ALOE
A Graphical Editor for OWL Ontologies

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Daria Spescha

2005

Leiter der Arbeit:

Prof. Dr. Gerhard Jäger,
Institut für Informatik und angewandte Mathematik

Abstract

Existing ontology editors are either too complex or do not offer enough functionality. Thus, the goal of this diploma thesis was the development of an easy-to-use, graphical browser and editor for ontologies. ALOE is an ontology editor written in Java and based on knOWLer, an ontology management system. It enables the user to edit ontologies in OWL format without having to change the source code. This diploma thesis describes the use and development of ALOE.

Acknowledgements

I would like to thank Professor G. Jäger for his supervision of my diploma thesis. Special thanks also go to Dr. Thomas Studer for his support and motivation.

Furthermore, many thanks to Iulian Ciorascu and Claudia Ciorascu from the University of Neuchâtel, Switzerland, for their support with, and for the customisation and enhancements to knOWLer.

I like to thank Norbert Kottmann and Marco Robertini for proofreading and everyday support. Finally, I'm grateful to everyone who supported me.

Contents

1 Introduction **5**
1.1 Structure of This Thesis 5

2 Theoretical Basics **7**
2.1 The Semantic Web 7
2.2 OWL - Web Ontology Language by W3C 7
2.3 knOWLer 12

3 User Manual **13**
3.1 Getting Started 13
3.1.1 Prerequisites 13
3.1.2 Installation and Starting 13
3.1.3 Mac OS X 14
3.2 Loading an Ontology 15
3.3 Browsing an Existing Ontology 15
3.3.1 Views for the Resources 15
3.3.2 The Navigation 16
3.3.3 Search for Resources 16
3.3.4 Navigating through the Resources 18
3.4 Creating and Editing Resources 19
3.4.1 Creating a Resource 19
3.4.2 Edit 19
3.4.3 Editing a Class 20
3.4.4 Editing a Property 21
3.4.5 Editing an Individual 22
3.5 Deleting Resources 23
3.6 Saving an Ontology 23
3.7 Example 24

4 Implementation **27**
4.1 Libraries Used 27
4.1.1 JGraph 27
4.1.2 JDOM 27

4.2 Design 28
4.3 Implementation Examples 29

5 Comparison with Other Ontology Editors 33
5.1 Protégé 33
5.2 SWOOP 35
5.3 SWeDE 35
5.4 KAON 36

6 Conclusion and Further Work 41
6.1 Conclusion 41
6.2 Further Work 42

Appendices 45

A Example Ontologies 45
A.1 Asterix & Obelix Ontologies 45
A.1.1 Story Ontology 45
A.1.2 Cartoon Ontology 47
A.2 Asterix & Obelix Graphs 52

B UML Diagrams 55

Bibliography 60

Chapter 1

Introduction

For my diploma thesis, I developed ALOE, a browser and editor for ontologies in OWL format. There exist already some ontology editors like Protégé or OntoEdit, however all of them offer either not enough possibilities or too many (the advantages and weaknesses of some of these editors are presented in detail in Chapter 5). Thus, the main goal of the diploma thesis was the development of an easy-to-use editor satisfying the following requirements:

Ease of use: The application should be useable for users with a basic knowledge about ontologies and OWL in general, but without profound knowledge.

General Overview: It should be possible to obtain an overview of existing ontologies with little effort.

Graphical: The ontology should be graphically represented to get an overview of the design and the relations between different resources.

Visibility: All attributes available for OWL should be visible and editable.

Navigation: Related resources should be accessible from each other.

Huge Ontologies: The editor should be able to handle also huge ontologies in a practical manner, i.e. also big ontologies should be presented concisely.

ALOE is implemented in Java and based on the Java interface of knOWLer (see 2.3). Therefore, it is platform independent (for restrictions see 2.3 and 1).

1.1 Structure of This Thesis

Chapter 2 presents the basics about the Semantic Web, OWL, and knOWLer. Chapter 1 contains the user manual with a detailed overview of the features of ALOE and an example of the creation of an ontology. Chapter 4 describes the design and the most important implementation decisions. A comparison with similar applications is provided in Chapter 5. Finally, the conclusions drawn from the development of ALOE, as well as some ideas for further work are presented in Chapter 6.

Chapter 2

Theoretical Basics

2.1 The Semantic Web

According to Wikipedia “The Semantic Web is a project that intends to create a universal medium for information exchange by giving meaning (semantics), in a manner understandable by machines, to the content of documents on the Web.”[8] The idea is to make computers not only present formated documents to the user, but also to enable computers to deal with the contents. Therefore, documents should be supplemented with additional mark-ups containing meta information. For the realization of this idea, vocabularies for the meta information, so-called ontologies, would be required.

An ontology “[...] is a document or file that formally defines the relations among terms.”[3] It “[...] is the product of an attempt to formulate an exhaustive and rigorous conceptual schema about a domain. An ontology is typically a hierarchical data structure containing all the relevant entities and their relationships and rules within that domain (e.g. a domain ontology).”[7] In other words, an ontology formalizes the concepts of the real world and defines the core glossary of a field. Webpage owners can then use this vocabulary to tag their documents. In combination with given rules, agents can execute reasoning tasks about the information and provide the additional knowledge to the user. For a more detailed introduction to the vision of the Semantic Web see [3] and [10].

2.2 OWL - Web Ontology Language by W3C

To be able to generate the meta tags mentioned above and the vocabularies, a special language called ontology language is required. At present, different ontology languages exist as several institutions created their own language. Among others, also the W3C developed a language, the Web Ontology Language OWL, for which the W3C Recommendations were released in February 2004. OWL tries to be as conform as possible to SHOE, OIL, and DAML+OIL and is based on Description Logic. It adopts and extends the XML syntax of the Resource Description Framework (RDF, also released as a W3C Recommendation). For the design principles behind OWL read [11].

OWL provides three sublanguages which are decreasingly expressive:

OWL Full has the maximum expressiveness. However, it misses guarantees for computational completeness and decidability. One main difference to OWL DL is the ability to treat classes as individuals.

OWL DL provides maximal expressiveness for computational completeness and decidability. Its expressiveness corresponds to the one of Description Logic (DL). Due to its computational properties, it is intended for reasoners. The main constraint of OWL DL, with respect to OWL Full, is the disjointness of classes, properties, individuals and datatypes.

OWL Lite has the ability to express hierarchical information and some easy constraints.

For creating a vocabulary for a field, OWL offers mainly three types of resources: classes, properties and individuals. To assure uniqueness of resource identifiers, each resource can be assigned a name with the attributes `rdf:ID` or `rdf:about` and a namespace. OWL uses namespaces as identifiers for ontologies, so each ontology should have one unique namespace. Thus, references to other ontologies and resources defined elsewhere are possible.

A class can be used as a representation of a concept or a type. Syntactically, a class is an instance of `owl:Class` (with `owl` being the prefix for the OWL namespace). There are two predefined classes in OWL: `owl:Thing`, the universal class containing all elements, and `owl:Nothing`, the empty concept. Furthermore, a class can be defined as a restriction on a property. This is a possibility to constrain the values possible for statements with instances of the restriction class as subject and the given property as predicate. The limitation can either be a cardinality constraint or a type restriction (cf. Table 2.4). Restrictions are always anonymous, that is they can not have a name. Classes can be related to each other by several axioms expressing equivalence, subclass hierarchy, or intersection, to name the most important ones. These axioms are interpreted as set operators: the axiom `owl:subclassOf` for example can be mapped to the subset operator $\subseteq$ or to the inclusion operator $\sqsubseteq$ from DL and implies that all elements of the subclass are also elements of the superclass.

Whereas a class represents a type, an individual stands for a single object. It can be an instance of one or more classes with the predicate `rdf:type`. As OWL does not support the unique-names assumption, two individuals can be different as well as point to the same real world object if no further information is provided. Therefore, the two axioms `owl:differentFrom` and `owl:sameAs` can be applied to individuals to make statements about identity. An individual in OWL is defined like in RDF with `rdf:Description`.

Relations between real world objects are modeled as properties. There are two different types: `owl:ObjectProperty` relating two objects and `owl:DatatypeProperty` assigning a (data)value to an object. One or more classes can define domain and range of a property. Like classes, they can be structured hierarchically with the predicate `owl:superProperty` and be equivalent to each other. Additionally, two properties can be the inverse of each other. Furthermore, properties can be assigned the following attributes which are of importance for reasoners: transitive, symmetric, functional and inverse functional. A functional property behaves like an injective function and implies that each individual can be assigned at most one value by this property. If a properterty is inverse functional, individuals can have at most one source for this property. After defining a property, it can be instantiated as a statement of the form `subject predicate object`.

Table 2.1, 2.3, and 2.2 show an overview of the available axioms, their corresponding DL axioms and the informal semantic.

Axiom	**DL-Axiom**	**Semantic**	**Lite**	**DL/Full**
`owl:Class`	concepts C	Represents a type or concept of the real world	✓	✓
`owl:Restriction`	see Table 2.4	Defines an anonymous class as a restriction on a Property. Requires exactly one `owl:onProperty` and one or more constraints (see 2.4).	✓	✓
`ow:subClassOf`	$\sqsubseteq$	Links two classes. Defines a specialization or type hierarchy	✓	✓
`owl:equivalentClass`	$\equiv$	Different names for the same type; same elements	✓	✓
`owl:disjointWith`	$A \sqcap B \equiv \bot$	Links two classes. The two classes don't have common instances.	✗	✓
`owl:oneOf`	$\{e_1, e_2, \ldots\}$	Class defined as enumeration of objects	✗	✓
`owl:intersectionOf`	$\sqcap$	Links a class with a list of classes. The class contains the elements which are in the intersection of the other classes.	✓	✓
`owl:unionOf`	$\sqcup$	Links a class with a list of classes. The class contains the elements which are in either of the classes.	✗	✓
`owl:complementOf`	$\neg C$	Links two classes. The classes are disjoint and each element is an instance of either of them.	✗	✓
`owl:Thing`	$\top$	The universal class.	✓	✓
`owl:Nothing`	$\bot$	The empty class.	✓	✓

Table 2.1: OWL Syntax for Classes

Axiom	**DL-Axiom**	**Semantic**	**Lite**	**DL/Full**
`rdf:Description`	individuals a	Defines a new individual.	✓	✓
⟨`Classname`⟩		Defines an individual as an instance of class `Classname`, equivalents to combining `rdf:Description` and `rdf:type`	✓	✓
`rdf:type`	$C(a)$	Links an individuals to a class. Defines a individual as an instance of the class.	✓	✓
⟨`Propertyname`⟩	$P(a, b)$	Defines a statement for the property `Propertyname`	✓	✓
`owl:sameAs`	$=$	Links two individuals. States that the individuals refer to the same thing.	✓	✓
`ow:differentFrom`	$\neq$	Links two individuals. States that the individuals refer to different things.	✓	✓
`owl:AllDifferent`		Requires `owl:distinctMembers`. A short form for `owl:differentFrom` to declare that all elements listed in `owl:distinctMembers` are pairwise disjoint.	✗	✓

Table 2.2: OWL Syntax for Individuals

Axiom	**DL-Axiom**	**Semantic**	**Lite**	**DL/Full**
`owl:ObjectProperty`	roles R	Defines a property for linking individuals to individuals	✓	✓
`owl:DatatypeProperty`		Defines a property for linking individuals to data values.	✓	✓
`owl:AnnotationProperty` `owl:OntologyProperty`	N.A.	Defines a new annotation for resources or ontologies	✓	✓
`rdfs:subPropertyOf`	$\sqsubseteq$	Links two properties. Defines a hierarchy for properties.	✓	✓
OWL Syntax for Properties				

Axiom	**DL-Axiom**	**Semantic**	**Lite**	**DL/ Full**
`rdfs:domain`		Links a property with a class. Restricts the domain for a property. In statement with this property, the subject will be inferred to be an instance of the domain.	✓	✓
`rdfs:range`		Links a property with a class. Restricts the range for a property. In statement with this property, the object will be inferred to be an instance of the range.	✓	✓
`owl:equivalentProperty`	$\equiv$	Links two properties.	✓	✓
`owl:inverseOf`	R^{-}	Links two properties. If a P is inverse to Q and in OWL the statement $P(a,b)$ is defined, then $Q(b,a)$ is inferred.	✓	✓
`owl:FunctionalProperty`		A functional property is a global cardinality constraint for a property: every individual can be assigned at maximum one value with this property.	✓	✓
`owl: InverseFunctionalProperty`		A inverse functional property is a global cardinality constraint for a property: every individual can be reached from at maximum one subject with this property.	✓	✓
`owl:SymmetricProperty`		If P is a symmetric property and $P(a,b)$ is defined, then $P(b,a)$ is inferred.	✓	✓
`owl:TransitiveProperty`		If P is a transitive property and $P(a,b)$ and $P(b,c)$ are defined, then $P(a,c)$ is inferred.	✓	✓

Table 2.3: OWL Syntax for Properties

A good introduction to the semantic web in general and especially OWL is [1]. For detailed information about OWL and the OWL syntax, see the different W3C Recommendations ([12], [13], [2], and [9]).

RestrictionConstraint	**DL**	**Lite**	**DL/Full**
hasValue	$R : a$	✓	✓
Property Type Restrictions			
allValuesFrom	$\forall R.C$	✓	✓
someValuesFrom	$\exists R.C$	✓	✓
Restricted Cardinality			
minCardinality	$\geqslant nR$	$\{0, 1\}$	✓
maxCardinality	$\leqslant nR$	$\{0, 1\}$	✓
cardinality	$\geqslant nR \sqcap \leqslant nR$	$\{0, 1\}$	✓

Table 2.4: Constraints for OWL Restrictions on a Property

2.3 knOWLer

knOWLer is a system for processing ontology files. It has been developed at the University of Neuchâtel, Switzerland, by Prof. Dr. Kilian Stoffel, Iulian Ciorascu and Claudia Ciorascu. It is able to read ontology files, reason about the resources and manipulate them. It is designed to be able to deal especially with big ontologies and provides good performance. It implements an extension of OWL Lite.

knOWLer consists of the kernel, responsible for the actual handling of the resources and for the reasoning, and different modules for import/export and storage. It provides several user interfaces, among other a Java API which I used for this diploma thesis. At present, it runs on Linux and Mac. For more information about knOWLer read [5] and [4].

Chapter 3

User Manual

This chapter describes the installation and usage of ALOE in details. At present, ALOE runs on Mac OS X and Linux.[1]

ALOE is a tool to browse, edit and create ontologies. It is not a reasoner, but it does some basic reasoning to check the validity of modifications. It enables the user to get an overview of the design of an ontology by navigating through the resources and showing its related resources (cf. 1.3.4). The details of a resource are available either in the detail view or in OWL format (see 1.3.1). All resources, new ones as well as those from existing ontologies, can be edited with the help of an edit mask like described in Section 1.4.

3.1 Getting Started

3.1.1 Prerequisites

To use ALOE, the following prerequisites need to be met:

Java: ALOE requires a properly installed Java Runtime Environment JRE 1.4.2.[2]

knOWLer Java: The Java API of knOWLer must be installed and set up correctly according to the installation instructions of knOWLer. Most important, the environment variables must be set. At the moment, knOWLer is available for Mac OS X and Linux.[3].

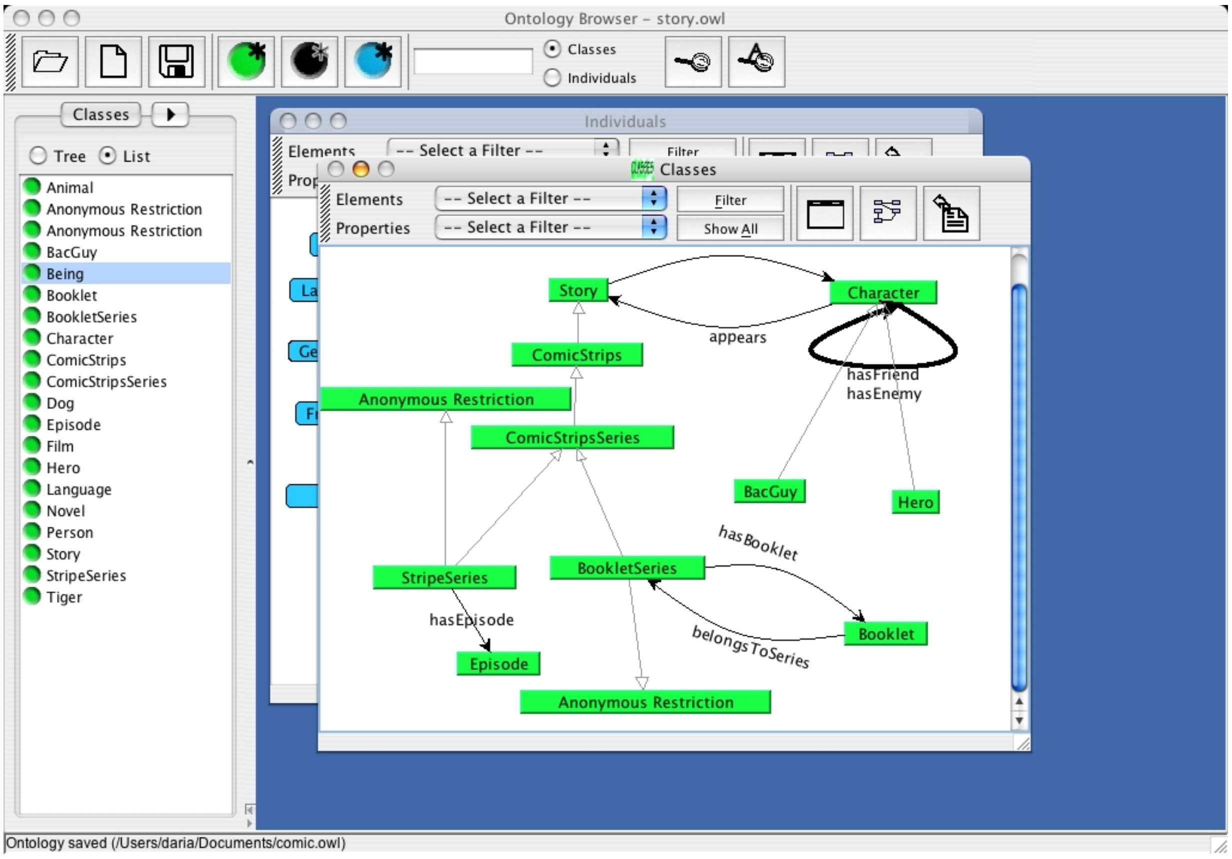
3.1.2 Installation and Starting

ALOE is available as a compressed archive (`.zip` or `.tar.gz` file). It should be unpacked to a user-defined location. Hereon the folder `ALOE` contains the following folder structure:

[1]ALOE is pure Java and therefore runs on all platforms with JRE and knOWLer installed (see 1.1.1). However, at present knOWLer runs only on Mac OS X and Linux.

[2]The JRE is available at `http://java.sun.com/j2se/1.4.2/download.html`

[3]The knOWLer home-page can be accessed at `http://taurus.unine.ch/knowler/`



Ontology saved (/Users/daria/Documents/comic.owl)

Figure 3.1: Main view of ALOE on Mac OS X

`aloe.jar`: the actual application
`doc`: the documentation including this document and the Javadocs.
`examples`: the examples used in Section 1.7 `icons`: the icons used by ALOE.
`lib`: additional jar libraries.
`log`: folder for the log file.
`setup.sh`: setup script
`src`: Java source files.

The script `./setup.sh` configures ALOE and it should be executed in a shell. The application can now be started by executing `./aloe` (or directly with `java -jar aloe.jar`. After starting the ALOE, the main view (see Figure 1.1) opens with an empty ontology and is ready to use. (If the installation directory of knOWLer is changed, `./setup.sh` needs to be executed again.)

3.1.3 Mac OS X

For Mac OS X, ALOE is also available as a bundled application. To install it, the archive needs to be extracted and the following commands have to be executed in a shell:

```
cd ALOE-macOSX
./setup
```

The created ALOE.app can now be dragged to any folder and started with a double click. If the installation directory of knOWLer is changed, ALOE also needs to be installed again.

3.2 Loading an Ontology

An existing ontology can either be loaded with the "open" button in the toolbar or through the menu "File". The file must be a valid OWL document. It can contain all elements of OWL Full, but only the elements conforming to OWL Lite will be handled and additional axioms like `disjointWith` (cf. 2.2) are ignored. If another ontology was already loaded/-generated, it is removed from the temporary storage and changes after the last save are lost.

An ontology can also be loaded into the current ontology. In this case, the resources already loaded/generated will be kept. This is useful if you have an ontology where resources from another one are referenced and you want to get the definitions of the referenced resources. It can be be achieved by selecting "Load Ontology Into" from the menu "File".

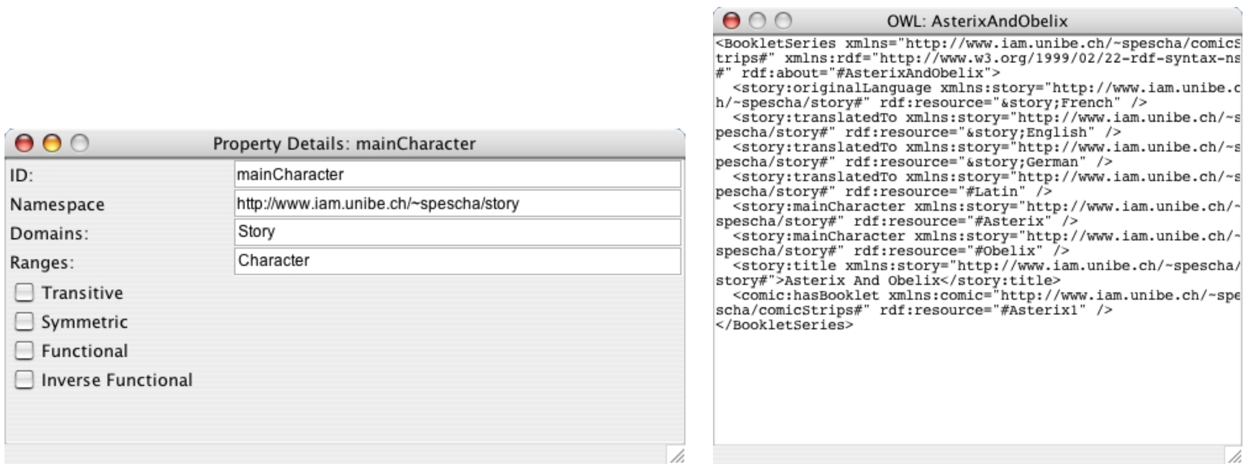
After an ontology is loaded, the navigation tab on the left hand side (cf. 1.3.2) is updated. During a normal load, the frames containing the graphical representation are reset.

3.3 Browsing an Existing Ontology

3.3.1 Views for the Resources

In ALOE, the user can inspect a resource in three different views: graphically as a node or vertex in a graph, in the detail view or as a preview of the OWL output. The detail view (cf. Figure 1.2(a)) shows the details of a single resource. All axioms, as mentioned in 2.2, and all annotations for this resource are listed. One of the main goals of ALOE is to enable users to edit ontologies without knowledge of the OWL XML syntax. However, it offers the OWL view for those interested in the OWL XML representation of a resource (vide Figure 1.2(b)).

The main view is the graphical view (cf. Figure 1.3). It shows not only single resources, but the relations between them as well. The selected resources are presented as a graph. Therfor, classes and individuals are represented as vertices, properties and axioms as edges. To easily distinguish between properties and axioms, they are displayed in different colors: the former are black and the latter grey. Classes and individuals are shown in separate graph frames, whereas properties can be found in both, but with a different semantic: If two classes are linked with a property, it means that the source class is in the domain and the target class in the range of the property. If two individuals are connected by a property, this is a property instance, that is, it is a statement of the form `subject predicate object`.



ID:	mainCharacter
Namespace	http://www.iam.unibe.ch/~spescha/story
Domains:	Story
Ranges:	Character

- [ ] Transitive
- [ ] Symmetric
- [ ] Functional
- [ ] Inverse Functional

```
<BookletSeries xmlns="http://www.iam.unibe.ch/~spescha/comicS
trips#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
#" rdf:about="#AsterixAndObelix">
  <story:originalLanguage xmlns:story="http://www.iam.unibe.c
h/~spescha/story#" rdf:resource="&story;French" />
  <story:translatedTo xmlns:story="http://www.iam.unibe.ch/~s
pescha/story#" rdf:resource="&story;English" />
  <story:translatedTo xmlns:story="http://www.iam.unibe.ch/~s
pescha/story#" rdf:resource="&story;German" />
  <story:translatedTo xmlns:story="http://www.iam.unibe.ch/~s
pescha/story#" rdf:resource="#Latin" />
  <story:mainCharacter xmlns:story="http://www.iam.unibe.ch/~
spescha/story#" rdf:resource="#Asterix" />
  <story:mainCharacter xmlns:story="http://www.iam.unibe.ch/~
spescha/story#" rdf:resource="#Obelix" />
  <story:title xmlns:story="http://www.iam.unibe.ch/~spescha/
story#">Asterix And Obelix</story:title>
  <comic:hasBooklet xmlns:comic="http://www.iam.unibe.ch/~spe
scha/comicStrips#" rdf:resource="#Asterix1" />
</BookletSeries>
```

(a) Detail View for a Property (b) OWL View of a property

Figure 3.2: Different views of a property

To connect classes and individuals, the classes of an individual are shown whenever the cursor is over the vertex. The actions available for this view are described in detail in Section 1.3.4.

If a graph has grown too big, it is possible to hide the vertices and edges visible by different criterias. For example, it is possible to hide all axiom edges and show only the properties. The filter to apply to the graph can be selected on top of the graph frame. On pressing the button "Filter", the vertices and edges are filtered. Several filters can be applied consecutively. The hidden elements can be made visible again by the button "Show All".

3.3.2 The Navigation

On the left of ALOE, there is the navigation panel with a tab for each resource type (classes, individuals, and properties). All resources are either displayed as a list or as a tree (see Figure 1.3.2). It is possible to switch between the two views with the radio button at the top of the panel.

A resource can be displayed in the appropriate graph (cf. 1.3.1) either by a double click or by a click of the right mouse button (`command` click on mac) and selecting "Show"in the pop-up menu. The pop-up menu for the right click on a resource offers also the possibility to show its details or to edit it.

3.3.3 Search for Resources

ALOE provides the possibility to search for individuals or classes by name. The resources found are shown in the respective graph. The easiest way to find a resource is by typing part of the name in the text input in the toolbar, selecting the type (classes or individuals) and

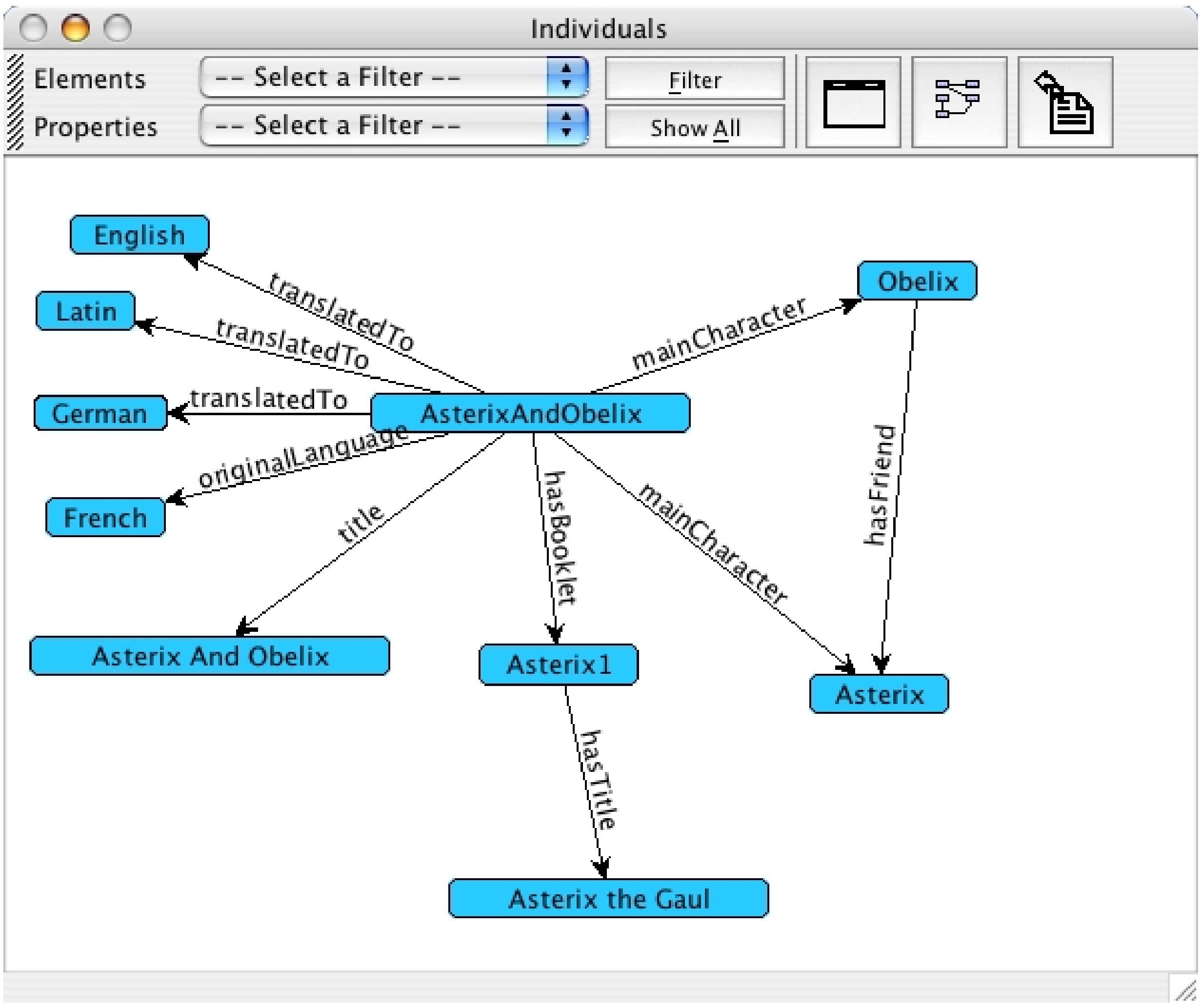
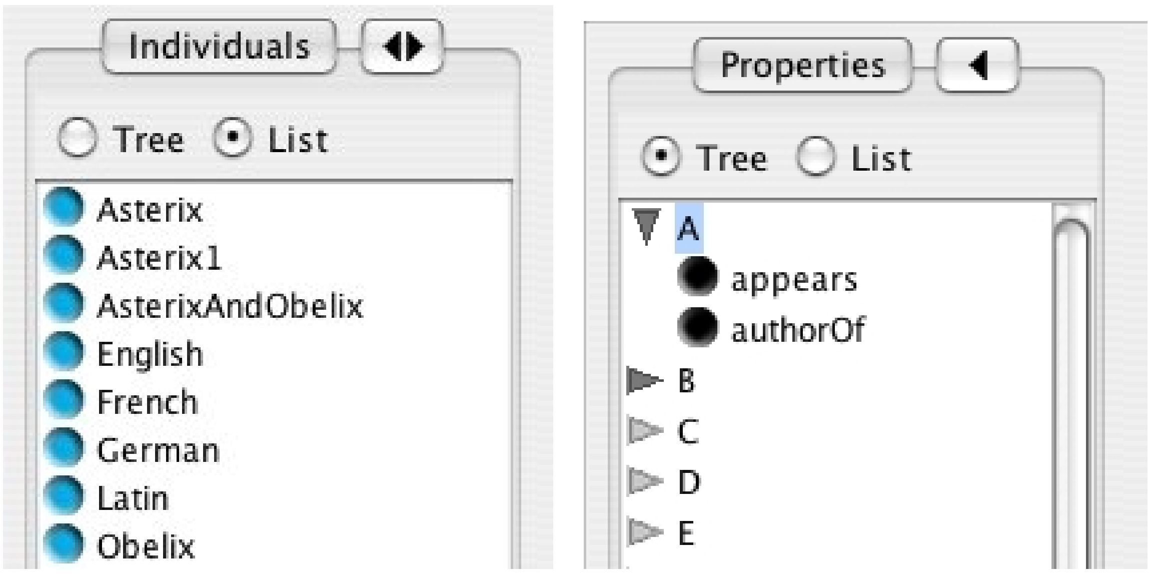


Figure 3.3: Graph view of the w6 Ontology (w6: the six “wh*”questions)

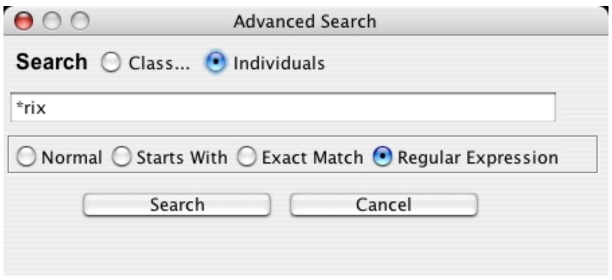
pressing the search button. If the field is empty, a dialog asks for an input. Furthermore, there is the Advanced Search for more sophisticated search criteria. It is accessible either from the toolbar button or from the menu “Navigation ➤ Advanced Search”. The user can select one of four possibilities to search: resources whose name start with the given characters, resources having exactly the given name or having partly the given name, or resources whose name match a given regular expression.

To keep the graph from growing too much because of big search results, there is a limit on the number of resources which are inserted. If more resources should be displayed, a list with all found resources appears and lets the user select which resources really should be visualised. This limit can be set in “Navigation ➤ Set Maximum Insert Size”.



(a) List View (b) Tree View

Figure 3.4: The Navigation Panel



Search Class... [✓ Individuals]

*rix

Normal Starts With Exact Match [✓ Regular Expression]

Figure 3.5: Search Dialog

3.3.4 Navigating through the Resources

A pop-up menu can be invoked for each resource in the graph view to navigate from one resource to its related resources. Depending on the type of the resource, different actions are available. The following actions are common to all types:

Show Details: Opens the detail view for the resource as described in 1.3.1.
Show OWL Representation: Shows the OWL XML preview for the resource.
Edit: Opens the edit wizard as explained in Section 1.4.

For individuals, the following additional actions are available:

Show Classes: Displays in the class graph all classes the individual is an instance of.

Show Related: Displays all individuals related to the selected individual either by a property instance or by an axiom.

As classes are the most complex resources, they offer the greatest number of possibilities for navigating through the ontology:

Show Superclasses: Shows all direct superclasses in the class graph.
Show Subclasses: Displays all direct subclasses.
Show All Properties: Presents in the class graph all properties the selected class is in the domain or range of.
Show All Related: Is a subsumption of the three above actions and displays also all equivalent classes.
Show Instances: Poses all direct instances of this class in the individual graph.
List Instances: Presents all direct instances of the selected class in a list.

3.4 Creating and Editing Resources

3.4.1 Creating a Resource

Resources can be created with a wizard. The wizard is accessible either from the buttons in the toolbar or from the menu "Edit". Each resource needs a namespace and a name. The user can either choose an existing namespace or type a new URI. The pair ⟨namespace#name⟩ must be unique, otherwise an error message will be displayed. When creating a property, it must be specified whether it is a `ObjectProperty` or `DatatypeProperty`. For individuals, one class must be given at creation. After the new resource is created, it is inserted in the graph view and an edit window opens on.

In addition to the three main resource types, it is also possible to create annotation properties. Annotation properties are used to add different annotations to resources (for further information on annotation properties see [2, "7.1 Annotations"]). Annotations can only be created, but neither displayed in a graph nor edited. They are used when editing other resources as explained below.

3.4.2 Edit

The edit dialog looks similar to the detail view, however it does not show all details available, but all axioms editable. This section describes the actions common to all resource types, the type specific actions are explained individually in the following sections.

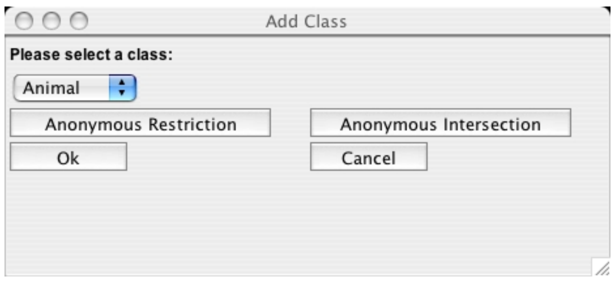
Every dialog has two buttons, "Change"and "Done", to adopt the values changed. "Change"reads the changes for the name and the annotations and updates the graphical view, "Done"does the same and closes the edit dialog. If the window is closed otherwise, changes to the name and to the annotations will be lost.

To rename a resource, the user can type a new name in the text field at the top of the dialog. The new name must not exist in the namespace of the resource, otherwise an error message will be shown on applying the change.

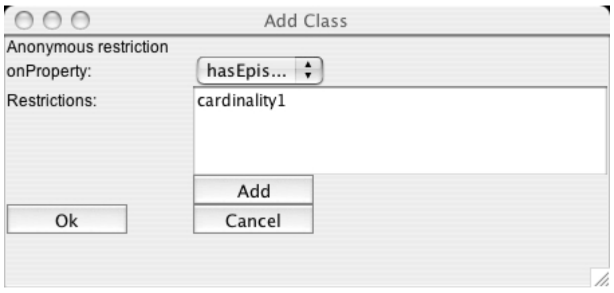
Furthermore, the resource can be annotated. There is a text field for each annotation. The annotations available are those predefined by OWL, such as `versionInfo`, `label`, `comment`, `seeAlso`, and `isDefinedBy`, as well as the user defined ones.

For each resource type, there are different axioms for relating two resources as described in Section 2.2. For each of these axioms, there is a list with the resources already related by this one and two buttons to add and remove resources. When resources are added, ALOE does some basic checks if the change does not lead to an inconsistent state of the ontology. It also checks whether the change has further effects, i.e. whether there are additional facts which can be deduced from the change. If there is such an effect, the user is warned. However, ALOE is not a reasoner and therefore may not detect all errors or all additional effects.

For putting a resource in relation to a class, there is a dialog where the user can select a class (cf. 1.6). It offers three kinds of classes for selection: The user can choose an already existing class, create a new anonymous restriction or make an anonymous intersection. If he decides to create an anonymous intersection, he can select two classes. When a new restriction is created, the dialog asks the user to select a property and to add restriction constraints. (Adding restrictions constraints to a restriction is explained later.) After pressing "Ok", a new anonymous class is created and then put in relation to the class currently edited by the axiom.



(a) Selecting an existing class



(b) Selecting an anonymous restriction

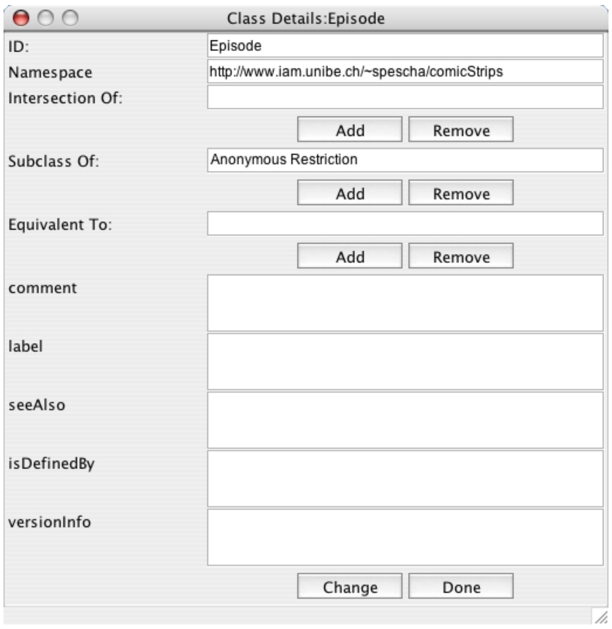
Figure 3.6: Adding a Class

3.4.3 Editing a Class

In addition to the general possibilities for modification, the user can manipulate the superclasses and the equivalent classes of a class. As OWL Lite does not facilitate a possibility to define two classes as disjoint, there are no restrictions on those axioms.

A class can be an intersection of at least two other classes. If a class is an intersection of two others and the user wants to remove one, he can either remove none or both, but not just one. Likewise, at least two classes must be chosen when adding a new intersection.

If a class is defined as a restriction, it is not possible to change the name as restrictions

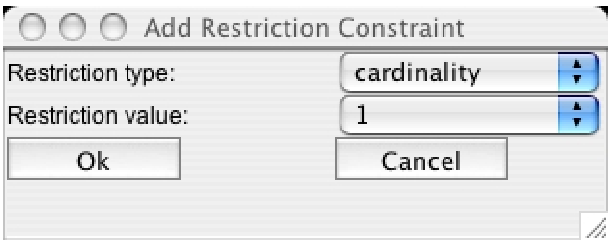


Class Details:Episode

ID:	Episode
Namespace	http://www.iam.unibe.ch/~spescha/comicStrips
Intersection Of:	
Subclass Of:	Anonymous Restriction
Equivalent To:	
comment	
label	
seeAlso	
isDefinedBy	
versionInfo	

Figure 3.7: Editing a Class

are always anonymous. The only manipulations allowed are addition and removal of constraints. When adding a constraint, the dialog shown in Figure 1.8 is opened. The user can select the type of the restriction and the value depending on the selection of the type.



Add Restriction Constraint

Restriction type: cardinality

Restriction value: 1

Figure 3.8: Dialog for adding a restriction constraint

3.4.4 Editing a Property

Each property should have at least one class in the domain and the range. When manipulating the domain and the range, it must be taken into account that they are interpreted as the intersection of the classes and not as the union of them if there is more than one.

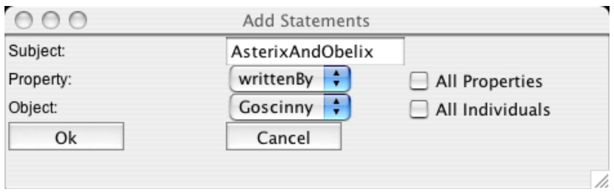
When a superproperty is being added, it must be of the same type as the property currently edited. In other words, a `DatatypeProperty` cannot be the superproperty of an `ObjectProperty` and vice versa. It is also checked whether the domain and range respectively of the subproperty are in the domain and the range of the superproperty or whether they are subclasses of a class of the domain and the range. If this is not the case, a warning appears.

The restrictions of a property (transitive, symmetric, functional and inverse functional) can be switched on and off with the checkboxes. If symmetric is selected, ALOE tests whether domain and range of the property comply with each other and warns if this is not the case.

3.4.5 Editing an Individual

An individual can be an instance of several classes. In OWL Lite there is no limitation because it is not possible to exclude individuals. Furthermore, two individuals can represent the same object or different ones.

Property instances can be created and deleted in the edit mask of the subject of the new statement. In the dialog for creating a new property instance (shown in Figure 1.9), the subject is already inserted and cannot be changed. First, a property should be selected as predicate. By default, only those properties are listed which have one of the classes of the subject in the domain. By checking the checkbox “All Properties”, all available properties are listed. As soon as the predicate is chosen, the list with the possible objects is updated with all individuals that are instances of the classes in the range of the prefix. Like for the predicates, all individuals can be displayed by enabling the checkbox “All Individuals”.



Subject: AsterixAndObelix
Property: writtenBy
Object: Goscinny
All Properties
All Individuals

Figure 3.9: Creating a new statement

A statement will not be accepted and an error message will be shown if one of the following conditions holds:

- The predicate is a `FunctionalProperty` and the subject already has a value for this property which cannot be inferred to be the same object as the value. Two individuals cannot be the same object if they are inferred to be different.
- The predicate is an `InverseFunctionalProperty` and the object already has a subject for this property which cannot be inferred to be the same object as the subject.
- The subject has a `hasValue` restriction on the predicate and the object is not equal to the required value.

The following deduced facts will be recognized, that is, ALOE is able to deduce the following additional facts:

- The subject has a cardinality restriction (`cardinality` or `maxCardinality`) on the predicate and has already reached the limit of objects. In this case, some of the objects will be deduced to be `sameAs`.

- The subject has a `allValuesFrom` restriction on the predicate and the object is not already an instance of the required class. Thus, the object is inferred to be an instance of the required class.

- The predicate is a `FunctionalProperty` and the subject already has a value for this property which can be the same as the object. Therefore the object will be deduced to be `sameAs` the other value(s).

- The predicate is an `InverseFunctionalProperty` and the object already has a subject for this property which can be the same as the new subject. Hence, the new subject will be inferred to be `sameAs` the other subject(s)

3.5 Deleting Resources

A resource can be deleted by selecting "Delete"in the popup menu for a resource in the graph view. If other resources depend on the resource to delete, an error message is shown explaining which resources depend on it and that it cannot be deleted. For example a property cannot be deleted if there is a restriction on it. If the resource to delete is referenced by other resources, a warning is displayed.

3.6 Saving an Ontology

An ontology can be saved in OWL format. First, a file must be chosen to write to. Then a dialog for configuring the ontology header and the namespaces is displayed. Values for all available annotations for an ontology can be defined. Additionally, the name of the author can be given. It will be inserted as a comment as OWL does not provide an author annotation. Furthermore, a prefix for each namespace must be defined. This prefix will be used as an abbreviation for the namespace as indicated in the OWL XML syntax definition. Also one namespace must be chosen as the default namespace. If constructs from other ontologies are referenced, the namespace defining these ontologies should be imported. As a namespace usually identifies an ontology, ALOE allows the export of selected namespaces. Only the definitions of the resources in the selected namespace(s) will be saved. Resources in other namespaces will just be referenced. This is useful because often several ontologies will be loaded (the ontology currently edited as well as the ontologies referenced), but only the ontology (ontologies) currently edited are supposed to be saved.

3.7 Example

In this section I demonstrate the usage of ALOE by creating an example ontology step by step. Therefore, I want to develop an ontology about cartoons based on an ontology about stories. The full OWL sources of the example ontologies can be found in Appendix A. Figure 1.10 shows the classes of the ontology I want to extend.

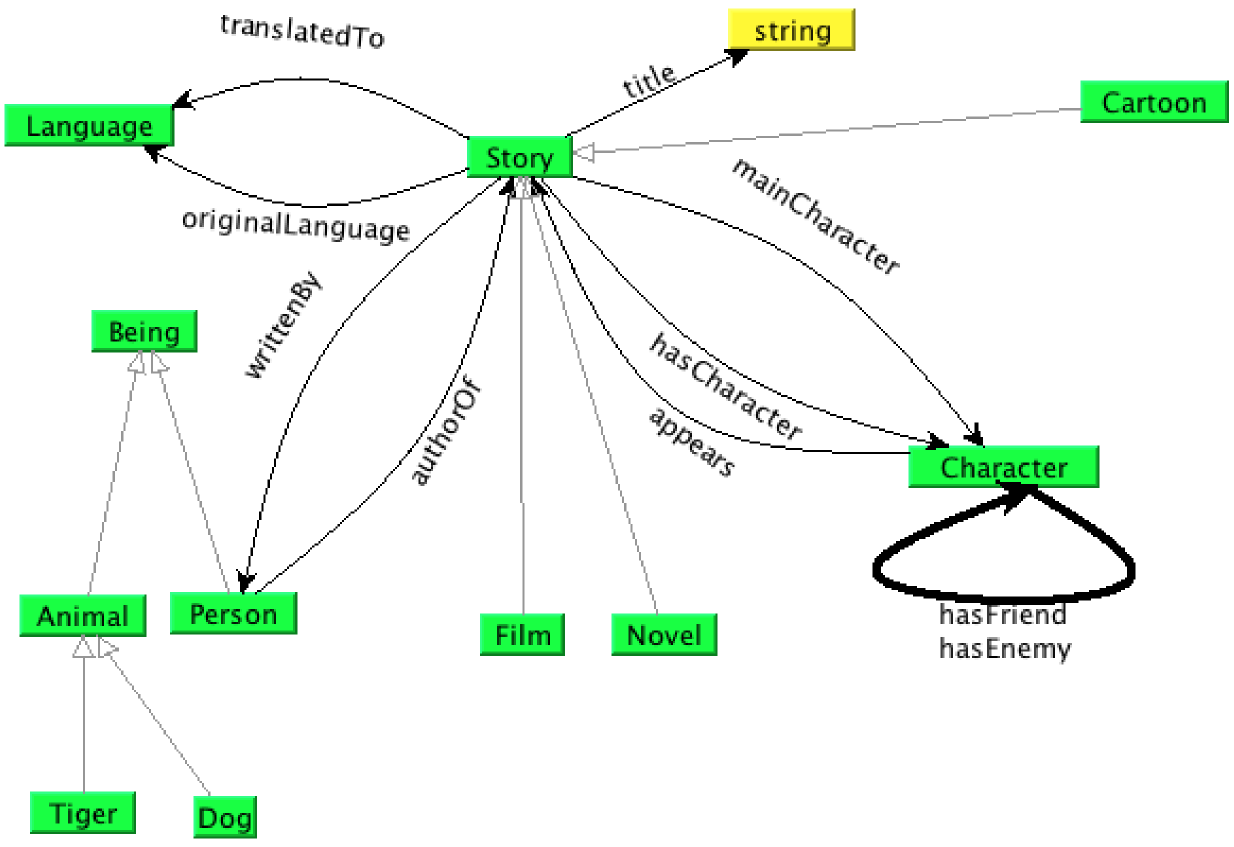


Figure 3.10: The ontology about stories

First, I want to model the fact that cartoons are a kind of stories. Therefore, I create a new class named `Cartoon` in the new namespace `http://www.iam.unibe.ch/~spescha/cartoons` and add `Story` as superclass. After inserting a comment, I'm done with this class. As there are cartoons appearing as a series, either as book like "Asterix & Obelix"or as a stripe in a newspaper like "Calvin and Hobbes", I create `CartoonSeries` as a subclass of `Cartoon` in the same namespace and `BookSeries` and `ComicStrip` as subclasses of `CartoonSeries`. A booklet series consists of several booklets. Thus, I first make a class `Book`. Then I generate the property `hasBook` as an `ObjectProperty`. In the edit mask for `hasBook` I add `BookSeries` as domain and `Book` as range and select it to be an inverse functional property. Now I create the `ObjectProperty` `belongsToSeries` with `Book` as domain and `BookSeries` as range. As this is the opposite of `hasBook`, I add the latter as inverse property and define `belongsToSeries` to be functional. A `BookSeries` must have at least one `Book`. Hence, I define `BookSeries` as subclass of an anonymous restriction. The restriction is a restriction on the property `hasBook` and has a `minCardinality` constraint with value `1`.

To specify the attributes for a `Book`, I define the two functional `DatatypeProperty` `title` and `isbn`, both with `Book` as domain and `string` from XML schema as range. Similar to `Book`, I define `Episode` as elements for `ComicStrip`. As a lot of comic strips are about heros and their bad antagonists, I add two subclasses of `Character`, `Hero` and `BadGuy`.

Now there are all classes I need and I can now start modeling "Asterix & Obelix". I create `Asterix` and `Obelix` as instances of `Hero`. As they are both humans, I add `Person` as class for both of them. Then I add a statement for `Asterix`. In the dialog for creating the property instance, I select `hasFriend` as predicate and `Obelix` as object. Now I am done with these characters.

As "Asterix & Obelix"is a booklet series, I generate an individual `AsterixAndObelix` as an instance of `BookletSeries`. I add two statements to define `Asterix` and `Obelix` as main characters of the comic strip. Then, I define `French` to be the original language and state that `AsterixAndObelix` was translated to other languages. Similarly, I create other characters from "Asterix & Obelix"and the comic strip "Calvin And Hobbes".

Finally, I want to save my ontology. I make some annotations for the resource and define prefixes for the namespaces. `http://www.iam.unibe.ch/~spescha/cartoons` is the default namespace of this ontology and also the only one I want to export. The resulting OWL file is available in the Appendix A.1.

Chapter 4

Implementation

This chapter describes the implementation of ALOE. In a short form, the libraries used in addition to knOWLer are introduced. The package structure and design principles are explained as well and some examples are presented. The source code is well documented with Javadoc comments. More details on the implementation can be found in the generated API specification on the supplemented CD-ROM.

4.1 Libraries Used

4.1.1 JGraph

JGraph[1] is a Java library for graph drawing with Swing. JGraph started as diploma thesis at the ETH Zurich and is available as open source under the LGPL license.

JGraph is pure Java and can be used in any Swing application. It is designed using the Model-View-Controller pattern of Swing. JGraph offers the ability to drag and drop, export the graphs, route the edges and to zoom, to name just a few. Furthermore, the layout of the edges and vertices is fully customizable. The auxiliary library JGraph Addons[2] contains useful features like layout algorithms and different shapes for the vertices.

ALOE employs JGraph and JGraph Addons for the graph view of the resources. JGraph is used in version 5.X and JGraph Addons in version 1.0.7.

4.1.2 JDOM

JDOM[3] is a Java library for manipulating XML data in Java. It permits the user to treat XML elements as Java objects and therefore makes it easy to manipulate them. It supports namespace declarations, which is very important for OWL. Furthermore, it provides the functionality to print XML data in a customizable form.

[1]`http://www.jgraph.com`

[2]JGraph Addons was lately replaced by a commercial library and will not be developped further.

[3]`http://www.jdom.org`

ALOE uses JDOM for saving an ontology in OWL XML format. JDOM version 1.0 is used.

4.2 Design

The implementation of ALOE consists of the following packages (cf. also Figure 4.1):

`ch.unibe.ontoBrowser` contains the main class starting the application and the controller responsible for the visualisation of the model. The controller also passes actions from the view to the model.

`ch.unibe.ontoBrowser.dialogs` includes all dialogs used for creating resources, adding relations and saving an ontology. All dialogs are subclasses of `javax.swing.JDialog`.

`ch.unibe.ontoBrowser.event` contains the classes and interfaces used for events, which are used for the communication between the model and the view. The model throws events whenever changes are made. This is the case whenever the ontology is changed by adding or deleting a resource or whenever a single resource is modified. Components interested in those changes must implement the corresponding interface and can register as listeners to the events. This mechanism is used to synchronize the different views and to keep track of the changes.

`ch.unibe.ontoBrowser.graph` holds the classes used for the graph view. Most of these classes are subclasses of JGraph. They either extend the functionality of JGraph with convenience methods or they customize the behaviour and the layout for ALOE.

`ch.unibe.ontoBrowser.log` includes the class `Logger` which is responsible for writing the log file. It implements the singleton pattern. All information which needs to be logged should be written to the log file with the help of this class.

`ch.unibe.ontoBrowser.model` consists of all classes and interfaces which are responsible for modelling the OWL resources. One part of the classes in this package subclass the resource classes of knOWLer. They extend the functionaliy of knOWLer by adding convenience methods and some basic reasoning. Furthermore, it contains the interface `EditManager` together with an implementation. `EditManager` defines an interface for basic validity checks for changes. All methods return an integer indicating whether the change is alright, alright implying additional facts or would lead to an inconsistent state.

`ch.unibe.ontoBrowser.test` contains unit tests for some of the classes of the model. `OntologyTestSuite` is responsible for creating the test suite with all unit tests.

`ch.unibe.ontoBrowser.view` consists of all classes needed for building the GUI. `OBrowser` is the main frame of the application. Furthermore, classes for the different views of the resources are included.

`ch.unibe.ontoBrowser.view.actions` contains the actions for the GUI. Only the actions encapsulating functionality are implemented as a class, other actions are written as anonymous classes when they are used.

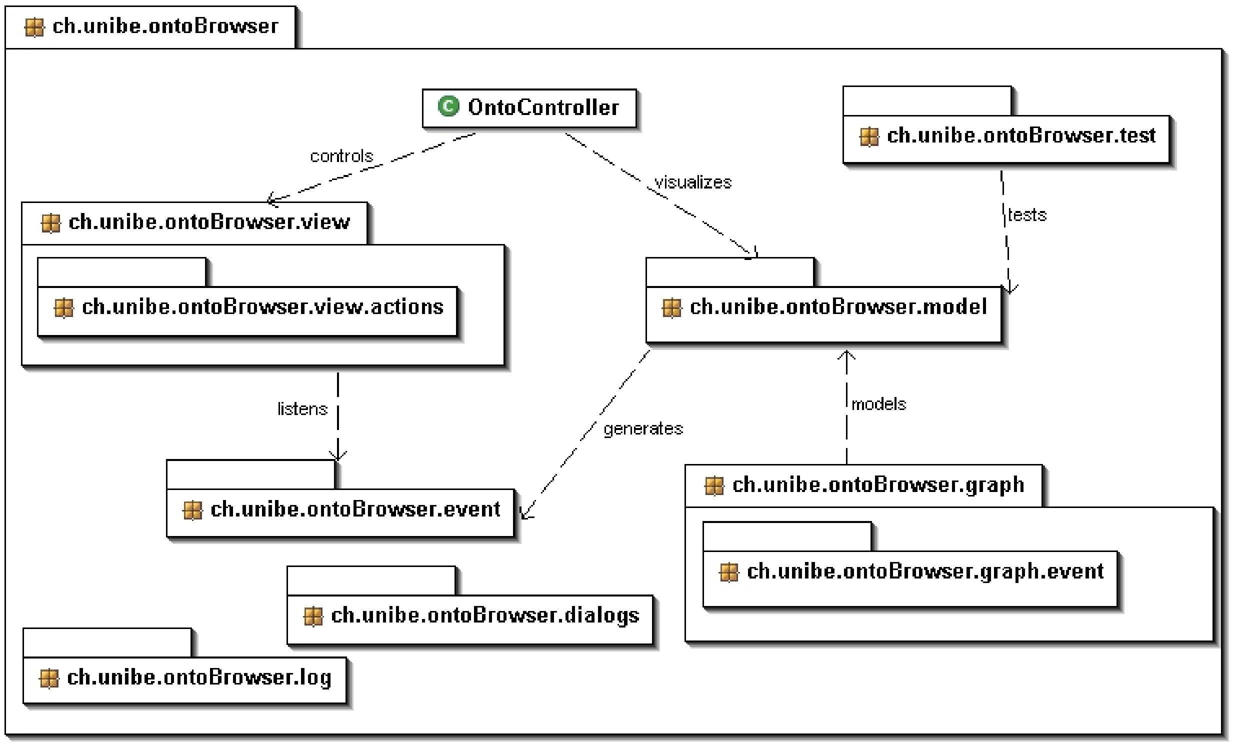

ch.unibe.ontoBrowser
OntoController
controls
visualizes
ch.unibe.ontoBrowser.test
tests
ch.unibe.ontoBrowser.view
ch.unibe.ontoBrowser.view.actions
ch.unibe.ontoBrowser.model
listens
generates
models
ch.unibe.ontoBrowser.event
ch.unibe.ontoBrowser.graph
ch.unibe.ontoBrowser.graph.event
ch.unibe.ontoBrowser.dialogs
ch.unibe.ontoBrowser.log

Figure 4.1: UML diagram of the package structure and dependencies.

ALOE implements the Model-View-Controller (MVC) pattern for the graph view. The model (package `ch.unibe.ontoBrowser.model`) consists of classes representing the OWL resources which are derived from the classes proviced by knOWLer. The view is made up of subclasses of classes of JGraph (package `ch.unibe.ontoBrowser.graph`). The controller is a class responsible for translating the objects of the model to graph objects for the view. Appendix B contains UML diagrams for the most important classes explaining the design of the packages.

4.3 Implementation Examples

This section contains the interface `EditManager`:

```
package ch.unibe.ontoBrowser.model;

import ch.unine.kid.knowler.OntologyResource;
import ch.unine.kid.knowler.RestrictionConstraint;
```

```
/**
 * An edit manager is responsible for checking the
 * validity of a statement. <code>EditManager</code>
 * defines the minimal requirements for the kind of statements
 * which needed to be checked. Implementing classes can especially
 * vary in the warning level. <br>
 * Each Method can return one of the following states:
 * <ul>
 * <li><code>EditManager.OK</code> if everything is ok.
 * <li><code>EditManager.WARNING</code> if the user should be warned
    that
 * after this change additional knowledge will be inferred.
 * <li><code>EditManager.ERROR</code> if the change would cause an
    invalid state
 * for the ontology.
 * </ul>
 * If <code>ERROR</code> or <code>WARNING</code> was returned, an
    explication of the
 * problem should be accessible by calling <code>getLastExplication</
    code>
 * @author Daria Spescha
 */
public interface EditManager {

    public static int OK = 0;
    public static int WARNING = 1;
    public static int ERROR = 2;

    /**
     * Checks whether an individual can be an instance of the given
        class.
     * @param individual the intended instance
     * @param classRes the class <code>individual</code> should be an
     * instance of.
     * @return See {@linkplain EditManager above} for the possible
        return values.
     */
    public int canBeInstanceOf(ExtendedIndividualResource individual,
       ExtendedClassResource classRes);

    /**
     * Checks whether two properties can be inverse.
     * @return See {@linkplain EditManager above} for the possible
        return values.
     */
```

```
public int canBeInverse(ExtendedPropertyResource property1,
   ExtendedPropertyResource property2);

/**
 * Checks whether the given property can be the superProperty
 * of the given subproperty.
 * @param sub the potential subproperty
 * @param superP the potential superproperty
 * @return See {@linkplain EditManager above} for the possible
    return values.
 */
public int canBeSuperproperty(ExtendedPropertyResource sub,
   ExtendedPropertyResource superP);

/**
 * Tests whether the property can be symmetric.
 * @return See {@linkplain EditManager above} for the possible
    return values.
 */
public int canBeSymmetric(ExtendedPropertyResource property);

/**
 * Collects all restrictions the given class has on the given
    property.
 * @param classRes the class to search the restrictions of.
 * @param property the property which should be in the onProperty
    part
 * of the restriction.
 * @return an array with the constraints found.
 */
public RestrictionConstraint[] getRestrictionsOn(
   ExtendedClassResource classRes, ExtendedPropertyResource
   property);

/**
 * Collects all restriction constraints the individual has on the
    given
 * property.
 * @param individual the individual to collect its constraints.
 * @param property the property for the onProperty part.
 * @return an array with the constraints found.
 */
public RestrictionConstraint[] getRestrictionsOn(
   ExtendedIndividualResource individual, ExtendedPropertyResource
    property);
    /**
     * Checks if the statement <code>(subject, predicate, object)</
        code>
     *  is a valid statement.
     * @return See {@linkplain EditManager above} for the possible
        return values.
     */
    public int isValidStatement(ExtendedIndividualResource subject,
       ExtendedPropertyResource predicate, ExtendedIndividualResource
       object);

    /**
     * Tests whether the given individuals can be sameAs.
     * @return See {@linkplain EditManager above} for the possible
        return values.
     */
    public int canBeSameAs(ExtendedIndividualResource res1,
       ExtendedIndividualResource res2);

    /**
     * Tests whether the given individuals can be defined to be
        differentFrom each other
     * @return See {@linkplain EditManager above} for the possible
        return values.
     */
    public int canBeDifferentFrom(ExtendedIndividualResource res1,
       ExtendedIndividualResource res2);

    /**
     * Gets the explication why the last check method returned
     * an error or warning.
     * @return The explication for the last error or warning.
     */
    public String getLastExplication();

    /**
     * Test whether a resource can be renamed to the given new name.
     * @return See {@linkplain EditManager above} for the possible
        return values.
     */
    public int canBeRenamed(OntologyResource res, String newname);
}
```

Chapter 5

Comparison with Other Ontology Editors

In this chapter, some other ontology editors are presented briefly and compared with ALOE. A short overview is given in table 5.1 where the criterias are split in four categories:

OS: The operating systems the editor runs on.
Languages: A selection of ontology languages the editor supports.
Browsing: Some criterias indicating how easy it is to browse and explore an existing ontology.
Editing Support: A selection of edit actions which are supported. (The possibility to do the edit in the source code is not counted as editing support.)

5.1 Protégé

Characteristics

Name: Protégé
Developer: Stanford Medical Informatics at the Stanford University School of Medicine
License: Freeware, open source
Homepage: `http://protege.stanford.edu/`
Supported Languages: RDF Schema, CLIPS, OWL with additional plug-in
Version: 3.0[1]
Comment: Requires JDK 1.3

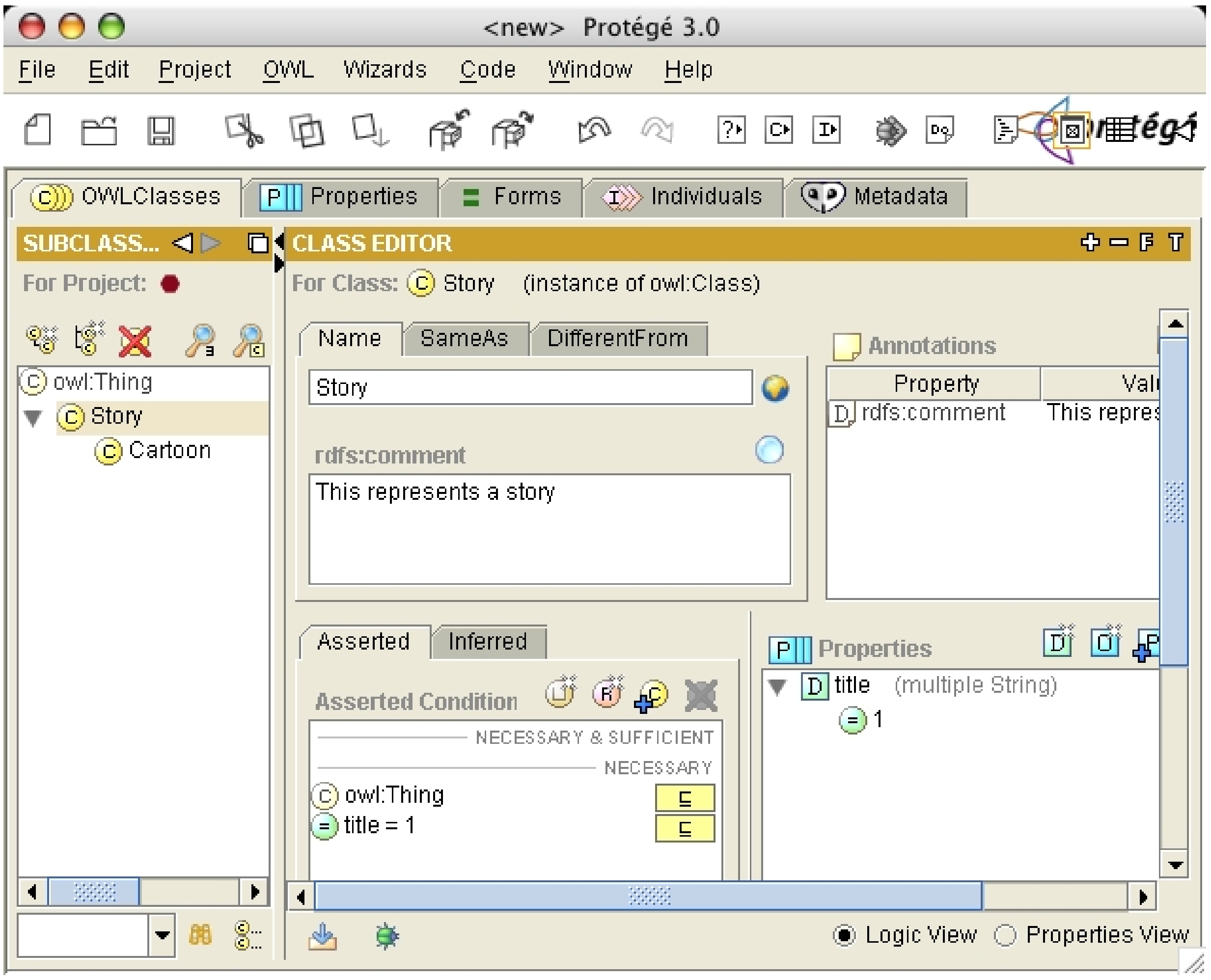
Protégé is intended as an editor for knowledge base systems. It offers a special OWL view (see Figure 5.1) for editing OWL ontologies. On the left hand side, it has a list of resources for each resource type, like ALOE. Classes are listed in a hierarchical tree. This is convenient for a first overview of the hierarchy, but it is confusing if a class has more than

[1]Latest version (June 2005)

one superclass. The edit masks for the resources offer many features and it is possible to edit all axioms available from OWL Full. However, it lacks uniformity as not all axioms are treated the same way. In the edit mask for classes, it is not clear which are the superclasses because they are not listed explicitly.

If an existing ontology is loaded, Protégé opens the general view and not the OWL specific one and it is not possible to switch the perspective. In the general view, some elements are named differently, for example properties are called slots, which is confusing. Unlike ALOE, it is not possible to load several ontologies at the same time. Furthermore Protégé does not support the use of multiple namespaces.

Protégé is suitable for advanced users, as profound knowledge about ontologies is required and knowledge of (description) logics is recommended because a lot of images used are taken from logics notation. Though, it does not provide an intuitive user interface for inexperienced users.



Name: Story

rdfs:comment: This represents a story

Property	Val
rdfs:comment	This repres

NECESSARY
owl:Thing ⊑
title = 1 ⊑

title (multiple String)
= 1

[✓ Logic View] Properties View

Figure 5.1: OWL view of Protégé

5.2 SWOOP

Characteristics

Name: SWOOP
Developer: MINDSWAP Research Group, University of Maryland, College Park
License: LGPL
Homepage: `http://www.mindswap.org/2004/SWOOP/`
Supported Languages: OWL
Version: 2.2.1[1]
Comment:

SWOOP looks like a common web-browser and has an intuitive user interface. The application lets the user load several ontologies and switch between them. Like ALOE, it has a list with all resources on the left hand side, but the resources are shown as a hierarchical tree.

SWOOP offers four different perspectives of the resources, including the OWL XML Syntax and one called "Concise Format". The latter enables the editing of resources with add and remove buttons as well as wizards for adding a relation. It is also possible to navigate from one resource to the detail view of its related resources.

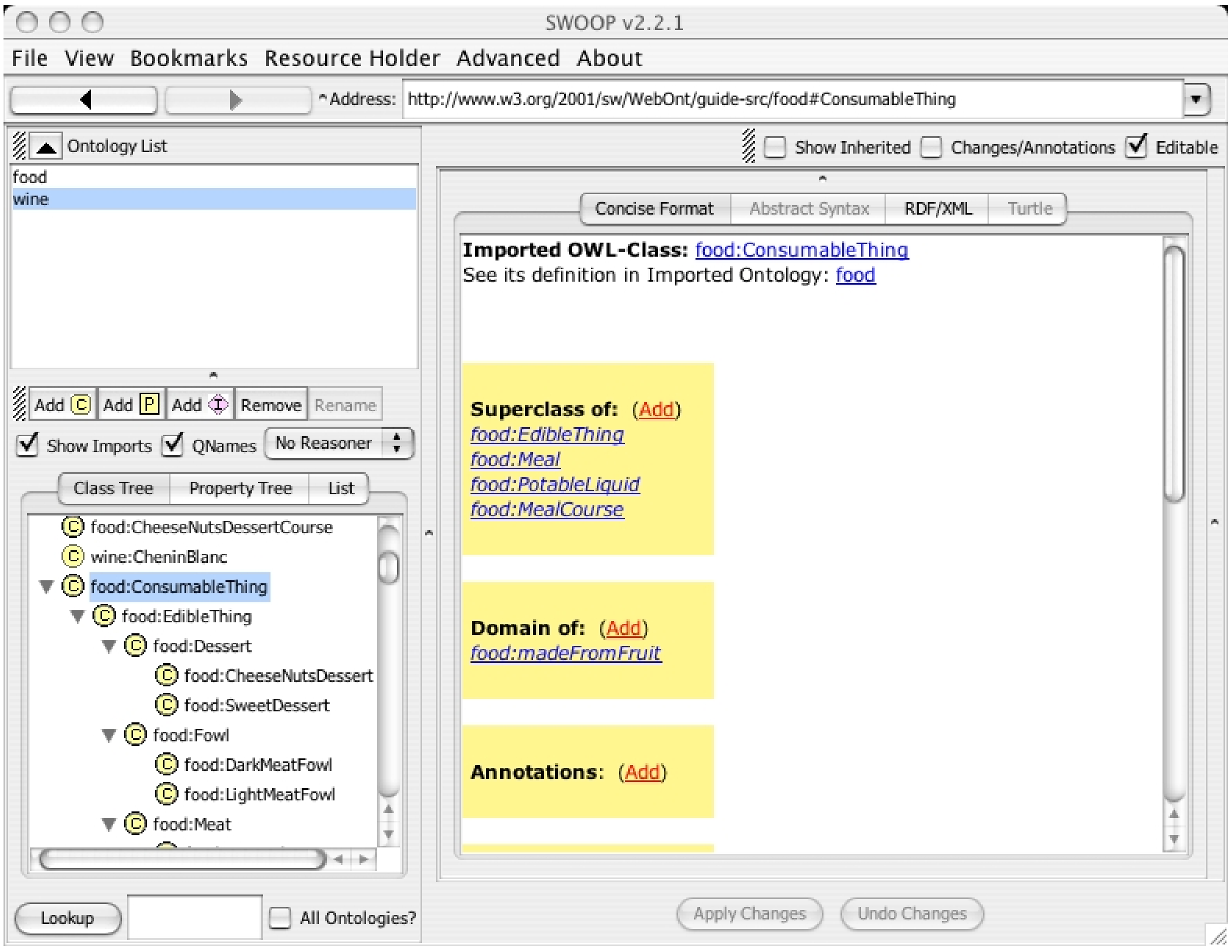
In contrast to ALOE, it is not possible to look at different resources at the same time. SWOOP also lacks a graphical representation.

5.3 SWeDE

Characteristics

Name: SWeDE (Semantic Web Development Environment)
Developer: BBN Technologies
License: Freeware
Homepage: `http://owl-eclipse.projects.semwebcentral.org/`
Supported Languages: OWL
Current version: 2.0.0[1]
Comment: Requires Eclipse

SWeDE is available as a plug-in for Eclipse 3.0. It offers an editor for the OWL XML syntax with syntax highlighting and auto-completion as well as an editor for restrictions. Furthermore, it has a validator which does not seem to work properly and does not provide very informative error messages. SWeDE is a good editor for editing OWL files directly, though.



Address: http://www.w3.org/2001/sw/WebOnt/guide-src/food#ConsumableThing

[✓ Show Imports] [✓ QNames] No Reasoner

Show Inherited Changes/Annotations [✓ Editable]

Imported OWL-Class: food:ConsumableThing
See its definition in Imported Ontology: food

Superclass of: (Add)
food:EdibleThing
food:Meal
food:PotableLiquid
food:MealCourse

Domain of: (Add)
food:madeFromFruit

Annotations: (Add)

Figure 5.2: SWOOP

5.4 KAON

Characteristics

Name: KAON
Developer: FZI (Forschungszentrum Informatik) and AIFB (Institut für Angewandte Informatik und Formale Beschreibungsverfahren, University of Karlsruhe)
License: LGPL
Homepage: `http://kaon.semanticweb.org/`
Supported Languages: RDF(S)
Version: 1.2.7[1]
Comment: Requires Java J2SE 1.4.0_01 or Java Webstart

Among the editors examined, KAON is the only one offering a graphical representation. However, KAON supports only RDF(S) and not OWL. Like ALOE, it displays the ontology as a graph. Though, KAON shows the properties also as vertices and not as edges, which is quite irritating. Furthermore, the layout of the graph is done automatically and it is not possible to drag vertices by hand. It does not have a list of all resources. Thus, the only possibility to explore an ontology is by navigating through the graph starting from "Root". As the graph grows too complex, some nodes can be hidden, but in opposition to ALOE, it is not possible to make them visible again.

Like ALOE, KAON uses colors to distinguish between the resource types. Also the edges have different colors for the different kinds of relations. This is practical to get an overview, but it would be difficult to adopt the coloring of axioms to ALOE as OWL supports a lot more axioms.

KAON lets the user connect two resource by drawing a line from one resource to another one. The type of the relation is identified depending on the type of the resources.

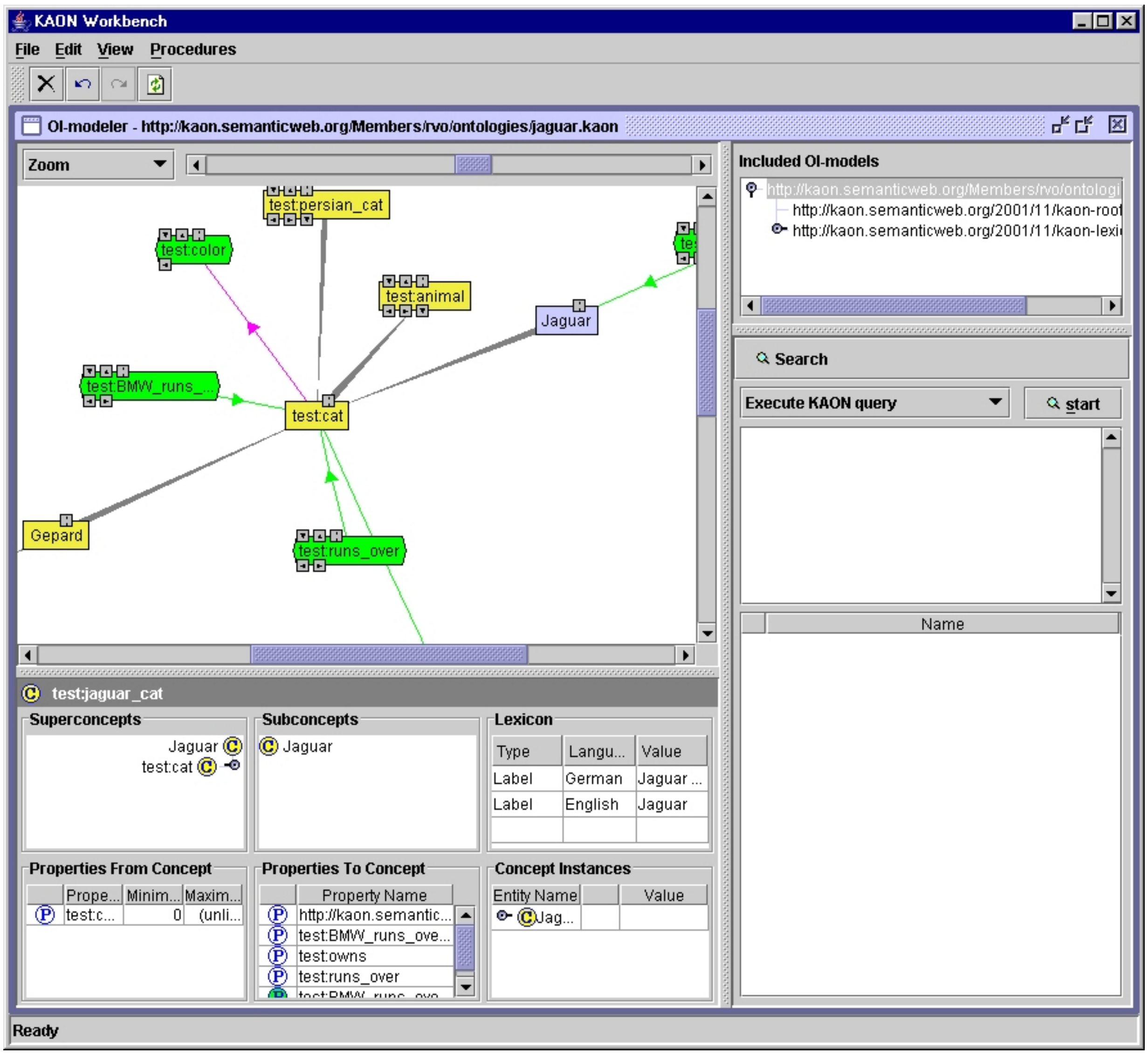


Figure 5.3: KAON

		ALOE	*SWOOP*	*KAON*	*Protégé*	*SWeDE*
OS	Linux	✓	✓	✓	✓	✓
	Mac	✓	✓	✗	✓	✓
	Windows	✗	✓	✓	✓	✓
Languages	OWL Lite	✓	✓	✗	✓	✓
	OWL DL	✗	✓	✗	✓	✓
	OWL Full	✗	✓	✗	✓	✓
	RDF	✗	✗	✓	✓	✗
	DAML	✗	✗	✗	✓	✗
Browsing	**Graphical Representation**	✓	✗	✓	✗	✗
	Selecting the visualisation	✓	✗	✗	✗	✗
	Intuitive User Interface	✓	✓	✗	✗	✓
	Navigation	✓	✓	✓	✗	✗
	Visibility of axioms	✓	✓	✓	✓	✓
	Loading several ontologies	✓	✓	✓	✗	✓
Editing Support	Individuals	✓	✓	✓	✓	✗
	Individual identity	✓	✓	✗	✓	✗
	Classes	✓	✓	✓	✓	✗
	Subclass	✓	✓	✓	✓	✗
	Intersection	✓	✓	✗	✓	✗
	Restriction	✓	✓	✗	✓	✗
	Equivalent classes	✓	✓	✗	✓	✗
	Properties	✓	✓	✓	✓	✗
	Subproperty	✓	✓	✓	✓	✗
	Domain and Range	✓	✓	✓	✓	✗
	Equivalent properties	✓	✓	✗	✓	✗
	Rename resources	✓	✓	✓	✓	✗
	User defined annotations	✓	✓	✗	✓	✗
	Save selected ontology	✓	✓	✓	✗	✓

Table 5.1: Comparison of Ontology Editors

Chapter 6

Conclusion and Further Work

6.1 Conclusion

Based on the requirements for ALOE and the previous chapters, this section explains in short terms how ALOE achieves this criteria and why it complies better with the requirements than other ontology editors.

Ease of use: As all changes are done in edit masks and dialogues, it is easy for users without any knowledge about the OWL syntax to edit or create an ontology. Furthermore, the user interface is intuitive as all main actions are available in the toolbars and navigation through the ontology is possible with a right click on the resources.

General Overview: The navigation tab on the left hand side offers a fast overview of the existing resources. To avoid too much scrolling, the resources can also be listed alphabetically, sorted by the first letter of their name. In addition, the relations between the resources are accessible in the graph view. It is also possible to look at the details of several resources at the same time and compare them.

Graphical: The graph view offers a very intuitive graphical representation of an ontology.

Visibility: All axioms available in OWL Lite are visible in the detail view or in the OWL view. All properties, relations and so on can be edited in the edit mask.

Navigation: Related resources can be displayed in the graph with different commands which let the user select the kind of relations he wants to see. Furthermore, the details of all related resources can be viewed in the detail view of a resource.

Huge Ontologies: ALOE lets the user select the resources which are to be displayed in the graph view. Therefore, it is easy to concentrate on one part of the ontology and even huge ontologies can be handled. The separation of classes and individuals prevents the graphs from getting too complex. If the graph gets too big anyhow, the vertices and edges can be filtered by different criteria and afterwards reshown.

Furthermore, there is a mechanism to impede that too many resources are inserted to the graph by accident (see section 1.3.3).

ALOE is able to handle for example the Wordnet ontology (see `http://taurus.unine.ch/knowler/wordnet.html`). Though, it takes a while until the ontology is loaded into knOWLer.

Moreover, ALOE can handle several ontologies at the same time. It is possible to load an additional ontology to the current one in order to extend it or in order to edit it in parallel. Furthermore, the resources which should be saved can be specified by the namespace.

6.2 Further Work

Finally, I want to present some ideas for useful extensions to ALOE:

- Queries: It would be useful to have an interface for composing queries and presenting the result.
- Reasoner: A reasoner explaining the inferred facts would be valuable for experienced users.
- As mistakes are made very easily, an "Undo"functionality would be useful.
- At the moment, the colours used for instances and classes are fixed. It would be nice to be able to select these colours and set them as preferences.
- Syntax highlighting for the OWL view would make the OWL Syntax more readable.
- As JGraph offers many features for editing graphs, it would be nice to connect resources by dragging edges from one vertex to another one.

Appendices

Appendix A

Example Ontologies

A.1 Asterix & Obelix Ontologies

A.1.1 Story Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
<!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
<!ENTITY cartoon 'http://www.iam.unibe.ch/~spescha/cartoons#'>
<!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
<!ENTITY story 'http://www.iam.unibe.ch/~spescha/story#'>
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
]>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:cartoon="http://www.iam.unibe.ch/~spescha/cartoons#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:story="http://www.iam.unibe.ch/~spescha/story#"
   xmlns="http://www.iam.unibe.ch/~spescha/story#"
   xml:base="http://www.iam.unibe.ch/~spescha/story#">
  <owl:Ontology>
    <!--Author: Daria Spescha-->
    <owl:versionInfo>v 1.0</owl:versionInfo>
    <rdfs:comment>A basic ontology about stories.</rdfs:comment>
  </owl:Ontology>
  <!--Annotations-->
  <!--Classes-->
  <owl:Class rdf:about="#Story" />
```

```
<owl:Class rdf:about="#Character" />
<owl:Class rdf:about="#Person">
  <rdfs:subClassOf rdf:resource="#Being" />
</owl:Class>
<owl:Class rdf:about="#Language" />
<owl:Class rdf:about="#Film">
  <rdfs:subClassOf rdf:resource="#Story" />
</owl:Class>
<owl:Class rdf:about="#Novel">
  <rdfs:subClassOf rdf:resource="#Story" />
</owl:Class>
<owl:Class rdf:about="#Being" />
<owl:Class rdf:about="#Animal">
  <rdfs:subClassOf rdf:resource="#Being" />
</owl:Class>
<owl:Class rdf:about="#Dog">
  <rdfs:subClassOf rdf:resource="#Animal" />
</owl:Class>
<owl:Class rdf:about="#Tiger">
  <rdfs:subClassOf rdf:resource="#Animal" />
</owl:Class>
<!--Properties-->
<owl:ObjectProperty rdf:about="#hasFriend">
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:range rdf:resource="#Character" />
  <rdfs:domain rdf:resource="#Character" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#originalLanguage">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:range rdf:resource="#Language" />
  <rdfs:domain rdf:resource="#Story" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#translatedTo">
  <rdfs:range rdf:resource="#Language" />
  <rdfs:domain rdf:resource="#Story" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#mainCharacter">
  <rdfs:range rdf:resource="#Character" />
  <rdfs:domain rdf:resource="#Story" />
  <rdfs:subPropertyOf rdf:resource="#hasCharacter" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#writtenBy">
  <rdfs:range rdf:resource="#Person" />
  <rdfs:domain rdf:resource="#Story" />
</owl:ObjectProperty>
```

```
  <owl:DatatypeProperty rdf:about="#title">
    <rdfs:range rdf:resource="&xsd;string" />
    <rdfs:domain rdf:resource="#Story" />
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:about="#appears">
    <rdfs:range rdf:resource="#Story" />
    <rdfs:domain rdf:resource="#Character" />
    <owl:inverseOf rdf:resource="#hasCharacter" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasEnemy">
    <rdf:type rdf:resource="&owl;SymmetricProperty" />
    <rdfs:range rdf:resource="#Character" />
    <rdfs:domain rdf:resource="#Character" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#authorOf">
    <rdfs:range rdf:resource="#Story" />
    <rdfs:domain rdf:resource="#Person" />
    <owl:inverseOf rdf:resource="#writtenBy" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasCharacter">
    <rdfs:range rdf:resource="#Character" />
    <rdfs:domain rdf:resource="#Story" />
  </owl:ObjectProperty>
  <!--Individuals-->
  <Language rdf:about="#French" />
  <Language rdf:about="#English" />
  <Language rdf:about="#German">
    <rdfs:label>Deutsch</rdfs:label>
  </Language>
</rdf:RDF>
```

A.1.2 Cartoon Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
<!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
<!ENTITY cartoon 'http://www.iam.unibe.ch/~spescha/cartoons#'>
<!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
<!ENTITY story 'http://www.iam.unibe.ch/~spescha/story#'>
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
]>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:cartoon="http://www.iam.unibe.ch/~spescha/cartoons#"
  xmlns="http://www.iam.unibe.ch/~spescha/cartoons#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:story="http://www.iam.unibe.ch/~spescha/story#"
  xml:base="http://www.iam.unibe.ch/~spescha/cartoons#">
<owl:Ontology>
  <!--Author: Daria Spescha-->
  <owl:versionInfo>v 1.0</owl:versionInfo>
  <rdfs:comment>A basic ontology about cartoons.</rdfs:comment>
  <owl:imports rdf:resource="http://www.iam.unibe.ch/~spescha/story"
      />
</owl:Ontology>
<!--Annotations-->
<!--Classes-->
<owl:Class rdf:about="#Cartoon">
  <rdfs:subClassOf rdf:resource="&story;Story" />
  <rdfs:comment>Cartoons are special stories, they are painted.</
     rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#CartoonSeries">
  <rdfs:subClassOf rdf:resource="#Cartoon" />
</owl:Class>
<owl:Class rdf:about="#ComicStripSeries">
  <rdfs:subClassOf rdf:resource="#CartoonSeries" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasEpisode" />
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</
         owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment>Some cartoons appear as comic strip series in
     newspapers like "Calvin␣And␣Hobbes".</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#BookSeries">
  <rdfs:subClassOf rdf:resource="#CartoonSeries" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasBook" />
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</
         owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
```

```
    <rdfs:comment>Some cartoons appear as books such as Asterix and
        Obelix.</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="#BookEpisode">
    <rdfs:subClassOf rdf:resource="#Episode" />
  </owl:Class>
  <owl:Class rdf:about="#Episode" />
  <owl:Class rdf:about="#Hero">
    <rdfs:subClassOf rdf:resource="&story;Character" />
  </owl:Class>
  <owl:Class rdf:about="#BadGuy">
    <rdfs:subClassOf rdf:resource="&story;Character" />
  </owl:Class>
  <owl:Class rdf:about="#newspaper" />
  <!--Properties-->
  <owl:ObjectProperty rdf:about="#hasEpisode">
    <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
    <rdfs:range rdf:resource="#Episode" />
    <rdfs:domain rdf:resource="#CartoonSeries" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasBook">
    <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
    <rdfs:range rdf:resource="#BookEpisode" />
    <rdfs:domain rdf:resource="#BookSeries" />
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="#title" />
  <owl:DatatypeProperty rdf:about="#isbn">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:range rdf:resource="&xsd;string" />
    <rdfs:domain rdf:resource="#BookEpisode" />
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:about="#belongsToSeries">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:range rdf:resource="#CartoonSeries" />
    <rdfs:domain rdf:resource="#Episode" />
    <owl:inverseOf rdf:resource="#hasEpisode" />
    <owl:inverseOf rdf:resource="#hasBook" />
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="#episodeTitle">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:range rdf:resource="&xsd;string" />
    <rdfs:domain rdf:resource="#Episode" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#episodeNo">
    <rdfs:range rdf:resource="&xsd;integer" />
    <rdfs:domain rdf:resource="#Episode" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#firstAppearance">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:range rdf:resource="&xsd;date" />
    <rdfs:domain rdf:resource="#Episode" />
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:about="#appearedIn">
    <rdfs:range rdf:resource="#newspaper" />
    <rdfs:domain rdf:resource="#ComicStripSeries" />
  </owl:ObjectProperty>
  <!--Individuals-->
  <rdf:Description rdf:about="#Asterix">
    <rdf:type rdf:resource="#Hero" />
    <rdf:type rdf:resource="&story;Person" />
  </rdf:Description>
  <rdf:Description rdf:about="#Obelix">
    <rdf:type rdf:resource="#Hero" />
    <rdf:type rdf:resource="&story;Person" />
    <story:hasFriend rdf:resource="#Asterix" />
  </rdf:Description>
  <BookSeries rdf:about="#AsterixAndObelix">
    <cartoon:hasBook rdf:resource="#Asterix1" />
    <story:originalLanguage rdf:resource="&story;French" />
    <story:translatedTo rdf:resource="&story;English" />
    <story:translatedTo rdf:resource="&story;German" />
    <story:translatedTo rdf:resource="#Latin" />
    <story:mainCharacter rdf:resource="#Asterix" />
    <story:mainCharacter rdf:resource="#Obelix" />
    <story:writtenBy rdf:resource="#Goscinny" />
    <story:writtenBy rdf:resource="#Uderzo" />
    <story:title>Asterix And Obelix</story:title>
  </BookSeries>
  <story:Language rdf:about="#Latin" />
  <story:Person rdf:about="#Goscinny">
    <rdfs:comment>Rene Goscinny</rdfs:comment>
  </story:Person>
  <story:Person rdf:about="#Uderzo">
    <rdfs:label>Albert Uderzo</rdfs:label>
  </story:Person>
  <BookEpisode rdf:about="#Asterix1">
    <cartoon:episodeTitle>Asterix the Gaul</cartoon:episodeTitle>
    <cartoon:episodeNo rdf:datatype="&xsd;nonNegativeInteger">1</
       cartoon:episodeNo>
  </BookEpisode>
  <rdf:Description rdf:about="#Idefix">
    <rdf:type rdf:resource="&story;Character" />
    <rdf:type rdf:resource="&story;Dog" />
    <story:hasFriend rdf:resource="#Obelix" />
    <story:appears rdf:resource="#AsterixAndObelix" />
  </rdf:Description>
  <newspaper rdf:about="#LosAngelesTimes" />
  <ComicStripSeries rdf:about="#CalvinAndHobbes">
    <cartoon:hasEpisode rdf:resource="#firstCH" />
    <story:mainCharacter rdf:resource="#Calvin" />
    <story:mainCharacter rdf:resource="#Hobbes" />
    <story:writtenBy rdf:resource="#Watterson" />
    <story:hasCharacter rdf:resource="#Moe" />
    <cartoon:appearedIn rdf:resource="#LosAngelesTimes" />
  </ComicStripSeries>
  <story:Person rdf:about="#Watterson">
    <rdfs:label>Bill Watterson</rdfs:label>
  </story:Person>
  <rdf:Description rdf:about="#Calvin">
    <rdf:type rdf:resource="&story;Character" />
    <rdf:type rdf:resource="&story;Person" />
  </rdf:Description>
  <rdf:Description rdf:about="#Hobbes">
    <rdf:type rdf:resource="&story;Character" />
    <rdf:type rdf:resource="&story;Tiger" />
    <story:hasFriend rdf:resource="#Calvin" />
  </rdf:Description>
  <Episode rdf:about="#firstCH">
    <cartoon:episodeNo>1</cartoon:episodeNo>
    <cartoon:firstAppearance>11-18-1985</cartoon:firstAppearance>
    <rdfs:comment>The first episode of Calvin And Hobbes</rdfs:comment
      >
  </Episode>
  <BadGuy rdf:about="#Moe">
    <story:hasEnemy rdf:resource="#Calvin" />
  </BadGuy>
</rdf:RDF>
```

A.2 Asterix & Obelix Graphs

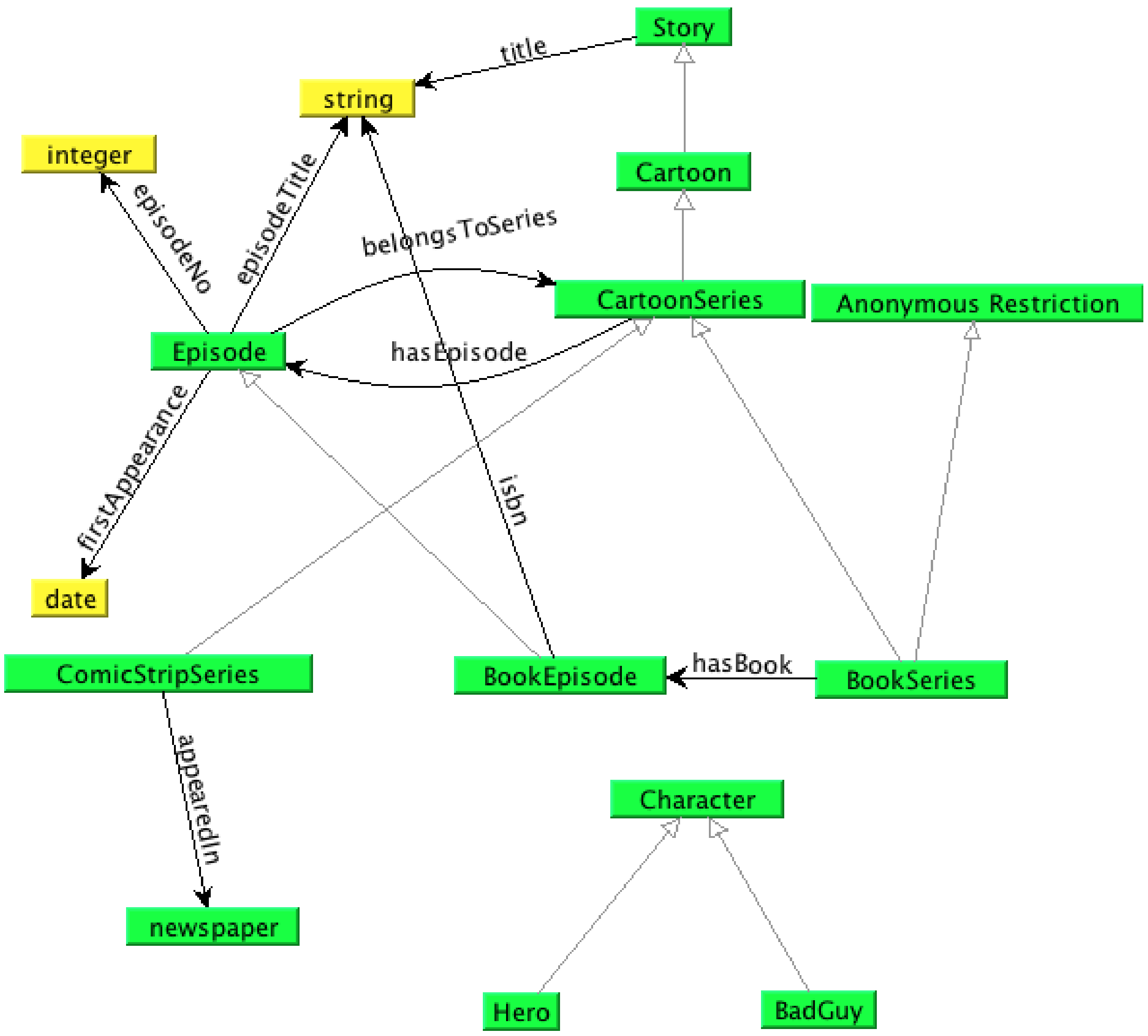


Figure A.1: Class graph of the cartoon ontology

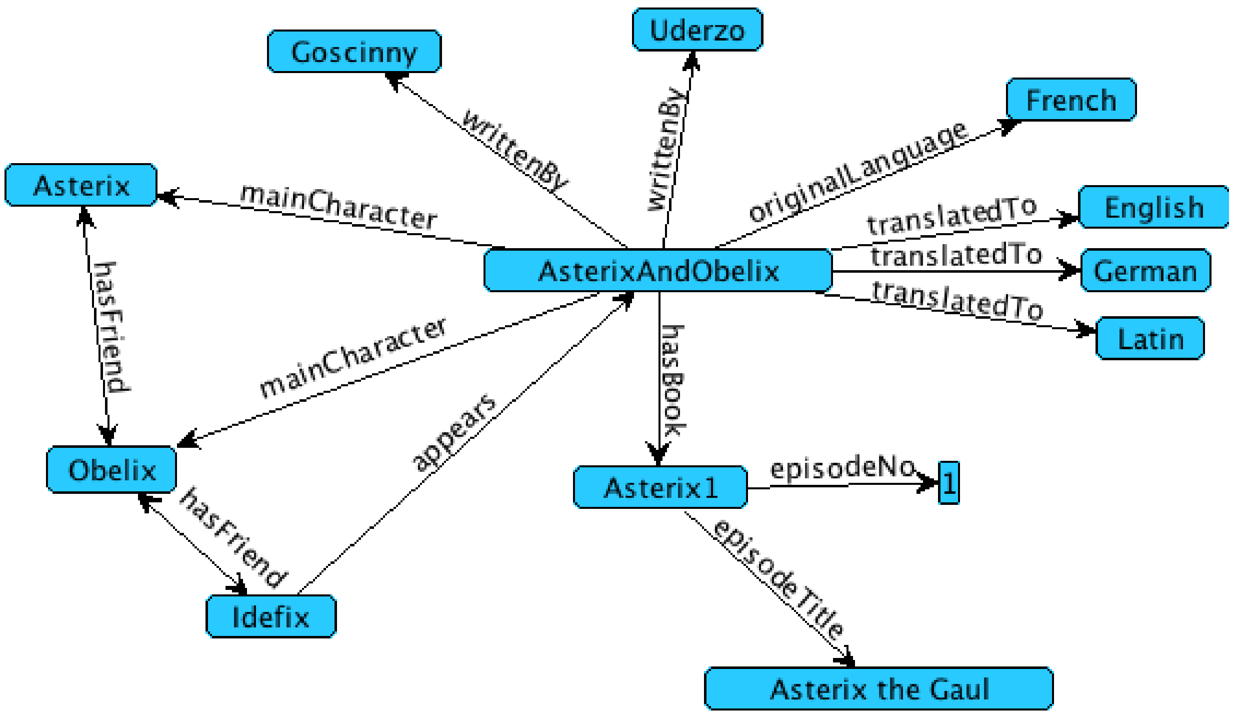


Figure A.2: Individual graph for "Asterix and Obelix"

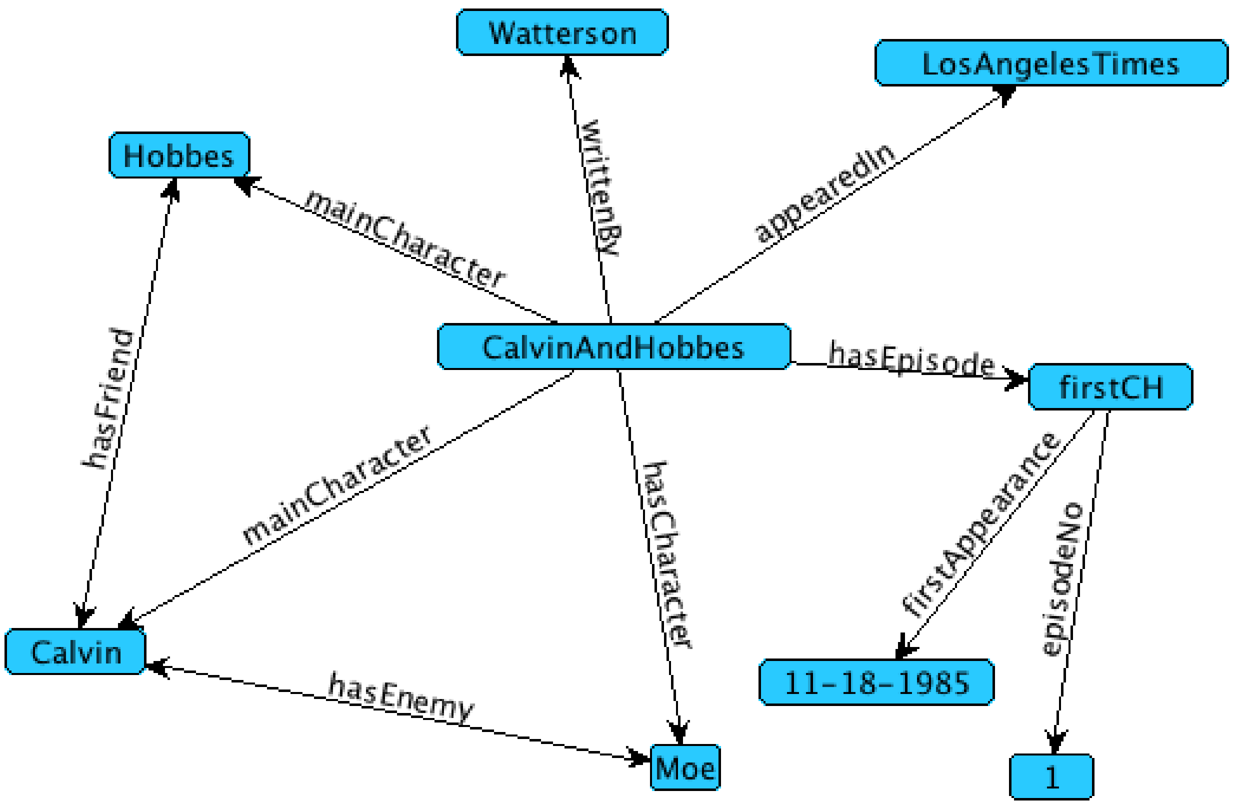


Figure A.3: Individual graph for “Calvin and Hobbes”

Appendix B

UML Diagrams

This appendix presents UML diagrams of the important classes by package:

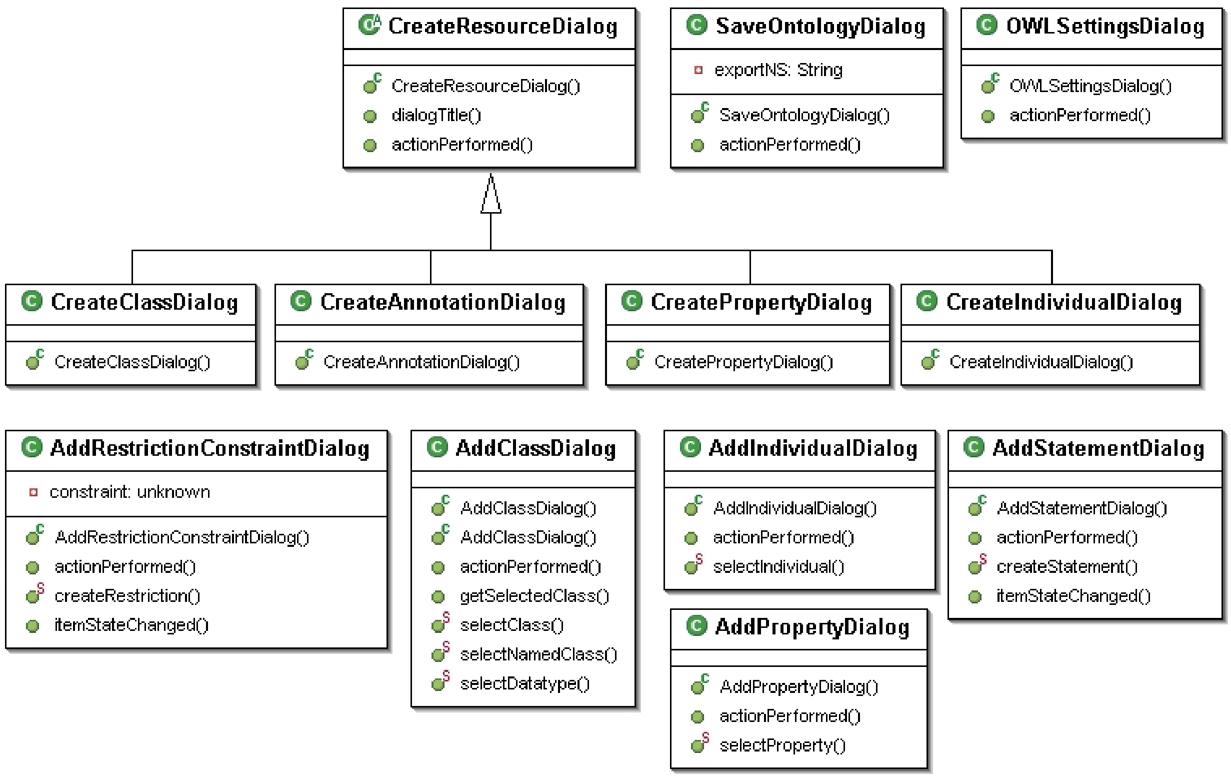


Figure B.1: Class Diagram for package ch.unibe.ontoBrowser.dialogs

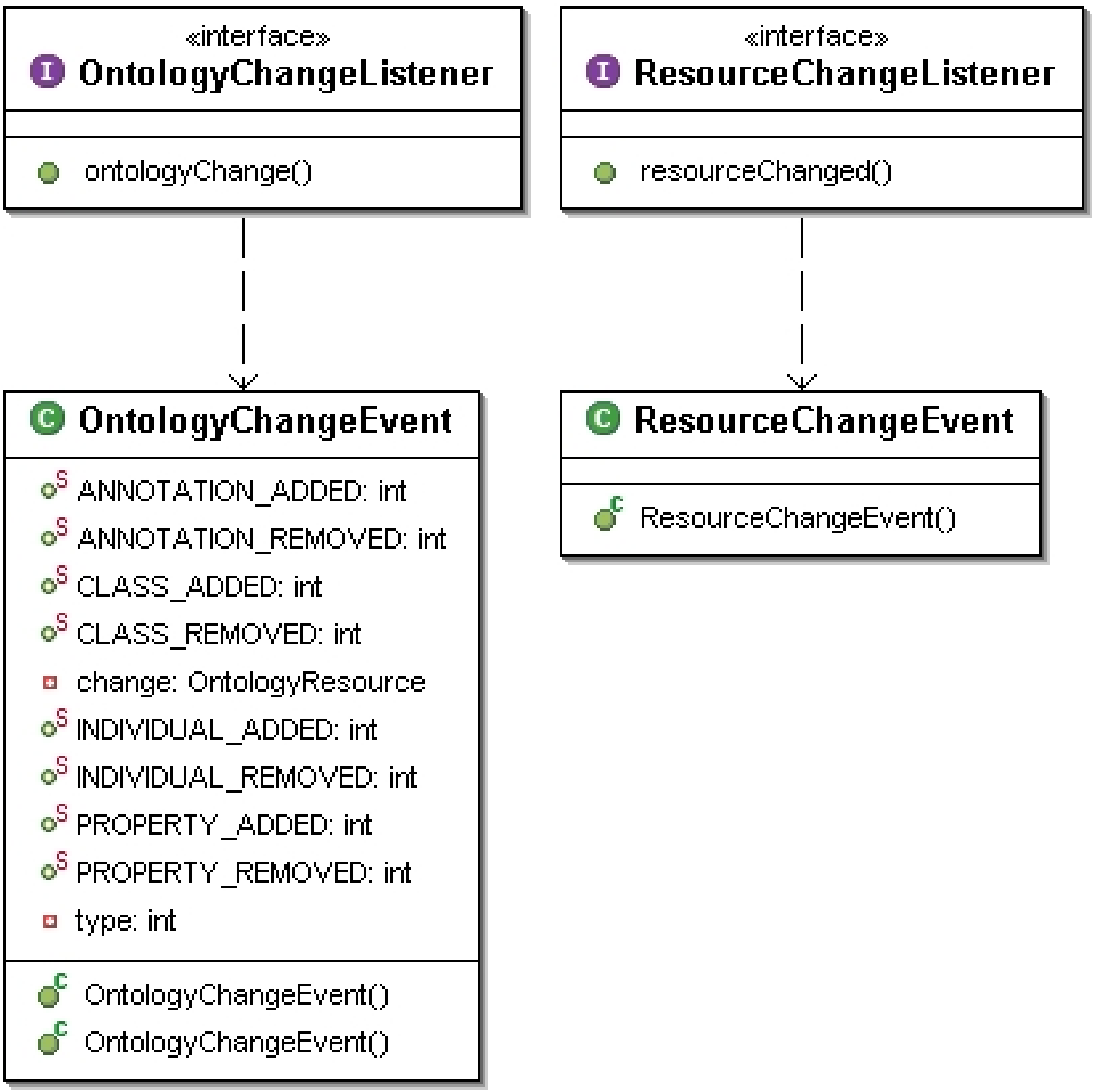


Figure B.2: Class Diagram for package ch.unibe.ontoBrowser.event

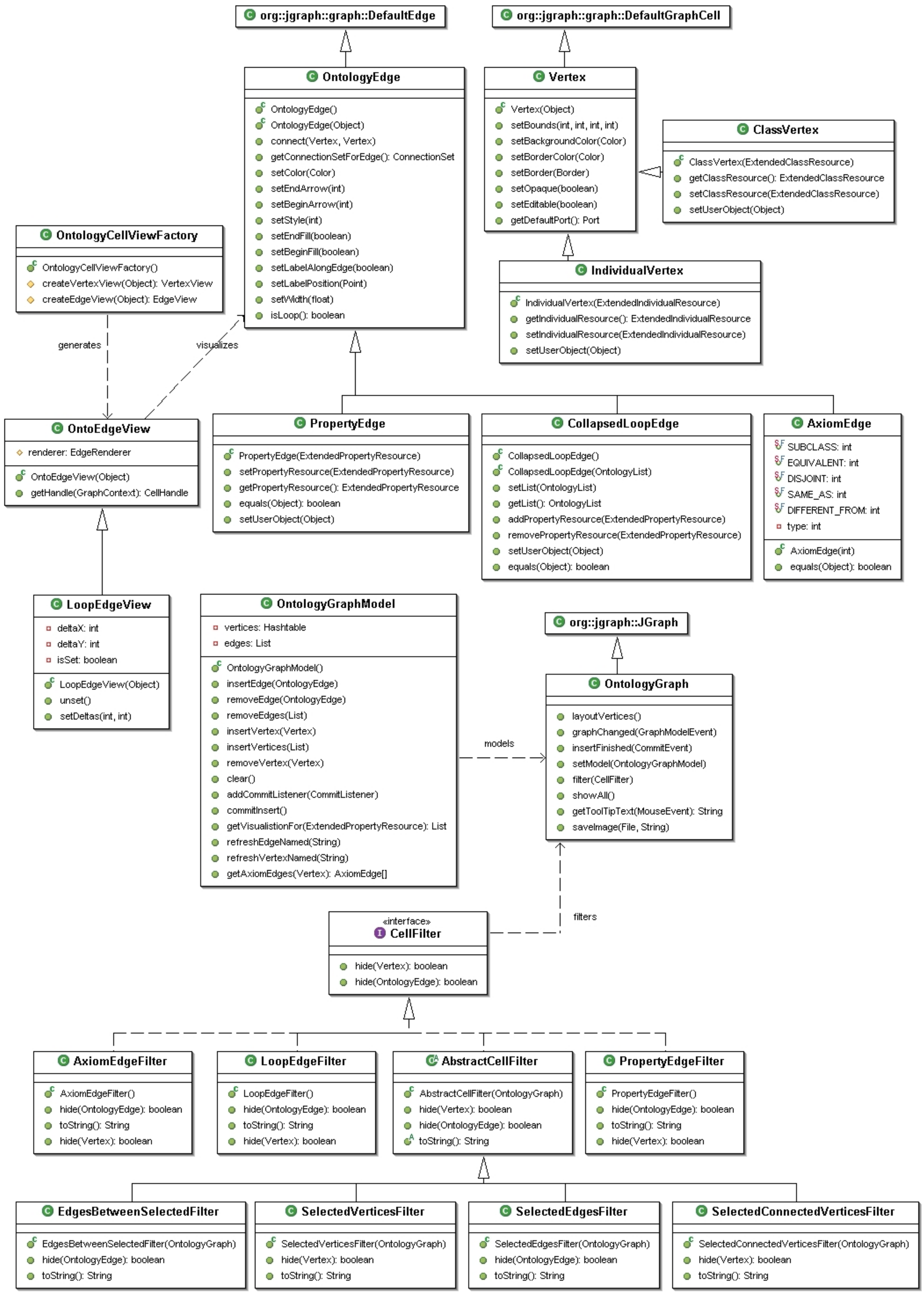


Figure B.3: Class Diagram for package ch.unibe.ontoBrowser.graph

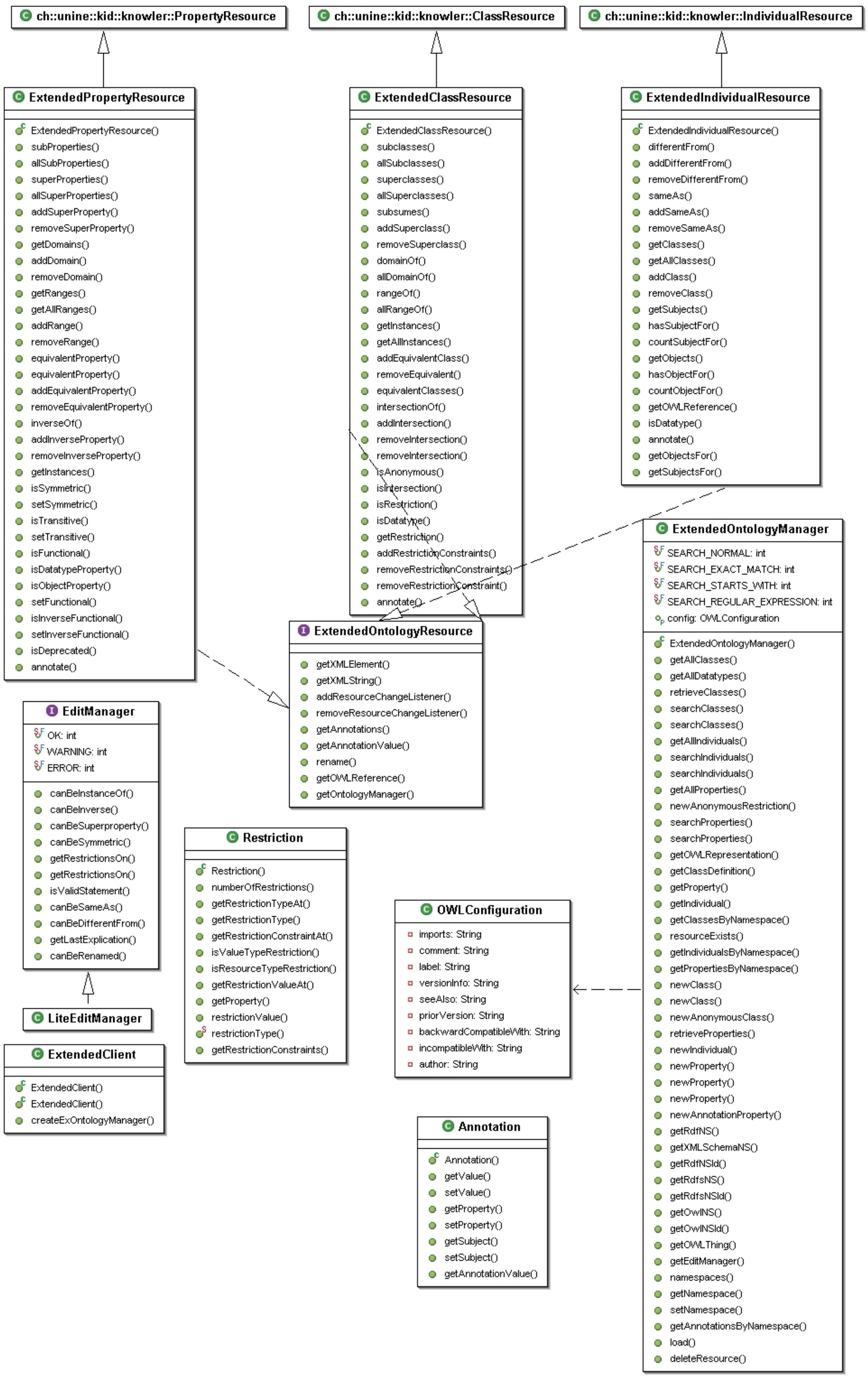


Figure B.4: Class Diagram for package ch.unibe.ontoBrowser.model

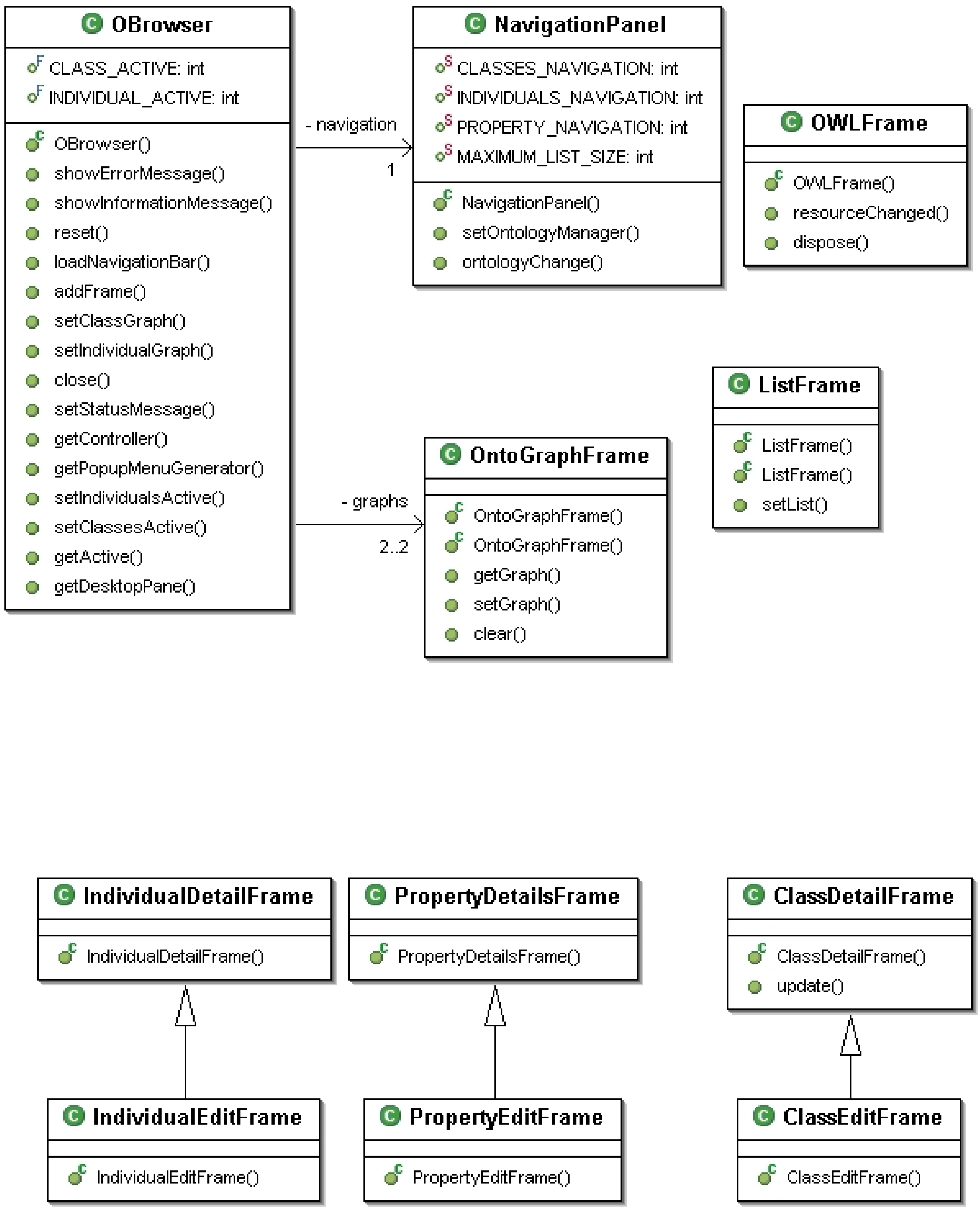


Figure B.5: Class Diagram for package ch.unibe.ontoBrowser.view

Bibliography

[1] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004. ISBN 0-262-01210-3.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, L. McGuinness, Deborah, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[4] Claudia Ciorascu, Iulian Ciorascu, and Kilian Stoffel. knOWLer - Ontological Support for Information Retrieval Systems. In *Proceedings of 26th Annual International ACM SIGIR 2003 Conference, Workshop on Semantic Web*, Toronto, Canada, August 2003. URL `http://taurus.unine.ch/knowler/sigir2003.pdf`.

[5] Iulian Ciorascu, Claudia Ciorascu, and Kilian Stoffel. Scalable Ontology Implementation Based on knOWLer. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003), Workshop on Practical and Scalable Semantic Systems*, Florida, USA, October 2003. URL `http://km.aifb.uni-karlsruhe.de/ws/psss03/proceedings/stoffel-et-al.pdf`.

[6] David De Roure, R. Jennings, Nicholas, and R. Shadbolt, Nigel. The Semantic Grid: A Future e-Science Infrastructure. In F. Berman, A.J.G. Hey, and G. Fox, editors, *Grid Computing*, pages 437–470. John Wiley, 2003. ISBN 0470853190.

[7] Wikipedia The Free Encyclopedia. Ontology (computer science), Mai 2005. URL `http://en.wikipedia.org/wiki/Ontology_%28computer_science%29`.

[8] Wikipedia The Free Encyclopedia. Semantic web, Mai 2005. URL `http://en.wikipedia.org/wiki/Semantic_Web`.

[9] Jeff Heflin. OWL Web Ontology Language Use Cases and Requirements, February 2004. URL `http://www.w3.org/TR/2004/REC-webont-req-20040210/`.

[10] Jeff Heflin and James Hendler. A Portrait of the Semantic Web in Action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[11] Ian Horrocks, F. Patel-Schneider, Peter, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. ISSN 1570-8268. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/HoPH03a.pdf`.

[12] L. McGuinness, Deborah and Frank van Harmelen. OWL Web Ontology Language Overview, February . URL `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[13] Michael K. Smith, Chris Welty, and L. McGuinness, Deborah. OWL Web Ontology Language Guide, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-guide-20040210/`.