitive editing abilities, and subsidiary reasoning services are key properties of ontology editors suitable even for users

[3] One could of course narrow this set to those classes that also have an explicit or inherited restriction on that particular property or a subproperty thereof. But this requires to have explicit knowledge about inherited restrictions or subproperties, which in turn may result in additional queries.

without much logical background. Traditional expand and contract style interfaces, template based editing forms, or tree-centered visualizations inherently have substantial drawbacks in this respect. To our knowledge, there is currently no appropriate tool for displaying the greater structure of more than 60 directly editable classes together with additional information about unsatisfiable classes, selective detail information and search results, as it is shown in Figure 9.

Our new ontology authoring tool OntoTrack combines navigation and direct manipulation in one single hierarchical view together with instant reasoning feedback and rudiments of an explanation component. We see OntoTrack as a first step towards an easy-to-use interactive ontology editor even for non experienced users and large ontologies.

Future work will be concerned with the visualization and manipulation of individuals in OntoTrack, either with help of an additional visualization component optimized for this purpose (e. g. like in [25]) or embedded within our current hierarchy-based layout.

Support for cross ontology references between different definitions is also a point of further investigations and are still under discussion within the Web Ontology Working Group [26]. In a first naive approach our plan was to draw a superclass from a different ontology as a thumbnail ancestor using a dashed subsumption link (in order to distinguish between local and external ancestors). In case of selecting such an ancestor the corresponding ontology should be loaded and expanded up to the referred class in a new tab of OntoTrack. Unfortunately, this functionality bears some problems regarding import statements and referencing mechanisms of OWL/RDF. E. g. OWL allows to distribute a class specification across multiple definitions at various RDF documents. It is even possible to define a class bearing a virtual URI not related to any of the defining documents. Now, when referring to this class from elsewhere its corresponding definition cannot be found at the URI implied by the class ID. In other words, we can't infer the location of a definition from its URI reference. Beyond that, the meaning of the referenced description can only be taken into account when importing those documents which contain RDF triples about the description. Furthermore, the notion of an ontology is an additional but optional structuring concept within OWL. In fact, the relationship between class/property descriptions and an ontology definition is unclear. More complex, even it is optional or may appear more than once in a document, an ontology header is the only way to import other documents. This means that a serious OWL authoring tool needs to carefully distinguish between documents, ontologies and references.

Another serious issue of general future research is concerned with debugging of ontologies not developed from scratch within OntoTrack. Here standard inference services provide only little help to resolve inconsistencies in logical incoherent ontologies. In [27] a new reasoning service for pinpointing logical contradictions within $\mathcal{ALC}$ ontologies has been introduced. A likewise methodology would obviously be helpful within OntoTrack.

Other novel inference services intended to support building an ontology have been developed (see sec. 6.3 in [28] for a summary). One interesting service

consists of matching of class patterns against class descriptions in order to find already defined classes with a similar structure. Another approach tries to create class definitions by generalizing one or more user given ABox assertions. Other non-standard inference services like least common subsumer (e. g. like in [29]) or most specific concept are also relevant during authoring of ontologies.

We also see OntoTrack as a platform for further tasks like cooperative ontology construction, ontology merging and alignment, or ontology evaluation.

This contribution tries to inspire the Semantic Web community in two ways. Concerning research we hope to stimulate the development of non-standard inference services and incremental reasoning systems. On user side we aim to motivate even non experienced domain experts to build, understand and use ontologies more widely in order to push the Semantic Web from academia to industry.

References

1. Baader, F., Horrocks, I., Sattler, U.: Description Logics as Ontology Languages for the Semantic Web. In Hutter, D., Stephan, W., eds.: Festschrift in honor of Jörg Siekmann. Lecture Notes in Artificial Intelligence, Springer (2003) To appear.
2. Project-Website: Project Halo. `http://www.projecthalo.com/` (2004)
3. Fiedland, N.S., Allen, P.G., Witbrock, M., Matthews, G., Salay, N., Miraglia, P., Angele, J., Stab, S., Israel, D., Chaudhri, V., Porter, B., Barker, K., Clark, P.: Towards a Quantitative, Plattform-Independent Analysis of Knowledge Systems. In: Proc. of the Ninth International Conference on Principles of Knowledge Representation and Reasoning, Whistler, BC, Canada, AAAI Press (2004) 507–514
4. Liebig, T., Noppens, O.: OntoTrack: Fast Browsing and Easy Editing of Large Ontologies. In: Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Sanibel Island, USA (2003)
5. Storey, M.A., Musen, M., Silvia, J., Best, C., Ernst, N., Fergerson, Noy, N.: Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé. In: Proc. of the Workshop on Interactive Tools for Knowledge Capture (K-CAP 2001), Victoria B.C., Canada (2001)
6. Kalyanpur, A.: Venn diagram approach for visualizing OWL in SVG. University of Maryland, `http://www.mindswap.org/~aditkal/svg_owl.shtml` (2003)
7. Mutton, P., Golbeck, J.: Visualization of Semantic Metadata and Ontologies. In: Proc. of Information Visualization 2003 (IV03), London, UK (2003)
8. Eklund, P., Green, S., Roberts, N.: OntoRama: Browsing RDF Ontologies Using a Hyperbolic Browser. In: Proc. of the 1st International Symposium on Cyber Worlds (CW2002), Tokyo, Japan (2001)
9. Hu, B., Shadbolt, N.: Visualising a DL Knowledge Base with DeLogViz. In: Proc. of the International Workshop on Description Logics (DL2003), Rome, Italy (2003)
10. Plaisant, C., Grosjean, J., Bederson, B.B.: SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In: Proc. of the IEEE Symposium on Information Visualization (INFOVIS 2002), Boston, USA (2002) 57 – 64
11. Haarslev, V., Möller, R.: RACER System Description. In: Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), Siena, Italy, Springer Verlag (2001) 701–705

12. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics **1** (2003) 7–26
13. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P., Stein, L.A.: OWL Web Ontology Language Reference. W3C Recommendation (2004)
14. Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation (2004)
15. Tempich, C., Volz, R.: Towards a benchmark for Semantic Web reasoners – an analysis of the DAML ontology library. In: Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Sanibel Island, USA (2003)
16. van Harmelen, F.: The Complexity of the Web Ontology Language. IEEE Intelligent Systems **17** (2002) 71 – 72
17. Shneiderman, B.: Dynamic queries for visual information seeking. IEEE Software **11** (1994) 70–77
18. McGuinness, D.L., Borgida, A.: Explaining Subsumption in Description Logics. Technical Report LCSR-TR-228, Dept. of Computer Sciences, Rutgers University (1994)
19. Borgida, A., Franconi, E., Horrocks, I., McGuinness, D., Patel-Schneider, P.F.: Explaining $\mathcal{ALC}$ subsumption. In: Proc. of the International Workshop on Description Logics (DL1999). (1999) 37–40
20. Bederson, B., Grosjean, J., Meyer, J.: Toolkit Design for Interactive Structured Graphics. Technical Report CS-TR-4432, University of Maryland (2002)
21. Niles, I., Pease, A.: Towards a Standard Upper Ontology. In: Proc. of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Ogunquit, Maine (2001)
22. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: Proc. of the 13th International World Wide Web Conference (WWW2004), New York, NY, USA (2004) 74–83
23. Liebig, T., Pfeifer, H., von Henke, F.: Reasoning Services for an OWL Authoring Tool: An Experience Report. In: Proc. of the 2004 International Workshop on Description Logics (DL2004), Whistler, BC, Canada (2004) 79–82
24. Möller, R., Haarslev, V.: Description Logics Systems. In: The Description Logic Handbook. Cambridge University Press (2003)
25. Fluit, C., Sabou, M., van Harmelen, F.: Supporting User Tasks through Visualisation of Light-weight Ontologies. In: Handbook on Ontologies in Information Systems. Springer Verlag (2004) 414–432
26. Mail-Archives: Web Ontology Working Group. `http://lists.w3.org/Archives/Public/www-webont-wg/` (2004)
27. Schlobach, S., Cornet, R.: Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In: Proc. of the Belgian-Dutch Conference on AI (BNAI03), Nijmegen, The Netherlands (2003)
28. Baader, F., Küsters, R., Wolter, F.: Extensions to Description Logics. In: The Description Logic Handbook. Cambridge University Press (2003)
29. Baader, F., Sertkaya, B., Turhan, A.Y.: Computing the Least Common Subsumer w.r.t. a Background Terminology. In: Proc. of the 2004 International Workshop on Description Logics (DL2004), Whistler, BC, Canada (2004) 11–20