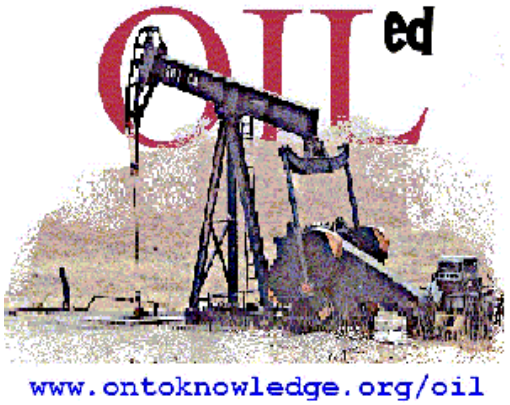

OilEd 3.4 Manual

Sean Bechhofer
Information Management Group
Department of Computer Science
University of Manchester

seanb@cs.man.ac.uk

Overview

OilEd is a simple editor that allows the user to create and edit OIL ontologies. The main intention behind OilEd is to provide a simple, freeware editor that demonstrates the use of, and stimulates interest in, DAML+OIL. OilEd is **not** intended as a full ontology development environment — it will **not** actively support the development of large-scale ontologies, the migration and integration of ontologies, versioning, argumentation and many other activities that are involved in ontology construction. It should, however, provide enough to allow the basic construction of OIL ontologies and demonstrate the power of the connection to the FaCT reasoner.

This is a quick user guide to the OilEd ontology editor. It assumes that the user knows what DAML+OIL is and the basics of DAML+OIL ontologies. If you don't, check out the OIL Home Page or the DAML Home Page.

System Requirements

OilEd requires Java1.2. If you wish to use the reasoning services, you will also need an installation of the CORBA-FaCT reasoner (available from `http://www.cs.man.ac.uk/~horrocks/software`). This is now bundled with the OilEd distribution. All of OilEd's editing functionality will be available without the reasoner, but you won't be able to verify models.

Installing OilEd

OilEd comes as a gzipped tar archive. If you've got as far as reading this, then you've probably been able to install it!

Acknowledgements

The original development of this tool was supported by Interprice GmbH and the Free University of Amsterdam.

RDF support is provided through Sergey Melnik's implementation of an RDI API. More details on this are available from:
`http://www-db.stanford.edu/~melnik/rdf/api.html`

The tool also makes use of the Jena API from HewlettPackard. More information on Jena is available from:

`http://www.hpl.hp.com/semweb/jena-top.html`

The look and feel of the editor has been strongly influenced by existing tools such as Protégé 2000.

This product includes software developed by the Apache Software Foundation (`http://www.apache.org`).

Warnings

> This is not a finished product. It has not been thoroughly tested. It lacks much of the necessary functionality required of a full scale ontology editor. It will almost certainly break or crash at some point and you will lose bits of work. You are advised to save your work frequently to avoid disappointment.

This release of the tool will no doubt contain a number of bugs. Any serious crashes, please report to seanb@cs.man.ac.uk. Note however that we have no maintenance strategy for OilEd, so the software is provided as-is. The tool is limited (see Limitations below), but hopefully offers sufficient functionality for the construction of example ontologies. You are advised to read this manual before using OilEd as it contains important information concerning some of the more idiosyncratic aspects of the tool!

Changes

This version of OilEd contains some significant changes. In particular, the tool is now primarily intended to support DAML+OIL as the input and output format rather than OIL. The Oil text format and OIL RDFS have both been dropped and are no longer supported. Efforts have been made to comply with the current schema as much as possible, and there may well be some incompatabilities with models produced using earlier versions of the tool.

Some of the DAML+OIL renderers in earlier OilEd versions used rdf:ID in some places rather rdf:about. The new Jena-based parser is more strict about the use of such things, so may well complain about ontologies produced by earlier versions. If this happens, replacing rdf:ID with rdf:about will sometimes do the trick.

Multiple class editors can now be used simultaneously.

There are a number of small bug fixes concerning the redrawing of panels when definitions are updated.

The Small Print

Licensing information relevant to the third party components used in OilEd are included in the `licensing` directory.

Main Menus

The menu bar provides access to functionality that in general applies to an entire ontology.

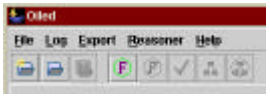

Figure 1 Main Menus

File Operations

The File menu provides a number of items relating to the creation, opening and saving of ontologies. OilEd works on projects, each of which contains an OIL ontology. Multiple projects can be open at one time, but classes cannot be moved between them — editing of each ontology is independent.

Figure 2 File Menu

Options are:

?? New	Start a new, empty, ontology
?? Open local	Open an existing DAML+OIL ontology from a local file.
?? Open URL	Open an ontology from a URL[1].
?? Include local	Include an ontology from a local file.
?? Include URL	Include an ontology from URL.
?? Save Model	Save the current ontology. DAML+OIL is used by default as the representation format. Use Export if you wish to save using a different format.
?? SaveAs Model	Save the current ontology with a new file name
?? Close Model	Close the ontology
?? Preferences	Adjust the editor's behaviour.
?? Exit	Exit from OilEd

Figure 3 File Toolbar Buttons

New, Open Model, Open URL and Save Model are also available via toolbar buttons.

[1] If you have to connect to the web via a proxy, then you will need to set the preferences up accordingly.

Logging

A number of the operations of the editor will cause logging information to be recorded. This will be written to the log file (available in the file OilEdlog.txt in directory logs). The logging information can also be viewed using Show Activity Log in the Log menu.

Ontology Editor

Once an ontology has been loaded or created, the editor has a number of panels showing classes, slots, individuals, axioms and the container. These panels are made up of a number of components, which display lists of expressions, slots or constraints. The panels allow editing of the respective lists, through both right-button menu actions and buttons beneath the panel. Key panels are described below.

Description Editor

A core aspect of the interface is the description editor. This pane will appear in a number of guises and allows the construction and editing of an arbitrary frame description, i.e. a specification of a class that has a number of superclasses and a number of restrictions.

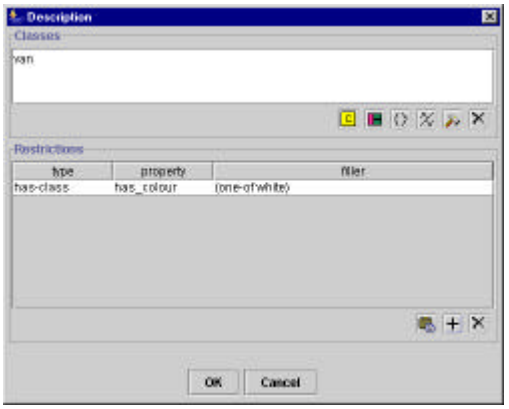

Figure 4 Description Editor

Figure 4 shows a description editor. The upper pane contains a list of superclasses and the lower pane shows a list of restrictions. Both these panes are described in more detail below.

Expression Editor

The expression editor allows the editing of arbitrary class expressions, including boolean expressions such as and/or or not. Figure 5 shows an example of the expression editor.

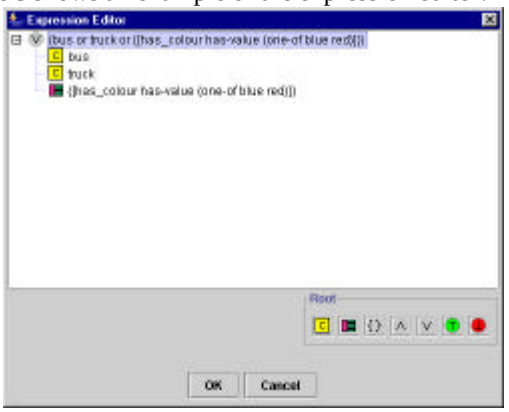

Figure 5 Expression Editor

The editor uses a tree to display the structure of the expression — the example shows an **or** expression with two class names and an arbitrary frame as arguments. Subexpressions can be expanded and collapsed using the standard Jtree controls. A right button menu gives access to various operations

such as adding, removing or editing arguments to a boolean expression as shown in Figure 6. The root of the expression can be reset using the buttons at the bottom of the screen — note that this will replace the entire expression.

Figure 6 Expression Editor menu

Note that boolean expressions such as **and** and **or** must have at least one argument.

Expression List

The expression list holds a number of expressions and allows editing of the list.

Figure 7 Expression List

Figure 7 shows an expression list along with its popup menu (accessible through a right mouse click). the selected expression can be edited, removed or copied, or a previously copied expression can be pasted[2]. Depending on the user preferences (see below), the menu may also include options for adding new expressions. Alternatively, new expressions (class names, frame descriptions or set expressions) can be added to the list using the buttons below the list. If `edit` is selected, an expression editor will be opened. Again, the user preferences may be set to open appropriate editors for each expression type (e.g. a class list, description editor or individuals list), but the default is to open the expression editor. In some cases (e.g. the range of a slot), it may be appropriate to allow concrete type expressions in the expression list, and corresponding options will be available.

Restriction List

The restriction list holds a number of restrictions and allows editing of the list.

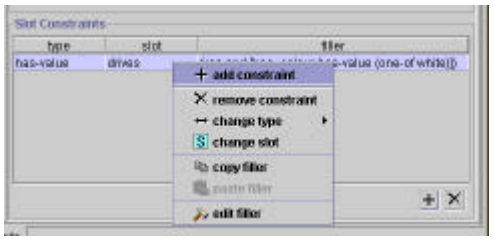

Figure 8 Slot Constraint List

Figure 8 show a restriction list along with its popup menu. New restrictions can be added, and the selected restriction can be removed or edited. When adding a new restriction, a list of properties is offered for selection, then depending on whether the property is an obect or datatype property, an appropriate filler can be selected. Restrictions are has-class (existential quantification) by default but

[2] Note that although multiple ontologies can be open at one time, editing is discrete — each ontology has its own clipboard and expressions cannot be copied between them.

can then be edited. Editing can involve a change to the type[3] of the slot constraint (has-class (existential quantification) to-class (universal quantification), cardinality), a change of the property, in which case a property list is provided for selection, editing of the filler — as with the expression list, an appropriate editor will be opened, copy of the filler to the clipboard or pasting of the filler from the clipboard.

Property List

The property list holds a number of slots and allows editing of the list.

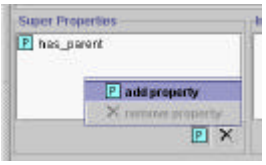

Figure 9 Slot List

Figure 9 shows a property list along with its popup menu. Properties can be added and removed.

Button Panels

For each of the panels described above (Expression List, Restriction List and Property List), buttons below the panel offer immediate access to a number of the functions without having to use the popup menu.

Set Expressions

A set expression is a "one-of" construction and consists of a list of individuals. When editing or creating set expressions, a list of individuals is provided — any selected will be appear in the one-of. Multi selections can be made using the appropriate combination of modifier keys, for example in Windows, holding down <Ctrl> while clicking will select/deselect a single item, while holding down <Shift> will select a range.

Figure 10 Set Expression Selection

Figure 10 shows an individual list with a number of individuals selected corresponding to the set expression one-of (adult, elderly, young).

Datatype Expressions

If a property has been defined as a datatype property, then Restrictions using the slot will have Datatype expressions as fillers. Currently, OilEd only supports the use of simple datatype expressions (integer, real, boolean etc.). These can be selected from a drop down menu when appropriate. If

[3] Slot constraint types can be confusion. An existential constraints asserts that there must be at least one filler with the given type. A universal constraint asserts that all fillers of the slot must be of the given type, but does not necessarily assert the existence of such a filler.

ontologies contain complex XSD expressions, the tool will probably break or behave in a strange manner.

Classes

The class pane displays a list of all classes on the left hand side[4], in alphabetical order. Selecting a class will then display its definition in the right hand pane. The definition consists of a description editor (see above) along with some documentation and a pair of radio buttons indicating whether the class is a *subclass* (necessary conditions) or *same class* (necessary and sufficient conditions)[5]. Although DAML+OIL language supports multiple definitions, OilEd does not and requires each class to have a single definition. If the editor encounters an input file that contains multiple definitions, it will convert the later definitions to axioms [6] — note that this will not in any way change the *semantics* of the ontology as a class definition is simply an alternative presentation of an axiom.

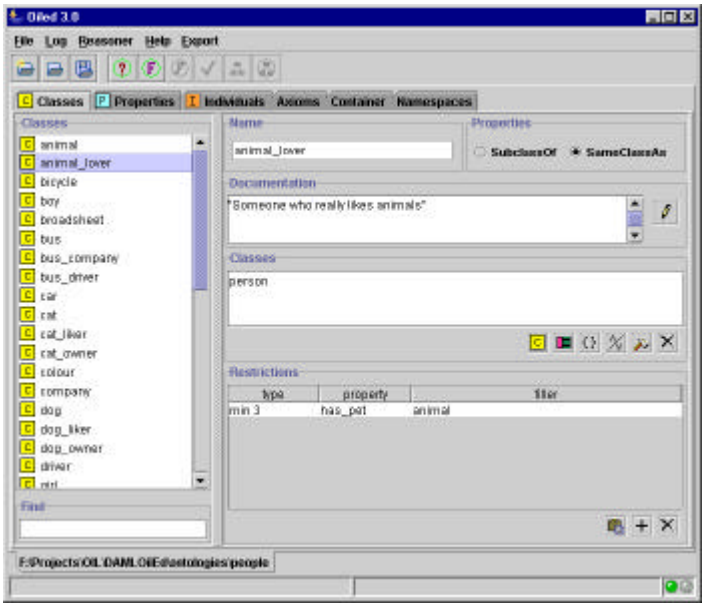

Figure 11 Class Panel

Figure 11 shows the class panel focused on the definition of an animal_lover: a person who has at least 3 pets that are animals. The editing facilities of the expression and slot constraint panels provide direct editing of the class definition. Documentation can be edited by clicking on the button below the documentation field.

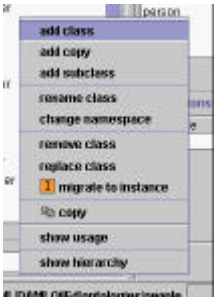

Figure 12 Class List Menu

The class list panel provides a menu allowing addition of classes, renaming of classes and removal. Options here are:

[4] No class hierarchies in the current version.
[5] It is important to understand the differences between defined and primitive classes. This document does not attempt to explain this — consult the OIL documentation for further elaboration on this point.
[6] A warning will be generated in the log.

- add class — add a new class (*subclass* by default).
- copy class — add a new class with the same definition as the currently selected class.
- add subclass — add a new class with the currently selected class as a super.
- rename class — rename the currently selected class.
- change namespace — change the namespace of the currently selected class (see remarks on namespaces below).
- remove class — remove the class from the ontology.
- replace class — replace the currently selected class with another.
- migrate to instance — migrate the class to an instance (see below).
- copy — copy the currently selected class to the clipboard.
- show usage — show where the class is used.
- show hierarchy — show the class's position in the hierarchy.

When adding a class, a name is requested — this must be a new name that is not already used by a class, slot or individual. Similarly when renaming. If classes are renamed, the renaming will be propagated throughout the ontology. Classes cannot be removed if they are used anywhere in the ontology (as part of a definition of class, slot or individual, or within an axiom) — in this case a warning will be given. When replacing classes, the class being replaced can be removed if required. See the comments below concerning class names and the reasoner.

Migration

If the class is not being used by another class, individual or slot, the class can be migrated to an instance. Any superclasses or restrictions will be converted to superclass assertions about the instance.

XML Schema Types

Limited support for XML Schema Types is now included. If a property is defined as being a datatype property (see below), then XML Schema types can be specified for range restrictions or over the property or fillers for restrictions in class definitions. As it currently stands, only simple, atomic types (integer, string, real, anyUri, date and boolean) can be given – these are selected from a drop down menu.

With respect to individuals, simple XML Schema type values can be given. These are simple specified by providing the type (chosen from those listed above), and a literal representing the value. Little, if any, checking is done as to the validity of the value supplied. Reasoning over concrete datatypes is not supported by the FaCT reasoner.

Class Hierarchy

Selecting show hierarchy in the class panel (or double clicking) will bring up a simple hierarchy viewer. This shows the position of a class w.r.t its super and sub classes. Refocusing the hierarchy viewer will refocus the class panel.

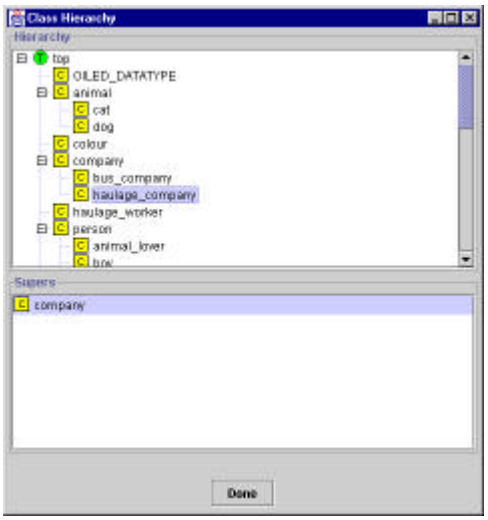

Figure 13 Class Hierarchy Viewer

The upper half of the pane shows the class hierarchy, the lower half shows other supers of the focus concept (the viewer uses a simple tree structure to display the hierarchy, and so cannot show multiple parents)[7]. Clicking a parent in the lower pane will refocus the upper pane to show the focus in the concept of the selected parent. Within the class hierarchy, defined classes (i.e. those which have been asserted to be the sameClassAs some description) will be shown in a slightly darker colour. This can help to spot misclassifications or erroneous definitions.

In addition, if two or more classes have been found to be equivalent during classification, this will be shown in the hierarchy by the inclusion of the equivalent classes in square brackets.

See the Reasoning section below for a discussion of the philosophy behind OilEd's hierarchies.

Properties

The property pane displays a list of all slots on the left hand side, in alphabetical order. Selecting a property will then display its definition in the right hand pane. The definition shows any super or inverse properties, range and domain restrictions. In addition, checkboxes indicate whether the property is transitive, symmetric or functional.

[7] This is very similar to the approach used in Protégé.

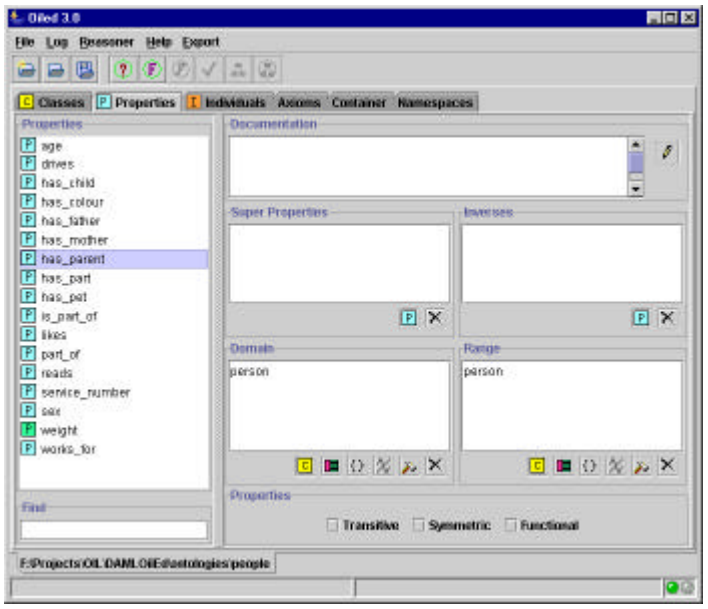

Figure 14 Property Panel

Figure 14 shows the property panel focused on the definition of a property has_parent. This property has a restriction on its domain and range, saying that the only fillers permissible are person.. The editing facilities of the expression and property constraint panels provide direct editing of the property definition. As with the class panel, a popup menu allows addition, renaming and removal of property definitions. Similar restrictions apply to the removal of slots when in use.
In addition, the popup menu allows you to specify whether a property is an object or datatype property. A datatype property can only have datatypes as range restrictions.

Individuals

The individual pane displays a list of all individuals on the left hand side, in alphabetical order. Selecting an individual will then display its definition in the right hand pane. The definition shows any superclasses of the individual along with relationships to other individuals or datatype values.

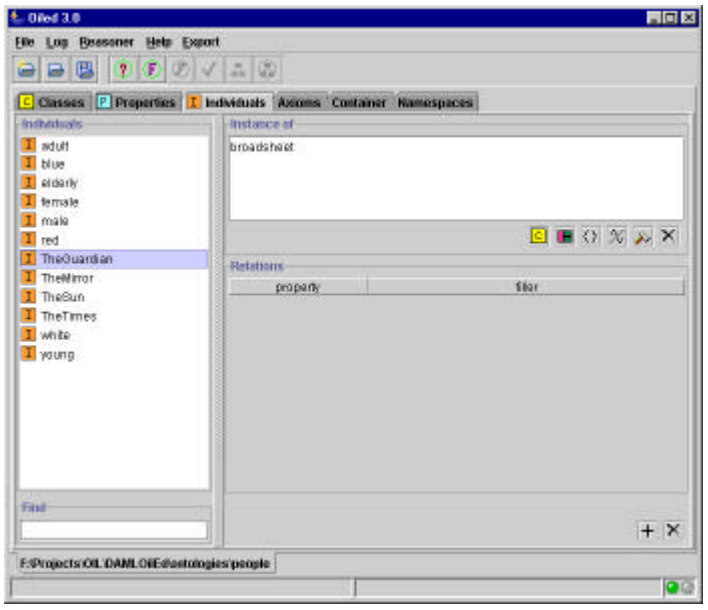

Figure 15 Individual Panel

Figure 15 shows the individual panel focused on the definition of TheGuardian, a broadsheet. The expression and relations list panel allow editing of the individual's definition and a popup menu on the individual list allows addition, removal and renaming as with the class and property panels. Relations

to other individuals can be added and removed. If a datatype property is selected, a datatype value can be entered. This must be explicitly typed. OilEd does not check that the values supplied are sensible.

Axioms

The axiom pane displays a list of all axioms on the left hand side. Selecting an axiom will then display its definition in the right hand pane. Different axioms have slightly different displays, but in general the axiom panels include expression lists for the arguments to the axiom.

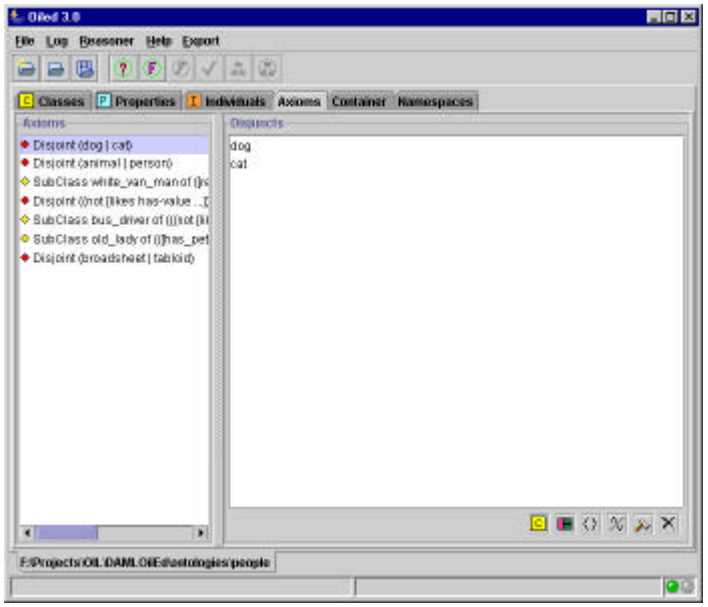

Figure 16 Axiom Panel

Figure 16 shows the axiom panel focused on a disjoint axiom which states that dog and cat are disjoint. The subclass axiom panel includes a check box for recording whether the subclass is disjoint. Panels for disjoint and same class as axioms simply show a list of the disjuncts or same class as which can be edited using the usual editing facilities.

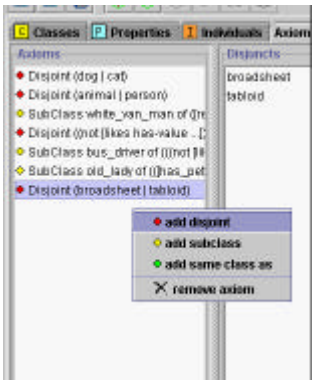

Figure 17 Axiom List Menu

The axiom list provides a popup menu (see Figure 17) which allows addition of new axioms and removal of the selected axiom[8].

[8] Axiom addition is rather clums y at present. Disjointness axioms should have at least two arguments, and same as axioms should have exactly two. This is not enforced at creation time. However, when the editor is asked to save the ontology, it will check that the axioms are valid. If not, a warning is issued and save does not take place. The log will show some indication of the offending axiom.

Container

A default container is written to any new ontologies. The container panel displays the current values of fields in the container.

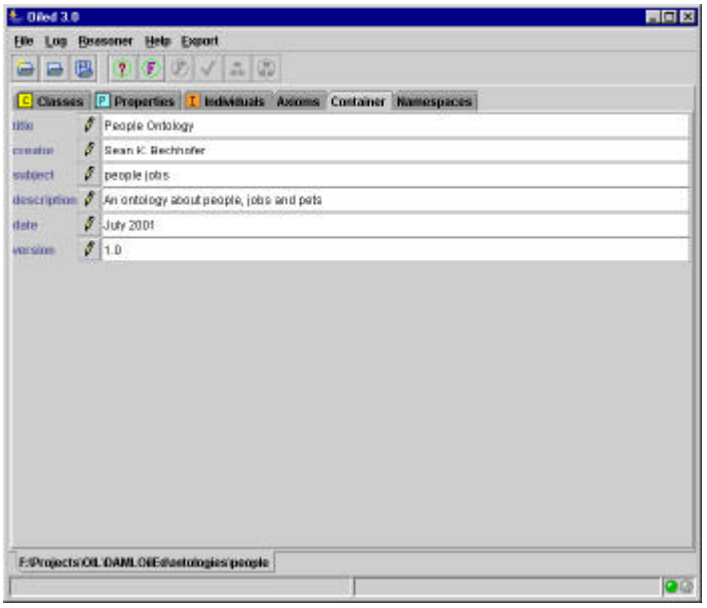

Figure 18 Container Panel

The small pen buttons allow editing of the fields within the container.

Spawning Editors

Earlier versions of OilEd required all the editing of classes to happen within the single pane provided next to the class list. Multiple class editors can now be spawned from the right click menu.

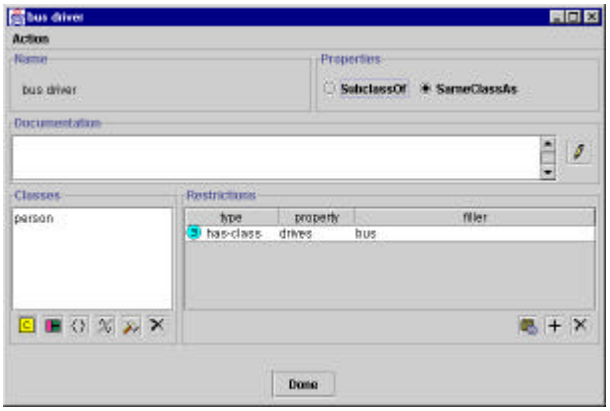

Figure 19 Spawned editor

Figure 19 shows a spawned editor. Any changes made here will propagate through other panels or screens in the application. Class editors can also be launched from the show usage panel. Properties and Individuals cannot be edited in this way at present. A drop down menu gives access to operations such as renaming or adding subclasses.

Reasoning

In order to use OilEd's reasoning capabilites, the CORBA-FaCT reasoner must be installed and running. This document does not provide instructions about the installation and use of CORBA-FaCT, but assumes that the reasoner is installed and working properly. **Please Note:** This version of OilEd

requires an up to date version of the CORBA-FaCT reasoner (version 2.30.5) running SHIQ. An installation of the reasoner is now supplied with OilEd.

OilEd provides a connection with the FaCT reasoner that allows satisfiability checking of models and discovery of implied subsumptions. Please see the note below concerning case sensitivity!

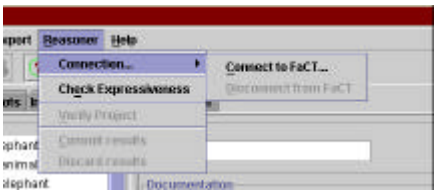

Figure 20 Connecting to FaCT

Use the Connect to FaCT... menu option to connect to a running CORBA-FaCT server. A dialog will prompt for host and port information. Once connected, Verify Project will become active. Selecting verification causes the following:

?? The ontology is translated to SHIQ/FaCT, and this is then transmitted to FaCT for classification
?? Each class in the ontology is checked for satisfiability.
?? For each class in the ontology, its supers (according to the classifier) are determined.

Depending on the operators used in the ontology (to be precise the presence of inverse roles or cardinality restrictions), the classifier may be able to classify using the FaCT reasoner rather than SHIQ (which will produce better performance). Check Expressiveness will check to see which reasoner is required for the current ontology[9].

Figure 21 Checking Expressiveness

Figure 21 shows the results of checking an ontology which requires the SHIQ classifier .

After the verification process, a short report is written to the activity log, recording any unsatisfiable concepts found. In addition, unsatisfiable concepts will now appear in red in the class list.

[9] Note, however, that it's your responsibility to ensure that the reasoner connected to is the appropriate one. Also, be aware that if FaCT is used rather than SHIQ and the ontology contains inverse roles or cardinality restrictions, then the reasoner may fail to spot inconsistencies or subsumptions.

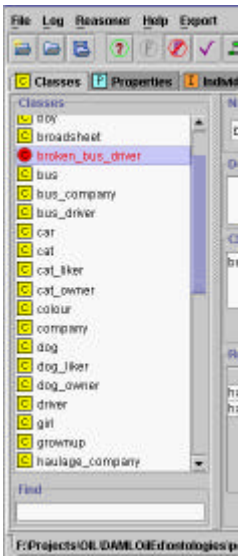

Figure 22 Unsatisfiable Classes

Figure 22 shows the class list after verification. In this case, the class broken_bus_driver has been found to be unsatisfiable. At this point, new subsumption relations and unsatisfiability information can be discarded or added to the model. See the section on Hierarchies below for a detailed discussion of the effects of these actions.

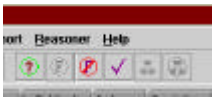

Figure 23 FaCT Toolbar Buttons

Control of the FaCT connection and verification are also available via toolbar buttons.

Hierarchies

The issue of hierarchies is a thorny one and there are still many issues that require resolving concerning the way that OiLEd deals with hierarchies. The current version takes a simple approach to displaying hierarchies, using a tree structure along with extra information about other parents. However, deciding exactly *what* that hierarchy should be raises many interesting questions.

The hierarchy viewer within OilEd is rather primitive. In particular, it's not very good at dealing with situations when there are cycles in the hierarchy (e.g. lots of equivalent concepts). For example, if you assert that a concept is a superclass of itself, that concept with then not appear in the hierarchy. This is an implementation issue that will hopefully be resolved in the near future.

When ontologies are read in, OilEd will warn (through a message in the log) if it encounters any cycles. Similarly, if superclass assertions introduce cycles, you will be warned. If such warning have appeared, it's safest to assume that the hierarchical display will **not** be showing you the true state of affairs.

OilEd without FaCT

If we disregard the FaCT reasoner, the superclasses shown in the concept hierarchy are those classes which have been *explicitly* asserted as being superclasses of a class. Note that OilEd itself does *no* reasoning at all. If a concept `driving_man` has the superclass `(man and driver)`, then the class hierarchy will *not* show `man` or `driver`. If the concept has the two classes `man` and `driver` asserted individually as superclasses, then they *will* appear in the class hierarchy. In addition, OilEd will *not* try and maintain a sparse concept hierarchy. Thus if we add classes `animal`, `mammal` and

dog, and assert that a mammal is a kind of animal, and a dog is both a mammal and an animal, the subclass link between dog and animal (which is now redundant) will still be present, as shown in Figure 24.

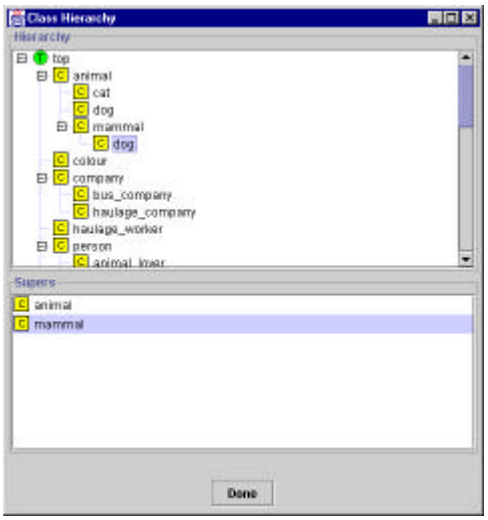

Figure 24 Redundancy in Hierarchies

The hierarchy viewer is also currently unable to cope with circular hierarchies. Thus if we have three classes a, b and c, with a a subclass of b, b a subclass of c and c a subclass of a (in other words they're all equivalent), the classes don't get displayed in the hierarchy. This is a bug.

OiLEd with FaCT

When the reasoner comes into play, the situation becomes rather more complicated. If the reasoner is asked to verify the model, the subclass hierarchy is replaced with the hierarchy determined by the reasoner. Thus in our example above, (but no change is made to the *definitions* of the classes). Figure 25 shows the state of the hierarchy after verification.

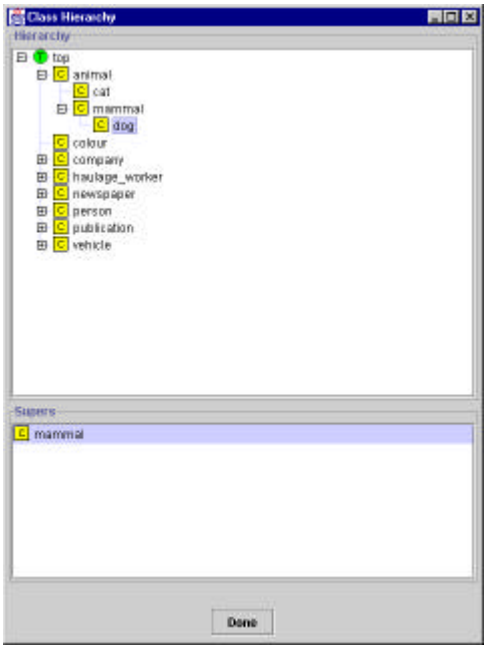

Figure 25 Hierarchy post Verification

At this point, if we elect to Discard changes, the hierarchy will be returned to its original state (and any information about unsatisfiability will be discarded). Alternatively, the subsumption relations can be added to the ontology through Commit changes. This will add new superclasses

corresponding to any discovered relationships that do not already occur. How such superclasses are added is determined by the preference settings. If Sparse Hierarchies is unselected, OiLEd will simply add any superclasses that it finds into the definition — this may result in the creation of non-sparse hierarchies and redundant superclasses. Alternatively, if Sparse Hierarchies is selected, OiLEd will attempt to prune any redundant parents and maintain a sparse hierarchy. Be aware that this behaviour may cause unexpected results.

Identifiers & Case Sensitivity

OilEd as a standalone application is case sensitive. Thus if we introduce classes `Cat`, `cat` and `CAT`, these will all be treated as different classes. The current implementation of the CORBA-FaCT reasoner, however is **not** case sensitive. Thus the reasoner will consider the three classes introduced above to be the same. OilEd attempts to resolve any problems, but this may cause unpredictable results. It is probably best to avoid the use of object names that are distinguishable only through case. In addition, the reasoner will not like class names containing some punctuation characters (for example the colon character `:`). Again, for these reasons, it is best to stick to alphanumeric names, using underscores in multi-word indentifiers.

Export

By default, OilEd uses DAML+OIL as it's format for storing ontologies. OilEd can also export ontologies in a number of different formats.

Figure 26 Export menu

Renderers which may be available include:

?? Simple RDFS	The ontology is exported as a simple RDFS description. The class hierarchy is simply exported as `rdfs:subClassOf` triples. Slots are also output along with `rdf:range` and `rdf:domain` expressions where possible.
?? DAML+OIL	The ontology is exported as a DAML-OIL description (both the schema from 2000/12 and 2001/03 are available).
?? SHIQ	The ontology is exported as a SHIQ knowledge base which can be read into FaCT at a later date for classificaton[10].
?? SHOQ(D)	The ontology is exported as a SHOQ(D) knowledge base which can be read into FaCT at a later date for classification.
?? HTML	A simple HTML rendering of the ontology is exported. This can be useful for static viewing of ontologies or for display on a web site.
?? Dotty	The basic class hierarchy is exported in a format that Dotty (AT&T's grapher) can read. If the model has been verified, the graph will

[10] The XML SHIQ knowledge bases produced by OilEd have no DTD declarations. This may cause a slight problem with the earlier releases of the FaCT demo client. If you have problems trying to compile SHIQ files with the original client, try inserting the text
`<!DOCTYPE KNOWLEDGEBASE SYSTEM "fact.dtd">`
after the initial `<?xml…>` declaration (ensuring that `fact.dtd` is in the same directory as the knowledge base).

represent the state of the inferred hierarchy. Note that Dotty's parser is rather fussy and objects to hyphens in node names. For this reason it's better to use underscores in class names.

?? Individual Graph — The graph formed by the individuals and relations between them is writtned out in Dotty format.

DAML-OIL output uses the schema:

`http://www.daml.org/2001/03/daml+oil`

Vanilla RDFS descriptions can be read — any resources that occur from outside of the RDF, RDFS and OIL namespaces will be ignored.

Namespaces

> Namespace support in OilEd is a little tricky, so you are advised to read the following section carefully!

Namespaces allow us to disambiguate between things that may use the same name. OilEd attempts to support this, and in doing so is explicit about the namespaces that it uses within ontologies. When reading an ontology, OilEd initially considers the URL from whence the ontology was obtained as the "default namespace". In the case of an ontology opened with Open…, this will be a file: URL, in the case of one opened with Open URL, this will be the URL supplied. If a New ontology is produced, the default namespace will initially be empty. If the file is saved, the default namespace will be assigned to be a file: URL based on the file being saved to. Alternatively, you can explicitly set the default namespace before saving (through the namespace panel) in which case the value you supply will be used.

Any classes, properties or individuals occuring within OilEd's default namespace will simply be shown as undecorated names. Any others will be given a suffix that indicates which namespace they come from[11].

Note that OilEd's default namespace is **not** the same as the "default" namespace that you may supply within an RDF description using an `xmlns="xxx"` declaration. Any classes, properties or individuals defined using the default namespace will simply be shown using their names. Any that come from other namespaces will be suffixed with a number corresponding to that namespace. The actual URLs for the namespaces can be edited using the namespace panel (Figure 27). No attempt is currently made to check whether the namespaces are syntactically valid. In addition, any of the namespaces can be set as the "default" – this will make no difference to the ontology other than changing the rendering of the objects *within* OilEd.

[11] This is, admittedly, not ideal, but it's a short term solution to a tricky problem.

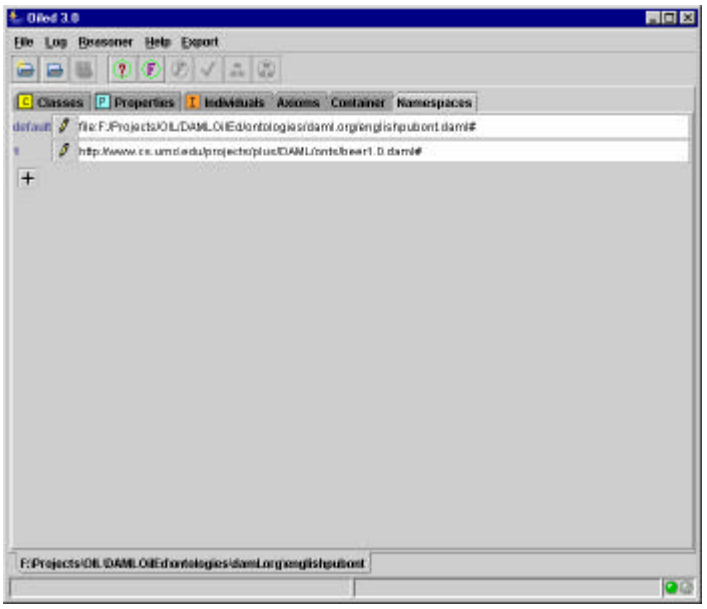

Figure 27 Namespace Panel

The namespace used for each class, property or individual can be altered using the list menu. You can then pick from the namespaces in use, as shown in Figure 28.

Figure 28 Namespace selection

When ontologies are written out by OilEd, every class, property and individual will be given an explicit URI based on its namespace and name.

Important!

Note that the namespaces used for concepts within an ontology are not explicitly connected or related to the URL of the ontology that they are being used in. Although, in general, it may be the case that the concept http://ontologies-r-us/ontologies/animals#Cat is defined in the ontology defined http://ontologies-r-us/ontologies/animals *this is simply convention*, and there's nothing to stop us supplying definitions for any classes within an ontology.

Including Ontologies

Ontologies can be included. When an ontology is included, any classes, properties and individuals not already in the ontology will be added along with their definitions. If information in the included ontology relates to an existing class (for example, the included ontology may also contain a definition for a particular class), that information will be added as an axiom of the existing class already has a definition. No attempt is made to prevent duplication, and no check is made at inclusion time that conflicts are not being introduced.

User Preferences

A limited amount of customisation is possible. This is accessible via the Preferences panel as shown in Figure 29.

Figure 29 Preferences Panel

Current options available are:

- Big Menus — Menu actions for addition of classes, frames, expressions and set expressions will be included in the expression list panel.
- Sensitive Menus — Appropriate editors will be opened up on expressions in the expression list panel. If this is unchecked, whenever an expression is to be edited, the generic expression editor will appear allowing the editing of the expressions. If checked, an editor targeted at that particular type of expression will instead be used. So, for example if the expression is a class, the class picking list will be presented. The use of the sensitive editor can make small changes easier, but may make more complex more difficult — if selected, it can be hard to migrate from, say, a class expression to a frame description. Any change to the checkbox will take place immediately.
- Allow Spaces — Allow the use of spaces in concept names (will cause problems with Oil text format).
- Show Inherited Properties — Show any inherited restrictions on classes in the class description pane (the display will not be editable).
- Strict Export Extensions — If checked, when exporting files, filenames have to have the corresponding file extension. If unchecked, filenames for export can have any extension.
- Include Top — Include the top concept in any graphs produced (GML/Dotty).
- Reasoner defaults — Set the host and port information for the reasoner.
- Check
  - Subsumption — OilEd will check the satisfiability of concepts and look for implied subsumption relationships.
  - Satisfiability — OilEd will only check for satisfiability.
  - Subsumption — OilEd will do no checking, but simple sends the model to the reasoner .
- Sparse Hierarchies — Controls the behaviour when adding inferred super classes. If checked, OilEd will remove any superclasses which are redundant.
- HTTP Proxy — Sets values for an http proxy. You may need to set this if you are behind a firewall and wish to access ontologies via URLs.

In addition, the configuration file `config.xml` specifies a number of options used by the application at start up, including default values for the reasoner location, http proxy and location of temporary and log files.

Plugins

There is a (kind of) plugin architecture parsers and renderers, and a certain amount of configuration is possible through the `config.xml` file. Currently two simple plugins are available. These are still very much prototypes and offered limited functionality.

Lexicon

This allows the description of a basic lexicon, i.e. a mapping from concepts to strings used to render the concept. These lexicons can then be used within the COHSE Ontology Service to aid in mapping from terms to concepts.

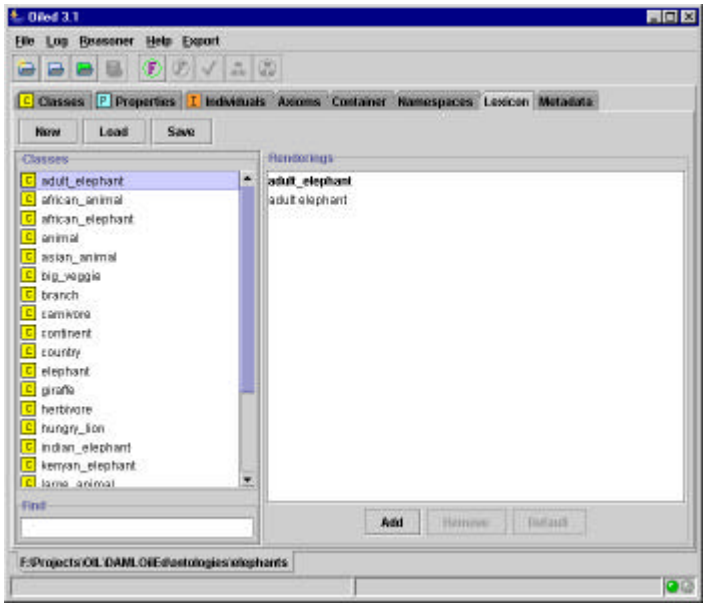

Figure 30 Lexicon Plugin

The functionality of the lexicon is very simple. For the selected class (only classes are mapped at present), the current mappings are shown on the right. Renderings can be added or removed, and a default can be set. The lexicon can then be saved in XML format (see the COHSE OS documentation for further discussion of this). Existing lexicons can be loaded in. Default lexicons can be generated – these will try some simple heuristics to produce strings (such as mapping to lower case and splitting inner capitalization, e.g. blackCat maps to “black cat”).

Metadata

Simple metadata for classes can be recorded. The creation date and creator are automatically logged by OilEd and cannot be changed via the interface. Additional information about the editorial status of the class can be added as free text. This will be appended to any existing history along with the date and the creator of the comment. If the ontology is saved as DAML+OIL (2001/03 schema), the editorial status will also be saved, using the OilEd RDF Schema.

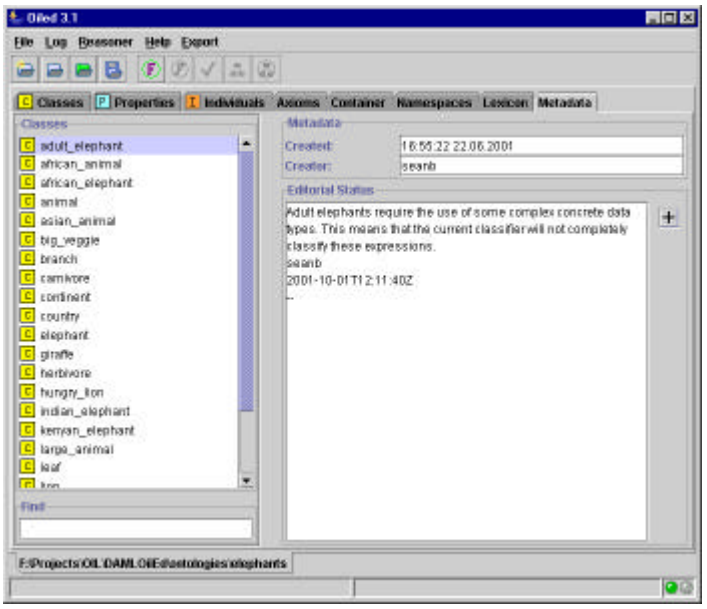

Figure 31 Metadata

Limitations and Known Bugs

This is a (non exhaustive) list of the limitations of the current version of OilEd. Most of these will be fixed during the next few iterations of development, however, some (such as import, include and the issues around modularisation) are longer term issues that will not be fully addressed.

?? No hierarchical slot displays.
?? The class hierarchy viewer can be temperamental. In particular, if the class hierarchy contains cycles, classes in the cycle may not show up in the hierarchy.
?? Concrete type support is miminal.
?? No undo. In particular be aware that if a popup editor has no `cancel` button, the changes you are making happen *immediately*.
?? Axiom editing is primitive. The covered expression cannot be edited in a covering expression.